

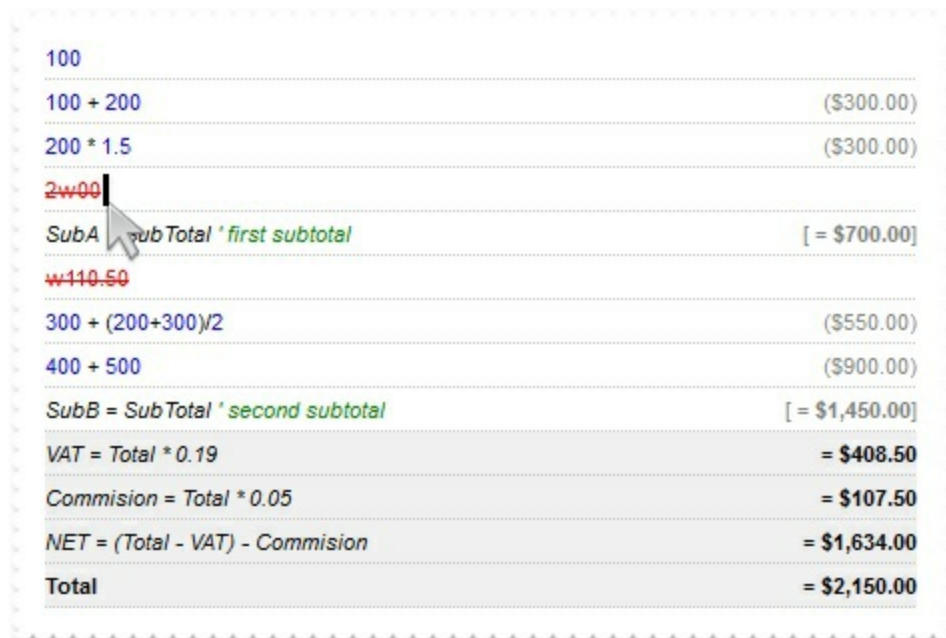


ExCalcEdit

Enrich your application with an easy-to-use edit control that supports arithmetic operations. Get results on the fly. The result is displayed as the user types the expression. The control handles double constants and arithmetic operations like + (addition), - (subtraction), / (division), or * (multiply). To enforce a priority, you can use parentheses (). The control is fully written in C++ using ATL and STL. The control is compatible with languages like VB, VB.NET, VBA, C++, C#, VFP, Access, HTML, and so on. The control doesn't require additional files or libraries like MFC, VB runtime, or else.

Features include:

- Standard or Complex Arithmetic operations
- Single or Multiple-Lines support
- Ability to define expressions using variables
- Total, SubTotal, Count and SubCount aggregate functions support
- Prefixes, Comments support
- Read-Only / Locked Text support
- Highly customizable format to display the results
- and more



100	
100 + 200	(\$300.00)
200 * 1.5	(\$300.00)
2w00	
SubA = SubTotal * first subtotal	[= \$700.00]
w110.50	
300 + (200+300)/2	(\$550.00)
400 + 500	(\$900.00)
SubB = SubTotal * second subtotal	[= \$1,450.00]
VAT = Total * 0.19	= \$408.50
Commision = Total * 0.05	= \$107.50
NET = (Total - VAT) - Commision	= \$1,634.00
Total	= \$2,150.00

Ž ExCalcEdit 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 control's appearance. Use the [Appearance](#) property to specify the control's appearance.

Name	Value	Description
None2	0	No border
Flat	1	Flat border
Sunken	2	The border has sunken border.
Raised	3	The border has raised border.
Etched	4	Etched border
Bump	5	Bump border

constants CalcTypeEnum

The CalcTypeEnum type indicates the operation the control supports. The [CalcType](#) property specifies the type of the control. The CalcTypeEnum type supports the following values:

Name	Value	Description
exCalcStandard	0	Allows only arithmetic operations.
exCalcIncludeAll	-1	Allows all operations, operators and functions.

constants `PictureDisplayEnum`

Specifies how a picture object is displayed.

Name	Value	Description
<code>exUpperLeft</code>	0	Aligns the picture to the upper left corner.
<code>exUpperCenter</code>	1	Centers the picture on the upper edge.
<code>exUpperRight</code>	2	Aligns the picture to the upper right corner.
<code>exMiddleLeft</code>	16	Aligns horizontally the picture on the left side, and centers the picture vertically.
<code>exMiddleCenter</code>	17	Puts the picture on the center of the source.
<code>exMiddleRight</code>	18	Aligns horizontally the picture on the right side, and centers the picture vertically.
<code>exLowerLeft</code>	32	Aligns the picture to the lower left corner.
<code>exLowerCenter</code>	33	Centers the picture on the lower edge.
<code>exLowerRight</code>	34	Aligns the picture to the lower right corner.
<code>exTile</code>	48	Tiles the picture on the source.
<code>exStretch</code>	49	The picture is resized to fit the source.

CalcEdit 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: {0D4EE794-3E13-4226-81F9-499EE6EDCCF7}. The object's program identifier is: "Exontrol.CalcEdit". The /COM object module is: "ExCalcEdit.dll"

The eXCalcEdit is an easy-to-use edit control that supports arithmetic operations. The Exontrol's eXCalcEdit control supports the following properties and methods:

Name	Description
AddDecimalSep	Specifies an additional decimal separator.
AddWildFormat	Formats the line based on the giving wild characters expression.
AllowComments	Specifies the HTML caption that starts the comment of the line. If empty, no comments are allowed.
AllowCount	Specifies the keyword that makes the control to display the count all lines being counted in a Total group.
AllowFormatInvalidOnTyping	Specifies whether the FormatInvalid property is applied on the current line, while typing into the control.
AllowPrefixes	Specifies the HTML caption that ends the prefix of the line. If empty, no prefixes are allowed.
AllowSubCount	Specifies the keyword that makes the control to display the subcounts.
AllowSubTotal	Specifies the keyword that makes the control to display the subtotals.
AllowTotal	Specifies the keyword that makes the control to display the sum/total of all lines.
AllowUndoRedo	Specifies whether the control allows undo/redo actions.
AllowVariables	Specifies the expression (no HTML) that defines the equal operator, so you can define variables.
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.
BackColorLockedLine	Retrieves or sets a value that indicates the line's background color when it is locked.
BackColorSubTotal	Specifies the background color to show the SubTotal lines.
BackColorTotal	Specifies the background color to show the Total line.

CalcType	Specifies the type of operations the control support.
CanRedo	Determines if the redo queue contains any actions.
CanUndo	Determines whether the last edit operation can be undone.
CaretLine	Indicates the line that displays the caret.
CaretPos	Retrieves or sets a value that indicates the position of the caret in the line.
ClearWildFormats	Clears the wild characters expressions collection into a sensitive control.
Count	Counts the lines in the control.
DeleteWildFormat	Deletes an entry from the wild characters expressions collection.
DrawGridLines	Returns or sets a value that determines whether lines are drawn between rows, or unpopulated areas.
Enabled	Enables or disables the control.
EvaluateSel	Specifies whether the control evaluates the selection.
ExecuteTemplate	Executes a template and returns the result.
Export	Exports the control's content as text, including the results.
Font	Retrieves or sets the control's font.
ForeColor	Specifies the control's foreground color.
ForeColorLockedLine	Retrieves or sets a value that indicates the line's foreground color when it is locked.
FormatABC	Formats the A,B,C values based on the giving expression and returns the result.
FormatCountResult	Specifies the HTML format to display the result of a Count line.
FormatInvalid	Specifies the HTML format to show invalid lines.
FormatLocal	Indicates the expression that defines the formatted value being replaced in FormatResult properties, when %I% is found.
FormatNumbers	Specifies the HTML format that's applied to numbers.
FormatResult	Specifies the HTML format of the result.
FormatSubCountResult	Specifies the HTML format to display the result of a SubCount line.
	Specifies the HTML format to display the result of a

[FormatSubTotalResult](#)

SubTotal line.

[FormatTotalResult](#)

Specifies the HTML format to display the result of a Total line.

[GridLineColor](#)

Specifies the grid line color.

[HideSelection](#)

Specifies whether the selection in the control is hidden when the control loses the focus.

[hWnd](#)

Retrieves the control's window handle.

[InsertLockedText](#)

Inserts locked text to the control.

[InsertText](#)

Inserts text to control.

[IsValid](#)

Specifies whether the expression is valid.

[LineHeight](#)

Specifies an expression that determines the height of the line within the editor.

[Locked](#)

Determines whether a control can be edited.

[Margin](#)

Defines the distance between text and inner border.

[MultiLine](#)

Specifies whether the control accepts multiple lines.

[Overtyp](#)

Specifies whether the control is running in overtyp mode.

[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

[Redo](#)

Redoes the next action in the control's redo queue.

[Refresh](#)

Refreshes the control.

[Result](#)

Retrieves the result.

[SelBackColor](#)

Specifies the selection's background color.

[SelForeColor](#)

Specifies the selection's foreground color.

[SelLength](#)

Returns or sets the number of characters selected.

[SelStart](#)

Returns or sets the starting point of text selected; indicates the position of the insertion point if no text is selected.

[SelText](#)

Returns or sets the string containing the currently selected text.

[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.

[Text](#)

Specifies the control's text.

[TextLine](#)

Specifies the line based on its index.

[Undo](#)

Call this function to undo the last edit-control operation.

[UseTabKey](#)

Specifies whether the control uses the TAB key.

[Variable](#)

Indicates the value of the specified variable.

[Version](#)

Retrieves the control's version.

property CalcEdit.AddDecimalSep as String

Specifies an additional decimal separator.

Type	Description
String	A String expression that defines the additional decimal separator.

By default, the AddDecimalSep property is ".", which indicates that the dot character is the default decimal separator. For instance, use the AddDecimalSep property on "1,234" to define the comma character as being your decimal separator.

method CalcEdit.AddWildFormat (Expression as String)

Formats the line based on the giving wild characters expression.

Type	Description
Expression as String	A string expression that specifies the HTML format for a wild characters expression. The wild characters supported are '*' and '?'. Also the wild expression supports escaped characters, AddWild("*") bolds the * character only, not including the rest of the line, while AddWild("**") bolds everything after a * character.

By default, the control has already the wild format defined as "<i>*=*</i>", which draws in italics any line that includes the = (equal) character (define the variables). The [AllowVariables](#) property specifies the expression (no HTML) that defines the equal operator, so you can define variables. The AddWild method adds an expression that may contain wild characters like '*' or '?'. Use the [FormatNumbers](#) property to specify the format of the numbers in the control. The [FormatResult](#) property specifies the HTML format of the result. The [FormatInvalid](#) property specifies the HTML format to show invalid lines. The [FormatTotalResult](#) / [FormatSubTotalResult](#) property formats the Total / Sub-Total lines. The [FormatCountResult](#) / [FormatSubCountResult](#) property specifies the HTML format of lines that contains Count or SubCount aggregate function.

The list of supported built-in HTML tags is:

- **bold**
- **<i>italic</i>**
- **<s>strikeout</s>**
- **<u>underline</u>**
- **<fgcolor=RRGGBB>fgcolor</fgcolor>**
- **<bcolor=RRGGBB>bgcolor</bgcolor>**
- **text ** 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.

The DeleteWildFormat deletes an entry from the wild characters expressions collection. Use the [ClearWildFormats](#) method to clear the all wild characters expressions. The [Refresh](#) method should be called after DeleteWild method was called to reflect the latest changes.

The following samples show how you can define new variables using the "is" keyword, and highlight lines that includes it:

A is 200

B is A + 0.22 [=200.22]

A + B [=400.22]

B is B * .19 [=38.04]

A + B [=238.04]

VBA (MS Access, Excell...)

```
With CalcEdit1
    .MultiLine = True
    .AllowVariables = "is"
    .ClearWildFormats
    .AddWildFormat "<b>*is*"
    .Text = "A is 200"
    .InsertText ""
    .InsertText "B is A + 0.22"
    .InsertText "A + B"
    .InsertText "B is B * .19"
    .InsertText "A + B"
End With
```

VB6

```
With CalcEdit1
    .MultiLine = True
    .AllowVariables = "is"
    .ClearWildFormats
    .AddWildFormat "<b>*is*"
    .Text = "A is 200"
    .InsertText ""
    .InsertText "B is A + 0.22"
    .InsertText "A + B"
    .InsertText "B is B * .19"
    .InsertText "A + B"
End With
```

VB.NET

```
With Excalcedit1
    .MultiLine = True
```

```

.AllowVariables = "is"
.ClearWildFormats()
.AddWildFormat("<b>*is*")
.Text = "A is 200"
.InsertText("")
.InsertText("B is A + 0.22")
.InsertText("A + B")
.InsertText("B is B * .19")
.InsertText("A + B")
End With

```

VB.NET for /COM

```

With AxCalcEdit1
.MultiLine = True
.AllowVariables = "is"
.ClearWildFormats()
.AddWildFormat("<b>*is*")
.Text = "A is 200"
.InsertText("")
.InsertText("B is A + 0.22")
.InsertText("A + B")
.InsertText("B is B * .19")
.InsertText("A + B")
End With

```

C++

```

/*
Copy and paste the following directives to your header file as
it defines the namespace 'EXCALCEDITLib' for the library: 'ExCalcEdit 1.0 Control
Library'

#import <ExCalcEdit.dll>
using namespace EXCALCEDITLib;
*/
EXCALCEDITLib::ICalcEditPtr spCalcEdit1 = GetDlgItem(IDC_CALCEDIT1)-
>GetControlUnknown();

```

```

spCalcEdit1->PutMultiLine(VARIANT_TRUE);
spCalcEdit1->PutAllowVariables(L"is");
spCalcEdit1->ClearWildFormats();
spCalcEdit1->AddWildFormat(L"<b>*is*");
spCalcEdit1->PutText(L"A is 200");
spCalcEdit1->InsertText(L"",vtMissing);
spCalcEdit1->InsertText(L"B is A + 0.22",vtMissing);
spCalcEdit1->InsertText(L"A + B",vtMissing);
spCalcEdit1->InsertText(L"B is B * .19",vtMissing);
spCalcEdit1->InsertText(L"A + B",vtMissing);

```

C++ Builder

```

CalcEdit1->MultiLine = true;
CalcEdit1->AllowVariables = L"is";
CalcEdit1->ClearWildFormats();
CalcEdit1->AddWildFormat(L"<b>*is*");
CalcEdit1->Text = L"A is 200";
CalcEdit1->InsertText(L"",TNoParam());
CalcEdit1->InsertText(L"B is A + 0.22",TNoParam());
CalcEdit1->InsertText(L"A + B",TNoParam());
CalcEdit1->InsertText(L"B is B * .19",TNoParam());
CalcEdit1->InsertText(L"A + B",TNoParam());

```

C#

```

excalcredit1.MultiLine = true;
excalcredit1.AllowVariables = "is";
excalcredit1.ClearWildFormats();
excalcredit1.AddWildFormat("<b>*is*");
excalcredit1.Text = "A is 200";
excalcredit1.InsertText("",null);
excalcredit1.InsertText("B is A + 0.22",null);
excalcredit1.InsertText("A + B",null);
excalcredit1.InsertText("B is B * .19",null);
excalcredit1.InsertText("A + B",null);

```

JScript/JavaScript

```
<BODY onload="Init()">
<OBJECT CLASSID="clsid:0D4EE794-3E13-4226-81F9-499EE6EDCCF7"
id="CalcEdit1"></OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    CalcEdit1.MultiLine = true;
    CalcEdit1.AllowVariables = "is";
    CalcEdit1.ClearWildFormats();
    CalcEdit1.AddWildFormat("<b>*is*");
    CalcEdit1.Text = "A is 200";
    CalcEdit1.InsertText("",null);
    CalcEdit1.InsertText("B is A + 0.22",null);
    CalcEdit1.InsertText("A + B",null);
    CalcEdit1.InsertText("B is B * .19",null);
    CalcEdit1.InsertText("A + B",null);
}
</SCRIPT>
</BODY>
```

VBScript

```
<BODY onload="Init()">
<OBJECT CLASSID="clsid:0D4EE794-3E13-4226-81F9-499EE6EDCCF7"
id="CalcEdit1"></OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With CalcEdit1
        .MultiLine = True
        .AllowVariables = "is"
        .ClearWildFormats
```

```

.AddWildFormat "<b>*is*"
.Text = "A is 200"
.InsertText ""
.InsertText "B is A + 0.22"
.InsertText "A + B"
.InsertText "B is B * .19"
.InsertText "A + B"
End With
End Function
</SCRIPT>
</BODY>

```

C# for /COM

```

axCalcEdit1.MultiLine = true;
axCalcEdit1.AllowVariables = "is";
axCalcEdit1.ClearWildFormats();
axCalcEdit1.AddWildFormat("<b>*is*");
axCalcEdit1.Text = "A is 200";
axCalcEdit1.InsertText("",null);
axCalcEdit1.InsertText("B is A + 0.22",null);
axCalcEdit1.InsertText("A + B",null);
axCalcEdit1.InsertText("B is B * .19",null);
axCalcEdit1.InsertText("A + B",null);

```

X++ (Dynamics Ax 2009)

```

public void init()
{
    ;

    super();

    excalcedit1.MultiLine(true);
    excalcedit1.AllowVariables("is");
    excalcedit1.ClearWildFormats();

```



```
excalcedit1.AddWildFormat("<b>*is*");
excalcedit1.Text("A is 200");
excalcedit1.InsertText("");
excalcedit1.InsertText("B is A + 0.22");
excalcedit1.InsertText("A + B");
excalcedit1.InsertText("B is B * .19");
excalcedit1.InsertText("A + B");
}
```

Delphi 8 (.NET only)

```
with AxCalcEdit1 do
begin
  MultiLine := True;
  AllowVariables := 'is';
  ClearWildFormats();
  AddWildFormat('<b>*is*');
  Text := 'A is 200';
  InsertText('',Nil);
  InsertText('B is A + 0.22',Nil);
  InsertText('A + B',Nil);
  InsertText('B is B * .19',Nil);
  InsertText('A + B',Nil);
end
```

Delphi (standard)

```
with CalcEdit1 do
begin
  MultiLine := True;
  AllowVariables := 'is';
  ClearWildFormats();
  AddWildFormat('<b>*is*');
  Text := 'A is 200';
  InsertText('',Null);
  InsertText('B is A + 0.22',Null);
  InsertText('A + B',Null);
  InsertText('B is B * .19',Null);
```

```
InsertText('A + B',Null);  
end
```

VFP

```
with thisform.CalcEdit1  
  .MultiLine = .T.  
  .AllowVariables = "is"  
  .ClearWildFormats  
  .AddWildFormat("<b>*is*")  
  .Text = "A is 200"  
  .InsertText("")  
  .InsertText("B is A + 0.22")  
  .InsertText("A + B")  
  .InsertText("B is B * .19")  
  .InsertText("A + B")  
endwith
```

dBASE Plus

```
local oCalcEdit  
  
oCalcEdit = form.EXCALCEDITACTIVEXCONTROL1.nativeObject  
oCalcEdit.MultiLine = true  
oCalcEdit.AllowVariables = "is"  
oCalcEdit.ClearWildFormats()  
oCalcEdit.AddWildFormat("<b>*is*")  
oCalcEdit.Text = "A is 200"  
oCalcEdit.InsertText("")  
oCalcEdit.InsertText("B is A + 0.22")  
oCalcEdit.InsertText("A + B")  
oCalcEdit.InsertText("B is B * .19")  
oCalcEdit.InsertText("A + B")
```

XBasic (Alpha Five)

```
Dim oCalcEdit as P
```

```
oCalcEdit = topparent:CONTROL_ACTIVEX1.activex
oCalcEdit.MultiLine = .t.
oCalcEdit.AllowVariables = "is"
oCalcEdit.ClearWildFormats()
oCalcEdit.AddWildFormat("<b>*is*")
oCalcEdit.Text = "A is 200"
oCalcEdit.InsertText("")
oCalcEdit.InsertText("B is A + 0.22")
oCalcEdit.InsertText("A + B")
oCalcEdit.InsertText("B is B * .19")
oCalcEdit.InsertText("A + B")
```

Visual Objects

```
oDCOCX_Exontrol1:MultiLine := true
oDCOCX_Exontrol1:AllowVariables := "is"
oDCOCX_Exontrol1:ClearWildFormats()
oDCOCX_Exontrol1:AddWildFormat("<b>*is*")
oDCOCX_Exontrol1:Text := "A is 200"
oDCOCX_Exontrol1:InsertText("",nil)
oDCOCX_Exontrol1:InsertText("B is A + 0.22",nil)
oDCOCX_Exontrol1:InsertText("A + B",nil)
oDCOCX_Exontrol1:InsertText("B is B * .19",nil)
oDCOCX_Exontrol1:InsertText("A + B",nil)
```

PowerBuilder

```
OleObject oCalcEdit
```

```
oCalcEdit = ole_1.Object
oCalcEdit.MultiLine = true
oCalcEdit.AllowVariables = "is"
oCalcEdit.ClearWildFormats()
oCalcEdit.AddWildFormat("<b>*is*")
```

```
oCalcEdit.Text = "A is 200"  
oCalcEdit.InsertText("")  
oCalcEdit.InsertText("B is A + 0.22")  
oCalcEdit.InsertText("A + B")  
oCalcEdit.InsertText("B is B * .19")  
oCalcEdit.InsertText("A + B")
```

Visual DataFlex

```
Procedure OnCreate  
  Forward Send OnCreate  
  Set ComMultiLine to True  
  Set ComAllowVariables to "is"  
  Send ComClearWildFormats  
  Send ComAddWildFormat "<b>*is*"   
  Set ComText to "A is 200"  
  Send ComInsertText "" Nothing  
  Send ComInsertText "B is A + 0.22" Nothing  
  Send ComInsertText "A + B" Nothing  
  Send ComInsertText "B is B * .19" Nothing  
  Send ComInsertText "A + B" Nothing  
End_Procedure
```

XBase++

```
#include "AppEvent.ch"  
#include "ActiveX.ch"  
  
PROCEDURE Main  
  LOCAL oForm  
  LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL  
  LOCAL oCalcEdit  
  
  oForm := XbpDialog():new( AppDesktop() )  
  oForm:drawingArea:clipChildren := .T.  
  oForm:create( ,, {100,100}, {640,480},,, .F. )  
  oForm:close := {|| PostAppEvent( xbeP_Quit )}
```

```

oCalcEdit := XbpActiveXControl():new( oForm:drawingArea )
oCalcEdit:CLSID := "Exontrol.CalcEdit.1" /*{0D4EE794-3E13-4226-81F9-
499EE6EDCCF7}*/
oCalcEdit:create(, {10,60},{610,370} )

oCalcEdit:MultiLine := .T.
oCalcEdit:AllowVariables := "is"
oCalcEdit:ClearWildFormats()
oCalcEdit:AddWildFormat("<b>*is*")
oCalcEdit:Text := "A is 200"
oCalcEdit:InsertText("")
oCalcEdit:InsertText("B is A + 0.22")
oCalcEdit:InsertText("A + B")
oCalcEdit:InsertText("B is B * .19")
oCalcEdit:InsertText("A + B")

oForm:Show()
DO WHILE nEvent != xbeP_Quit
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
    oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO
RETURN

```

property CalcEdit.AllowComments as String

Specifies the HTML caption that starts the comment of the line. If empty, no comments are allowed.

Type	Description
String	A string expression that defines the HTML caption that starts the comment of the line

By default, the AllowComments property is "", which indicates that the control supports no comments. A line can have a prefix delimited by the [AllowPrefixes](#) property, and can have a comment delimited by the AllowComments property. The prefix of the line and its comment are never evaluated. In conclusion, the expression of the line starts after [AllowPrefixes](#) property, and ends before AllowComments property. The [FormatInvalid](#) property specifies the HTML format to show invalid lines. The [AddWildFormat](#) method formats the line based on the giving wild characters expression.

