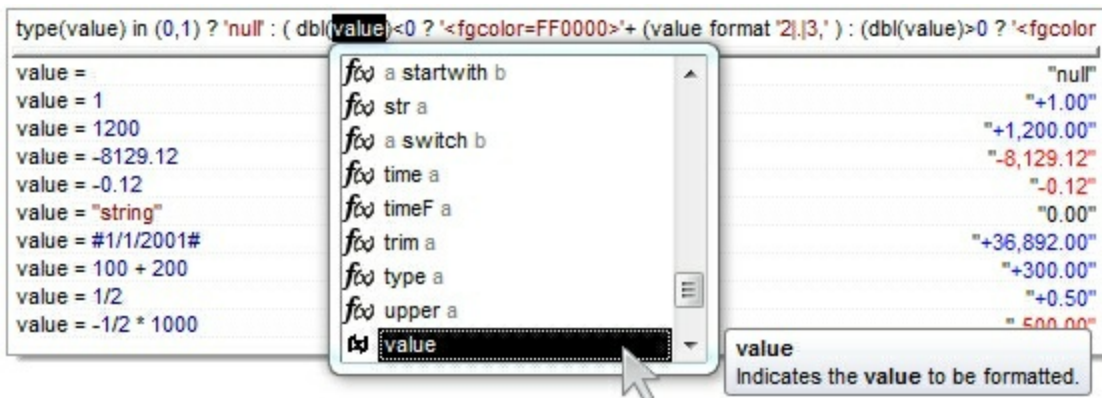


Expression

The Exontrol's eXpression component is a syntax-editor that helps you to define, view, edit and evaluate expressions. Most of our UI components support formatting based on arithmetic expressions. For instance, the Column.FormatColumn property specifies the format to display the column's content. In other words, if the Column.FormatColumn property is "currency(value)" the column displays values as currency, like \$1,234.00, instead 1234. Having 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.

Features include:

- Highlighting support for keywords, functions, numbers, strings, dates, result
- Custom Operators support
- Different colors, while the expression is not syntactically correct
- Code Completion Support
- Context Sensitive support
- Find, Replace, Incremental Search support
- OLE Drag and Drop support
- Overstrike/Overtyping or Insert mode support



Ž Expression 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 AlignmentEnum

The AlignmentEnum type specifies the alignment of the object.

Name	Value	Description
LeftAlignment	0	The source is left aligned.
CenterAlignment	1	The source is centered.
RightAlignment	2	The source is right aligned.

constants AppearanceEnum

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

Name	Value	Description
exNone	0	The control has no border.
exSingle	1	The control has single border.
exSunken	2	The border has sunken border.
exRaised	3	The border has raised border.

constants BackgroundPartEnum

The BackgroundPartEnum type indicates parts in the control. Use the [Background](#) property to specify a background color or a visual appearance for specific parts in the control. A Color expression that indicates the background color for a specified part. The last 7 bits in the high significant byte of the color to indicates the identifier of the skin being used. Use the [Add](#) method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the background's part.

If you refer a part of the scroll bar please notice the following:

- All BackgroundPartEnum expressions that starts with **exVS** changes a part in a vertical scroll bar
- All BackgroundPartEnum expressions that starts with **exHS** changes a part in the horizontal scroll bar
- Any BackgroundPartEnum expression that ends with **P** (and starts with exVS or exHS) specifies a part of the scrollbar when it is pressed.
- Any BackgroundPartEnum expression that ends with **D** (and starts with exVS or exHS) specifies a part of the scrollbar when it is disabled.
- Any BackgroundPartEnum expression that ends with **H** (and starts with exVS or exHS) specifies a part of the scrollbar when the cursor hovers it.
- Any BackgroundPartEnum expression that ends with no **H**, **P** or **D** (and starts with exVS or exHS) specifies a part of the scrollbar on normal state

Name	Value	Description
exHSplitterApp	0	Specifies the visual appearance for the horizontal split bar.
exVSplitterApp	1	Specifies the visual appearance for the vertical split bar.
exISplitterApp	2	Specifies the visual appearance for the intersection between splitters.
exSizeGrip	3	Specifies the visual appearance for control's size grip.
exToolTipAppearance	64	Indicates the visual appearance of the borders of the tooltips. Use the ToolTipPopDelay property specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. Use the ToolTipWidth property to specify the width of the tooltip window. The

[ToolTipDelay](#) property specifies the time in ms that passes before the ToolTip appears.

exToolTipBackColor	65	Specifies the tooltip's background color.
exToolTipForeColor	66	Specifies the tooltip's foreground color.
exVSUp	256	The up button in normal state.
exVSUpP	257	The up button when it is pressed.
exVSUpD	258	The up button when it is disabled.
exVSUpH	259	The up button when the cursor hovers it.
exVSThumb	260	The thumb part (exThumbPart) in normal state.
exVSThumbP	261	The thumb part (exThumbPart) when it is pressed.
exVSThumbD	262	The thumb part (exThumbPart) when it is disabled.
exVSThumbH	263	The thumb part (exThumbPart) when cursor hovers it.
exVSDown	264	The down button in normal state.
exVSDownP	265	The down button when it is pressed.
exVSDownD	266	The down button when it is disabled.
exVSDownH	267	The down button when the cursor hovers it.
exVSLower	268	The lower part (exLowerBackPart) in normal state.
exVSLowerP	269	The lower part (exLowerBackPart) when it is pressed.
exVSLowerD	270	The lower part (exLowerBackPart) when it is disabled.
exVSLowerH	271	The lower part (exLowerBackPart) when the cursor hovers it.
exVSUpper	272	The upper part (exUpperBackPart) in normal state.
exVSUpperP	273	The upper part (exUpperBackPart) when it is pressed.
exVSUpperD	274	The upper part (exUpperBackPart) when it is disabled.
exVSUpperH	275	The upper part (exUpperBackPart) when the cursor hovers it.
exVSBack	276	The background part (exLowerBackPart and

		exUpperBackPart) in normal state.
exVSBackP	277	The background part (exLowerBackPart and exUpperBackPart) when it is pressed.
exVSBackD	278	The background part (exLowerBackPart and exUpperBackPart) when it is disabled.
exVSBackH	279	The background part (exLowerBackPart and exUpperBackPart) when the cursor hovers it.
exHSLeft	384	The left button in normal state.
exHSLeftP	385	The left button when it is pressed.
exHSLeftD	386	The left button when it is disabled.
exHSLeftH	387	The left button when the cursor hovers it.
exHSThumb	388	The thumb part (exThumbPart) in normal state.
exHSThumbP	389	The thumb part (exThumbPart) when it is pressed.
exHSThumbD	390	The thumb part (exThumbPart) when it is disabled.
exHSThumbH	391	The thumb part (exThumbPart) when the cursor hovers it.
exHSRight	392	The right button in normal state.
exHSRightP	393	The right button when it is pressed.
exHSRightD	394	The right button when it is disabled.
exHSRightH	395	The right button when the cursor hovers it.
exHSLower	396	The lower part (exLowerBackPart) in normal state.
exHSLowerP	397	The lower part (exLowerBackPart) when it is pressed.
exHSLowerD	398	The lower part (exLowerBackPart) when it is disabled.
exHSLowerH	399	The lower part (exLowerBackPart) when the cursor hovers it.
exHSUpper	400	The upper part (exUpperBackPart) in normal state.
exHSUpperP	401	The upper part (exUpperBackPart) when it is pressed.
exHSUpperD	402	The upper part (exUpperBackPart) when it is disabled.
exHSUpperH	403	The upper part (exUpperBackPart) when the cursor hovers it.

exHSBack	404	The background part (exLowerBackPart and exUpperBackPart) in normal state.
exHSBackP	405	The background part (exLowerBackPart and exUpperBackPart) when it is pressed.
exHSBackD	406	The background part (exLowerBackPart and exUpperBackPart) when it is disabled.
exHSBackH	407	The background part (exLowerBackPart and exUpperBackPart) when the cursor hovers it.
exSBtn	324	All button parts (L1-L5, LButton, exThumbPart, RButton, R1-R6), in normal state.
exSBtnP	325	All button parts (L1-L5, LButton, exThumbPart, RButton, R1-R6), when it is pressed.
exSBtnD	326	All button parts (L1-L5, LButton, exThumbPart, RButton, R1-R6), when it is disabled.
exSBtnH	327	All button parts (L1-L5, LButton, exThumbPart, RButton, R1-R6), when the cursor hovers it .
exVSTThumbExt	503	The thumb-extension part in normal state.
exVSTThumbExtP	504	The thumb-extension part when it is pressed.
exVSTThumbExtD	505	The thumb-extension part when it is disabled.
exVSTThumbExtH	506	The thumb-extension when the cursor hovers it.
exHSTThumbExt	507	The thumb-extension in normal state.
exHSTThumbExtP	508	The thumb-extension when it is pressed.
exHSTThumbExtD	509	The thumb-extension when it is disabled.
exHSTThumbExtH	510	The thumb-extension when the cursor hovers it.

constants ClientAreaEnum

The ClientAreaEnum type specifies the area inside the control. Use the [Cursor](#) property to change the mouse pointer when cursor hovers the control.

Name	Value	Description
exEditArea	0	exEditArea. The cursor is on the control's edit area.
exIncrementalSearchArea	3	The cursor is on the control's incremental search bar.
exSelectedText	4	The cursor hovers the selected text and the control supports drag and drop operations.

constants DialogEnum

The DialogEnum type identifies the Find or Replace dialog of the control. Use the [Caption](#) property to internationalize your dialogs.

Name	Value	Description
exFindDialog	0	Identifies the control's Find dialog.
exReplaceDialog	1	Identifies the control's Replace dialog.
exContextMenu	2	Identifies the control's context menu.
exIncrementalSearchField	3	Identifies the incremental search field.

constants exClipboardFormatEnum

Defines the clipboard format constants. Use [GetFormat](#) property to check whether the clipboard data is of given type

Name	Value	Description
exCFText	1	Null-terminated, plain ANSI text in a global memory bloc
exCFBitmap	2	A bitmap compatible with Windows 2.X
exCFMetafile	3	A Windows metafile with some additional information about how the metafile should be displayed
exCFDIB	8	A global memory block containing a Windows device-independent bitmap (DIB)
exCFPalette	9	A color-palette handle
exCFEMetafile	14	A Windows enhanced metafile
exCFFiles	15	A collection of files. Use Files property to get the collection of files
exCFRTF	-16639A	RTF document

constants exOLEDragOverEnum

State transition constants for the OLEDragOver event.