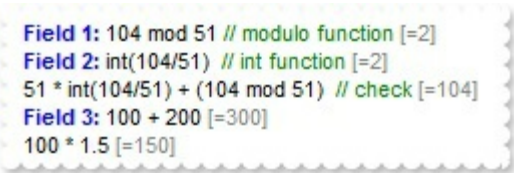
The list of supported built-in HTML tags is:

- **bold**
- **<i>italic</i>**
- **<s>strikeout</s>**
- **<u>underline</u>**
- **<fgcolor=RRGGBB>fgcolor</fgcolor>**
- **<bgcolor=RRGGBB>bgcolor</bgcolor>**
- **text ** 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.

For instance, if:

- the AllowComments property is "**<fgcolor=008080>//</fgcolor>**", it specifies that the comments starts after // expression, and show in green.
- the AllowPrefixes property is "**<fgcolor=0000FF>:</fgcolor>**", it specifies that prefix of the line starts before : character, and it shows in bold and blue.

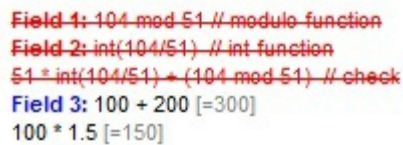
The following screen shot shows lines with/without comments and prefixes:



The

- the black portion of each line is the expression being evaluated
- the gray portion of each line indicates the result of evaluation the line
- the blue portion of each line indicates its prefix, and it is not evaluated
- the green portion of each line is its comment and it is not evaluated
- if present, the red line indicates an invalid line

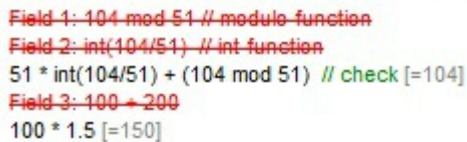
In case, the AllowComments property is empty, we get the following:



```
Field 1: 104 mod 51 // module function
Field 2: int(104/51) // int function
51 * int(104/51) + (104 mod 51) // check
Field 3: 100 + 200 [=300]
100 * 1.5 [=150]
```

so only lines with or without a prefix are evaluated:

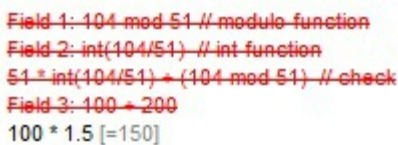
In case, the AllowPrefixes property is empty, we get the following:



```
Field 1: 104 mod 51 // module function
Field 2: int(104/51) // int function
51 * int(104/51) + (104 mod 51) // check [=104]
Field 3: 100 + 200
100 * 1.5 [=150]
```

so only lines with or without a comment are evaluated:

or if both are empty we get the following:



```
Field 1: 104 mod 51 // module function
Field 2: int(104/51) // int function
51 * int(104/51) + (104 mod 51) // check
Field 3: 100 + 200
100 * 1.5 [=150]
```

so only lines without a prefix and comment are evaluated:

property CalcEdit.AllowCount as String

Specifies the keyword that makes the control to display the count all lines being counted in a Total group.

Type	Description
String	A String expression that defines the HTML expression that indicates the keyword that computes the valid lines.

By default, the AllowCount property is "Count", which indicates that the **Count** keyword specifies the count of all lines (valid) in the control. If the AllowCount property is "", the control supports no Count aggregate function. The [FormatCountResult](#) / [FormatLocal](#) property defines the format to display the result of a Count line. The [AllowVariables](#) property specifies the expression (no HTML) that defines the equal operator, so you can define variables.

By default, the control supports the following aggregate functions:

- [AllowTotal](#) property defines the Total keyword. The **Total** keyword, specifies the sum of all lines that are not empty, valid and defines no variables.
- [AllowSubTotal](#) property defines the SubTotal keyword. The **SubTotal** keyword, specifies the sum of all previously lines that are not empty, valid, defines no variables until another SubTotal keyword is found.
- AllowCount property defines the Count keyword. The **Count** keyword, counts all lines that are not empty, valid and defines no variables.
- [AllowSubCount](#) property defines the SubCount keyword. The **SubCount** keyword, counts all previously lines that are not empty, valid, defines no variables until another SubCount keyword is found.

The following screen show shows a control that displays a Count line:



property CalcEdit.AllowFormatInvalidOnTyping as Boolean

Specifies whether the FormatInvalid property is applied on the current line, while typing into the control.

Type	Description
Boolean	A Boolean expression that specifies whether the FormatInvalid property is applied on the current line, while typing into the control.

By default, the AllowFormatInvalidOnTyping property is True, which indicates that the current line is highlighted as soon as the user types. The AllowFormatInvalidOnTyping property specifies whether the [FormatInvalid](#) property is applied on the current line, while typing into the control. By default, the FormatInvalid property is "<fgcolor=FF0000><s></s></fgcolor>", which indicates that invalid lines are shown in red as showing in the following screen shot. If the FormatInvalid property is "", the control does not highlight the invalid lines.



An invalid line is not evaluated, and so no result is being shown. The [FormatResult](#) property specifies the HTML format of the result. The [FormatTotalResult](#) / [FormatSubTotalResult](#) property formats the Total / Sub-Total lines. The [FormatCountResult](#) / [FormatSubCountResult](#) property specifies the HTML format of lines that contains Count or SubCount aggregate function.

The list of supported built-in HTML tags is:

- **bold**
- **<i>italic</i>**
- **<s>strikeout</s>**
- **<u>underline</u>**
- **<fgcolor=RRGGBB>fgcolor</fgcolor>**
- **<bgcolor=RRGGBB>bgcolor</bgcolor>**
- **text ** 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.

property CalcEdit.AllowPrefixes as String

Specifies the HTML caption that ends the prefix of the line. If empty, no prefixes are allowed.

Type	Description
String	A string expression that defines the HTML caption that ends the prefix of the line

By default, the AllowPrefixes property is "", which indicates that the control supports no prefixes. A line can have a prefix delimited by the AllowPrefixes property, and can have a comment delimited by the [AllowComments](#) property. The prefix of the line and its comment are never evaluated. In conclusion, the expression of the line starts after AllowPrefixes property, and ends before [AllowComments](#) property. The [FormatInvalid](#) property specifies the HTML format to show invalid lines. The [AddWildFormat](#) method formats the line based on the giving wild characters expression.

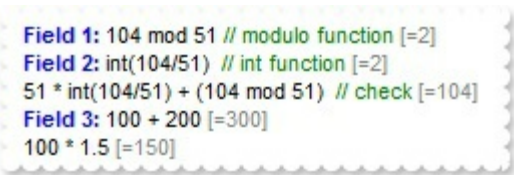
The list of supported built-in HTML tags is:

- **bold**
- **<i>italic</i>**
- **<s>strikeout</s>**
- **<u>underline</u>**
- **<fgcolor=RRGGBB>fgcolor</fgcolor>**
- **<bgcolor=RRGGBB>bgcolor</bgcolor>**
- **text ** 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.

For instance, if:

- the AllowComments property is "**<fgcolor=008080>//</fgcolor>**", it specifies that the comments starts after // expression, and show in green.
- the AllowPrefixes property is "**<fgcolor=0000FF>:</fgcolor>**", it specifies that prefix of the line starts before : character, and it shows in bold and blue.

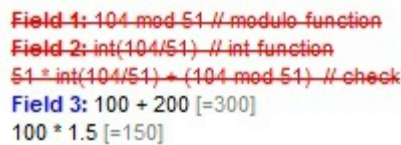
The following screen shot shows lines with/without comments and prefixes:



The

- the black portion of each line is the expression being evaluated
- the gray portion of each line indicates the result of evaluation the line
- the blue portion of each line indicates its prefix, and it is not evaluated
- the green portion of each line is its comment and it is not evaluated
- if present, the red line indicates an invalid line

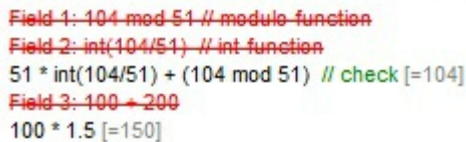
In case, the AllowComments property is empty, we get the following:



```
Field 1: 104 mod 51 // module function
Field 2: int(104/51) // int function
51 * int(104/51) + (104 mod 51) // check
Field 3: 100 + 200 [=300]
100 * 1.5 [=150]
```

so only lines with or without a prefix are evaluated:

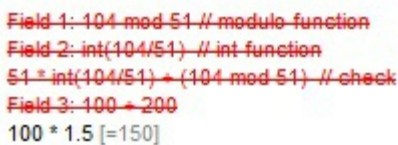
In case, the AllowPrefixes property is empty, we get the following:



```
Field 1: 104 mod 51 // module function
Field 2: int(104/51) // int function
51 * int(104/51) + (104 mod 51) // check [=104]
Field 3: 100 + 200
100 * 1.5 [=150]
```

so only lines with or without a comment are evaluated:

or if both are empty we get the following:



```
Field 1: 104 mod 51 // module function
Field 2: int(104/51) // int function
51 * int(104/51) + (104 mod 51) // check
Field 3: 100 + 200
100 * 1.5 [=150]
```

so only lines without a prefix and comment are evaluated:

property CalcEdit.AllowSubCount as String

Specifies the keyword that makes the control to display the subcounts.

Type	Description
String	A String expression that defines the HTML expression that indicates the keyword that computes all previously valid lines.

By default, the AllowSubCount property is "<fgcolor=808080>SubCount</fgcolor>", which indicates that the **SubCount** keyword specifies the count of all lines (valid) in the control. If the AllowSubCount property is "", the control supports no SubCount aggregate function. The [FormatSubCountResult](#) / [FormatLocal](#) property defines the format to display the result of a SubCount line. The [AllowVariables](#) property specifies the expression (no HTML) that defines the equal operator, so you can define variables.

By default, the control supports the following aggregate functions:

- [AllowTotal](#) property defines the Total keyword. The **Total** keyword, specifies the sum of all lines that are not empty, valid and defines no variables.
- [AllowSubTotal](#) property defines the SubTotal keyword. The **SubTotal** keyword, specifies the sum of all previously lines that are not empty, valid, defines no variables until another SubTotal keyword is found.
- [AllowCount](#) property defines the Count keyword. The **Count** keyword, counts all lines that are not empty, valid and defines no variables.
- AllowSubCount property defines the SubCount keyword. The **SubCount** keyword, counts all previously lines that are not empty, valid, defines no variables until another SubCount keyword is found.

The following screen show shows a control that displays a Count line:



property CalcEdit.AllowSubTotal as String

Specifies the keyword that makes the control to display the subtotals.

Type	Description
String	A String expression that defines the HTML expression that indicates the keyword that computes the sub-total of lines.

By default, the AllowSubTotal property is "<fgcolor=808080>SubTotal</fgcolor>", which indicates that the **SubTotal** keyword specifies the sub-total of previously lines in the control. If the AllowSubTotal property is "", the control supports no SubTotal aggregate function. The [FormatSubTotalResult](#) / [FormatLocal](#) property defines the format to display the result of a Total line. The [BackColorSubTotalLine](#) property specifies the background color to show the SubTotal line. The [AllowVariables](#) property specifies the expression (no HTML) that defines the equal operator, so you can define variables.

By default, the control supports the following aggregate functions:

- [AllowTotal](#) property defines the Total keyword. The **Total** keyword, specifies the sum of all lines that are not empty, valid and defines no variables.
- AllowSubTotal property defines the SubTotal keyword. The **SubTotal** keyword, specifies the sum of all previously lines that are not empty, valid, defines no variables until another SubTotal keyword is found.
- [AllowCount](#) property defines the Count keyword. The **Count** keyword, counts all lines that are not empty, valid and defines no variables.
- [AllowSubCount](#) property defines the SubCount keyword. The **SubCount** keyword, counts all previously lines that are not empty, valid, defines no variables until another SubCount keyword is found.

The following screen show shows a control that displays SubTotal lines:



property CalcEdit.AllowTotal as String

Specifies the keyword that makes the control to display the sum/total of all lines.

Type	Description
String	A String expression that defines the HTML expression that indicates the keyword that computes the total of lines.

By default, the AllowTotal property is "Total", which indicates that the **Total** keyword specifies the total of all lines in the control. if the AllowTotal property is "", the control supports no Total aggregate function. The [FormatTotalResult](#) / [FormatLocal](#) property defines the format to display the result of a Total line. The [BackColorTotalLine](#) property specifies the background color to show the Total line. The [AllowVariables](#) property specifies the expression (no HTML) that defines the equal operator, so you can define variables.

By default, the control supports the following aggregate functions:

- AllowTotal property defines the Total keyword. The **Total** keyword, specifies the sum of all lines that are not empty, valid and defines no variables.
- [AllowSubTotal](#) property defines the SubTotal keyword. The **SubTotal** keyword, specifies the sum of all previously lines that are not empty, valid, defines no variables until another SubTotal keyword is found.
- [AllowCount](#) property defines the Count keyword. The **Count** keyword, counts all lines that are not empty, valid and defines no variables.
- [AllowSubCount](#) property defines the SubCount keyword. The **SubCount** keyword, counts all previously lines that are not empty, valid, defines no variables until another SubCount keyword is found.

The following screen show shows a control that displays a Total line:



property CalcEdit.AllowUndoRedo as Boolean

Specifies whether the control allows undo/redo actions.

Type	Description
Boolean	A boolean expression that indicates whether the control allows undo/redo actions.

The control supports multi levels undo/redo support. The CTRL + Z reverses the last editing action, The CTRL + Y restores the previously undone action. Use the [CanUndo](#) property to determine by code whether an undo operation is available. Use the [CanRedo](#) property to determine by code whether a redo operation is available. Use the [Redo](#) method to redo the next action in the control's redo queue. Use the [Undo](#) method to undo the last edit-control operation.

property CalcEdit.AllowVariables as String

Specifies the expression (no HTML) that defines the equal operator, so you can define variables.

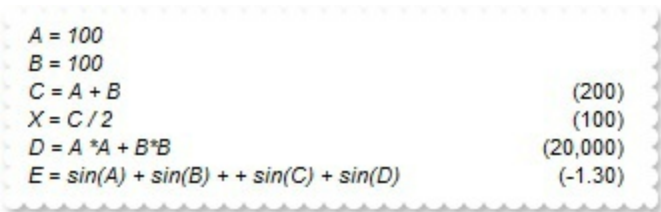
Type	Description
String	A String expression that defines the expression (no HTML) that defines the equal operator, so you can define variables.

By default, the AllowVariables property is "=" (equal character), which indicates that a line can be divided in parts as var = expression, which indicates defining the var variable. The [AddWildFormat](#) method formats the line based on the giving wild characters expression. By default, the control has already the wild format defined as "<i>*=*</i>", which draws in italics any line that includes the = (equal) character (define the variables). If the AllowVariables property is "", the control does not support defining any variable. The [Variable](#) property indicates the value of the specified variable. By default, the control supports variables such as Total and Count, which defines the Total of all valid lines, and count of them. The [CalcType](#) property specifies the type of operations the control supports.

By default, the control supports the following aggregate functions:

- [AllowTotal](#) property defines the Total keyword. The **Total** keyword, specifies the sum of all lines that are not empty, valid and defines no variables.
- [AllowSubTotal](#) property defines the SubTotal keyword. The **SubTotal** keyword, specifies the sum of all previously lines that are not empty, valid, defines no variables until another SubTotal keyword is found.
- [AllowCount](#) property defines the Count keyword. The **Count** keyword, counts all lines that are not empty, valid and defines no variables.
- [AllowSubCount](#) property defines the SubCount keyword. The **SubCount** keyword, counts all previously lines that are not empty, valid, defines no variables until another SubCount keyword is found.

The following screen shot defines a few variables:



property CalcEdit.Appearance as AppearanceEnum

Retrieves or sets the control's appearance.

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

Use the Appearance property to hide the control's border. Use the [BackColor](#) property to specify the control's background color. Use the [ForeColor](#) property to specify the control's foreground color.

method CalcEdit.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 CacEdit1_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 CalcEdit.BackColor as Color

Specifies the control's background color.

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

Use the BackColor and [ForeColor](#) properties to define the control's background and foreground colors. Use the [Picture](#) property to assign a picture on the control's background.

The following VB sample changes the control's background color:

```
With CalcEdit1
    .BackColor = ColorConstants.vbWhite
End With
```

The following C++ sample changes the control's background color:

```
m_calcEdit.SetBackColor( RGB(255,255,255) );
```

The following VB.NET sample changes the control's background color:

```
With AxCalcEdit1
    .BackColor = Color.White
End With
```

The following C# sample changes the control's background color:

```
axCalcEdit1.BackColor = Color.White;
```

The following VFP sample changes the control's background color:

```
With thisform.CalcEdit1.Object
    .BackColor = RGB(255,255,255)
endwith
```

property CalcEdit.BackColorLockedLine as Color

Retrieves or sets a value that indicates the line's background color when it is locked.

Type	Description
Color	A Color expression that indicates the background color for locked line.

The BackColorLockedLine property specifies the foreground color for locked lines. The property has effect while it is not zero. The [ForeColorLockedLine](#) property specifies the foreground color for locked lines. Use the [InsertLockedText](#) method inserts locked text/lines to control.

The following screen shot shows the locked lines with a different back/foreground color:



property CalcEdit.BackColorSubTotal as Color

Specifies the background color to show the SubTotal lines.

Type	Description
Color	A Color expression that indicates the background color to show the SubTotal line.

The BackColorSubTotalLine property specifies the background color to show the SubTotal line. The [AllowSubTotal](#) property specifies the keyword that makes the control to display the subtotal of all lines. The [FormatSubTotalResult](#) / [FormatLocal](#) property defines the format to display the result of a SubTotal line. The [AllowVariables](#) property specifies the expression (no HTML) that defines the equal operator, so you can define variables.

By default, the control supports the following aggregate functions:

- [AllowTotal](#) property defines the Total keyword. The **Total** keyword, specifies the sum of all lines that are not empty, valid and defines no variables.
- [AllowSubTotal](#) property defines the SubTotal keyword. The **SubTotal** keyword, specifies the sum of all previously lines that are not empty, valid, defines no variables until another SubTotal keyword is found.
- [AllowCount](#) property defines the Count keyword. The **Count** keyword, counts all lines that are not empty, valid and defines no variables.
- [AllowSubCount](#) property defines the SubCount keyword. The **SubCount** keyword, counts all previously lines that are not empty, valid, defines no variables until another SubCount keyword is found.

The following screen show shows a control that displays a Total line, with a different background color:



property CalcEdit.BackColorTotal as Color

Specifies the background color to show the Total line.

Type	Description
Color	A Color expression that indicates the background color to show the Total line.

The BackColorTotalLine property specifies the background color to show the Total line. The [AllowTotal](#) property specifies the keyword that makes the control to display the sum/total of all lines. The [FormatTotalResult](#) / [FormatLocal](#) property defines the format to display the result of a Total line. The [AllowVariables](#) property specifies the expression (no HTML) that defines the equal operator, so you can define variables.

By default, the control supports the following aggregate functions:

- [AllowTotal](#) property defines the Total keyword. The **Total** keyword, specifies the sum of all lines that are not empty, valid and defines no variables.
- [AllowSubTotal](#) property defines the SubTotal keyword. The **SubTotal** keyword, specifies the sum of all previously lines that are not empty, valid, defines no variables until another SubTotal keyword is found.
- [AllowCount](#) property defines the Count keyword. The **Count** keyword, counts all lines that are not empty, valid and defines no variables.
- [AllowSubCount](#) property defines the SubCount keyword. The **SubCount** keyword, counts all previously lines that are not empty, valid, defines no variables until another SubCount keyword is found.

The following screen show shows a control that displays a Total line, with a different background color:



property CalcEdit.CalcType as CalcTypeEnum

Specifies the type of operations the control support.

Type	Description
CalcTypeEnum	A CalcTypeEnum expression that specifies the type of operations the control supports.

By default, the CalcType property is exCalcStandard. For instance, you can use the CalcType property on exCalcIncludeAll, to allow sqrt (square root function) to be used in the control. Use the [Text](#) property to specify the control's text. The control's text is evaluated using arithmetic operators. Use the [Result](#) property to get the result, if the expression is valid. Use the [IsValid](#) property to specify whether the Text property is syntactically correct, and may be evaluated. The [AllowVariables](#) property specifies the expression (no HTML) that defines the equal operator, so you can define variables.

By default, the control supports the following aggregate functions:

- [AllowTotal](#) property defines the Total keyword. The **Total** keyword, specifies the sum of all lines that are not empty, valid and defines no variables.
- [AllowSubTotal](#) property defines the SubTotal keyword. The **SubTotal** keyword, specifies the sum of all previously lines that are not empty, valid, defines no variables until another SubTotal keyword is found.
- [AllowCount](#) property defines the Count keyword. The **Count** keyword, counts all lines that are not empty, valid and defines no variables.
- [AllowSubCount](#) property defines the SubCount keyword. The **SubCount** keyword, counts all previously lines that are not empty, valid, defines no variables until another SubCount keyword is found.

If the CalcType property is exCalcStandard, the control supports all operations and functions as listed below:

- * (multiplicity operator), priority 5
- / (divide operator), priority 5
- + (addition operator), priority 4
- - (subtraction operator), priority 4

If the CalcType property is exCalcIncludeAll, the control supports all operations and functions as listed below:

The constants are (DPI-Aware components):

- **dpi** (DPI constant), specifies the current DPI setting. and it indicates the minimum value between **dpix** and **dpiy** constants. For instance, if current DPI setting is 100%,

the dpi constant returns 1, if 150% it returns 1.5, and so on. For instance, the expression value * dpi returns the value if the DPI setting is 100%, or value * 1.5 in case, the DPI setting is 150%

- **dpix** (DPIX constant), specifies the current DPI setting on x-scale. For instance, if current DPI setting is 100%, the dpix constant returns 1, if 150% it returns 1.5, and so on. For instance, the expression value * dpix returns the value if the DPI setting is 100%, or value * 1.5 in case, the DPI setting is 150%
- **dpiy** (DPIY constant), specifies the current DPI setting on y-scale. For instance, if current DPI setting is 100%, the dpiy constant returns 1, if 150% it returns 1.5, and so on. For instance, the expression value * dpiy returns the value if the DPI setting is 100%, or value * 1.5 in case, the DPI setting is 150%

The supported binary arithmetic operators are:

- * (multiplicity operator), priority 5
- / (divide operator), priority 5
- **mod** (remainder operator), priority 5
- + (addition operator), priority 4 (concatenates two strings, if one of the operands is of string type)
- - (subtraction operator), priority 4

The supported unary boolean operators are:

- **not** (not operator), priority 3 (high priority)

The supported binary boolean operators are:

- **or** (or operator), priority 2
- **and** (and operator), priority 1

The supported binary boolean operators, all these with the same priority 0, are :

- < (less operator)
- <= (less or equal operator)
- = (equal operator)
- != (not equal operator)
- >= (greater or equal operator)
- > (greater operator)

The supported binary range operators, all these with the same priority 5, are :

- **MIN** (min operator), indicates the minimum value, so a **MIN** b returns the value of a, if it is less than b, else it returns b. For instance, the expression value MIN 10 returns always a value greater than 10.

- **MAX** (max operator), indicates the maximum value, so a **MAX** b returns the value of a, if it is greater than b, else it returns b. For instance, the expression value MAX 100 returns always a value less than 100.