Name	Value	Description
exOLEDragEnter	0	Source component is being dragged within the range of a target.
exOLEDragLeave	1	Source component is being dragged out of the range of a target.
exOLEDragOver	2	Source component has moved from one position in the target to another.

constants exOLEDDropEffectEnum

Drop effect constants for OLE drag and drop events.

Name	Value	Description
exOLEDDropEffectNone	0	Drop target cannot accept the data, or the drop operation was cancelled
exOLEDDropEffectCopy	1	Drop results in a copy of data from the source to the target. The original data is unaltered by the drag operation.
exOLEDDropEffectMove	2	Drop results in data being moved from drag source to drop source. The drag source should remove the data from itself after the move.
exOLEDDropEffectScroll	-2147483648	Not implemented.

constants exOLEDropModeEnum

Constants for the OLEDropMode property, that defines how the control accepts OLE drag and drop operations. Use the [OLEDropMode](#) property to set how the component handles drop operations.

Name	Value	Description
exOLEDropNone	0	The control is not used OLE drag and drop functionality
exOLEDropManual	1	The control triggers the OLE drop events, allowing the programmer to handle the OLE drop operation in code
exOLEDropAutomatic	-1	The control triggers the OLE drop events, but automatically moves or copy the selected text to the new position.

constants FieldDialogEnum

The FieldDialogEnum type describes a field in the Find or Replace dialog. Use the [Caption](#) property to internationalize the Find and Replace dialogs.

Name	Value	Description
exCaption	0	Specifies the caption of the dialog. By default, it is "Find" or "Replace".
exFieldFindWhat	202	Identifies the 'Find what' field. By default, it is "Fi&nd what:".
exFieldWordOnly	104	Identifies the 'Match whole word only' field. By default, it is "Match &whole word only".
exFieldMatchCase	105	Identifies the 'Match case' field. By default, it is "Match &case".
exFieldDirection	203	Identifies the 'Direction' field. By default, it is "Direction".
exFieldUp	113	Identifies the 'Up' field. By default, it is "&Up".
exFieldDown	114	Identifies the 'Down' field. By default, it is "&Down".
exFieldFindNext	103	Identifies the 'Find Next' field. By default, it is "&Find Next".
exFieldMarkAll	21199	Identifies the 'Mark All' field. By default, it is "&Mark All".
exFieldCancel	2	Identifies the 'Cancel' field. By default, it is "Cancel".
exFieldReplaceWith	204	Identifies the 'Replace with' field. By default, it is "Re&place with:".
exFieldReplaceIn	205	Identifies the 'Replace in' field. By default, it is "Replace in".
exFieldSelection	113	Identifies the 'Selection' field. By default, it is "Selection".
exFieldWholeFile	114	Identifies the 'Whole file' field. By default, it is "Wh&ole file".
exFieldReplace	21199	Identifies the 'Replace' field. By default, it is "&Replace".
exFieldReplaceAll	21200	Identifies the 'Replace All' field. By default, it is "Replace &All".
exErrorTitle	32000	Specifies the title of the message if an error occurs.

exErrorFindNext	32001	Specifies the message when 'Find Next' operation fails.
exReplaceDone	32002	Specifies the message that's displayed when replacing is done.
exContextUndo	16384	Specifies the Undo caption in the control's context menu.
exContextRedo	16385	Specifies the Redo caption in the control's context menu.
exContextCut	16387	Specifies the Cut caption in the control's context menu.
exContextCopy	16388	Specifies the Copy caption in the control's context menu.
exContextPaste	16389	Specifies the Paste caption in the control's context menu.
exContextDelete	16390	Specifies the Delete caption in the control's context menu.
exContextSelectAll	16392	Specifies the Select All caption in the control's context menu.

constants FindOptionEnum

Specifies the options for the [Find](#) method.

Name	Value	Description
exSearchDown	0	Searches the string down to the cursor position
exSearchUp	1	Searches the string up to the cursor position
exMatchWholeWordOnly	2	The string being searched needs to match the exactly word
exMatchCase	4	The searching is case sensitive
exReplaceFile	8	Finds and replaces the string within the file.

constants KeywordEnum

The KeywordEnum type specifies the keywords the control's expression supports. The [Description](#) property specifies the description for giving keyword. The KeywordEnum type supports the following values:

Name	Value	Description
exKeywordValue	0	Indicates the value keyword.
exKeywordOr	1	Indicates the or keyword.
exKeywordAnd	2	Indicates the and keyword.
exKeywordNot	3	Indicates the not keyword.
exKeywordStartWith	4	Indicates the startwith keyword.
exKeywordEndWith	5	Indicates the endwith keyword.
exKeywordContains	6	Indicates the contains keyword.
exKeywordCount	7	Indicates the count keyword.
exKeywordLeft	8	Indicates the left keyword.
exKeywordRight	9	Indicates the right keyword.
exKeywordMid	10	Indicates the mid keyword.
exKeywordFormat	11	Indicates the format keyword.
exKeywordIn	12	Indicates the in keyword.
exKeywordSwitch	13	Indicates the switch keyword.
exKeywordArray	14	Indicates the array keyword.
exKeywordSplit	15	Indicates the split keyword.
exKeywordCase	16	Indicates the case keyword.
exKeywordReplace	17	Indicates the replace keyword.
exKeywordWith	18	Indicates the with keyword.
exKeywordLen	19	Indicates the len keyword.
exKeywordType	20	Indicates the type keyword.
exKeywordLower	21	Indicates the lower keyword.
exKeywordUpper	22	Indicates the upper keyword.
exKeywordProper	23	Indicates the proper keyword.
exKeywordLTrim	24	Indicates the ltrim keyword.
exKeywordRTrim	25	Indicates the rtrim keyword.

exKeywordTrim	26	Indicates the trim keyword.
exKeywordInt	27	Indicates the int keyword.
exKeywordRound	28	Indicates the round keyword.
exKeywordFloor	29	Indicates the floor keyword.
exKeywordAbs	30	Indicates the abs keyword.
exKeywordStr	31	Indicates the str keyword.
exKeywordCurrency	32	Indicates the currency keyword.
exKeywordTime	33	Indicates the time keyword.
exKeywordTimeF	34	Indicates the timeF keyword.
exKeywordShortDate	35	Indicates the shortdate keyword.
exKeywordShortDateF	36	Indicates the shortdateF keyword.
exKeywordLongDate	37	Indicates the longdate keyword.
exKeywordDbl	38	Indicates the dbl keyword.
exKeywordDate	39	Indicates the date keyword.
exKeywordDateS	40	Indicates the dateS keyword.
exKeywordDateF	41	Indicates the dateF keyword.
exKeywordYear	42	Indicates the year keyword.
exKeywordMonth	43	Indicates the month keyword.
exKeywordDay	44	Indicates the day keyword.
exKeywordYearDay	45	Indicates the yearday keyword.
exKeywordWeekday	46	Indicates the weekday keyword.
exKeywordHour	47	Indicates the hour keyword.
exKeywordMin	48	Indicates the min keyword.
exKeywordSec	49	Indicates the sec keyword.
exKeywordStore	50	Indicates the := keyword.
exKeywordRestore	51	Indicates the =: keyword.
exKeywordIf	52	Indicates the ? keyword.
exKeywordMod	53	Indicates the mod keyword.
exKeywordLFind	54	Indicates the lfind keyword.
exKeywordRFind	55	Indicates the rfind keyword.
exKeywordReverse	56	Indicates the reverse keyword.

exKeywordDPI	57	Indicates the dpi keyword.
exKeywordDPIX	58	Indicates the dpix keyword.
exKeywordDPIY	59	Indicates the dpiy keyword.
exKeywordMINimum	60	Indicates the MIN keyword.
exKeywordMAXimum	61	Indicates the MAX keyword.
exKeywordSin	62	Indicates the sin keyword.
exKeywordCos	63	Indicates the cos keyword.
exKeywordASin	64	Indicates the asin keyword.
exKeywordACos	65	Indicates the acos keyword.
exKeywordSqrt	66	Indicates the sqrt keyword.
exKeywordLike	67	Indicates the like keyword.
exKeywordHex	68	Indicates the hex keyword.
exKeywordBitAnd	69	Indicates the bitand keyword.
exKeywordBitOr	70	Indicates the bitor keyword.
exKeywordBitXor	71	Indicates the bitxor keyword.
exKeywordBitShift	72	Indicates the bitshift keyword.
exKeywordBitNot	73	Indicates the bitxor keyword.
exKeywordLPad	74	Indicates the lpad keyword.
exKeywordRPad	75	Indicates the rpad keyword.
exKeywordConcat	76	Indicates the concat keyword.
exKeywordBIAS	77	Indicates the bias keyword.

constants OperatorTypeEnum

The OperatorTypeEnum type specifies the type of custom operators the you can add to expression. The [AddCustomOperator](#) method adds a custom operator. The OperatorTypeEnum type supports the following values:

Name	Value	Description
exKeywordOperator	0	Indicates keyword operator (operator with no parameters). For instance, the value is a keyword, and accept no parameters such as "value + 100".
exUnaryOperator	1	Indicates an unary operator (operator with one parameter). For instance, the len function is an unary-operator, and accept only a single parameter, such as "len(`string`)".
exBinaryOperator	2	Indicates a binary operator. For instance, the like function is a binary-operator, and can be used as "`adv` like `avans*`"

constants PictureBoxDisplayEnum

Specifies how a picture object is displayed.

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

constants ScrollBarEnum

The ScrollBarEnum type specifies the vertical or horizontal scroll bar in the control. Use the [ScrollBars](#) property to specify whether the vertical or horizontal scroll bar is visible or hidden. Use the [ScrollPartVisible](#) property to specify the visible parts in the control's scroll bars.

Name	Value	Description
exVScroll	0	Indicates the vertical scroll bar.
exHScroll	1	Indicates the horizontal scroll bar.