The supported binary operators, all these with the same priority 0, are :

- **:= (Store operator)**, stores the result of expression to variable. The syntax for := operator is

variable := expression

where variable is a integer between 0 and 9. You can use the **:=** operator to restore any stored variable (please make the difference between **:=** and **=:**). For instance, `(0:=dbl(value)) = 0 ? "zero" : =:0`, stores the value converted to double, and prints zero if it is 0, else the converted number. Please pay attention that the **:=** and **=:** are two distinct operators, the first for storing the result into a variable, while the second for restoring the variable

- **=: (Restore operator)**, restores the giving variable (previously saved using the store operator). The syntax for **=:** operator is

=: variable

where variable is a integer between 0 and 9. You can use the **:=** operator to store the value of any expression (please make the difference between **:=** and **=:**). For instance, `(0:=dbl(value)) = 0 ? "zero" : =:0`, stores the value converted to double, and prints zero if it is 0, else the converted number. Please pay attention that the **:=** and **=:** are two distinct operators, the first for storing the result into a variable, while the second for restoring the variable

The supported ternary operators, all these with the same priority 0, are :

- **? (Immediate If operator)**, returns and executes one of two expressions, depending on the evaluation of an expression. The syntax for **?** operator is

expression ? true_part : false_part

, while it executes and returns the true_part if the expression is true, else it executes and returns the false_part. For instance, the `%0 = 1 ? 'One' : (%0 = 2 ? 'Two' : 'not found')` returns 'One' if the value is 1, 'Two' if the value is 2, and 'not found' for any other value. A n-ary equivalent operation is the case() statement, which is available in newer versions of the component.

The supported n-ary operators are (with priority 5):

- **array** (*at operator*), returns the element from an array giving its index (0 base). The *array* operator returns empty if the element is found, else the associated element in the collection if it is found. The syntax for *array* operator is

expression array (c1,c2,c3,...cn)

, where the c1, c2, ... are constant elements. The constant elements could be numeric, date or string expressions. For instance the *month(value)-1 array* ('J','F','M','A','M','Jun','J','A','S','O','N','D') is equivalent with *month(value)-1 case* (default:"; 0:'J';1:'F';2:'M';3:'A';4:'M';5:'Jun';6:'J';7:'A';8:'S';9:'O';10:'N';11:'D').

- **in** (*include operator*), specifies whether an element is found in a set of constant elements. The *in* operator returns -1 (True) if the element is found, else 0 (false) is retrieved. The syntax for *in* operator is

expression in (c1,c2,c3,...cn)

, where the c1, c2, ... are constant elements. The constant elements could be numeric, date or string expressions. For instance the *value in (11,22,33,44,13)* is equivalent with (*expression = 11*) or (*expression = 22*) or (*expression = 33*) or (*expression = 44*) or (*expression = 13*). The *in* operator is not a time consuming as the equivalent *or* version is, so when you have large number of constant elements it is recommended using the *in* operator. Shortly, if the collection of elements has 1000 elements the *in* operator could take up to 8 operations in order to find if an element fits the set, else if the *or* statement is used, it could take up to 1000 operations to check, so by far, the *in* operator could save time on finding elements within a collection.

- **switch** (*switch operator*), returns the value being found in the collection, or a predefined value if the element is not found (default). The syntax for *switch* operator is

expression switch (default,c1,c2,c3,...,cn)

, where the c1, c2, ... are constant elements, and the default is a constant element being returned when the element is not found in the collection. The constant elements could be numeric, date or string expressions. The equivalent syntax is "%0 = c 1 ? c 1 : (%0 = c 2 ? c 2 : (... ? . : default))". The *switch* operator is very similar with the *in* operator excepts that the first element in the switch is always returned by the statement if the element is not found, while the returned value is the value itself instead -1. For instance, the *%0 switch ('not found',1,4,7,9,11)* gets 1, 4, 7, 9 or 11, or 'not found' for any other value. As the *in* operator the *switch* operator uses binary searches for fitting the element, so it is quicker than *if* (immediate if operator) alternative.

- **case()** (*case operator*) returns and executes one of n expressions, depending on the evaluation of the expression (*IIF* - immediate IF operator is a binary *case()* operator).

The syntax for `case()` operator is:

expression case ([default : default_expression ;] c1 : expression1 ; c2 : expression2 ; c3 : expression3 ;....)

If the default part is missing, the `case()` operator returns the value of the expression if it is not found in the collection of cases (`c1`, `c2`, ...). For instance, if the value of expression is not any of `c1`, `c2`, the `default_expression` is executed and returned. If the value of the expression is `c1`, then the `case()` operator executes and returns the *expression1*. The *default*, *c1*, *c2*, *c3*, ... must be constant elements as numbers, dates or strings. For instance, the `date(shortdate(value)) case (default:0 ; #1/1/2002#:1 ; #2/1/2002#:1; #4/1/2002#:1; #5/1/2002#:1)` indicates that only `#1/1/2002#`, `#2/1/2002#`, `#4/1/2002#` and `#5/1/2002#` dates returns 1, since the others returns 0. For instance the following sample specifies the hour being non-working for specified dates: `date(shortdate(value)) case(default:0;#4/1/2009# : hour(value) >= 6 and hour(value) <= 12 ; #4/5/2009# : hour(value) >= 7 and hour(value) <= 10 or hour(value) in(15,16,18,22); #5/1/2009# : hour(value) <= 8)` statement indicates the working hours for dates as follows:

- `#4/1/2009#`, from hours 06:00 AM to 12:00 PM
- `#4/5/2009#`, from hours 07:00 AM to 10:00 AM and hours 03:00PM, 04:00PM, 06:00PM and 10:00PM
- `#5/1/2009#`, from hours 12:00 AM to 08:00 AM

The *in*, *switch* and *case()* use binary search to look for elements so they are faster then using *iif* and *or* expressions. Obviously, the priority of the operations inside the expression is determined by () parenthesis and the priority for each operator.

The supported conversion unary operators are:

- **type** (unary operator) retrieves the type of the object. For instance `type(%1) = 8` specifies the cells (on the column 1) that contains string values.

Here's few predefined types:

- 0 - empty (not initialized)
- 1 - null
- 2 - short
- 3 - long
- 4 - float
- 5 - double
- 6 - currency
- 7 - date
- 8 - string

- 9 - object
- 10 - error
- 11 - boolean
- 12 - variant
- 13 - any
- 14 - decimal
- 16 - char
- 17 - byte
- 18 - unsigned short
- 19 - unsigned long
- 20 - long on 64 bits
- 21 - unsigned long on 64 bites
- **str** (unary operator) converts the expression to a string. The str operator converts the expression to a string. For instance, the *str(-12.54)* returns the string "-12.54".
- **dbl** (unary operator) converts the expression to a number. The dbl operator converts the expression to a number. For instance, the *dbl("12.54")* returns 12.54
- **date** (unary operator) converts the expression to a date, based on your regional settings. For instance, the *date(``)* gets the current date (no time included), the *date(`now`)* gets the current date-time, while the *date("01/01/2001")* returns #1/1/2001#
- **dateS** (unary operator) converts the string expression to a date using the format MM/DD/YYYY HH:MM:SS. For instance, the *dateS("01/01/2001 14:00:00")* returns #1/1/2001 14:00:00#

Other known operators for numbers are:

- **int** (unary operator) retrieves the integer part of the number. For instance, the *int(12.54)* returns 12
- **round** (unary operator) rounds the number ie 1.2 gets 1, since 1.8 gets 2. For instance, the *round(12.54)* returns 13
- **floor** (unary operator) returns the largest number with no fraction part that is not greater than the value of its argument. For instance, the *floor(12.54)* returns 12
- **abs** (unary operator) retrieves the absolute part of the number ie -1 gets 1, 2 gets 2. For instance, the *abs(-12.54)* returns 12.54
- **sin** (unary operator) returns the sine of an angle of x radians. For instance, the *sin(3.14)* returns 0.001593.
- **cos** (unary operator) returns the cosine of an angle of x radians. For instance, the *cos(3.14)* returns -0.999999.
- **asin** (unary operator) returns the principal value of the arc sine of x, expressed in radians. For instance, the *2*asin(1)* returns the value of PI.
- **acos** (unary operator) returns the principal value of the arc cosine of x, expressed in radians. For instance, the *2*acos(0)* returns the value of PI

- **sqrt** (unary operator) returns the square root of x. For instance, the *sqrt(81)* returns 9.
- **currency** (unary operator) formats the giving number as a currency string, as indicated by the control panel. For instance, *currency(value)* displays the value using the current format for the currency ie, 1000 gets displayed as \$1,000.00, for US format.
- value **format** 'flags' (binary operator) formats the value with specified flags. If flags is empty, the number is displayed as shown in the field "Number" in the "Regional and Language Options" from the Control Panel. For instance the *1000 format "* displays 1,000.00 for English format, while 1.000,00 is displayed for German format. 1000 format '2|.|3|,' will always displays 1,000.00 no matter of settings in the control panel. If formatting the number fails for some invalid parameter, the value is displayed with no formatting.

The ' flags' for format operator is a list of values separated by | character such as '*NumDigits|DecimalSep|Grouping|ThousandSep|NegativeOrder|LeadingZero*' with the following meanings:

- *NumDigits* - specifies the number of fractional digits, If the flag is missing, the field "No. of digits after decimal" from "Regional and Language Options" is using.
- *DecimalSep* - specifies the decimal separator. If the flag is missing, the field "Decimal symbol" from "Regional and Language Options" is using.
- *Grouping* - indicates the number of digits in each group of numbers to the left of the decimal separator. Values in the range 0 through 9 and 32 are valid. The most significant grouping digit indicates the number of digits in the least significant group immediately to the left of the decimal separator. Each subsequent grouping digit indicates the next significant group of digits to the left of the previous group. If the last value supplied is not 0, the remaining groups repeat the last group. Typical examples of settings for this member are: 0 to group digits as in 123456789.00; 3 to group digits as in 123,456,789.00; and 32 to group digits as in 12,34,56,789.00. If the flag is missing, the field "Digit grouping" from "Regional and Language Options" indicates the grouping flag.
- *ThousandSep* - specifies the thousand separator. If the flag is missing, the field "Digit grouping symbol" from "Regional and Language Options" is using.
- *NegativeOrder* - indicates the negative number mode. If the flag is missing, the field "Negative number format" from "Regional and Language Options" is using. The valid values are 0, 1, 2, 3 and 4 with the following meanings:
 - 0 - Left parenthesis, number, right parenthesis; for example, (1.1)
 - 1 - Negative sign, number; for example, -1.1
 - 2 - Negative sign, space, number; for example, - 1.1
 - 3 - Number, negative sign; for example, 1.1-
 - 4 - Number, space, negative sign; for example, 1.1 -
- *LeadingZero* - indicates if leading zeros should be used in decimal fields. If the flag is missing, the field "Display leading zeros" from "Regional and Language Options" is using. The valid values are 0, 1

Other known operators for strings are:

- **len** (unary operator) retrieves the number of characters in the string. For instance, the *len("Mihai")* returns 5.
- **lower** (unary operator) returns a string expression in lowercase letters. For instance, the *lower("MIHAI")* returns "mihai"
- **upper** (unary operator) returns a string expression in uppercase letters. For instance, the *upper("mihai")* returns "MIHAI"
- **proper** (unary operator) returns from a character expression a string capitalized as appropriate for proper names. For instance, the *proper("mihai")* returns "Mihai"
- **ltrim** (unary operator) removes spaces on the left side of a string. For instance, the *ltrim(" mihai")* returns "mihai"
- **rtrim** (unary operator) removes spaces on the right side of a string. For instance, the *rtrim("mihai ")* returns "mihai"
- **trim** (unary operator) removes spaces on both sides of a string. For instance, the *trim(" mihai ")* returns "mihai"
- **reverse** (unary operator) reverses the order of the characters in the string a. For instance, the *reverse("Mihai")* returns "iahIM"
- **startswith** (binary operator) specifies whether a string starts with specified string (0 if not found, -1 if found). For instance *"Mihai" startwith "Mi"* returns -1
- **endwith** (binary operator) specifies whether a string ends with specified string (0 if not found, -1 if found). For instance *"Mihai" endwith "ai"* returns -1
- **contains** (binary operator) specifies whether a string contains another specified string (0 if not found, -1 if found). For instance *"Mihai" contains "ha"* returns -1
- **left** (binary operator) retrieves the left part of the string. For instance *"Mihai" left 2* returns "Mi".
- **right** (binary operator) retrieves the right part of the string. For instance *"Mihai" right 2* returns "ai"
- a **lfind** b (binary operator) The a lfind b (binary operator) searches the first occurrence of the string b within string a, and returns -1 if not found, or the position of the result (zero-index). For instance *"ABCABC" lfind "C"* returns 2
- a **rfind** b (binary operator) The a rfind b (binary operator) searches the last occurrence of the string b within string a, and returns -1 if not found, or the position of the result (zero-index). For instance *"ABCABC" rfind "C"* returns 5.
- a **mid** b (binary operator) retrieves the middle part of the string a starting from b (1 means first position, and so on). For instance *"Mihai" mid 2* returns "ihai"
- a **count** b (binary operator) retrieves the number of occurrences of the b in a. For instance *"Mihai" count "i"* returns 2.
- a **replace** b with c (double binary operator) replaces in a the b with c, and gets the result. For instance, the *"Mihai" replace "i" with ""* returns "Mha" string, as it replaces all "i" with nothing.
- a **split** b, splits the a using the separator b, and returns an array. For instance, the

weekday(value) array 'Sun Mon Thu Wed Thu Fri Sat' split ' ' gets the weekday as string. This operator can be used with the array.

Other known operators for dates are:

- **time** (unary operator) retrieves the time of the date in string format, as specified in the control's panel. For instance, the *time(#1/1/2001 13:00#)* returns "1:00:00 PM"
- **timeF** (unary operator) retrieves the time of the date in string format, as "HH:MM:SS". For instance, the *timeF(#1/1/2001 13:00#)* returns "13:00:00"
- **shortdate** (unary operator) formats a date as a date string using the short date format, as specified in the control's panel. For instance, the *shortdate(#1/1/2001 13:00#)* returns "1/1/2001"
- **shortdateF** (unary operator) formats a date as a date string using the "MM/DD/YYYY" format. For instance, the *shortdateF(#1/1/2001 13:00#)* returns "01/01/2001"
- **dateF** (unary operator) converts the date expression to a string expression in "MM/DD/YYYY HH:MM:SS" format. For instance, the *dateF(#01/01/2001 14:00:00#)* returns #01/01/2001 14:00:00#
- **longdate** (unary operator) formats a date as a date string using the long date format, as specified in the control's panel. For instance, the *longdate(#1/1/2001 13:00#)* returns "Monday, January 01, 2001"
- **year** (unary operator) retrieves the year of the date (100,...,9999). For instance, the *year(#12/31/1971 13:14:15#)* returns 1971
- **month** (unary operator) retrieves the month of the date (1, 2,...,12). For instance, the *month(#12/31/1971 13:14:15#)* returns 12.
- **day** (unary operator) retrieves the day of the date (1, 2,...,31). For instance, the *day(#12/31/1971 13:14:15#)* returns 31
- **yearday** (unary operator) retrieves the number of the day in the year, or the days since January 1st (0, 1,...,365). For instance, the *yearday(#12/31/1971 13:14:15#)* returns 365
- **weekday** (unary operator) retrieves the number of days since Sunday (0 - Sunday, 1 - Monday,..., 6 - Saturday). For instance, the *weekday(#12/31/1971 13:14:15#)* returns 5.
- **hour** (unary operator) retrieves the hour of the date (0, 1, ..., 23). For instance, the *hour(#12/31/1971 13:14:15#)* returns 13
- **min** (unary operator) retrieves the minute of the date (0, 1, ..., 59). For instance, the *min(#12/31/1971 13:14:15#)* returns 14
- **sec** (unary operator) retrieves the second of the date (0, 1, ..., 59). For instance, the *sec(#12/31/1971 13:14:15#)* returns 15

The Exontrol's [eXPression](#) component is a syntax-editor that helps you to define, view, edit and evaluate expressions. Using the eXPression component you can easily view or check if

the expression you have used is syntactically correct, and you can evaluate what is the result you get giving different values to be tested. The Exontrol's eXPression component can be used as an user-editor, to configure your applications.

property CalcEdit.CanRedo as Boolean

Determines if the redo queue contains any actions.

Type	Description
Boolean	A boolean expression that determines if the redo queue contains any actions.

The control supports multi levels undo/redo support. The CTRL + Z reverses the last editing action, The CTRL + Y restores the previously undone action. Use the CanRedo property to determine by code whether a redo operation is available. Use the [CanUndo](#) property to determine by code whether an undo operation is available. Use the [Redo](#) method to redo the next action in the control's redo queue. Use the [Undo](#) method to undo the last edit-control operation.

property CalcEdit.CanUndo as Boolean

Determines whether the last edit operation can be undone.

Type	Description
Boolean	A boolean expression that indicates whether the last edit operation can be undone.

The control supports multi levels undo/redo support. The CTRL + Z reverses the last editing action, The CTRL + Y restores the previously undone action. Use the CanUndo property to determine by code whether an undo operation is available. Use the [CanRedo](#) property to determine by code whether a redo operation is available. Use the [Redo](#) method to redo the next action in the control's redo queue. Use the [Undo](#) method to undo the last edit-control operation.

property CalcEdit.CaretLine as Long

Indicates the line that displays the caret.

Type	Description
Long	A long expression that defines the caret's line (index of the line , 1 based).

Use the CaretLine and [CaretPos](#) properties to determine the caret's position. The CaretLine property is 1-based. The index for the first line in the control's text is 1. Use the [TextLine](#) property to get the line based on its index.

property CalcEdit.CaretPos as Long

Retrieves or sets a value that indicates the position of the caret in the line.

Type	Description
Long	A long expression that defines the position of the caret in the current line (0 based).

Use the [CaretLine](#) and CaretPos properties to determine the caret's position. The CaretPos property is 0-based. The first character in a line is 0. Use the [TextLine](#) property to get the line based on its index.

method CalcEdit.ClearWildFormats ()

Clears the wild characters expressions collection into a sensitive control.

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

Clears the wild characters expression into a sensitive control. Use the [DeleteWildFormat](#) method to delete a specific wild characters expression

property CalcEdit.Count as Long

Counts the lines in the control.

Type	Description
Long	A long expression that specifies the number of lines in the control.

The Count property gets the number of lines in the control. The [MultiLine](#) property specifies whether the control accepts multiple lines. Use the [InsertText](#) method inserts text/lines to control. Use the [InsertLockedText](#) method inserts locked text/lines to control. Use the [TextLine](#) property to access the line based on its index. Use the [Text](#) property to access the control's text. Use the [InsertText](#) method to insert lines to the control. Use the DeleteLine method to delete a specific line.

The following VB sample prints the line in the control:

```
With CalcEdit1
    Dim i As Long
    For i = 1 To .Count
        Debug.Print .TextLine(i)
    Next
End With
```

The following C++ sample prints the line in the control:

```
for ( long i = 1; i <= m_edit.GetCount(); i++ )
    OutputDebugString( m_edit.GetTextLine( i ) );
```

The following VB.NET sample prints the line in the control:

```
With AxCalcEdit1
    Dim i As Integer
    For i = 1 To .Count
        Debug.WriteLine(.get_TextLine(i))
    Next
End With
```

The following C# sample prints the line in the control:

```
for (int i = 1; i <= axCalcEdit1.Count; i++)
```



```
System.Diagnostics.Debug.WriteLine(axCalcEdit1.get_TextLine(i));
```

The following VFP sample prints the line in the control:

```
with thisform.CalcEdit1.Object  
  local i  
  for i = 1 to .Count  
    wait window nowait .TextLine(i)  
  next  
endwith
```

method CalcEdit.DeleteWildFormat (Expression as String)

Deletes an entry from the wild characters expressions collection.

Type	Description
Expression as String	Deletes a wild characters expression being defined by AddWildFormat method.

You have to be carefully when deleting a wild characters expression. For instance, let's say that we defined the wild expression like follows:

```
With CalcEdit1
    .AddWild ("<b> <fgcolor=FF0000> *;</fgcolor> </b> ")
End With
```

The sample highlights everything that ends with ';'. Use the following sample to delete the wild characters expression:

```
With CalcEdit1
    .DeleteWild " *;"
    .Refresh
End With
```

Use the [ClearWildFormats](#) method to clear the all wild characters expressions. The [Refresh](#) method should be called after DeleteWildFormat method was called to reflect the latest cha

property CalcEdit.DrawGridLines as Boolean

Returns or sets a value that determines whether lines are drawn between rows, or unpopulated areas.

Type	Description
Boolean	A boolean expression that determines whether lines are drawn between rows, or unpopulated areas.

By default, the DrawGridLines property is False, which indicates that the control shows no grid lines. The [LineHeight](#) property specifies an expression that determines the height of the line within the editor. The [GridLineColor](#) property specifies the color to show the grid lines.

The following screen shot shows how grid line colors are displayed:



property CalcEdit.Enabled as Boolean

Enables or disables the control.

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

Use the Enabled property to disable the control. Use the [Locked](#) property to lock the control. If the control is disabled the user cannot change the control's content. The scrollbars are disabled. If the control's is disabled the control's caret is hidden too. If the control is disabled, the control's content looks grayed.

property CalcEdit.EvaluateSel as Boolean

Specifies whether the control evaluates the selection.

Type	Description
Boolean	A Boolean expression that specifies whether the control evaluates the selection.

By default, the EvaluateSel property is True, which indicates that the control evaluates the selection, while it is changed. Set the EvaluateSel property on False, to prevent evaluating the current selection.

method CalcEdit.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 Varian expression that holds the result of the call.

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, you can use the EXPRINT.PrintExt = CONTROL.ExecuteTemplate("me") to print the control's content.

For instance, the following sample retrieves the the handle of the first visible item:

```
Debug.Print CalcEdit1.ExecuteTemplate("Items.FirstVisibleItem()")
```

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 CalcEdit.Export as String

Exports the control's content as text, including the results.

Type	Description
String	A String expression that includes the control's Text plus all results.

The Export property exports each line of the control including its result. Use the [Text](#) property to specify the control's text. The [MultiLine](#) property specifies whether the control accepts multiple lines. Use the [TextLine](#) property to access the line based on its index. Use the [InsertText](#) method inserts text/lines to control. Use the [InsertLockedText](#) method inserts locked text/lines to control. Use the [Result](#) property to get the result, if the expression is valid.

The following screen shot shows how the control is:



while the following shows what the Export gets:

```
100 * 1.5 [=150]
120 * 1.5 [=180]
130 * 1.5 [=195]
Total [=525]
```


property CalcEdit.Font as IFontDisp

Retrieves or sets the control's font.

Type	Description
IFontDisp	A font object that indicates the control's font.

Specifies the control's font. Use the [ForeColor](#) property to specify the control's foreground color. Use the [FormatNumbers](#) property to specify the HTML format for numbers. use the [FormatResult](#) property to specify the HTML format being used to display the result.

property CalcEdit.ForeColor as Color

Specifies the control's foreground color.

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

Use the [BackColor](#) and ForeColor properties to define the control's background and foreground colors. The ForeColor property has no effect if the control's [Enabled](#) property is False. Use the [Picture](#) property to assign a picture on the control's background.

The following VB sample changes the control's foreground color:

```
With CalcEdit1
    .ForeColor = ColorConstants.vbBlack
End With
```

The following C++ sample changes the control's foreground color:

```
m_calcEdit.SetForeColor( RGB(0,0,0) );
```

The following VB.NET sample changes the control's foreground color:

```
With AxCalcEdit1
    .ForeColor = Color.Black
End With
```

The following C# sample changes the control's foreground color:

```
axCalcEdit1.ForeColor = Color.Black;
```

The following VFP sample changes the control's foreground color:

```
With thisform.CalcEdit1.Object
    .ForeColor = RGB(0,0,0)
endwith
```

property CalcEdit.ForeColorLockedLine as Color

Retrieves or sets a value that indicates the line's foreground color when it is locked.

Type	Description
Color	A Color expression that indicates the foreground color for locked line.