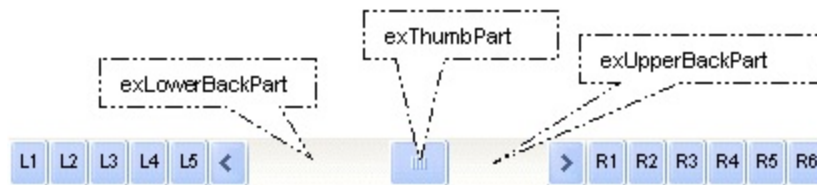
constants ScrollBarEnum

Specifies what scrollbars the control supports.

Name	Value	Description
exNoScroll	0	No scroll bars are shown
exHorizontal	1	Only horizontal scroll bars are shown.
exVertical	2	Only vertical scroll bars are shown.
exBoth	3	Both horizontal and vertical scroll bars are shown.

constants ScrollPartEnum

The ScrollPartEnum type defines the parts in the control's scrollbar. Use the [ScrollPartVisible](#) property to specify the visible parts in the control's scroll bar. Use the [ScrollPartCaption](#) property to specify the caption being displayed in any part of the control's scrollbar. The control fires the [ScrollButtonClick](#) event when the user clicks any button in the control's scrollbar.



Name	Value	Description
exExtentThumbPart	65536	The thumb-extension part.
exLeftB1Part	32768	(L1) The first additional button, in the left or top area. By default, this button is hidden.
exLeftB2Part	16384	(L2) The second additional button, in the left or top area. By default, this button is hidden.
exLeftB3Part	8192	(L3) The third additional button, in the left or top area. By default, this button is hidden.
exLeftB4Part	4096	(L4) The fourth additional button, in the left or top area. By default, this button is hidden.
exLeftB5Part	2048	(L5) The fifth additional button, in the left or top area. By default, this button is hidden.
exLeftBPart	1024	(<) The left or top button. By default, this button is visible.
exLowerBackPart	512	The area between the left/top button and the thumb. By default, this part is visible.
exThumbPart	256	The thumb part or the scroll box region. By default, the thumb is visible.
exUpperBackPart	128	The area between the thumb and the right/bottom button. By default, this part is visible.
exBackgroundPart	640	The union between the exLowerBackPart and the exUpperBackPart parts. By default, this part is visible.
exRightBPart	64	(>) The right or down button. By default, this button is visible.

exRightB1Part	32	(R1) The first additional button in the right or down side. By default, this button is hidden.
exRightB2Part	16	(R2) The second additional button in the right or down side. By default, this button is hidden.
exRightB3Part	8	(R3) The third additional button in the right or down side. By default, this button is hidden.
exRightB4Part	4	(R4) The forth additional button in the right or down side. By default, this button is hidden
exRightB5Part	2	(R5) The fifth additional button in the right or down side. By default, this button is hidden.
exRightB6Part	1	(R6) The sixth additional button in the right or down side. By default, this button is hidden.
exPartNone	0	No part.

constants SplitterEnum

The SplitterEnum expression defines the control's splitters. Use the [AllowSplitter](#) property to specify the visible splitters inside the expression control. A splitter is shown only if the expression control has scroll bars.

Name	Value	Description
exNoSplitter	0	The control's splitter is hidden.
exHSplitter	1	The control displays the horizontal splitter.
exVSplitter	2	The control displays the vertical splitter.

Appearance object

The component lets the user changes its visual appearance using **skins**, each one providing an additional visual experience that enhances viewing pleasure. Skins are relatively easy to build and put on any part of the control. The Appearance object holds a collection of skins. The Appearance object supports the following properties and methods:

Name	Description
Add	Adds or replaces a skin object to the control.
Clear	Removes all skins in the control.
Remove	Removes a specific skin from the control.

method Appearance.Add (ID as Long, Skin as Variant)

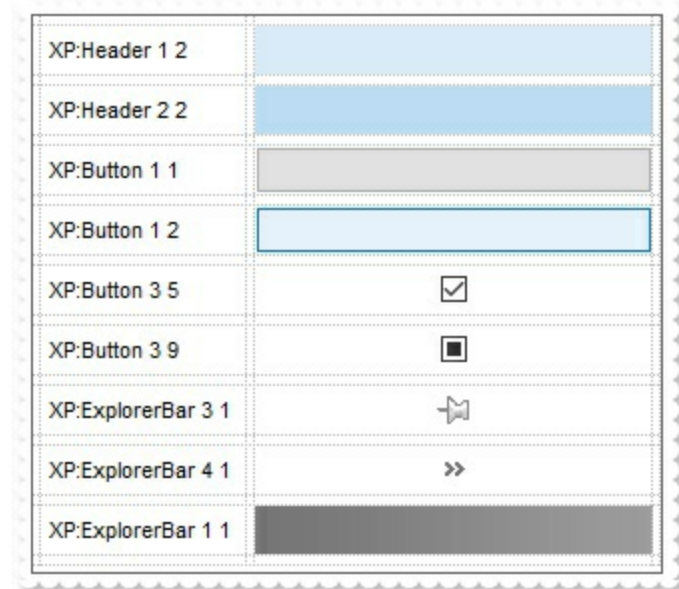
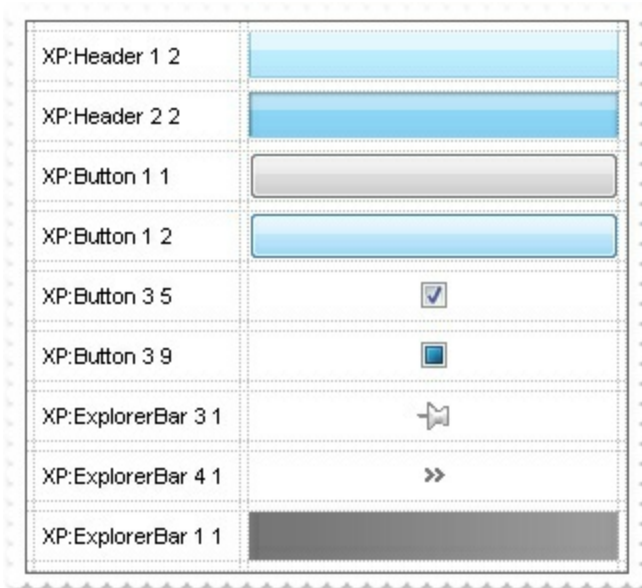
Adds or replaces a skin object to the control.

Type	Description
ID as Long	<p>A Long expression that indicates the index of the skin being added or replaced. The value must be between 1 and 126, so Appearance collection should holds no more than 126 elements.</p> <hr/> <p>The Skin parameter of the Add method can a STRING as explained bellow, a BYTE[] / safe arrays of VT_I1 or VT_UI1 expression that indicates the content of the EBN file. You can use the BYTE[] / safe arrays of VT_I1 or VT_UI1 option when using the EBN file directly in the resources of the project. For instance, the VB6 provides the LoadResData to get the safe array o bytes for specified resource, while in VB/.NET or C# the internal class Resources provides definitions for all files being inserted. (ResourceManager.GetObject("ebn", resourceCulture))</p> <p>If the Skin parameter points to a string expression, it can be one of the following:</p> <ul style="list-style-type: none">• A path to the skin file (*.EBN). The ExButton component or ExEBN tool can be used to create, view or edit EBN files. For instance, "C:\Program Files\Exontrol\ExButton\Sample\EBN\MSOffice-Ribbon\msor_frameh.ebn"• A BASE64 encoded string that holds the skin file (*.EBN). Use the ExImages tool to build BASE 64 encoded strings of the skin file (*.EBN). The BASE64 encoded string starts with "gBFLBCJw..."• An Windows XP theme part, if the Skin parameter starts with "XP:". Use this option, to display any UI element of the Current Windows XP Theme, on any part of the control. In this case, the syntax of the Skin parameter is: "XP:ClassName Part State" where the ClassName defines the window/control class name in the Windows XP Theme, the Part indicates a long expression that defines the part, and the State indicates the state of the part to be shown. All known values for window/class, part and start are defined at

the end of this document. For instance the "XP:Header 1 2" indicates the part 1 of the Header class in the state 2, in the current Windows XP theme.

The following screen shots show a few Windows XP Theme Elements, running on Windows Vista and Windows 10, using the XP options:

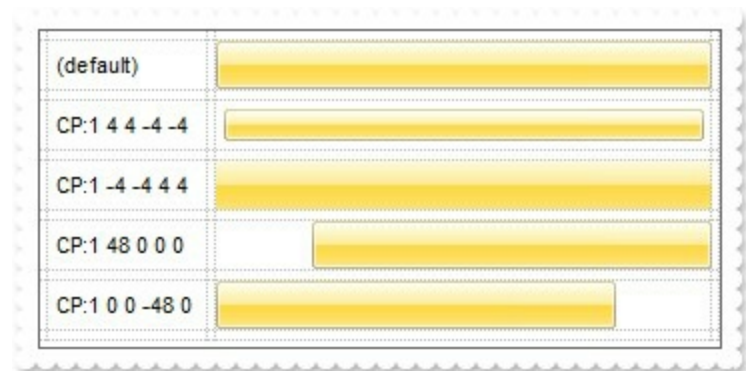
Skin as Variant



- A copy of another skin with different coordinates (position, size), if the Skin parameter starts with "**CP:**". Use this option, to display the EBN, using different coordinates (position, size). By default, the EBN skin object is rendered on the part's client area. Using this option, you can display the same EBN, on a different position / size. In this case, the syntax of the Skin parameter is: "**CP:ID Left Top Right Bottom**"

where the ID is the identifier of the EBN to be used (it is a number that specifies the ID parameter of the Add method), Left, Top, Right and Bottom parameters/numbers specifies the relative position to the part's client area, where the EBN should be rendered. The Left, Top, Right and Bottom parameters are numbers (negative, zero or positive values, with no decimal), that can be followed by the D character which indicates the value according to the current DPI settings. For instance, "CP:1 -2 -2 2 2", uses the EBN with the identifier 1, and displays it on a 2-pixels wider rectangle no matter of the DPI settings, while "CP:1 -2D -2D 2D 2D" displays it on a 2-pixels wider rectangle if DPI settings is 100%, and on on a 3-pixels wider rectangle if DPI settings is 150%.