The ForeColorLockedLine property specifies the foreground color for locked lines. The property has effect while it is not zero. The [BackColorLockedLine](#) property specifies the foreground color for locked lines. Use the [InsertLockedText](#) method inserts locked text/lines to control.

The following screen shot shows the locked lines with a different back/foreground color:



method CalcEdit.FormatABC (Expression as String, [A as Variant], [B as Variant], [C as Variant])

Formats the A,B,C values based on the giving expression and returns the result.

Type	Description
Expression as String	A String that defines the expression to be evaluated.
A as Variant	A VARIANT expression that indicates the value of the A keyword.
B as Variant	A VARIANT expression that indicates the value of the B keyword.
C as Variant	A VARIANT expression that indicates the value of the C keyword.

Return	Description
Variant	A VARIANT expression that indicates the result of the evaluation the CacEdit.

The FormatABC method formats the A,B,C values based on the giving expression and returns the result.

For instance:

- "A + B + C", adds / concatenates the values of the A, B and C
- "value MIN 0 MAX 99", limits the value between 0 and 99
- "value format ``, formats the value with two decimals, according to the control's panel setting
- "date(`now`)" returns the current time as double

The FormatABC method supports the following keywords, constants, operators and functions:

- **A** or **value** keyword, indicates a variable A whose value is giving by the A parameter
- **B** keyword, indicates a variable B whose value is giving by the B parameter
- **C** keyword, indicates a variable C whose value is giving by the C parameter

This property/method supports predefined constants and operators/functions as described [here](#).

property CalcEdit.FormatCountResult as String

Specifies the HTML format to display the result of a Count line.

Type	Description
String	A string expression that indicates the HTML format being used to display the result for a Count line. The FormatCountResult property should include %% sequence that's replaced with the result. The FormatCountResult supports also %I%, which is replaced by the evaluation of the FormatLocal property using the current result.

By default, the FormatCountResult property is "(%%)". The [AllowCount](#) property defines the Count keyword. The **Count** keyword, counts all lines that are not empty, valid and defines no variables. Use the [Result](#) property to retrieve the result on specified line. The [TextLine](#) property specifies the content of the giving line. Use the [IsValid](#) property to specify whether the expression on giving line, is syntactically correct and may be evaluated. The Result is not displayed, if the FormatCountResult property is empty. For instance, the format "</r>%%" displays the result in the right side of the control.

The list of supported built-in HTML tags is:

- bold
- <i>italic</i>
- <s>strikeout</s>
- <u>underline</u>
- <fgcolor=RRGGBB>fgcolor</fgcolor>
- <bcolor=RRGGBB>**bcolor**</bcolor>
- text 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.

For instance, let's say we have the following sample:

```
With CalcEdit1
    .MultiLine = True
    .InsertText "100 * 200"
    .InsertText "300 * 400 * 1.5"
    .InsertText "200 + ( 400 * 1.5 + 300 / 1.19)"
End With
```

The following screen shot shows the control if the FormatResult property is "
 <fgcolor=808080>[=%%]</fgcolor>" (default), FormatLocal property is "" (default)



```

100 * 200 [=20000]
300 * 400 * 1.5 [=180000]
200 + ( 400 * 1.5 + 300 / 1.19) [=1052.10]
  
```

The following screen shot shows the control if the FormatResult property is "
 <fgcolor=808080><r> = %%", FormatLocal property is "" (default)



```

100 * 200                = 20000
300 * 400 * 1.5          = 180000
200 + ( 400 * 1.5 + 300 / 1.19) = 1052.10
  
```

The following screen shot shows the control if the FormatResult property is "
 <fgcolor=808080><r> = %l%", FormatLocal property is "" (default)



```

100 * 200                = 20,000
300 * 400 * 1.5          = 180,000
200 + ( 400 * 1.5 + 300 / 1.19) = 1,052.10
  
```

The following screen shot shows the control if the FormatResult property is "
 <fgcolor=808080><r> = %l%", FormatLocal property is "currency(value)"



```

100 * 200                = $20,000.00
300 * 400 * 1.5          = $180,000.00
200 + ( 400 * 1.5 + 300 / 1.19) = $1,052.10
  
```

The following screen shot shows the control if the FormatResult property is "
 <fgcolor=808080><r> = %l%", FormatLocal property is "value format `2`"



```

100 * 200                = 20,000.00
300 * 400 * 1.5          = 180,000.00
200 + ( 400 * 1.5 + 300 / 1.19) = 1,052.10
  
```

The following screen shot shows the control if the FormatResult property is "
 <fgcolor=808080><r> = %l%", FormatLocal property is "value"



```

100 * 200                = 20000
300 * 400 * 1.5          = 180000
200 + ( 400 * 1.5 + 300 / 1.19) = 1052.10084033613
  
```

The following screen shot shows the control if the FormatResult property is "
 <fgcolor=808080><r> = %l%", FormatLocal property is "(value > 10000 ?
 `<fgcolor=FF0000>` : ``) + (value format `2`)"

100 * 200	= 20,000.00
300 * 400 * 1.5	= 180,000.00
200 + (400 * 1.5 + 300 / 1.19)	= 1,052.10

The following screen shot shows the control if the FormatResult property is "
 <fgcolor=808080><r> = %l%", FormatLocal property is "(value < 10000 ?
 `<fgcolor=000000>` : ``) + (value format `2`)"

100 * 200	= 20,000.00
300 * 400 * 1.5	= 180,000.00
200 + (400 * 1.5 + 300 / 1.19)	= 1,052.10

property CalcEdit.FormatInvalid as String

Specifies the HTML format to show invalid lines.

Type	Description
String	A String expression that defines the HTML format to show invalid lines.

By default, the FormatInvalid property is "<fgcolor=FF0000><s> </s></fgcolor>", which indicates that invalid lines are shown in red as showing in the following screen shot. If the FormatInvalid property is "", the control does not highlight the invalid lines. The [AllowFormatInvalidOnTyping](#) property specifies whether the FormatInvalid property is applied on the current line, while typing into the control.



An invalid line is not evaluated, and so no result is being shown. The [FormatResult](#) property specifies the HTML format of the result. The [FormatTotalResult](#) / [FormatSubTotalResult](#) property formats the Total / Sub-Total lines. The [FormatCountResult](#) / [FormatSubCountResult](#) property specifies the HTML format of lines that contains Count or SubCount aggregate function.

The list of supported built-in HTML tags is:

- **bold**
- **<i>italic</i>**
- **<s>strikeout</s>**
- **<u>underline</u>**
- **<fgcolor=RRGGBB>fgcolor</fgcolor>**
- **<bbgcolor=RRGGBB>bgcolor</bgcolor>**
- **text ** 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.

property CalcEdit.FormatLocal as String

Indicates the expression that defines the formatted value being replaced in FormatResult properties, when %I% is found.

Type	Description
String	A String expression that defines the formula to display the result when %I% is found in the FormatResult properties.

By default, the FormatLocal property is "", which indicates that the %I% displays the result as indicated by the current locale (regional settings for numbers).

The FormatLocal property has effect for any of the following properties:

- [FormatResult](#) property specifies the HTML format of the result of each line.
- [FormatTotalResult](#) property specifies the HTML format to display the result of a Total line.
- [FormatSubTotalResult](#) property specifies the HTML format to display the result of a SubTotal line.
- [FormatCountResult](#) property specifies the HTML format to display the result of a Count line.
- [FormatSubCountResult](#) property specifies the HTML format to display the result of a SubCount line.

if it contains a %I% sequence.

For instance,

if the FormatResult property is:

- "<r> = %I%", displays the result of each line aligned to the right of the control using the evaluation of the FormatLocal property

and the FormatLocal property is:

- "currency(value)", gets the value(result) and formats it using current regional setting including the current currency symbol

each line displays the result using current regional setting including the current currency symbol.

For instance, let's say we have the following sample:

With CalcEdit1

```
.MultiLine = True  
.InsertText "100 * 200"  
.InsertText "300 * 400 * 1.5"  
.InsertText "200 + ( 400 * 1.5 + 300 / 1.19)"  
End With
```

The following screen shot shows the control if the FormatResult property is "`<fgcolor=808080>[=%%]</fgcolor>`" (default), FormatLocal property is "" (default)



100 * 200 [=20000]
300 * 400 * 1.5 [=180000]
200 + (400 * 1.5 + 300 / 1.19) [=1052.10]

The following screen shot shows the control if the FormatResult property is "`<fgcolor=808080><r> = %%`", FormatLocal property is "" (default)



100 * 200	= 20000
300 * 400 * 1.5	= 180000
200 + (400 * 1.5 + 300 / 1.19)	= 1052.10

The following screen shot shows the control if the FormatResult property is "`<fgcolor=808080><r> = %l%`", FormatLocal property is "" (default)



100 * 200	= 20,000
300 * 400 * 1.5	= 180,000
200 + (400 * 1.5 + 300 / 1.19)	= 1,052.10

The following screen shot shows the control if the FormatResult property is "`<fgcolor=808080><r> = %l%`", FormatLocal property is "currency(value)"



100 * 200	= \$20,000.00
300 * 400 * 1.5	= \$180,000.00
200 + (400 * 1.5 + 300 / 1.19)	= \$1,052.10

The following screen shot shows the control if the FormatResult property is "`<fgcolor=808080><r> = %l%`", FormatLocal property is "value format `2`"



100 * 200	= 20,000.00
300 * 400 * 1.5	= 180,000.00
200 + (400 * 1.5 + 300 / 1.19)	= 1,052.10

The following screen shot shows the control if the FormatResult property is "`<fgcolor=808080><r> = %l%`", FormatLocal property is "value"

100 * 200 = 20000
 300 * 400 * 1.5 = 180000
 200 + (400 * 1.5 + 300 / 1.19) = 1052.10084033613

The following screen shot shows the control if the FormatResult property is "`<fgcolor=808080><r> = %l%`", FormatLocal property is "(value > 10000 ? `<fgcolor=FF0000>` : ```) + (value format `2`)"

100 * 200 = 20,000.00
 300 * 400 * 1.5 = 180,000.00
 200 + (400 * 1.5 + 300 / 1.19) = 1,052.10

The following screen shot shows the control if the FormatResult property is "`<fgcolor=808080><r> = %l%`", FormatLocal property is "(value < 10000 ? `<fgcolor=000000>` : ```) + (value format `2`)"

100 * 200 = 20,000.00
 300 * 400 * 1.5 = 180,000.00
 200 + (400 * 1.5 + 300 / 1.19) = 1,052.10

The FormatLocal property supports the **value** keyword which indicates the result/number to be formatted. The FormatLocal property supports the following functions, operators and constants:

The constants are (DPI-Aware components):

- **dpi** (DPI constant), specifies the current DPI setting. and it indicates the minimum value between **dpix** and **dpiy** constants. For instance, if current DPI setting is 100%, the dpi constant returns 1, if 150% it returns 1.5, and so on. For instance, the expression value * dpi returns the value if the DPI setting is 100%, or value * 1.5 in case, the DPI setting is 150%
- **dpix** (DPIX constant), specifies the current DPI setting on x-scale. For instance, if current DPI setting is 100%, the dpix constant returns 1, if 150% it returns 1.5, and so on. For instance, the expression value * dpix returns the value if the DPI setting is 100%, or value * 1.5 in case, the DPI setting is 150%
- **dpiy** (DPIY constant), specifies the current DPI setting on y-scale. For instance, if current DPI setting is 100%, the dpiy constant returns 1, if 150% it returns 1.5, and so on. For instance, the expression value * dpiy returns the value if the DPI setting is 100%, or value * 1.5 in case, the DPI setting is 150%

The supported binary arithmetic operators are:

- * (multiplicity operator), priority 5
- / (divide operator), priority 5

- **mod** (reminder operator), priority 5
- **+** (addition operator), priority 4 (concatenates two strings, if one of the operands is of string type)
- **-** (subtraction operator), priority 4

The supported unary boolean operators are:

- **not** (not operator), priority 3 (high priority)

The supported binary boolean operators are:

- **or** (or operator), priority 2
- **and** (and operator), priority 1

The supported binary boolean operators, all these with the same priority 0, are :

- **<** (less operator)
- **<=** (less or equal operator)
- **=** (equal operator)
- **!=** (not equal operator)
- **>=** (greater or equal operator)
- **>** (greater operator)

The supported binary range operators, all these with the same priority 5, are :

- **MIN** (min operator), indicates the minimum value, so a **MIN** b returns the value of a, if it is less than b, else it returns b. For instance, the expression value MIN 10 returns always a value greater than 10.
- **MAX** (max operator), indicates the maximum value, so a **MAX** b returns the value of a, if it is greater than b, else it returns b. For instance, the expression value MAX 100 returns always a value less than 100.

The supported binary operators, all these with the same priority 0, are :

- **:= (Store operator)**, stores the result of expression to variable. The syntax for := operator is

variable := expression

where variable is a integer between 0 and 9. You can use the **=:** operator to restore any stored variable (please make the difference between **:=** and **=:**). For instance, **(0:=dbl(value)) = 0 ? "zero" : =:0**, stores the value converted to double, and prints zero if it is 0, else the converted number. Please pay attention that the **:=** and **=:** are two distinct operators, the first for storing the result into a variable, while the second for

restoring the variable

- **=: (Restore operator)**, restores the giving variable (previously saved using the store operator). The syntax for =: operator is

=: variable

where variable is a integer between 0 and 9. You can use the := operator to store the value of any expression (please make the difference between := and =:). For instance, $(0:=dbl(value)) = 0 ? "zero" : =:0$, stores the value converted to double, and prints zero if it is 0, else the converted number. Please pay attention that the := and =: are two distinct operators, the first for storing the result into a variable, while the second for restoring the variable

The supported ternary operators, all these with the same priority 0, are :

- **? (Immediate If operator)**, returns and executes one of two expressions, depending on the evaluation of an expression. The syntax for ? operator is

expression ? true_part : false_part

, while it executes and returns the true_part if the expression is true, else it executes and returns the false_part. For instance, the $\%0 = 1 ? 'One' : (\%0 = 2 ? 'Two' : 'not found')$ returns 'One' if the value is 1, 'Two' if the value is 2, and 'not found' for any other value. A n-ary equivalent operation is the case() statement, which is available in newer versions of the component.

The supported n-ary operators are (with priority 5):

- **array (at operator)**, returns the element from an array giving its index (0 base). The array operator returns empty if the element is found, else the associated element in the collection if it is found. The syntax for array operator is

expression array (c1,c2,c3,...cn)

, where the c1, c2, ... are constant elements. The constant elements could be numeric, date or string expressions. For instance the *month(value)-1 array* ('J','F','M','A','M','Jun','J','A','S','O','N','D') is equivalent with *month(value)-1 case* (default:"; 0:'J';1:'F';2:'M';3:'A';4:'M';5:'Jun';6:'J';7:'A';8:'S';9:'O';10:'N';11:'D').

- **in (include operator)**, specifies whether an element is found in a set of constant elements. The in operator returns -1 (True) if the element is found, else 0 (false) is retrieved. The syntax for in operator is

expression in (c1,c2,c3,...cn)

, where the c1, c2, ... are constant elements. The constant elements could be numeric, date or string expressions. For instance the *value in (11,22,33,44,13)* is equivalent with *(expression = 11)* or *(expression = 22)* or *(expression = 33)* or *(expression = 44)* or *(expression = 13)*. The *in* operator is not a time consuming as the equivalent *or* version is, so when you have large number of constant elements it is recommended using the *in* operator. Shortly, if the collection of elements has 1000 elements the *in* operator could take up to 8 operations in order to find if an element fits the set, else if the *or* statement is used, it could take up to 1000 operations to check, so by far, the *in* operator could save time on finding elements within a collection.

- **switch** (*switch operator*), returns the value being found in the collection, or a predefined value if the element is not found (default). The syntax for *switch* operator is

expression switch (default,c1,c2,c3,...,cn)

, where the c1, c2, ... are constant elements, and the default is a constant element being returned when the element is not found in the collection. The constant elements could be numeric, date or string expressions. The equivalent syntax is "%0 = c 1 ? c 1 : (%0 = c 2 ? c 2 : (... ? . : default))". The *switch* operator is very similar with the *in* operator excepts that the first element in the switch is always returned by the statement if the element is not found, while the returned value is the value itself instead -1. For instance, the %0 *switch ('not found',1,4,7,9,11)* gets 1, 4, 7, 9 or 11, or 'not found' for any other value. As the *in* operator the *switch* operator uses binary searches for fitting the element, so it is quicker than *iif* (immediate if operator) alternative.

- **case()** (*case operator*) returns and executes one of n expressions, depending on the evaluation of the expression (*IIF* - immediate IF operator is a binary case() operator). The syntax for *case()* operator is:

expression case ([default : default_expression ;] c1 : expression1 ; c2 : expression2 ; c3 : expression3 ;....)

If the default part is missing, the *case()* operator returns the value of the expression if it is not found in the collection of cases (c1, c2, ...). For instance, if the value of expression is not any of c1, c2, the *default_expression* is executed and returned. If the value of the expression is c1, then the *case()* operator executes and returns the *expression1*. The *default*, c1, c2, c3, ... must be constant elements as numbers, dates or strings. For instance, the *date(shortdate(value)) case (default:0 ; #1/1/2002#:1 ; #2/1/2002#:1; #4/1/2002#:1; #5/1/2002#:1)* indicates that only #1/1/2002#, #2/1/2002#, #4/1/2002# and #5/1/2002# dates returns 1, since the others returns 0. For instance the following sample specifies the hour being non-working for specified

dates: *date(shortdate(value)) case(default:0;#4/1/2009# : hour(value) >= 6 and hour(value) <= 12 ; #4/5/2009# : hour(value) >= 7 and hour(value) <= 10 or hour(value) in(15,16,18,22); #5/1/2009# : hour(value) <= 8)* statement indicates the working hours for dates as follows:

- *#4/1/2009#*, from hours 06:00 AM to 12:00 PM
- *#4/5/2009#*, from hours 07:00 AM to 10:00 AM and hours 03:00PM, 04:00PM, 06:00PM and 10:00PM
- *#5/1/2009#*, from hours 12:00 AM to 08:00 AM

The *in*, *switch* and *case()* use binary search to look for elements so they are faster then using *iif* and *or* expressions. Obviously, the priority of the operations inside the expression is determined by () parenthesis and the priority for each operator.

The supported conversion unary operators are:

- **type** (unary operator) retrieves the type of the object. For instance *type(%1) = 8* specifies the cells (on the column 1) that contains string values.

Here's few predefined types:

- 0 - empty (not initialized)
- 1 - null
- 2 - short
- 3 - long
- 4 - float
- 5 - double
- 6 - currency
- 7 - date
- 8 - string
- 9 - object
- 10 - error
- 11 - boolean
- 12 - variant
- 13 - any
- 14 - decimal
- 16 - char
- 17 - byte
- 18 - unsigned short
- 19 - unsigned long
- 20 - long on 64 bits
- 21 - unsigned long on 64 bites
- **str** (unary operator) converts the expression to a string. The *str* operator converts the expression to a string. For instance, the *str(-12.54)* returns the string "-12.54".

- **dbl** (unary operator) converts the expression to a number. The *dbl* operator converts the expression to a number. For instance, the *dbl("12.54")* returns 12.54
- **date** (unary operator) converts the expression to a date, based on your regional settings. For instance, the *date(``)* gets the current date (no time included), the *date(`now`)* gets the current date-time, while the *date("01/01/2001")* returns #1/1/2001#
- **dateS** (unary operator) converts the string expression to a date using the format MM/DD/YYYY HH:MM:SS. For instance, the *dateS("01/01/2001 14:00:00")* returns #1/1/2001 14:00:00#

Other known operators for numbers are:

- **int** (unary operator) retrieves the integer part of the number. For instance, the *int(12.54)* returns 12
- **round** (unary operator) rounds the number ie 1.2 gets 1, since 1.8 gets 2. For instance, the *round(12.54)* returns 13
- **floor** (unary operator) returns the largest number with no fraction part that is not greater than the value of its argument. For instance, the *floor(12.54)* returns 12
- **abs** (unary operator) retrieves the absolute part of the number ie -1 gets 1, 2 gets 2. For instance, the *abs(-12.54)* returns 12.54
- **sin** (unary operator) returns the sine of an angle of x radians. For instance, the *sin(3.14)* returns 0.001593.
- **cos** (unary operator) returns the cosine of an angle of x radians. For instance, the *cos(3.14)* returns -0.999999.
- **asin** (unary operator) returns the principal value of the arc sine of x, expressed in radians. For instance, the *2*asin(1)* returns the value of PI.
- **acos** (unary operator) returns the principal value of the arc cosine of x, expressed in radians. For instance, the *2*acos(0)* returns the value of PI
- **sqrt** (unary operator) returns the square root of x. For instance, the *sqrt(81)* returns 9.
- **currency** (unary operator) formats the giving number as a currency string, as indicated by the control panel. For instance, *currency(value)* displays the value using the current format for the currency ie, 1000 gets displayed as \$1,000.00, for US format.
- value **format** 'flags' (binary operator) formats the value with specified flags. If flags is empty, the number is displayed as shown in the field "Number" in the "Regional and Language Options" from the Control Panel. For instance the *1000 format "* displays 1,000.00 for English format, while 1.000,00 is displayed for German format. *1000 format '2|.|3|,'* will always displays 1,000.00 no matter of settings in the control panel. If formatting the number fails for some invalid parameter, the value is displayed with no formatting.

The ' flags' for format operator is a list of values separated by | character such as 'NumDigits|DecimalSep|Grouping|ThousandSep|NegativeOrder|LeadingZero' with the

following meanings:

- *NumDigits* - specifies the number of fractional digits, If the flag is missing, the field "No. of digits after decimal" from "Regional and Language Options" is using.
- *DecimalSep* - specifies the decimal separator. If the flag is missing, the field "Decimal symbol" from "Regional and Language Options" is using.
- *Grouping* - indicates the number of digits in each group of numbers to the left of the decimal separator. Values in the range 0 through 9 and 32 are valid. The most significant grouping digit indicates the number of digits in the least significant group immediately to the left of the decimal separator. Each subsequent grouping digit indicates the next significant group of digits to the left of the previous group. If the last value supplied is not 0, the remaining groups repeat the last group. Typical examples of settings for this member are: 0 to group digits as in 123456789.00; 3 to group digits as in 123,456,789.00; and 32 to group digits as in 12,34,56,789.00. If the flag is missing, the field "Digit grouping" from "Regional and Language Options" indicates the grouping flag.
- *ThousandSep* - specifies the thousand separator. If the flag is missing, the field "Digit grouping symbol" from "Regional and Language Options" is using.
- *NegativeOrder* - indicates the negative number mode. If the flag is missing, the field "Negative number format" from "Regional and Language Options" is using. The valid values are 0, 1, 2, 3 and 4 with the following meanings:
 - 0 - Left parenthesis, number, right parenthesis; for example, (1.1)
 - 1 - Negative sign, number; for example, -1.1
 - 2 - Negative sign, space, number; for example, - 1.1
 - 3 - Number, negative sign; for example, 1.1-
 - 4 - Number, space, negative sign; for example, 1.1 -
- *LeadingZero* - indicates if leading zeros should be used in decimal fields. If the flag is missing, the field "Display leading zeros" from "Regional and Language Options" is using. The valid values are 0, 1

Other known operators for strings are:

- **len** (unary operator) retrieves the number of characters in the string. For instance, the *len("Mihai")* returns 5.
- **lower** (unary operator) returns a string expression in lowercase letters. For instance, the *lower("MIHAI")* returns "mihai"
- **upper** (unary operator) returns a string expression in uppercase letters. For instance, the *upper("mihai")* returns "MIHAI"
- **proper** (unary operator) returns from a character expression a string capitalized as appropriate for proper names. For instance, the *proper("mihai")* returns "Mihai"
- **ltrim** (unary operator) removes spaces on the left side of a string. For instance, the *ltrim(" mihai")* returns "mihai"
- **rtrim** (unary operator) removes spaces on the right side of a string. For instance, the

rtrim("mihai ") returns "mihai"

- **trim** (unary operator) removes spaces on both sides of a string. For instance, the *trim(" mihai ")* returns "mihai"
- **reverse** (unary operator) reverses the order of the characters in the string a. For instance, the *reverse("Mihai")* returns "iahIM"
- **startswith** (binary operator) specifies whether a string starts with specified string (0 if not found, -1 if found). For instance *"Mihai" startwith "Mi"* returns -1
- **endwith** (binary operator) specifies whether a string ends with specified string (0 if not found, -1 if found). For instance *"Mihai" endwith "ai"* returns -1
- **contains** (binary operator) specifies whether a string contains another specified string (0 if not found, -1 if found). For instance *"Mihai" contains "ha"* returns -1
- **left** (binary operator) retrieves the left part of the string. For instance *"Mihai" left 2* returns "Mi".
- **right** (binary operator) retrieves the right part of the string. For instance *"Mihai" right 2* returns "ai"
- a **lfind** b (binary operator) The a lfind b (binary operator) searches the first occurrence of the string b within string a, and returns -1 if not found, or the position of the result (zero-index). For instance *"ABCABC" lfind "C"* returns 2
- a **rfind** b (binary operator) The a rfind b (binary operator) searches the last occurrence of the string b within string a, and returns -1 if not found, or the position of the result (zero-index). For instance *"ABCABC" rfind "C"* returns 5.
- a **mid** b (binary operator) retrieves the middle part of the string a starting from b (1 means first position, and so on). For instance *"Mihai" mid 2* returns "ihai"
- a **count** b (binary operator) retrieves the number of occurrences of the b in a. For instance *"Mihai" count "i"* returns 2.
- a **replace** b with c (double binary operator) replaces in a the b with c, and gets the result. For instance, the *"Mihai" replace "i" with ""* returns "Mha" string, as it replaces all "i" with nothing.
- a **split** b, splits the a using the separator b, and returns an array. For instance, the *weekday(value) array 'Sun Mon Thu Wed Thu Fri Sat' split ' '* gets the weekday as string. This operator can be used with the array.