The following screen shot shows the same EBN being displayed, using different CP options:



Return

Boolean

Description

A Boolean expression that indicates whether the new skin was added or replaced.

Use the Add method to add or replace skins to the control. The skin method, in it's simplest form, uses a single graphic file (*.ebn) assigned to a part of the control. By using a collection of objects laid over the graphic, it is possible to define which sections of the graphic will be used as borders, corners and other possible elements, fixing them to their proper position regardless of the size of the part. Use the [Remove](#) method to remove a specific skin from the control. Use the [Clear](#) method to remove all skins in the control. Use the [Refresh](#) method to refresh the control.

The identifier you choose for the skin is very important to be used in the background properties like explained bellow. Shortly, the color properties uses 4 bytes (DWORD, double WORD, and so on) to hold a RGB value. More than that, the first byte (most significant byte in the color) is used only to specify system color. if the first bit in the byte is

1, the rest of bits indicates the index of the system color being used. So, we use the last 7 bits in the high significant byte of the color to indicates the identifier of the skin being used. So, since the 7 bits can cover 127 values, excluding 0, we have 126 possibilities to store an identifier in that byte. This way, a DWORD expression indicates the background color stored in RRGGBB format and the index of the skin (ID parameter) in the last 7 bits in the high significant byte of the color.

method Appearance.Clear ()

Removes all skins in the control.

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

Use the Clear method to clear all skins from the control. Use the [Remove](#) method to remove a specific skin. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the background's part.

method Appearance.Remove (ID as Long)

Removes a specific skin from the control.

Type	Description
ID as Long	A Long expression that indicates the index of the skin being removed.

Use the Remove method to remove a specific skin. The identifier of the skin being removed should be the same as when the skin was added using the [Add](#) method. Use the [Clear](#) method to clear all skins from the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the background's part.

ExDataObject object

Defines the object that contains OLE drag and drop information.

Name	Description
Clear	Deletes the contents of the ExDataObject object.
Files	Returns an ExDataObjectFiles collection, which in turn contains a list of all filenames used by an ExDataObject object.
GetData	Returns data from an ExDataObject object in the form of a variant.
GetFormat	Returns a value indicating whether an item in the ExDataObject object matches a specified format.
SetData	Inserts data into an ExDataObject object using the specified data format.

method `ExDataObject.Clear ()`

Deletes the contents of the `DataObject` object.

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

The `Clear` method can be called only for drag sources. The [OleDragDrop](#) event notifies your application that the user drags some data on the control.

property `ExDataObject.Files` as `ExDataObjectFiles`

Returns a `DataObjectFiles` collection, which in turn contains a list of all filenames used by a `DataObject` object.

Type	Description
ExDataObjectFiles	An <code>ExDataObjectFiles</code> object that contains a list of filenames used in OLE drag and drop operations

The `Files` property is valid only if the format of the clipboard data is `exCFFiles`.

method `ExDataObject.GetData` (Format as Integer)

Returns data from a `DataObject` object in the form of a variant.

Type	Description
Format as Integer	An exClipboardFormatEnum expression that defines the data's format
Return	Description
Variant	A Variant value that contains the <code>ExDataObject</code> 's data in the given format

Use `GetData` property to retrieve the clipboard's data that has been dragged to the control. It's possible for the `GetData` and [SetData](#) methods to use data formats other than [exClipboardFormatEnum](#) , including user-defined formats registered with Windows via the `RegisterClipboardFormat()` API function. The `GetData` method always returns data in a byte array when it is in a format that it is not recognized. Use the [Files](#) property to retrieves the filenames if the format of data is `exCFFiles`

method `ExDataObject.Format` (Format as Integer)

Returns a value indicating whether the `ExDataObject`'s data is of the specified format.

Type	Description
Format as Integer	A constant or value that specifies a clipboard data format like described in exClipboardFormatEnum enum.
Return	Description
Boolean	A boolean value that indicates whether the <code>ExDataObject</code> 's data is of specified format.

Use the `GetFormat` property to verify if the `ExDataObject`'s data is of a specified clipboard format. The `GetFormat` property retrieves `True`, if the `ExDataObject`'s data format matches the given data format.

The following VB sample inserts the names of the files being dragged at the cursor position:

```
Private Sub Expression1_OLEDragDrop(ByVal Data As EXPRESSIONLibCtl.IExDataObject,
Effect As Long, ByVal Button As Integer, ByVal Shift As Integer, ByVal X As Single, ByVal Y As
Single)
    If (Data.GetFormat(exCFFiles)) Then
        With Data.Files
            Dim i As Long, s As String
            s = ""
            For i = 0 To .Count - 1
                s = s + .Item(i) + vbCrLf
            Next
            Expression1.SelText = s
        End With
    End If
End Sub
```

The following C++ sample inserts the names of the files being dragged at the cursor position:

```
void OnOLEDragDropExpression1(LPDISPATCH Data, long FAR* Effect, short Button, short
Shift, long X, long Y)
{
    if ( EXPRESSIONLib::IExDataObjectPtr spData( Data ) )
```

```

if ( spData->GetFormat( EXPRESSIONLib::exCFFiles ) )
{
    EXPRESSIONLib::IExDataObjectFilesPtr spFiles = spData->Files;
    if ( spFiles != NULL )
    {
        CString s;
        for ( long i = 0; i < spFiles->Count; i++ )
            s += spFiles->Item[i];
        m_edit.SetSelText( s );
    }
}
}

```

the sample needs to include definitions for IExDataObject and IExDataObjectFiles objects, by using the #import <expression.dll> statement. The #import <expression.dll> includes the EXPRESSIONLib namespace where are defined all objects in the control's type library.