Other known operators for dates are:

- **time** (unary operator) retrieves the time of the date in string format, as specified in the control's panel. For instance, the *time(#1/1/2001 13:00#)* returns "1:00:00 PM"
- **timeF** (unary operator) retrieves the time of the date in string format, as "HH:MM:SS". For instance, the *timeF(#1/1/2001 13:00#)* returns "13:00:00"
- **shortdate** (unary operator) formats a date as a date string using the short date format, as specified in the control's panel. For instance, the *shortdate(#1/1/2001 13:00#)* returns "1/1/2001"
- **shortdateF** (unary operator) formats a date as a date string using the

"MM/DD/YYYY" format. For instance, the *shortdateF*(#1/1/2001 13:00#) returns "01/01/2001"

- **dateF** (unary operator) converts the date expression to a string expression in "MM/DD/YYYY HH:MM:SS" format. For instance, the *dateF*(#01/01/2001 14:00:00#) returns #01/01/2001 14:00:00#
- **longdate** (unary operator) formats a date as a date string using the long date format, as specified in the control's panel. For instance, the *longdate*(#1/1/2001 13:00#) returns "Monday, January 01, 2001"
- **year** (unary operator) retrieves the year of the date (100,...,9999). For instance, the *year*(#12/31/1971 13:14:15#) returns 1971
- **month** (unary operator) retrieves the month of the date (1, 2,...,12). For instance, the *month*(#12/31/1971 13:14:15#) returns 12.
- **day** (unary operator) retrieves the day of the date (1, 2,...,31). For instance, the *day*(#12/31/1971 13:14:15#) returns 31
- **yearday** (unary operator) retrieves the number of the day in the year, or the days since January 1st (0, 1,...,365). For instance, the *yearday*(#12/31/1971 13:14:15#) returns 365
- **weekday** (unary operator) retrieves the number of days since Sunday (0 - Sunday, 1 - Monday,..., 6 - Saturday). For instance, the *weekday*(#12/31/1971 13:14:15#) returns 5.
- **hour** (unary operator) retrieves the hour of the date (0, 1, ..., 23). For instance, the *hour*(#12/31/1971 13:14:15#) returns 13
- **min** (unary operator) retrieves the minute of the date (0, 1, ..., 59). For instance, the *min*(#12/31/1971 13:14:15#) returns 14
- **sec** (unary operator) retrieves the second of the date (0, 1, ..., 59). For instance, the *sec*(#12/31/1971 13:14:15#) returns 15

The Exontrol's [eXPression](#) component is a syntax-editor that helps you to define, view, edit and evaluate expressions. Using the eXPression component you can easily view or check if the expression you have used is syntactically correct, and you can evaluate what is the result you get giving different values to be tested. The Exontrol's eXPression component can be used as an user-editor, to configure your applications.

property CalcEdit.FormatNumbers as String

Specifies the HTML format that's applied to numbers.

Type	Description
String	A string expression that defines the HTML expression being used when control displays numbers.

By default, the FormatNumbers property is "<fgcolor=0000FF> </fgcolor>". By default the numbers get colored in blue. For instance, use the FormatNumbers property on "", and so no numbers will be shown in colors. Use the FormatNumbers to define the appearance for the numbers in the control. If the FormatNumbers property is empty no format is applied to numbers in the control. The [FormatResult](#) property specifies the HTML format of the result. The [FormatInvalid](#) property specifies the HTML format to show invalid lines. The [FormatTotalResult](#) / [FormatSubTotalResult](#) property formats the Total / Sub-Total lines. The [FormatCountResult](#) / [FormatSubCountResult](#) property specifies the HTML format of lines that contains Count or SubCount aggregate function.

$$1+2*(3+4/(1-2*(1+0)+2*(1+1)/2))|[=15]$$

The list of supported built-in HTML tags is:

- **bold**
- **<i>italic</i>**
- **<s>strikeout</s>**
- **<u>underline</u>**
- **<fgcolor=RRGGBB>fgcolor</fgcolor>**
- **<bgcolor=RRGGBB>bgcolor</bgcolor>**
- **text ** 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.

property CalcEdit.FormatResult as String

Specifies the HTML format of the result.

Type	Description
String	A string expression that indicates the HTML format being used to display the result. The FormatResult property should include %% sequence that's replaced with the result. The FormatResult supports also %I%, which is replaced by the evaluation of the FormatLocal property using the current result.

By default, the FormatResult property is "<fgcolor=808080>[=%%]</fgcolor>". Use the [Result](#) property to retrieve the result. The [Text](#) property indicates the control's expression. Use the [IsValid](#) property to specify whether the expression is syntactically correct and may be evaluated. The Result is not displayed, if the FormatResult property is empty. For instance, the format "</r>%%" displays the result in the right side of the control.

$$1+2*(3+4/(1-2*(1+0)+2*(1+1)/2))\text{[=15]}$$

The list of supported built-in HTML tags is:

- **bold**
- **<i>italic</i>**
- **<s>strikeout</s>**
- **<u>underline</u>**
- **<fgcolor=RRGGBB>fgcolor</fgcolor>**
- **<bgcolor=RRGGBB>bgcolor</bgcolor>**
- **text ** 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.

For instance, let's say we have the following sample:

```
With CalcEdit1
  .MultiLine = True
  .InsertText "100 * 200"
  .InsertText "300 * 400 * 1.5"
  .InsertText "200 + ( 400 * 1.5 + 300 / 1.19)"
```

The following screen shot shows the control if the FormatResult property is "`<fgcolor=808080>[=%%]</fgcolor>`" (default), FormatLocal property is "" (default)



A screenshot of a control with a light blue background and a decorative border. It displays three lines of text: "100 * 200 [=20000]", "300 * 400 * 1.5 [=180000]", and "200 + (400 * 1.5 + 300 / 1.19) [=1052.10]". The numbers are in a dark blue font, and the results are in a lighter blue font.

The following screen shot shows the control if the FormatResult property is "`<fgcolor=808080><r> = %%`", FormatLocal property is "" (default)



A screenshot of a control with a light blue background and a decorative border. It displays three lines of text: "100 * 200", "300 * 400 * 1.5", and "200 + (400 * 1.5 + 300 / 1.19)". To the right of each line is a result: "= 20000", "= 180000", and "= 1052.10". The numbers are in a dark blue font, and the results are in a lighter blue font.

The following screen shot shows the control if the FormatResult property is "`<fgcolor=808080><r> = %l%`", FormatLocal property is "" (default)



A screenshot of a control with a light blue background and a decorative border. It displays three lines of text: "100 * 200", "300 * 400 * 1.5", and "200 + (400 * 1.5 + 300 / 1.19)". To the right of each line is a result: "= 20,000", "= 180,000", and "= 1,052.10". The numbers are in a dark blue font, and the results are in a lighter blue font.

The following screen shot shows the control if the FormatResult property is "`<fgcolor=808080><r> = %l%`", FormatLocal property is "currency(value)"



A screenshot of a control with a light blue background and a decorative border. It displays three lines of text: "100 * 200", "300 * 400 * 1.5", and "200 + (400 * 1.5 + 300 / 1.19)". To the right of each line is a result: "= \$20,000.00", "= \$180,000.00", and "= \$1,052.10". The numbers are in a dark blue font, and the results are in a lighter blue font.

The following screen shot shows the control if the FormatResult property is "`<fgcolor=808080><r> = %l%`", FormatLocal property is "value format `2`"



A screenshot of a control with a light blue background and a decorative border. It displays three lines of text: "100 * 200", "300 * 400 * 1.5", and "200 + (400 * 1.5 + 300 / 1.19)". To the right of each line is a result: "= 20,000.00", "= 180,000.00", and "= 1,052.10". The numbers are in a dark blue font, and the results are in a lighter blue font.

The following screen shot shows the control if the FormatResult property is "`<fgcolor=808080><r> = %l%`", FormatLocal property is "value"



A screenshot of a control with a light blue background and a decorative border. It displays three lines of text: "100 * 200", "300 * 400 * 1.5", and "200 + (400 * 1.5 + 300 / 1.19)". To the right of each line is a result: "= 20000", "= 180000", and "= 1052.10084033613". The numbers are in a dark blue font, and the results are in a lighter blue font.

The following screen shot shows the control if the FormatResult property is "

<fgcolor=808080><r> = %l%", FormatLocal property is "(value > 10000 ?
<fgcolor=FF0000>` : ``) + (value format `2`)"



100 * 200	= 20,000.00
300 * 400 * 1.5	= 180,000.00
200 + (400 * 1.5 + 300 / 1.19)	= 1,052.10

The following screen shot shows the control if the FormatResult property is "
<fgcolor=808080><r> = %l%", FormatLocal property is "(value < 10000 ?
<fgcolor=000000>` : ``) + (value format `2`)"



100 * 200	= 20,000.00
300 * 400 * 1.5	= 180,000.00
200 + (400 * 1.5 + 300 / 1.19)	= 1,052.10

property CalcEdit.FormatSubCountResult as String

Specifies the HTML format to display the result of a SubCount line.

Type	Description
String	A string expression that indicates the HTML format being used to display the result for a SubCount line. The FormatSubCountResult property should include %% sequence that's replaced with the result. The FormatSubCountResult supports also %I%, which is replaced by the evaluation of the FormatLocal property using the current result.

By default, the FormatSubCountResult property is "<fgcolor=808080>(%%)</fgcolor>". The [AllowSubCount](#) property defines the SubCount keyword. The **SubCount** keyword, counts all previously lines that are not empty, valid, defines no variables until another SubCount keyword is found. Use the [Result](#) property to retrieve the result on specified line. The [TextLine](#) property specifies the content of the giving line. Use the [IsValid](#) property to specify whether the expression on giving line, is syntactically correct and may be evaluated. The Result is not displayed, if the FormatSubCountResult property is empty. For instance, the format "</r>%%" displays the result in the right side of the control.

The list of supported built-in HTML tags is:

- bold
- <i>italic</i>
- <s>strikeout</s>
- <u>underline</u>
- <fgcolor=RRGGBB>fgcolor</fgcolor>
- <bgcolor=RRGGBB>[bgcolor](#)</bgcolor>
- text 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.

For instance, let's say we have the following sample:

```
With CalcEdit1
    .MultiLine = True
    .InsertText "100 * 200"
    .InsertText "300 * 400 * 1.5"
```


.InsertText "200 + (400 * 1.5 + 300 / 1.19)"
End With

The following screen shot shows the control if the FormatResult property is "
<fgcolor=808080>[=%%]</fgcolor>" (default), FormatLocal property is "" (default)



100 * 200 [=20000]
300 * 400 * 1.5 [=180000]
200 + (400 * 1.5 + 300 / 1.19) [=1052.10]

The following screen shot shows the control if the FormatResult property is "
<fgcolor=808080><r> = %%", FormatLocal property is "" (default)



100 * 200 = 20000
300 * 400 * 1.5 = 180000
200 + (400 * 1.5 + 300 / 1.19) = 1052.10

The following screen shot shows the control if the FormatResult property is "
<fgcolor=808080><r> = %l%", FormatLocal property is "" (default)



100 * 200 = 20,000
300 * 400 * 1.5 = 180,000
200 + (400 * 1.5 + 300 / 1.19) = 1,052.10

The following screen shot shows the control if the FormatResult property is "
<fgcolor=808080><r> = %l%", FormatLocal property is "currency(value)"



100 * 200 = \$20,000.00
300 * 400 * 1.5 = \$180,000.00
200 + (400 * 1.5 + 300 / 1.19) = \$1,052.10

The following screen shot shows the control if the FormatResult property is "
<fgcolor=808080><r> = %l%", FormatLocal property is "value format `2`"



100 * 200 = 20,000.00
300 * 400 * 1.5 = 180,000.00
200 + (400 * 1.5 + 300 / 1.19) = 1,052.10

The following screen shot shows the control if the FormatResult property is "
<fgcolor=808080><r> = %l%", FormatLocal property is "value"



100 * 200 = 20000
300 * 400 * 1.5 = 180000
200 + (400 * 1.5 + 300 / 1.19) = 1052.10084033613

The following screen shot shows the control if the FormatResult property is "
<fgcolor=808080><r> = %l%", FormatLocal property is "(value > 10000 ?
<fgcolor=FF0000>` : ``) + (value format `2`)"



100 * 200 = 20,000.00
300 * 400 * 1.5 = 180,000.00
200 + (400 * 1.5 + 300 / 1.19) = 1,052.10

The following screen shot shows the control if the FormatResult property is "
<fgcolor=808080><r> = %l%", FormatLocal property is "(value < 10000 ?
<fgcolor=000000>` : ``) + (value format `2`)"



100 * 200 = 20,000.00
300 * 400 * 1.5 = 180,000.00
200 + (400 * 1.5 + 300 / 1.19) = 1,052.10

property CalcEdit.FormatSubTotalResult as String

Specifies the HTML format to display the result of a SubTotal line.

Type	Description
String	A string expression that indicates the HTML format being used to display the result for a SubTotal line. The FormatSubTotalResult property should include %% sequence that's replaced with the result. The FormatSubTotalResult supports also %I%, which is replaced by the evaluation of the FormatLocal property using the current result.

By default, the FormatSubTotalResult property is "<fgcolor=808080>[=%%]</fgcolor>". The [AllowSubTotal](#) property defines the SubTotal keyword. The **SubTotal** keyword, specifies the sum of all previously lines that are not empty, valid, defines no variables until another SubTotal keyword is found. Use the [Result](#) property to retrieve the result on specified line. The [TextLine](#) property specifies the content of the giving line. Use the [IsValid](#) property to specify whether the expression on giving line, is syntactically correct and may be evaluated. The Result is not displayed, if the FormatSubTotalResult property is empty. For instance, the format "</r>%%" displays the result in the right side of the control.

$$1+2*(3+4/(1-2*(1+0)+2*(1+1)/2)) \text{ [=15]}$$

The list of supported built-in HTML tags is:

- bold
- <i>italic</i>
- <s>strikeout</s>
- <u>underline</u>
- <fgcolor=RRGGBB>fgcolor</fgcolor>
- <bgcolor=RRGGBB>[bgcolor](#)</bgcolor>
- text 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.

For instance, let's say we have the following sample:

```
With CalcEdit1
    .MultiLine = True
```

```
.InsertText "100 * 200"
```

```
.InsertText "300 * 400 * 1.5"
```

```
.InsertText "200 + ( 400 * 1.5 + 300 / 1.19)"
```

```
End With
```

The following screen shot shows the control if the FormatResult property is "`<fgcolor=808080>[=%%]</fgcolor>`" (default), FormatLocal property is "" (default)



```
100 * 200 [=20000]  
300 * 400 * 1.5 [=180000]  
200 + ( 400 * 1.5 + 300 / 1.19) [=1052.10]
```

The following screen shot shows the control if the FormatResult property is "`<fgcolor=808080><r> = %%`", FormatLocal property is "" (default)



```
100 * 200           = 20000  
300 * 400 * 1.5     = 180000  
200 + ( 400 * 1.5 + 300 / 1.19) = 1052.10
```

The following screen shot shows the control if the FormatResult property is "`<fgcolor=808080><r> = %l%`", FormatLocal property is "" (default)



```
100 * 200           = 20,000  
300 * 400 * 1.5     = 180,000  
200 + ( 400 * 1.5 + 300 / 1.19) = 1,052.10
```

The following screen shot shows the control if the FormatResult property is "`<fgcolor=808080><r> = %l%`", FormatLocal property is "currency(value)"



```
100 * 200           = $20,000.00  
300 * 400 * 1.5     = $180,000.00  
200 + ( 400 * 1.5 + 300 / 1.19) = $1,052.10
```

The following screen shot shows the control if the FormatResult property is "`<fgcolor=808080><r> = %l%`", FormatLocal property is "value format `2`"



```
100 * 200           = 20,000.00  
300 * 400 * 1.5     = 180,000.00  
200 + ( 400 * 1.5 + 300 / 1.19) = 1,052.10
```

The following screen shot shows the control if the FormatResult property is "`<fgcolor=808080><r> = %l%`", FormatLocal property is "value"

100 * 200	= 20000
300 * 400 * 1.5	= 180000
200 + (400 * 1.5 + 300 / 1.19)	= 1052.10084033613

The following screen shot shows the control if the FormatResult property is "
 <fgcolor=808080><r> = %l%", FormatLocal property is "(value > 10000 ?
 `<fgcolor=FF0000>` : ``) + (value format `2`)"

100 * 200	= 20,000.00
300 * 400 * 1.5	= 180,000.00
200 + (400 * 1.5 + 300 / 1.19)	= 1,052.10

The following screen shot shows the control if the FormatResult property is "
 <fgcolor=808080><r> = %l%", FormatLocal property is "(value < 10000 ?
 `<fgcolor=000000>` : ``) + (value format `2`)"

100 * 200	= 20,000.00
300 * 400 * 1.5	= 180,000.00
200 + (400 * 1.5 + 300 / 1.19)	= 1,052.10

property CalcEdit.FormatTotalResult as String

Specifies the HTML format to display the result of a Total line.

Type	Description
String	A string expression that indicates the HTML format being used to display the result for a Total line. The FormatTotalResult property should include %% sequence that's replaced with the result. The FormatTotalResult supports also %I%, which is replaced by the evaluation of the FormatLocal property using the current result.

By default, the FormatTotalResult property is "[=%%]". The [AllowTotal](#) property defines the Total keyword. The **Total** keyword, specifies the sum of all lines that are not empty, valid and defines no variables. Use the [Result](#) property to retrieve the result on specified line. The [TextLine](#) property specifies the content of the giving line. Use the [IsValid](#) property to specify whether the expression on giving line, is syntactically correct and may be evaluated. The Result is not displayed, if the FormatTotalResult property is empty. For instance, the format "</r>%%" displays the result in the right side of the control.

$$1+2*(3+4/(1-2*(1+0)+2*(1+1)/2))$$
[=15]

The list of supported built-in HTML tags is:

- **bold**
- **<i>italic</i>**
- **<s>strikeout</s>**
- **<u>underline</u>**
- **<fgcolor=RRGGBB>fgcolor</fgcolor>**
- **<bcolor=RRGGBB>bgcolor</bcolor>**
- **text ** 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.

For instance, let's say we have the following sample:

```
With CalcEdit1
    .MultiLine = True
    .InsertText "100 * 200"
    .InsertText "300 * 400 * 1.5"
```

.InsertText "200 + (400 * 1.5 + 300 / 1.19)"
End With

The following screen shot shows the control if the FormatResult property is "
<fgcolor=808080>[=%%]</fgcolor>" (default), FormatLocal property is "" (default)



100 * 200	[=20000]
300 * 400 * 1.5	[=180000]
200 + (400 * 1.5 + 300 / 1.19)	[=1052.10]

The following screen shot shows the control if the FormatResult property is "
<fgcolor=808080><r> = %%", FormatLocal property is "" (default)



100 * 200	= 20000
300 * 400 * 1.5	= 180000
200 + (400 * 1.5 + 300 / 1.19)	= 1052.10

The following screen shot shows the control if the FormatResult property is "
<fgcolor=808080><r> = %I%", FormatLocal property is "" (default)



100 * 200	= 20,000
300 * 400 * 1.5	= 180,000
200 + (400 * 1.5 + 300 / 1.19)	= 1,052.10

The following screen shot shows the control if the FormatResult property is "
<fgcolor=808080><r> = %I%", FormatLocal property is "currency(value)"



100 * 200	= \$20,000.00
300 * 400 * 1.5	= \$180,000.00
200 + (400 * 1.5 + 300 / 1.19)	= \$1,052.10

The following screen shot shows the control if the FormatResult property is "
<fgcolor=808080><r> = %I%", FormatLocal property is "value format `2`"



100 * 200	= 20,000.00
300 * 400 * 1.5	= 180,000.00
200 + (400 * 1.5 + 300 / 1.19)	= 1,052.10

The following screen shot shows the control if the FormatResult property is "
<fgcolor=808080><r> = %I%", FormatLocal property is "value"



100 * 200	= 20000
300 * 400 * 1.5	= 180000
200 + (400 * 1.5 + 300 / 1.19)	= 1052.10084033613

The following screen shot shows the control if the FormatResult property is "
<fgcolor=808080><r> = %l%", FormatLocal property is "(value > 10000 ?
<fgcolor=FF0000>` : ``) + (value format `2`)"



100 * 200	= 20,000.00
300 * 400 * 1.5	= 180,000.00
200 + (400 * 1.5 + 300 / 1.19)	= 1,052.10

The following screen shot shows the control if the FormatResult property is "
<fgcolor=808080><r> = %l%", FormatLocal property is "(value < 10000 ?
<fgcolor=000000>` : ``) + (value format `2`)"



100 * 200	= 20,000.00
300 * 400 * 1.5	= 180,000.00
200 + (400 * 1.5 + 300 / 1.19)	= 1,052.10

property CalcEdit.GridLineColor as Color

Specifies the grid line color.

Type	Description
Color	A Color expression that specifies the color to show the grid lines.

The GridLineColor property specifies the color to show the grid lines. The [DrawGridLines](#) property specifies a value that determines whether lines are drawn between rows, or unpopulated areas. The [LineHeight](#) property specifies an expression that determines the height of the line within the editor.

The following screen shot shows how grid line colors are displayed:



property CalcEdit.HideSelection as Boolean

Specifies whether the selection in the control is hidden when the control loses the focus.

Type	Description
Boolean	A boolean expression that specifies whether the selection is visible when the control loses the focus.

Use the HideSelection property to hide the selection when control loses the focus. Use the [SelForeColor](#) and [SelBackColor](#) properties to define the colors used to paint the selection. Use the [SelStart](#), [SelLength](#) and [SelText](#) properties to access the selection. Set the [EvaluateSel](#) property on False, to prevent evaluating the current selection

property CalcEdit.hWnd as Long

Retrieves the control's window handle.

Type	Description
Long	A long expression that indicates the window's handle.

The Microsoft Windows operating environment identifies each form 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. Because the value of this property can change while a program is running, you cannot rely on its value (e.g., when stored in a variable)

method CalcEdit.InsertLockedText (Text as String, [Index as Variant])

Inserts locked text to the control.

Type	Description
Text as String	A string expression being inserted.
Index as Variant	A long expression that defines the index of line where the text follows to be inserted. If missing or negative, the text is added at the end of the control's text

Use the InsertLockedText method inserts locked text/lines to control. A locked line / text can not be removed or deleted at runtime. For instance, you can add a total line, that user can not edit or remove it. Use the [InsertText](#) method inserts text/lines to control. By default, the [MultiLine](#) property is False, which indicates that the control can display a single line only. Use the [Text](#) property to specify the control's text. The control's text is evaluated using arithmetic operators. Use the [Result](#) property to get the result, if the expression is valid. Use the [IsValid](#) property to specify whether the Text property is syntactically correct, and may be evaluated. The result is displayed as the user types the expression. The control fires the [Change](#) event when the user alters the expression. The [BackColorLockedLine](#) property specifies the foreground color for locked lines. The [ForeColorLockedLine](#) property specifies the foreground color for locked lines.

method CalcEdit.InsertText (Text as String, [Index as Variant])

Inserts text to control.

Type	Description
Text as String	A string expression being inserted.
Index as Variant	A long expression that defines the index of line where the text follows to be inserted. If missing or negative, the text is added at the end of the control's text.

Use the InsertText method inserts text/lines to control. By default, the [MultiLine](#) property is False, which indicates that the control can display a single line only. Use the [Text](#) property to specify the control's text. The control's text is evaluated using arithmetic operators. Use the [InsertLockedText](#) method inserts locked text/lines to control. Use the [Result](#) property to get the result, if the expression is valid. Use the [IsValid](#) property to specify whether the Text property is syntactically correct, and may be evaluated. The result is displayed as the user types the expression. The control fires the [Change](#) event when the user alters the expression.

property CalcEdit.IsValid ([Line as Variant]) as Boolean

Specifies whether the expression is valid.

Type	Description
Line as Variant	A Long expression that specifies the index of the line. 1 indicates the first line in the control, 2 indicates the second and so on...
Boolean	A boolean expression that indicates whether the expression being evaluated is syntactically correct.

The [Text](#) property indicates the expression being evaluated. The [Result](#) property returns the result of the evaluation, if the expression is valid, else 0 is returned. Use the IsValid property to programmatically determine when the control's expression is valid or not. The control does not display the result of the evaluation, if the expression is not valid. The [TextLine](#) property specifies the text / expression on specified line.

property CalcEdit.LineHeight as String

Specifies an expression that determines the height of the line within the editor.

Type	Description
String	A String expression that determines the height of the line within the editor.

By default, the LineHeight property is empty, which indicates that the control computes automatically the line height based on the control's [Font](#) property. If the LineHeight's expression is empty, invalid, evaluated to zero or negative, the line height is automatically computed based on the control's Font property. You can use the LineHeight property to increase or decrease the default's line height. The DrawGridLines property returns or sets a value that determines whether lines are drawn between rows, or unpopulated areas.

For instance:

- "value + 4*dpi", increases the default line height with 4 dots (4 pixels for 100% DPI settings, 6 pixels for 150% DPI settings, and so on)
- "value - 4*dpi", decreases the default line height with 4 dots (4 pixels for 100% DPI settings, 6 pixels for 150% DPI settings, and so on)
- "18", specifies that the line height is exactly 18 pixels.
- "18*dpi", specifies that the line height is exactly 18 dots (18 pixels for 100% DPI settings, 27 pixels for 150% DPI settings, and so on)

The **value** keyword in the LineHeight's expression indicates the default line height based on the control's font.

The Exontrol's [eXPression](#) component is a syntax-editor that helps you to define, view, edit and evaluate expressions. Using the eXPression component you can easily view or check if the expression you have used is syntactically correct, and you can evaluate what is the result you get giving different values to be tested. The Exontrol's eXPression component can be used as an user-editor, to configure your applications.

The constants are (DPI-Aware components):

- **dpi** (DPI constant), specifies the current DPI setting. and it indicates the minimum value between **dpix** and **dpiy** constants. For instance, if current DPI setting is 100%, the dpi constant returns 1, if 150% it returns 1.5, and so on. For instance, the expression value * dpi returns the value if the DPI setting is 100%, or value * 1.5 in case, the DPI setting is 150%
- **dpix** (DPIX constant), specifies the current DPI setting on x-scale. For instance, if current DPI setting is 100%, the dpix constant returns 1, if 150% it returns 1.5, and so on. For instance, the expression value * dpix returns the value if the DPI setting is

100%, or value * 1.5 in case, the DPI setting is 150%

- **dpiy** (DPIY constant), specifies the current DPI setting on x-scale. For instance, if current DPI setting is 100%, the dpiy constant returns 1, if 150% it returns 1.5, and so on. For instance, the expression value * dpiy returns the value if the DPI setting is 100%, or value * 1.5 in case, the DPI setting is 150%

The supported binary arithmetic operators are:

- * (multiplicity operator), priority 5
- / (divide operator), priority 5
- **mod** (reminder operator), priority 5
- + (addition operator), priority 4 (concatenates two strings, if one of the operands is of string type)
- - (subtraction operator), priority 4

The supported unary boolean operators are:

- **not** (not operator), priority 3 (high priority)

The supported binary boolean operators are:

- **or** (or operator), priority 2
- **and** (or operator), priority 1

The supported binary boolean operators, all these with the same priority 0, are :

- < (less operator)
- <= (less or equal operator)
- = (equal operator)
- != (not equal operator)
- >= (greater or equal operator)
- > (greater operator)

The supported binary range operators, all these with the same priority 5, are :

- **MIN** (min operator), indicates the minimum value, so a **MIN** b returns the value of a, if it is less than b, else it returns b. For instance, the expression value MIN 10 returns always a value greater than 10.
- **MAX** (max operator), indicates the maximum value, so a **MAX** b returns the value of a, if it is greater than b, else it returns b. For instance, the expression value MAX 100 returns always a value less than 100.

The supported binary operators, all these with the same priority 0, are :

- **:= (Store operator)**, stores the result of expression to variable. The syntax for := operator is

variable := expression

where variable is a integer between 0 and 9. You can use the := operator to restore any stored variable (please make the difference between := and =:). For instance, $(0:=dbl(value)) = 0 ? "zero" : =:0$, stores the value converted to double, and prints zero if it is 0, else the converted number. Please pay attention that the := and =: are two distinct operators, the first for storing the result into a variable, while the second for restoring the variable

- **=: (Restore operator)**, restores the giving variable (previously saved using the store operator). The syntax for =: operator is

=: variable

where variable is a integer between 0 and 9. You can use the := operator to store the value of any expression (please make the difference between := and =:). For instance, $(0:=dbl(value)) = 0 ? "zero" : =:0$, stores the value converted to double, and prints zero if it is 0, else the converted number. Please pay attention that the := and =: are two distinct operators, the first for storing the result into a variable, while the second for restoring the variable

The supported ternary operators, all these with the same priority 0, are :

- **? (Immediate If operator)**, returns and executes one of two expressions, depending on the evaluation of an expression. The syntax for ? operator is

expression ? true_part : false_part

, while it executes and returns the true_part if the expression is true, else it executes and returns the false_part. For instance, the $\%0 = 1 ? 'One' : (\%0 = 2 ? 'Two' : 'not found')$ returns 'One' if the value is 1, 'Two' if the value is 2, and 'not found' for any other value. A n-ary equivalent operation is the case() statement, which is available in newer versions of the component.

The supported n-ary operators are (with priority 5):

- **array (at operator)**, returns the element from an array giving its index (0 base). The array operator returns empty if the element is found, else the associated element in the collection if it is found. The syntax for array operator is

expression array (c1,c2,c3,...cn)

, where the c_1, c_2, \dots are constant elements. The constant elements could be numeric, date or string expressions. For instance the *month(value)-1 array* ('J','F','M','A','M','Jun','J','A','S','O','N','D') is equivalent with *month(value)-1 case* (default:"; 0:'J';1:'F';2:'M';3:'A';4:'M';5:'Jun';6:'J';7:'A';8:'S';9:'O';10:'N';11:'D').

- **in** (*include operator*), specifies whether an element is found in a set of constant elements. The *in* operator returns -1 (True) if the element is found, else 0 (false) is retrieved. The syntax for *in* operator is

expression in (c1,c2,c3,...cn)

, where the c_1, c_2, \dots are constant elements. The constant elements could be numeric, date or string expressions. For instance the *value in (11,22,33,44,13)* is equivalent with (*expression = 11*) or (*expression = 22*) or (*expression = 33*) or (*expression = 44*) or (*expression = 13*). The *in* operator is not a time consuming as the equivalent *or* version is, so when you have large number of constant elements it is recommended using the *in* operator. Shortly, if the collection of elements has 1000 elements the *in* operator could take up to 8 operations in order to find if an element fits the set, else if the *or* statement is used, it could take up to 1000 operations to check, so by far, the *in* operator could save time on finding elements within a collection.

- **switch** (*switch operator*), returns the value being found in the collection, or a predefined value if the element is not found (default). The syntax for *switch* operator is

expression switch (default,c1,c2,c3,...,cn)

, where the c_1, c_2, \dots are constant elements, and the default is a constant element being returned when the element is not found in the collection. The constant elements could be numeric, date or string expressions. The equivalent syntax is "%0 = c 1 ? c 1 : (%0 = c 2 ? c 2 : (... ? : default))". The *switch* operator is very similar with the *in* operator excepts that the first element in the switch is always returned by the statement if the element is not found, while the returned value is the value itself instead -1. For instance, the %0 switch ('not found',1,4,7,9,11) gets 1, 4, 7, 9 or 11, or 'not found' for any other value. As the *in* operator the *switch* operator uses binary searches for fitting the element, so it is quicker than *iif* (immediate if operator) alternative.

- **case()** (*case operator*) returns and executes one of n expressions, depending on the evaluation of the expression (*IIF* - immediate IF operator is a binary case() operator). The syntax for *case()* operator is:

expression case ([default : default_expression ;] c1 : expression1 ; c2 : expression2 ; c3 : expression3 ;....)

If the default part is missing, the *case()* operator returns the value of the expression if it

is not found in the collection of cases (c1, c2, ...). For instance, if the value of expression is not any of c1, c2, the `default_expression` is executed and returned. If the value of the expression is c1, then the `case()` operator executes and returns the *expression1*. The *default, c1, c2, c3, ...* must be constant elements as numbers, dates or strings. For instance, the `date(shortdate(value)) case (default:0 ; #1/1/2002#:1 ; #2/1/2002#:1; #4/1/2002#:1; #5/1/2002#:1)` indicates that only #1/1/2002#, #2/1/2002#, #4/1/2002# and #5/1/2002# dates returns 1, since the others returns 0. For instance the following sample specifies the hour being non-working for specified dates: `date(shortdate(value)) case(default:0;#4/1/2009# : hour(value) >= 6 and hour(value) <= 12 ; #4/5/2009# : hour(value) >= 7 and hour(value) <= 10 or hour(value) in(15,16,18,22); #5/1/2009# : hour(value) <= 8)` statement indicates the working hours for dates as follows:

- #4/1/2009#, from hours 06:00 AM to 12:00 PM
- #4/5/2009#, from hours 07:00 AM to 10:00 AM and hours 03:00PM, 04:00PM, 06:00PM and 10:00PM
- #5/1/2009#, from hours 12:00 AM to 08:00 AM

The *in*, *switch* and *case()* use binary search to look for elements so they are faster then using *iif* and *or* expressions. Obviously, the priority of the operations inside the expression is determined by () parenthesis and the priority for each operator.

The supported conversion unary operators are:

- **type** (unary operator) retrieves the type of the object. For instance `type(%1) = 8` specifies the cells (on the column 1) that contains string values.

Here's few predefined types:

- 0 - empty (not initialized)
- 1 - null
- 2 - short
- 3 - long
- 4 - float
- 5 - double
- 6 - currency
- 7 - date
- 8 - string
- 9 - object
- 10 - error
- 11 - boolean
- 12 - variant
- 13 - any
- 14 - decimal

- 16 - char
- 17 - byte
- 18 - unsigned short
- 19 - unsigned long
- 20 - long on 64 bits
- 21 - unsigned long on 64 bites
- **str** (unary operator) converts the expression to a string. The str operator converts the expression to a string. For instance, the *str(-12.54)* returns the string "-12.54".
- **dbl** (unary operator) converts the expression to a number. The dbl operator converts the expression to a number. For instance, the *dbl("12.54")* returns 12.54
- **date** (unary operator) converts the expression to a date, based on your regional settings. For instance, the *date(``)* gets the current date (no time included), the *date(`now`)* gets the current date-time, while the *date("01/01/2001")* returns #1/1/2001#
- **dateS** (unary operator) converts the string expression to a date using the format MM/DD/YYYY HH:MM:SS. For instance, the *dateS("01/01/2001 14:00:00")* returns #1/1/2001 14:00:00#

Other known operators for numbers are:

- **int** (unary operator) retrieves the integer part of the number. For instance, the *int(12.54)* returns 12
- **round** (unary operator) rounds the number ie 1.2 gets 1, since 1.8 gets 2. For instance, the *round(12.54)* returns 13
- **floor** (unary operator) returns the largest number with no fraction part that is not greater than the value of its argument. For instance, the *floor(12.54)* returns 12
- **abs** (unary operator) retrieves the absolute part of the number ie -1 gets 1, 2 gets 2. For instance, the *abs(-12.54)* returns 12.54
- **sin** (unary operator) returns the sine of an angle of x radians. For instance, the *sin(3.14)* returns 0.001593.
- **cos** (unary operator) returns the cosine of an angle of x radians. For instance, the *cos(3.14)* returns -0.999999.
- **asin** (unary operator) returns the principal value of the arc sine of x, expressed in radians. For instance, the *2*asin(1)* returns the value of PI.
- **acos** (unary operator) returns the principal value of the arc cosine of x, expressed in radians. For instance, the *2*acos(0)* returns the value of PI
- **sqrt** (unary operator) returns the square root of x. For instance, the *sqrt(81)* returns 9.
- **currency** (unary operator) formats the giving number as a currency string, as indicated by the control panel. For instance, *currency(value)* displays the value using the current format for the currency ie, 1000 gets displayed as \$1,000.00, for US format.
- value **format** 'flags' (binary operator) formats the value with specified flags. If flags is empty, the number is displayed as shown in the field "Number" in the "Regional and

Language Options" from the Control Panel For instance the *1000 format "* displays 1,000.00 for English format, while 1.000,00 is displayed for German format. 1000 format '*2|.|3|,*' will always displays 1,000.00 no matter of settings in the control panel. If formatting the number fails for some invalid parameter, the value is displayed with no formatting.

The ' flags' for format operator is a list of values separated by | character such as '*NumDigits|DecimalSep|Grouping|ThousandSep|NegativeOrder|LeadingZero*' with the following meanings:

- *NumDigits* - specifies the number of fractional digits, If the flag is missing, the field "No. of digits after decimal" from "Regional and Language Options" is using.
- *DecimalSep* - specifies the decimal separator. If the flag is missing, the field "Decimal symbol" from "Regional and Language Options" is using.
- *Grouping* - indicates the number of digits in each group of numbers to the left of the decimal separator. Values in the range 0 through 9 and 32 are valid. The most significant grouping digit indicates the number of digits in the least significant group immediately to the left of the decimal separator. Each subsequent grouping digit indicates the next significant group of digits to the left of the previous group. If the last value supplied is not 0, the remaining groups repeat the last group. Typical examples of settings for this member are: 0 to group digits as in 123456789.00; 3 to group digits as in 123,456,789.00; and 32 to group digits as in 12,34,56,789.00. If the flag is missing, the field "Digit grouping" from "Regional and Language Options" indicates the grouping flag.
- *ThousandSep* - specifies the thousand separator. If the flag is missing, the field "Digit grouping symbol" from "Regional and Language Options" is using.
- *NegativeOrder* - indicates the negative number mode. If the flag is missing, the field "Negative number format" from "Regional and Language Options" is using. The valid values are 0, 1, 2, 3 and 4 with the following meanings:
 - 0 - Left parenthesis, number, right parenthesis; for example, (1.1)
 - 1 - Negative sign, number; for example, -1.1
 - 2 - Negative sign, space, number; for example, - 1.1
 - 3 - Number, negative sign; for example, 1.1-
 - 4 - Number, space, negative sign; for example, 1.1 -
- *LeadingZero* - indicates if leading zeros should be used in decimal fields. If the flag is missing, the field "Display leading zeros" from "Regional and Language Options" is using. The valid values are 0, 1

Other known operators for strings are:

- **len** (unary operator) retrieves the number of characters in the string. For instance, the *len("Mihai")* returns 5.
- **lower** (unary operator) returns a string expression in lowercase letters. For instance,

the *lower("MIHAI")* returns "mihai"

- **upper** (unary operator) returns a string expression in uppercase letters. For instance, the *upper("mihai")* returns "MIHAI"
- **proper** (unary operator) returns from a character expression a string capitalized as appropriate for proper names. For instance, the *proper("mihai")* returns "Mihai"
- **ltrim** (unary operator) removes spaces on the left side of a string. For instance, the *ltrim(" mihai")* returns "mihai"
- **rtrim** (unary operator) removes spaces on the right side of a string. For instance, the *rtrim("mihai ")* returns "mihai"
- **trim** (unary operator) removes spaces on both sides of a string. For instance, the *trim(" mihai ")* returns "mihai"
- **reverse** (unary operator) reverses the order of the characters in the string a. For instance, the *reverse("Mihai")* returns "iahIM"
- **startswith** (binary operator) specifies whether a string starts with specified string (0 if not found, -1 if found). For instance *"Mihai" startwith "Mi"* returns -1
- **endwith** (binary operator) specifies whether a string ends with specified string (0 if not found, -1 if found). For instance *"Mihai" endwith "ai"* returns -1
- **contains** (binary operator) specifies whether a string contains another specified string (0 if not found, -1 if found). For instance *"Mihai" contains "ha"* returns -1
- **left** (binary operator) retrieves the left part of the string. For instance *"Mihai" left 2* returns "Mi".
- **right** (binary operator) retrieves the right part of the string. For instance *"Mihai" right 2* returns "ai"
- a **lfind** b (binary operator) The a lfind b (binary operator) searches the first occurrence of the string b within string a, and returns -1 if not found, or the position of the result (zero-index). For instance *"ABCABC" lfind "C"* returns 2
- a **rfind** b (binary operator) The a rfind b (binary operator) searches the last occurrence of the string b within string a, and returns -1 if not found, or the position of the result (zero-index). For instance *"ABCABC" rfind "C"* returns 5.
- a **mid** b (binary operator) retrieves the middle part of the string a starting from b (1 means first position, and so on). For instance *"Mihai" mid 2* returns "ihai"
- a **count** b (binary operator) retrieves the number of occurrences of the b in a. For instance *"Mihai" count "i"* returns 2.
- a **replace** b with c (double binary operator) replaces in a the b with c, and gets the result. For instance, the *"Mihai" replace "i" with ""* returns "Mha" string, as it replaces all "i" with nothing.
- a **split** b, splits the a using the separator b, and returns an array. For instance, the *weekday(value) array 'Sun Mon Thu Wed Thu Fri Sat' split ' '* gets the weekday as string. This operator can be used with the array.

Other known operators for dates are:

- **time** (unary operator) retrieves the time of the date in string format, as specified in the control's panel. For instance, the *time(#1/1/2001 13:00#)* returns "1:00:00 PM"
- **timeF** (unary operator) retrieves the time of the date in string format, as "HH:MM:SS". For instance, the *timeF(#1/1/2001 13:00#)* returns "13:00:00"
- **shortdate** (unary operator) formats a date as a date string using the short date format, as specified in the control's panel. For instance, the *shortdate(#1/1/2001 13:00#)* returns "1/1/2001"
- **shortdateF** (unary operator) formats a date as a date string using the "MM/DD/YYYY" format. For instance, the *shortdateF(#1/1/2001 13:00#)* returns "01/01/2001"
- **dateF** (unary operator) converts the date expression to a string expression in "MM/DD/YYYY HH:MM:SS" format. For instance, the *dateF(#01/01/2001 14:00:00#)* returns #01/01/2001 14:00:00#
- **longdate** (unary operator) formats a date as a date string using the long date format, as specified in the control's panel. For instance, the *longdate(#1/1/2001 13:00#)* returns "Monday, January 01, 2001"
- **year** (unary operator) retrieves the year of the date (100,...,9999). For instance, the *year(#12/31/1971 13:14:15#)* returns 1971
- **month** (unary operator) retrieves the month of the date (1, 2,...,12). For instance, the *month(#12/31/1971 13:14:15#)* returns 12.
- **day** (unary operator) retrieves the day of the date (1, 2,...,31). For instance, the *day(#12/31/1971 13:14:15#)* returns 31
- **yearday** (unary operator) retrieves the number of the day in the year, or the days since January 1st (0, 1,...,365). For instance, the *yearday(#12/31/1971 13:14:15#)* returns 365
- **weekday** (unary operator) retrieves the number of days since Sunday (0 - Sunday, 1 - Monday,..., 6 - Saturday). For instance, the *weekday(#12/31/1971 13:14:15#)* returns 5.
- **hour** (unary operator) retrieves the hour of the date (0, 1, ..., 23). For instance, the *hour(#12/31/1971 13:14:15#)* returns 13
- **min** (unary operator) retrieves the minute of the date (0, 1, ..., 59). For instance, the *min(#12/31/1971 13:14:15#)* returns 14
- **sec** (unary operator) retrieves the second of the date (0, 1, ..., 59). For instance, the *sec(#12/31/1971 13:14:15#)* returns 15

property CalcEdit.Locked as Boolean

Determines whether a control can be edited.

Type	Description
Boolean	A boolean expression that determines whether the control can be edited.

Use the Locked property to make the control read-only. The user can select or moves the caret. Use the [Enabled](#) property to disable the control. If the control is disabled, the user can't select or move the caret. Use the [InsertLockedText](#) method inserts locked text/lines to control. The [BackColorLockedLine](#) property specifies the foreground color for locked lines. The [ForeColorLockedLine](#) property specifies the foreground color for locked lines.

property CalcEdit.Margin as Long

Defines the distance between text and inner border.

Type	Description
Long	A long expression that defines the distance between text and inner border.

By default, the Margin property is 2 pixels. Use the Margin property to define the distance between text and inner border.

property CalcEdit.MultiLine as Boolean

Specifies whether the control accepts multiple lines.

Type	Description
Boolean	A Boolean expression that specifies whether the control supports multiple lines.

By default, the MultiLine property is False, which indicates that the control can display a single line only. Use the [Text](#) property to specify the control's text. The control's text is evaluated using arithmetic operators. Use the [TextLine](#) property to access the line based on its index. Use the [InsertText](#) method inserts text/lines to control. Use the [InsertLockedText](#) method inserts locked text/lines to control. Use the [Result](#) property to get the result, if the expression is valid. Use the [IsValid](#) property to specify whether the Text property is syntactically correct, and may be evaluated. The result is displayed as the user types the expression. The control fires the [Change](#) event when the user alters the expression. The Count property gets the number of lines in the control.

property CalcEdit.Overtyp

as Boolean

Specifies whether the control is running in overtyp

Type	Description
Boolean	A boolean expression that indicates whether the control is running the overtyp/overstrike or insert mode.

By default, the Overtyp property is False. The INSERT key toggles between overtyp/overstrike and insert mode. overtyp/overstrike replaces existing characters, insert adds new text where you start typing.

The following VB sample disables Overtyp/Overstrike mode, when the user presses Insert key:

```
Private Sub CalcEdit1_KeyDown(KeyCode As Integer, Shift As Integer)
    If (KeyCode = vbKeyInsert) Then
        KeyCode = 0
    End If
End Sub
```

property CalcEdit.Picture as IPictureDisp

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

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

Use the Picture and [PictureDisplay](#) properties to put a picture on the control's background. If the Picture property is empty no picture is displayed on the control's background. The VB provides method like LoadPicture that loads a picture from a file. Use the [BackColor](#) and [ForeColor](#) properties to define the control's background and foreground colors.

property CalcEdit.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 defines how the control's picture is arranged on control's background.

Use the PictureDisplay property to arrange the control's picture on its background. The PictureDisplay property has no effect if the control's [Picture](#) property is empty. Use the [BackColor](#) property to specify the control's background color.

method CalcEdit.Redo ()

Redoes the next action in the control's redo queue.

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

The control supports multi levels undo/redo support. The CTRL + Z reverses the last editing action, The CTRL + Y restores the previously undone action. Use the Redo method to redo the next action in the control's redo queue. Use the [CanUndo](#) property to determine by code whether an undo operation is available. Use the [CanRedo](#) property to determine by code whether a redo operation is available. Use the [Undo](#) method to undo the last edit-control operation.

method CalcEdit.Refresh ()

Refreshes the control.

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

property CalcEdit.Result ([Line as Variant]) as Double

Retrieves the result.

Type	Description
Line as Variant	A Long expression that specifies the index of the line where to request the result. 1 indicates the first line in the control, 2 indicates the second and so on...
Double	A Double expression that indicates the result of evaluation of the text being typed.

The Result property retrieves the result of the [Text](#) expression. The control does not display the result of the evaluation, if the entered expression is not valid, or the [FormatResult](#) property is empty. Use the FormatResult property to specify how the result should be displayed. The control fires the [Change](#) event when the user alters the expression.

property CalcEdit.SelBackColor as Color

Specifies the selection's background color.

Type	Description
Color	A color expression that specifies the selection's background color.

Use the [SelForeColor](#) and [SelBackColor](#) properties to define the colors used to paint the selection. Use the [SelStart](#), [SelLength](#) and [SelText](#) properties to access the selection. Use the [HideSelection](#) property to specify whether the control hides the selection when the control loses the focus. The control fires the [SelChange](#) event when user changes the selection.

property CalcEdit.SelForeColor as Color

Specifies the selection's foreground color.

Type	Description
Color	A color expression that specifies the selection's foreground color.

Use the SelForeColor and [SelBackColor](#) properties to define the colors used to paint the selection. Use the [SelStart](#), [SelLength](#) and [SelText](#) properties to access the selection. Use the [HideSelection](#) property to specify whether the control hides the selection when the control loses the focus. The control fires the [SelChange](#) event when user changes the selection.

property CalcEdit.SelLength as Long

Returns or sets the number of characters selected.

Type	Description
Long	A long expression that indicates the number of characters selected.

Returns the number of characters the user selects in a text-entry area of a control, or specifies the number of characters to select. Not available at design time; read-write at run time. Use the [SelText](#) property to get the current selection. The [SelStart](#) indicates the starting point of text selected. Set the [EvaluateSel](#) property on False, to prevent evaluating the current selection

property CalcEdit.SelStart as Long

Returns or sets the starting point of text selected; indicates the position of the insertion point if no text is selected.

Type	Description
Long	A long expression that indicates the starting point of text selected.

Returns the starting point of a text selection made by the user in a text-entry area of a control, or indicates the position of the insertion point if no text is selected. Also, specifies the starting point of a text selection in a text-entry area of a control. Not available at design time; read-write at run time. Use the [SelLenght](#) property to get the selection's length. Use the [SelText](#) property to set or get the selected text. Set the [EvaluateSel](#) property on False, to prevent evaluating the current selection

property CalcEdit.SelText as String

Returns or sets the string containing the currently selected text.

Type	Description
String	A string expression that indicates the current selection's text.

Returns the text that the user selected in a text-entry area of a control, or returns an empty string ("") if no characters are selected. Specifies the string containing the selected text. Not available at design time; read-write at run time. The [SelStart](#) property returns or sets the starting point of text selected; indicates the position of the insertion point if no text is selected. The [SelLength](#) property determines the length of the selected text. The control fires the [SelChange](#) event when the user changes the selection. Set the [EvaluateSel](#) property on False, to prevent evaluating the current selection

The following VB sample displays the selected text when the user changes it:

```
Private Sub CalcCalcEdit1_SelChange()  
    Debug.Print CalcEdit1.SelText  
End Sub
```

The following C++ sample displays the selected text when the user changes it:

```
void OnSelChangeCalcEdit1()  
{  
    OutputDebugString( m_edit.GetSelText() );  
}
```

The following VB.NET sample displays the selected text when the user changes it:

```
Private Sub AxCalcCalcEdit1_SelChange(ByVal sender As Object, ByVal e As  
System.EventArgs) Handles AxCalcEdit1.SelChange  
    With AxCalcEdit1  
        Debug.WriteLine(.SelText)  
    End With  
End Sub
```

The following C# sample displays the selected text when the user changes it:

```
private void axCalcCalcEdit1_SelChange(object sender, EventArgs e)
```

```
{  
    System.Diagnostics.Debug.WriteLine(axCalcEdit1.SelText);  
}
```

The following VFP sample displays the selected text when the user changes it:

```
*** ActiveX Control Event ***  
  
with thisform.CalcEdit1  
    wait window nowait .SelText  
endwith
```

property CalcEdit.Template as String

Specifies the control's template.

Type	Description
String	A string expression that defines 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 get the result of executing a template script.

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. For instance, the following code creates an `ADOR.Recordset` and pass it to the control using the `DataSource` property:*

The following sample loads the Orders table:

```
Dim rs
ColumnAutoResize = False
rs = CreateObject("ADOR.Recordset")
{
Open("Orders","Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Program
Files\Exontrol\ExCalcEdit\Sample\SAMPLE.MDB", 3, 3 )
}
DataSource = rs
```


property CalcEdit.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. The TemplateDef and [TemplatePut](#) are equivalents, excepts that the TemplateDef is a property, while the TemplateDef is a method with a parameter.

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)
```

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 CalcEdit.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 methiod 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. The [TemplateDef](#) and TemplatePut are equivalents, excepts that the TemplateDef is a property, while the TemplateDef is a method with a parameter.

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)
```

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.*

property CalcEdit.Text as String

Specifies the control's text.

Type	Description
String	A String expression that indicates the control's text (no HTML included)

Use the Text property to specify the control's text. The control's text is evaluated using arithmetic operators. The [MultiLine](#) property specifies whether the control accepts multiple lines. Use the [TextLine](#) property to access the line based on its index. Use the [InsertText](#) method inserts text/lines to control. Use the [InsertLockedText](#) method inserts locked text/lines to control. Use the [Result](#) property to get the result, if the expression is valid. Use the [IsValid](#) property to specify whether the Text property is syntactically correct, and may be evaluated. The result is displayed as the user types the expression. The control handles double constants and arithmetic operations like +(addition), - (subtraction), / (division), or * (multiply). To enforce a priority, you can use parentheses (). The control fires the [Change](#) event when the user alters the expression. Use the [SelText](#) property to retrieve the selected text. Use the [FormatNumbers](#) property to specify the HTML format for the numbers, and the [FormatResult](#) to specify the HTML format for the result being displayed while editing. The [CalcType](#) property specifies the type of operations the control support. The [Export](#) property exports each line of the control including its result.

property CalcEdit.TextLine(Index as Long) as String

Specifies the line based on its index.

Type	Description
Index as Long	A long expression that defines the index of line being accessed. The Index is 1 based
String	A string expression that defines the line's text (no HTML included)

Use the TextLine property to access a particular line in the control's text. The [MultiLine](#) property specifies whether the control accepts multiple lines. Use the [InsertText](#) method inserts text/lines to control. Use the [InsertLockedText](#) method inserts locked text/lines to control. Use the [Text](#) property to access the control's text when MultiLine property is False. Use the DeleteLine method to delete a particular line. Passing an empty string to the TextLine property doesn't remove the line. It just erases the line's content. The [Count](#) property counts the number of lines in the control.

The following VB sample prints the line in the control:

```
With CalcEdit1
    Dim i As Long
    For i = 1 To .Count
        Debug.Print .TextLine(i)
    Next
End With
```

The following C++ sample prints the line in the control:

```
for ( long i = 1; i <= m_edit.GetCount(); i++ )
    OutputDebugString( m_edit.GetTextLine( i ) );
```

The following VB.NET sample prints the line in the control:

```
With AxCalcEdit1
    Dim i As Integer
    For i = 1 To .Count
        Debug.WriteLine(.get_TextLine(i))
    Next
End With
```

The following C# sample prints the line in the control:

```
for (int i = 1; i <= axCalcEdit1.Count; i++)  
    System.Diagnostics.Debug.WriteLine(axCalcEdit1.get_TextLine(i));
```

The following VFP sample prints the line in the control:

```
with thisform.CalcEdit1.Object  
    local i  
    for i = 1 to .Count  
        wait window nowait .TextLine(i)  
    next  
endwith
```

method CalcEdit.Undo ()

Call this function to undo the last edit-control operation.

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

The control supports multi levels undo/redo support. The CTRL + Z reverses the last editing action, The CTRL + Y restores the previously undone action. Use the Undo method to undo the last edit-control operation. Use the [CanUndo](#) property to determine by code whether an undo operation is available. Use the [CanRedo](#) property to determine by code whether a redo operation is available. Use the [Redo](#) method to redo the next action in the control's redo queue.

property CalcEdit.UseTabKey as Boolean

Specifies whether the control uses the TAB key.

Type	Description
Boolean	A boolean expression that specifies whether the control uses the TAB key.

By default, the UseTabKey is False. If the UseTabKey property is True, the control inserts a TAB character at the caret position, or indents the selection (if multiple lines are selected). If the UseTabKey property is False, the control doesn't handle the TAB key. If the UseTabKey property is False, the TAB key focuses the next visible control in the form. If the [Locked](#) property is True, the UseTabKey property is False.

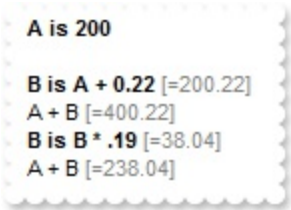
property CalcEdit.Variable(Name as String) as Variant

Indicates the value of the specified variable.

Type	Description
Name as String	A String expression that defines the name of the variable to be requested.
Variant	A VARIANT expression that indicates the value of the requested variable.

The Variable(Name) property specifies the value of giving variable. By default, the control supports variables such as Total and Count, which defines the Total of all valid lines, and count of them. The [AllowVariables](#) property specifies the expression (no HTML) that defines the equal operator, so you can define variables. The [AddWildFormat](#) method formats the line based on the giving wild characters expression. By default, the control has already the wild format defined as "<i>*=*</i>", which draws in italics any line that includes the = (equal) character (define the variables). If the AllowVariables property is "", the control does not support defining any variable. The [CalcType](#) property specifies the type of operations the control supports.

The following samples show how you can define new variables using the "is" keyword, and highlight lines that includes it:



property CalcEdit.Version as String

Retrieves the control's version.

Type	Description
String	A String expression that indicates the version of the control.

The Version property is read-only. Use the Version property to identify the version of the control you are running.

ExCalcEdit 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: {0D4EE794-3E13-4226-81F9-499EE6EDCCF7}. The object's program identifier is: "Exontrol.CalcEdit". The /COM object module is: "ExCalcEdit.dll"

The Exontrol's eXCalcEdit component supports the following events:

Name	Description
Change	Indicates that the control's text has changed.
Click	Occurs when the user presses and then releases the left mouse button over the control.
DbClick	Occurs when the user dblclk the left mouse button over an object.
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.
SelChange	Occurs when the user selects text in the control.

event Change ()

Indicates that the control's text has changed.

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

Use the Change event to notify you application that the user changes the text in the control. Use the [Text](#) property to access the control's text. Use the [Result](#) property to access the result. Use the [SelChange](#) event to notify your application when the user changes the selection, or the cursor is moved to a new position. Use the [CaretLine](#) and [CaretPos](#) properties to determine the caret's position.

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(oCalcEdit)  
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
```

C++

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

XBasic

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

dBASE

```
function nativeObject_Change()  
return
```

The following VB sample displays the result in the output window:

```
Private Sub CalcEdit1_Change()  
    Debug.Print CalcEdit1.Result  
End Sub
```

The following VB.NET sample displays the result in the output window:

```
Private Sub AxCalcEdit1_Change(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles AxCalcEdit1.Change  
    System.Diagnostics.Debug.WriteLine(AxCalcEdit1.Result.ToString())  
End Sub
```

The following C# sample displays the result in the output window:

```
private void axCalcEdit1_Change(object sender, EventArgs e)  
{  
    System.Diagnostics.Debug.WriteLine(axCalcEdit1.Result.ToString());  
}
```

The following C++ sample displays the result in the output window:

```
void OnChangeCalcredit1()  
{
```

```
TCHAR szText[1024] = _T("");  
_stprintf( szText, _T("%f\n"), m_calcEdit.GetResult() );  
OutputDebugString( szText );  
}
```

The following VFP sample displays the result in the output window:

```
*** ActiveX Control Event ***
```

```
with thisform.CalcEdit1
```

```
    ? Str(.Result)
```

```
endwith
```

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

Occurs when the user dbtclk 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

Use the DbtClick event to notify your application that user double clicked the control. By default, the control selects the word from the cursor when the user double clicks the control's client area.

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

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

VBPrivate 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,
AxExCALCEDITLib._ICalcEditEvents_DbtClickEvent e)
{
}

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


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

Delphi

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

Delphi 8
(.NET
only)

```
procedure DbClick(sender: System.Object; e:
AxExCALCEDITLib._ICalcEditEvents_DblClickEvent);
begin
end;
```

Powe...

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

VB.NET

```
Private Sub DbClick(ByVal sender As System.Object, ByVal e As
AxExCALCEDITLib._ICalcEditEvents_DblClickEvent) Handles DbClick
End Sub
```

VB6

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

VBA

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

VFP

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

Xbas...

```
PROCEDURE OnDbClick(oCalcEdit,Shift,X,Y)
RETURN
```

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

Java...

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

VBS...

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

**Visual
Data...**

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

**Visual
Objects**

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

X++

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

XBasic

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

dBASE

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

event KeyDown (ByRef KeyCode as Integer, Shift as Integer)

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

Type	Description
KeyCode as Integer	(By Reference) 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. The control fires the [Change](#) event when the user alters the control's content. Use the [SelChange](#) event to notify your application when the user changes the selection, or the cursor is moved to a new position. 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,
AxExCALCEDITLib._ICalcEditEvents_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:
AxExCALCEDITLib._ICalcEditEvents_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
AxExCALCEDITLib._ICalcEditEvents_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(oCalcEdit,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 (ByRef KeyAscii as Integer)

Occurs when the user presses and releases an ANSI key.

Type	Description
KeyAscii as Integer	(By Reference) 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. The control fires the [Change](#) event when the user alters the control's content.

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,
AxExCALCEDITLib._ICalcEditEvents_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:  
AxExCALCEDITLib._ICalcEditEvents_KeyPressEvent);  
begin  
end;
```

Powe...

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

VB.NET

```
Private Sub KeyPressEvent(ByVal sender As System.Object, ByVal e As  
AxExCALCEDITLib._ICalcEditEvents_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(oCalcEdit,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 (ByRef KeyCode as Integer, Shift as Integer)

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

Type	Description
KeyCode as Integer	(By Reference) 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. The control fires the [Change](#) event when the user alters the control's content.

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,AxExCALCEDITLib._ICalcEditEvents_KeyUpEvent e){}

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

```
void __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:
AxExCALCEDITLib._ICalcEditEvents_KeyUpEvent);
begin
end;
```

Power...

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

VB.NET

```
Private Sub KeyUpEvent(ByVal sender As System.Object, ByVal e As
AxExCALCEDITLib._ICalcEditEvents_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(oCalcEdit,KeyCode,Shift)
RETURN
```

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

Java...

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

VBS...

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

**Visual
Data...**

```
Procedure OnComKeyUp Short IIKeyCode Short IIShift  
    Forward Send OnComKeyUp IIKeyCode IIShift  
End_Procedure
```

**Visual
Objects**

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

X++

```
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,
AxExCALCEDITLib._ICalcEditEvents_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: AxExCALCEDITLib._ICalcEditEvents_MouseDownEvent);
begin
end;

PowerBuilder 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 AxExCALCEDITLib._ICalcEditEvents_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

Xbase++ PROCEDURE OnMouseDown(oCalcEdit,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.CalcEdit.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.CalcEdit.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.

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

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

VBPrivate 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#private void MouseMoveEvent(object sender,
AxExCALCEDITLib._ICalcEditEvents_MouseMoveEvent e)
{
}

C++void OnMouseMove(short Button,short Shift,long X,long Y)

```
{  
}
```

C++
Builder

```
void __fastcall MouseMove(TObject *Sender,short Button,short Shift,int X,int Y)  
{  
}
```

Delphi

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

Delphi 8
(.NET
only)

```
procedure MouseMoveEvent(sender: System.Object; e:  
AxExCALCEDITLib._ICalcEditEvents_MouseMoveEvent);  
begin  
end;
```

Powe...

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

VB.NET

```
Private Sub MouseMoveEvent(ByVal sender As System.Object, ByVal e As  
AxExCALCEDITLib._ICalcEditEvents_MouseMoveEvent) Handles MouseMoveEvent  
End Sub
```

VB6

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

VBA

```
Private Sub MouseMove(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 OnMouseMove(oCalcEdit,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.CalcEdit.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.CalcEdit.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 the [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,
AxExCALCEDITLib._ICalcEditEvents_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: AxExCALCEDITLib._ICalcEditEvents_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 AxExCALCEDITLib._ICalcEditEvents_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(oCalcEdit,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.CalcEdit.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.CalcEdit.1::OLE_YPOS_PIXELS)
end function`

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

event SelChange ()

Occurs when the user selects text in the control.

Type

Description

Use the SelChange event to notify your application that the user changes the selection, or the cursor is moved to a new position. Use the [SelText](#) property to get the selected text. The [SelStart](#) and [SelLenght](#) properties determine the position of the selected text. Use the [SelForeColor](#) and [SelBackColor](#) properties to specify the colors for the selected text. Use the [Change](#) event to notify your application when the user alters the control's text. Use the [CaretLine](#) and [CaretPos](#) properties to determine the caret's position.

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

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

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

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

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

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

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

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

Delphi 8
(.NET
only)

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

Power...

```
begin event SelChange()  
end event SelChange
```

VB.NET

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

VB6

```
Private Sub SelChange()  
End Sub
```

VBA

```
Private Sub SelChange()  
End Sub
```

VFP

```
LPARAMETERS nop
```

Xbas...

```
PROCEDURE OnSelChange(oCalcEdit)  
RETURN
```

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

Java...

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

VBSc...

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

Visual
Data...

```
Procedure OnComSelChange  
Forward Send OnComSelChange
```

End_Procedure

Visual
Objects

METHOD OCX_SelChange() CLASS MainDialog
RETURN NIL

X++

```
void onEvent_SelChange()
{
}
```

XBasic

```
function SelChange as v ()
end function
```

dBASE

```
function nativeObject_SelChange()
return
```

The following VB sample displays the selected text when the user changes it:

```
Private Sub CalcCalcEdit1_SelChange()
    Debug.Print CalcEdit1.SelText
End Sub
```

The following C++ sample displays the selected text when the user changes it:

```
void OnSelChangeCalcEdit1()
{
    OutputDebugString( m_edit.GetSelText() );
}
```

The following VB.NET sample displays the selected text when the user changes it:

```
Private Sub AxCalcCalcEdit1_SelChange(ByVal sender As Object, ByVal e As
System.EventArgs) Handles AxCalcEdit1.SelChange
    With AxCalcEdit1
        Debug.WriteLine(.SelText)
    End With
End Sub
```

The following C# sample displays the selected text when the user changes it:

```
private void axCalcCalcEdit1_SelChange(object sender, EventArgs e)
{
    System.Diagnostics.Debug.WriteLine(axCalcEdit1.SelText);
}
```

The following VFP sample displays the selected text when the user changes it:

```
*** ActiveX Control Event ***

with thisform.CalcEdit1
    wait window nowait .SelText
endwith
```


Expressions

An expression is a string which defines a formula or criteria, that's evaluated at runtime. The expression may be a combination of variables, constants, strings, dates and operators/functions. For instance `1000 format ``` gets `1,000.00` for US format, while `1.000,00` is displayed for German format.

The Exontrol's [eXPression](#) component is a syntax-editor that helps you to define, view, edit and evaluate expressions. Using the eXPression component you can easily view or check if the expression you have used is syntactically correct, and you can evaluate what is the result you get giving different values to be tested. The Exontrol's eXPression component can be used as an user-editor, to configure your applications.

Usage examples:

- `100 + 200`, adds two numbers and returns `300`
- `"100" + 200`, concatenates the strings, and returns `"100200"`
- `currency(1000)` displays the value in currency format based on the current regional setting, such as `"$1,000.00"` for US format.
- `1000 format ``` gets `1,000.00` for English format, while `1.000,00` is displayed for German format
- `1000 format `2|.|3|,`` always gets `1,000.00` no matter of settings in the control panel.
- `date(value) format `MMM d, yyyy``, returns the date such as `Sep 2, 2023`, for English format
- `upper("string")` converts the giving string in uppercase letters, such as `"STRING"`
- `date(dateS('3/1/' + year(9:=#1/1/2018#)) + ((1:=(((255 - 11 * (year(=9) mod 19)) - 21) mod 30) + 21) + (=:1 > 48 ? -1 : 0) + 6 - ((year(=9) + int(year(=9) / 4)) + =:1 + (=:1 > 48 ? -1 : 0) + 1) mod 7))` returns the date the Easter Sunday will fall, for year 2018. In this case the expression returns `#4/1/2018#`. If `#1/1/2018#` is replaced with `#1/1/2019#`, the expression returns `#4/21/2019#`.

Listed bellow are all predefined constants, operators and functions the general-expression supports:

The constants can be represented as:

- numbers in **decimal** format (where dot character specifies the decimal separator). For instance: `-1`, `100`, `20.45`, `.99` and so on
- numbers in **hexa-decimal** format (preceded by `0x` or `0X` sequence), uses sixteen distinct symbols, most often the symbols 0-9 to represent values zero to nine, and A, B, C, D, E, F (or alternatively a, b, c, d, e, f) to represent values ten to fifteen. Hexadecimal numerals are widely used by computer system designers and

programmers. As each hexadecimal digit represents four binary digits (bits), it allows a more human-friendly representation of binary-coded values. For instance, `0xFF`, `0x00FF00`, and so so.

- **date-time** in format `#mm/dd/yyyy hh:mm:ss#`, For instance, `#1/31/2001 10:00#` means the `January 31th, 2001, 10:00 AM`
- **string**, if it starts / ends with any of the ' or ` or " characters. If you require the starting character inside the string, it should be escaped (preceded by a \ character). For instance, ``Mihai``, `"Filimon"`, `'has'`, `"\"a quote\""`, and so on

The predefined constants are:

- **bias** (BIAS constant), defines the difference, in minutes, between Coordinated Universal Time (UTC) and local time. For example, Middle European Time (MET, GMT+01:00) has a time zone bias of "-60" because it is one hour ahead of UTC. Pacific Standard Time (PST, GMT-08:00) has a time zone bias of "+480" because it is eight hours behind UTC. For instance, `date(value - bias/24/60)` converts the UTC time to local time, or `date(date('now') + bias/24/60)` converts the current local time to UTC time. For instance, `"date(value - bias/24/60)"` converts the value date-time from UTC to local time, while `"date(value + bias/24/60)"` converts the local-time to UTC time.
- **dpi** (DPI constant), specifies the current DPI setting. and it indicates the minimum value between **dpix** and **dpiy** constants. For instance, if current DPI setting is 100%, the dpi constant returns 1, if 150% it returns 1.5, and so on. For instance, the expression `value * dpi` returns the value if the DPI setting is 100%, or `value * 1.5` in case, the DPI setting is 150%
- **dpix** (DPIX constant), specifies the current DPI setting on x-scale. For instance, if current DPI setting is 100%, the dpix constant returns 1, if 150% it returns 1.5, and so on. For instance, the expression `value * dpix` returns the value if the DPI setting is 100%, or `value * 1.5` in case, the DPI setting is 150%
- **dpiy** (DPIY constant), specifies the current DPI setting on y-scale. For instance, if current DPI setting is 100%, the dpiy constant returns 1, if 150% it returns 1.5, and so on. For instance, the expression `value * dpiy` returns the value if the DPI setting is 100%, or `value * 1.5` in case, the DPI setting is 150%

The supported binary arithmetic operators are:

- ***** (multiplicity operator), priority 5
- **/** (divide operator), priority 5
- **mod** (remainder operator), priority 5
- **+** (addition operator), priority 4 (concatenates two strings, if one of the operands is of string type)
- **-** (subtraction operator), priority 4

The supported unary boolean operators are:

- **not** (not operator), priority 3 (high priority)

The supported binary boolean operators are:

- **or** (or operator), priority 2
- **and** (or operator), priority 1

The supported binary boolean operators, all these with the same priority 0, are :

- **<** (less operator)
- **<=** (less or equal operator)
- **=** (equal operator)
- **!=** (not equal operator)
- **>=** (greater or equal operator)
- **>** (greater operator)

The supported binary range operators, all these with the same priority 5, are :

- a **MIN** b (min operator), indicates the minimum value, so a **MIN** b returns the value of a, if it is less than b, else it returns b. For instance, the expression **value MIN 10** returns always a value greater than 10.
- a **MAX** b (max operator), indicates the maximum value, so a **MAX** b returns the value of a, if it is greater than b, else it returns b. For instance, the expression **value MAX 100** returns always a value less than 100.

The supported binary operators, all these with the same priority 0, are :

- **:= (Store operator)**, stores the result of expression to variable. The syntax for := operator is

variable := expression

where variable is a integer between 0 and 9. You can use the **=:** operator to restore any stored variable (please make the difference between **:=** and **=:**). For instance, **(0:=dbl(value)) = 0 ? "zero" : =:0**, stores the value converted to double, and prints zero if it is 0, else the converted number. Please pay attention that the **:=** and **=:** are two distinct operators, the first for storing the result into a variable, while the second for restoring the variable

- **=: (Restore operator)**, restores the giving variable (previously saved using the store operator). The syntax for **=:** operator is

=: variable

where variable is a integer between 0 and 9. You can use the `:=` operator to store the value of any expression (please make the difference between `:=` and `=`:). For instance, `(0:=dbl(value)) = 0 ? "zero" : :=0`, stores the value converted to double, and prints zero if it is 0, else the converted number. Please pay attention that the `:=` and `=` are two distinct operators, the first for storing the result into a variable, while the second for restoring the variable

The supported ternary operators, all these with the same priority 0, are :

- **? (Immediate If operator)**, returns and executes one of two expressions, depending on the evaluation of an expression. The syntax for `?` operator is

expression ? true_part : false_part

, while it executes and returns the `true_part` if the expression is true, else it executes and returns the `false_part`. For instance, the `%0 = 1 ? 'One' : (%0 = 2 ? 'Two' : 'not found')` returns 'One' if the value is 1, 'Two' if the value is 2, and 'not found' for any other value. A n-ary equivalent operation is the `case()` statement, which is available in newer versions of the component.

The supported n-ary operators are (with priority 5):

- **array (at operator)**, returns the element from an array giving its index (0 base). The `array` operator returns empty if the element is found, else the associated element in the collection if it is found. The syntax for `array` operator is

expression array (c1,c2,c3,...cn)

, where the `c1`, `c2`, ... are constant elements. The constant elements could be numeric, date or string expressions. For instance the `month(value)-1 array ('J','F','M','A','M','Jun','J','A','S','O','N','D')` is equivalent with `month(value)-1 case (default:"; 0:'J';1:'F';2:'M';3:'A';4:'M';5:'Jun';6:'J';7:'A';8:'S';9:'O';10:'N';11:'D')`.

- **in (include operator)**, specifies whether an element is found in a set of constant elements. The `in` operator returns -1 (True) if the element is found, else 0 (false) is retrieved. The syntax for `in` operator is

expression in (c1,c2,c3,...cn)

, where the `c1`, `c2`, ... are constant elements. The constant elements could be numeric, date or string expressions. For instance the `value in (11,22,33,44,13)` is equivalent with `(expression = 11) or (expression = 22) or (expression = 33) or (expression = 44) or (expression = 13)`. The `in` operator is not a time consuming as the equivalent `or` version is, so when you have large number of constant elements it is recommended using the

in operator. Shortly, if the collection of elements has 1000 elements the *in* operator could take up to 8 operations in order to find if an element fits the set, else if the *or* statement is used, it could take up to 1000 operations to check, so by far, the *in* operator could save time on finding elements within a collection.

- **switch** (*switch operator*), returns the value being found in the collection, or a predefined value if the element is not found (default). The syntax for *switch* operator is

expression switch (default,c1,c2,c3,...,cn)

, where the *c1*, *c2*, ... are constant elements, and the *default* is a constant element being returned when the element is not found in the collection. The constant elements could be numeric, date or string expressions. The equivalent syntax is "%0 = c 1 ? c 1 : (%0 = c 2 ? c 2 : (... ? . : default))". The *switch* operator is very similar with the *in* operator excepts that the first element in the switch is always returned by the statement if the element is not found, while the returned value is the value itself instead -1. For instance, the *%0 switch ('not found',1,4,7,9,11)* gets 1, 4, 7, 9 or 11, or 'not found' for any other value. As the *in* operator the *switch* operator uses binary searches for fitting the element, so it is quicker than the *if* (immediate if operator) alternative.

- **case()** (*case operator*) returns and executes one of *n* expressions, depending on the evaluation of the expression (*IIF* - immediate IF operator is a binary *case()* operator). The syntax for *case()* operator is:

expression case ([default : default_expression ;] c1 : expression1 ; c2 : expression2 ; c3 : expression3 ;....)

If the default part is missing, the *case()* operator returns the value of the expression if it is not found in the collection of cases (*c1*, *c2*, ...). For instance, if the value of expression is not any of *c1*, *c2*, the *default_expression* is executed and returned. If the value of the expression is *c1*, then the *case()* operator executes and returns the *expression1*. The *default*, *c1*, *c2*, *c3*, ... must be constant elements as numbers, dates or strings. For instance, the *date(shortdate(value)) case (default:0 ; #1/1/2002#:1 ; #2/1/2002#:1; #4/1/2002#:1; #5/1/2002#:1)* indicates that only #1/1/2002#, #2/1/2002#, #4/1/2002# and #5/1/2002# dates returns 1, since the others returns 0. For instance the following sample specifies the hour being non-working for specified dates: *date(shortdate(value)) case(default:0;#4/1/2009# : hour(value) >= 6 and hour(value) <= 12 ; #4/5/2009# : hour(value) >= 7 and hour(value) <= 10 or hour(value) in(15,16,18,22); #5/1/2009# : hour(value) <= 8)* statement indicates the working hours for dates as follows:

- #4/1/2009#, from hours 06:00 AM to 12:00 PM
- #4/5/2009#, from hours 07:00 AM to 10:00 AM and hours 03:00PM,

04:00PM, 06:00PM and 10:00PM

- #5/1/2009#, from hours 12:00 AM to 08:00 AM

The *in*, *switch* and *case()* use binary search to look for elements so they are faster than using *if* and *or* expressions. Obviously, the priority of the operations inside the expression is determined by () parenthesis and the priority for each operator.

The supported conversion unary operators are:

- **type** (unary operator) retrieves the type of the object. The type operator may return any of the following: 0 - empty (not initialized), 1 - null, 2 - short, 3 - long, 4 - float, 5 - double, 6 - currency, **7 - date**, **8 - string**, 9 - object, 10 - error, **11 - boolean**, 12 - variant, 13 - any, 14 - decimal, 16 - char, 17 - byte, 18 - unsigned short, 19 - unsigned long, 20 - long on 64 bits, 21 - unsigned long on 64 bits. For instance `type(%1) = 8` specifies the cells (on the column with the index 1) that contains string values.
- **str** (unary operator) converts the expression to a string. The str operator converts the expression to a string. For instance, the `str(-12.54)` returns the string "-12.54".
- **dbl** (unary operator) converts the expression to a number. The dbl operator converts the expression to a number. For instance, the `dbl("12.54")` returns 12.54
- **date** (unary operator) converts the expression to a date, based on your regional settings. For instance, the `date(``)` gets the current date (no time included), the `date(`now`)` gets the current date-time, while the `date("01/01/2001")` returns #1/1/2001#
- **dateS** (unary operator) converts the string expression to a date using the format MM/DD/YYYY HH:MM:SS. For instance, the `dateS("01/01/2001 14:00:00")` returns #1/1/2001 14:00:00#
- **hex** (unary operator) converts the giving string from hexa-representation to a numeric value, or converts the giving numeric value to hexa-representation as string. For instance, `hex(`FF`)` returns 255, while the `hex(255)` or `hex(0xFF)` returns the `FF` string. The `hex(hex(`FFFFFFFF`))` always returns `FFFFFFFF` string, as the second hex call converts the giving string to a number, and the first hex call converts the returned number to string representation (hexa-representation).

The bitwise operators for numbers are:

- a **bitand** b (binary operator) computes the AND operation on bits of a and b, and returns the unsigned value. For instance, `0x01001000 bitand 0x10111000` returns 0x00001000.
- a **bitor** b (binary operator) computes the OR operation on bits of a and b, and returns the unsigned value. For instance, `0x01001000 bitor 0x10111000` returns 0x11111000.
- a **bitxor** b (binary operator) computes the XOR (exclusive-OR) operation on bits of a and b, and returns the unsigned value. For instance, `0x01110010 bitxor 0x10101010` returns 0x11011000.

- a **bitshift** (b) (binary operator) shifts every bit of a value to the left if b is negative, or to the right if b is positive, for b times, and returns the unsigned value. For instance, `128 bitshift 1` returns 64 (dividing by 2) or `128 bitshift (-1)` returns 256 (multiplying by 2)
- **bitnot** (unary operator) flips every bit of x, and returns the unsigned value. For instance, `bitnot(0x00FF0000)` returns 0xFF00FFFF.

The operators for numbers are:

- **int** (unary operator) retrieves the integer part of the number. For instance, the `int(12.54)` returns 12
- **round** (unary operator) rounds the number ie 1.2 gets 1, since 1.8 gets 2. For instance, the `round(12.54)` returns 13
- **floor** (unary operator) returns the largest number with no fraction part that is not greater than the value of its argument. For instance, the `floor(12.54)` returns 12
- **abs** (unary operator) retrieves the absolute part of the number ie -1 gets 1, 2 gets 2. For instance, the `abs(-12.54)` returns 12.54
- **sin** (unary operator) returns the sine of an angle of x radians. For instance, the `sin(3.14)` returns 0.001593.
- **cos** (unary operator) returns the cosine of an angle of x radians. For instance, the `cos(3.14)` returns -0.999999.
- **asin** (unary operator) returns the principal value of the arc sine of x, expressed in radians. For instance, the `2*asin(1)` returns the value of PI.
- **acos** (unary operator) returns the principal value of the arc cosine of x, expressed in radians. For instance, the `2*acos(0)` returns the value of PI
- **sqrt** (unary operator) returns the square root of x. For instance, the `sqrt(81)` returns 9.
- **currency** (unary operator) formats the giving number as a currency string, as indicated by the control panel. For instance, `currency(value)` displays the value using the current format for the currency ie, 1000 gets displayed as \$1,000.00, for US format.
- value **format** 'flags' (binary operator) formats a numeric value with specified flags. The format method formats numeric or date expressions (depends on the type of the value, explained at operators for dates). If flags is empty, the number is displayed as shown in the field "Number" in the "Regional and Language Options" from the Control Panel. For instance the "`1000 format ''`" displays 1,000.00 for English format, while 1.000,00 is displayed for German format. "`1000 format '2|.|3|,'`" will always displays 1,000.00 no matter of the settings in your control panel. If formatting the number fails for some invalid parameter, the value is displayed with no formatting.

The ' flags' for format operator is a list of values separated by | character such as 'NumDigits|DecimalSep|Grouping|ThousandSep|NegativeOrder|LeadingZero' with the following meanings:

- *NumDigits* - specifies the number of fractional digits, If the flag is missing, the

field "No. of digits after decimal" from "Regional and Language Options" is using.

- *DecimalSep* - specifies the decimal separator. If the flag is missing, the field "Decimal symbol" from "Regional and Language Options" is using.
- *Grouping* - indicates the number of digits in each group of numbers to the left of the decimal separator. Values in the range 0 through 9 and 32 are valid. The most significant grouping digit indicates the number of digits in the least significant group immediately to the left of the decimal separator. Each subsequent grouping digit indicates the next significant group of digits to the left of the previous group. If the last value supplied is not 0, the remaining groups repeat the last group. Typical examples of settings for this member are: 0 to group digits as in 123456789.00; 3 to group digits as in 123,456,789.00; and 32 to group digits as in 12,34,56,789.00. If the flag is missing, the field "Digit grouping" from "Regional and Language Options" indicates the grouping flag.
- *ThousandSep* - specifies the thousand separator. If the flag is missing, the field "Digit grouping symbol" from "Regional and Language Options" is using.
- *NegativeOrder* - indicates the negative number mode. If the flag is missing, the field "Negative number format" from "Regional and Language Options" is using. The valid values are 0, 1, 2, 3 and 4 with the following meanings:
 - 0 - Left parenthesis, number, right parenthesis; for example, (1.1)
 - 1 - Negative sign, number; for example, -1.1
 - 2 - Negative sign, space, number; for example, - 1.1
 - 3 - Number, negative sign; for example, 1.1-
 - 4 - Number, space, negative sign; for example, 1.1 -
- *LeadingZero* - indicates if leading zeros should be used in decimal fields. If the flag is missing, the field "Display leading zeros" from "Regional and Language Options" is using. The valid values are 0, 1

The operators for strings are:

- **len** (unary operator) retrieves the number of characters in the string. For instance, the *len("Mihai")* returns 5.
- **lower** (unary operator) returns a string expression in lowercase letters. For instance, the *lower("MIHAI")* returns "mihai"
- **upper** (unary operator) returns a string expression in uppercase letters. For instance, the *upper("mihai")* returns "MIHAI"
- **proper** (unary operator) returns from a character expression a string capitalized as appropriate for proper names. For instance, the *proper("mihai")* returns "Mihai"
- **ltrim** (unary operator) removes spaces on the left side of a string. For instance, the *ltrim(" mihai")* returns "mihai"
- **rtrim** (unary operator) removes spaces on the right side of a string. For instance, the *rtrim("mihai ")* returns "mihai"

- **trim** (unary operator) removes spaces on both sides of a string. For instance, the `trim(" mihai ")` returns "mihai"
- **reverse** (unary operator) reverses the order of the characters in the string a. For instance, the `reverse("Mihai")` returns "iahIM"
- a **startswith** b (binary operator) specifies whether a string starts with specified string (0 if not found, -1 if found). For instance `"Mihai" startswith "Mi"` returns -1
- a **endwith** b (binary operator) specifies whether a string ends with specified string (0 if not found, -1 if found). For instance `"Mihai" endwith "ai"` returns -1
- a **contains** b (binary operator) specifies whether a string contains another specified string (0 if not found, -1 if found). For instance `"Mihai" contains "ha"` returns -1
- a **left** b (binary operator) retrieves the left part of the string. For instance `"Mihai" left 2` returns "Mi".
- a **right** b (binary operator) retrieves the right part of the string. For instance `"Mihai" right 2` returns "ai"
- a **lfind** b (binary operator) The a lfind b (binary operator) searches the first occurrence of the string b within string a, and returns -1 if not found, or the position of the result (zero-index). For instance `"ABCABC" lfind "C"` returns 2
- a **rfind** b (binary operator) The a rfind b (binary operator) searches the last occurrence of the string b within string a, and returns -1 if not found, or the position of the result (zero-index). For instance `"ABCABC" rfind "C"` returns 5.
- a **mid** b (binary operator) retrieves the middle part of the string a starting from b (1 means first position, and so on). For instance `"Mihai" mid 2` returns "ihai"
- a **count** b (binary operator) retrieves the number of occurrences of the b in a. For instance `"Mihai" count "i"` returns 2.
- a **replace** b with c (double binary operator) replaces in a the b with c, and gets the result. For instance, the `"Mihai" replace "i" with ""` returns "Mha" string, as it replaces all "i" with nothing.
- a **split** b (binary operator) splits the a using the separator b, and returns an array. For instance, the `weekday(value) array 'Sun Mon Thu Wed Thu Fri Sat' split ' '` gets the weekday as string. This operator can be used with the array.
- a **like** b (binary operator) compares the string a against the pattern b. The pattern b may contain wild-characters such as *, ?, # or [] and can have multiple patterns separated by space character. In order to have the space, or any other wild-character inside the pattern, it has to be escaped, or in other words it should be preceded by a \ character. For instance `value like 'F*e'` matches all strings that start with F and ends on e, or `value like 'a* b*'` indicates any strings that start with a or b character.
- a **lpad** b (binary operator) pads the value of a to the left with b padding pattern. For instance, `12 lpad "0000"` generates the string "0012".
- a **rpadd** b (binary operator) pads the value of a to the right with b padding pattern. For instance, `12 lpad "____"` generates the string "12__".
- a **concat** b (binary operator) concatenates the a (as string) for b times. For instance, `"x" concat 5`, generates the string "xxxxx".

The operators for dates are:

- **time** (unary operator) retrieves the time of the date in string format, as specified in the control's panel. For instance, the `time(#1/1/2001 13:00#)` returns "1:00:00 PM"
- **timeF** (unary operator) retrieves the time of the date in string format, as "HH:MM:SS". For instance, the `timeF(#1/1/2001 13:00#)` returns "13:00:00"
- **shortdate** (unary operator) formats a date as a date string using the short date format, as specified in the control's panel. For instance, the `shortdate(#1/1/2001 13:00#)` returns "1/1/2001"
- **shortdateF** (unary operator) formats a date as a date string using the "MM/DD/YYYY" format. For instance, the `shortdateF(#1/1/2001 13:00#)` returns "01/01/2001"
- **dateF** (unary operator) converts the date expression to a string expression in "MM/DD/YYYY HH:MM:SS" format. For instance, the `dateF(#01/01/2001 14:00:00#)` returns #01/01/2001 14:00:00#
- **longdate** (unary operator) formats a date as a date string using the long date format, as specified in the control's panel. For instance, the `longdate(#1/1/2001 13:00#)` returns "Monday, January 01, 2001"
- **year** (unary operator) retrieves the year of the date (100,...,9999). For instance, the `year(#12/31/1971 13:14:15#)` returns 1971
- **month** (unary operator) retrieves the month of the date (1, 2,...,12). For instance, the `month(#12/31/1971 13:14:15#)` returns 12.
- **day** (unary operator) retrieves the day of the date (1, 2,...,31). For instance, the `day(#12/31/1971 13:14:15#)` returns 31
- **yearday** (unary operator) retrieves the number of the day in the year, or the days since January 1st (0, 1,...,365). For instance, the `yearday(#12/31/1971 13:14:15#)` returns 365
- **weekday** (unary operator) retrieves the number of days since Sunday (0 - Sunday, 1 - Monday,..., 6 - Saturday). For instance, the `weekday(#12/31/1971 13:14:15#)` returns 5.
- **hour** (unary operator) retrieves the hour of the date (0, 1, ..., 23). For instance, the `hour(#12/31/1971 13:14:15#)` returns 13
- **min** (unary operator) retrieves the minute of the date (0, 1, ..., 59). For instance, the `min(#12/31/1971 13:14:15#)` returns 14
- **sec** (unary operator) retrieves the second of the date (0, 1, ..., 59). For instance, the `sec(#12/31/1971 13:14:15#)` returns 15
- value **format** 'flags' (binary operator) formats a date expression with specified flags. The format method formats numeric (depends on the type of the value, explained at operators for numbers) or date expressions. If not supported, the value is formatted as a number (the date format is supported by newer version only). The flags specifies the format picture string that is used to form the date. Possible values for the format picture string are defined below. For instance, the `date(value) format 'MMM d, yyyy'`

returns "Sep 2, 2023"

The following table defines the format types used to represent days:

- d, day of the month as digits without leading zeros for single-digit days (8)
- dd, day of the month as digits with leading zeros for single-digit days (08)
- ddd, abbreviated day of the week as specified by the current locale ("Mon" in English)
- dddd, day of the week as specified by the current locale ("Monday" in English)

The following table defines the format types used to represent months:

- M, month as digits without leading zeros for single-digit months (4)
- MM, month as digits with leading zeros for single-digit months (04)
- MMM, abbreviated month as specified by the current locale ("Nov" in English)
- MMMM, month as specified by the current locale ("November" for English)

The following table defines the format types used to represent years:

- y, year represented only by the last digit (3)
- yy, year represented only by the last two digits. A leading zero is added for single-digit years (03)
- yyy, year represented by a full four or five digits, depending on the calendar used. Thai Buddhist and Korean calendars have five-digit years. The "yyyy" pattern shows five digits for these two calendars, and four digits for all other supported calendars. Calendars that have single-digit or two-digit years, such as for the Japanese Emperor era, are represented differently. A single-digit year is represented with a leading zero, for example, "03". A two-digit year is represented with two digits, for example, "13". No additional leading zeros are displayed.
- yyyy, behaves identically to "yyyy"

The following table defines the format types used to represent era:

- g, period/era string formatted as specified by the CAL_SERASTRING value (ignored if there is no associated era or period string)
- gg, period/era string formatted as specified by the CAL_SERASTRING value (ignored if there is no associated era or period string)

The following table defines the format types used to represent hours:

- h, hours with no leading zero for single-digit hours; 12-hour clock
- hh, hours with leading zero for single-digit hours; 12-hour clock
- H, hours with no leading zero for single-digit hours; 24-hour clock

- HH, hours with leading zero for single-digit hours; 24-hour clock

The following table defines the format types used to represent minutes:

- m, minutes with no leading zero for single-digit minutes
- mm, minutes with leading zero for single-digit minutes

The following table defines the format types used to represent seconds:

- s, seconds with no leading zero for single-digit seconds
- ss, seconds with leading zero for single-digit seconds

The following table defines the format types used to represent time markers:

- t, one character time marker string, such as A or P
- tt, multi-character time marker string, such as AM or PM

The expression supports also **immediate if** (similar with iif in visual basic, or ? : in C++) ie `cond ? value_true : value_false`, which means that once that cond is true the value_true is used, else the value_false is used. Also, it supports variables, up to 10 from 0 to 9. For instance, `0:="Abc"` means that in the variable 0 is "Abc", and `=:0` means retrieves the value of the variable 0. For instance, the `len(%0) ? (0:=(%1+%2) ? currency(=:0) else ``) : ``` gets the sum between second and third column in currency format if it is not zero, and only if the first column is not empty. As you can see you can use the variables to avoid computing several times the same thing (in this case the sum %1 and %2).