The following VB.NET sample inserts the names of the files being dragged at the cursor position:

```

Private Sub AxExpression1_OLEDragDrop(ByVal sender As Object, ByVal e As
AxEXPRESSIONLib._IExpressionEvents_OLEDragDropEvent) Handles
AxExpression1.OLEDragDrop
    If (e.data.GetFormat(EXPRESSIONLib.exClipboardFormatEnum.exCFFiles)) Then
        With e.data.Files
            Dim i As Long, s As String = ""
            For i = 0 To .Count - 1
                s = s + .Item(i) + vbCrLf
            Next
            AxExpression1.SetText = s
        End With
    End If
End Sub

```

The following C# sample inserts the names of the files being dragged at the cursor position:

```

private void axExpression1_OLEDragDrop(object sender,
AxEXPRESSIONLib._IExpressionEvents_OLEDragDropEvent e)
{

```

```

if
(e.data.GetFormat(Convert.ToInt16(EXPRESSIONLib.exClipboardFormatEnum.exCFFiles)))
{
    EXPRESSIONLib.IExDataObjectFiles files = e.data.Files;
    if (files != null)
    {
        string s = "";
        for (int i = 0; i < files.Count; i++)
            s += files[i] + "\r\n";
        axExpression1.SelText = s;
    }
}
}
}

```

The following VFP sample inserts the names of the files being dragged at the cursor position:

```

*** ActiveX Control Event ***
LPARAMETERS data, effect, button, shift, x, y

if ( data.GetFormat( 15 ) ) then
    local i, s
    s = ""
    with data.Files
        for i = 0 to .Count - 1
            s = s + .Item(i) + chr(13) + chr(10)
        next
    thisform.Expression1.SelText = s
    endwith
endif

```

method `ExDataObject.SetData ([Value as Variant], [Format as Variant])`

Inserts data into a `ExDataObject` object using the specified data format.

Type	Description
Value as Variant	A data that is going to be inserted to <code>ExDataObject</code> object.
Format as Variant	A constant or value that specifies the data format, as described in exClipboardFormatEnum enum

Use `SetData` property to insert data for OLE drag and drop operations. Use the [Files](#) property is to add files to the clipboard data, when `Format` parameter is `exCFFiles`.

The following VB sample puts the selected text to the clipboard when the drag and drop operation starts (The `OLEDropMode` property is `exOLEDropManual`):

```
Private Sub Expression1_OLEStartDrag(ByVal Data As EXPRESSIONLibCtl.IExDataObject,
AllowedEffects As Long)
    Dim s As String
    s = Expression1.SelText
    If (Len(s) > 0) Then
        AllowedEffects = 1
        Data.SetData s, 1
    End If
End Sub
```

The following C++ sample puts the selected text to the clipboard when the drag and drop operation starts:

```
void OnOLEStartDragExpression1(LPDISPATCH Data, long FAR* AllowedEffects)
{
    CString s( m_edit.GetSelText() );
    if ( s.GetLength() > 0 )
    {
        *AllowedEffects = 1; /*exOLEDropEffectCopy*/
        if ( EXPRESSIONLib::IExDataObjectPtr spData( Data ) )
            spData->SetData( COleVariant( s ), COleVariant( long(EXPRESSIONLib::exCFTText) ) );
    }
}
```

The C++ sample uses the `#import <expression.dll>` statement to include definitions for the `IExDataObject` and `IExDataObjectFiles` objects.

The following VB.NET sample puts the selected text to the clipboard when the drag and drop operation starts:

```
Private Sub AxExpression1_OLEStartDrag(ByVal sender As Object, ByVal e As
AxEXPRESSIONLib._IExpressionEvents_OLEStartDragEvent) Handles
AxExpression1.OLEStartDrag
    Dim s As String = AxExpression1.SelText
    If (Len(s) > 0) Then
        e.allowedEffects = 1
        e.data.SetData(s, 1)
    End If
End Sub
```

The following C# sample puts the selected text to the clipboard when the drag and drop operation starts:

```
private void axExpression1_OLEStartDrag(object sender,
AxEXPRESSIONLib._IExpressionEvents_OLEStartDragEvent e)
{
    string s = axExpression1.SelText;
    if (s.Length > 0)
    {
        e.allowedEffects = 1;
        e.data.SetData(s, EXPRESSIONLib.exClipboardFormatEnum.exCFText );
    }
}
```

The following VFP sample puts the selected text to the clipboard when the drag and drop operation starts:

```
*** ActiveX Control Event ***
LPARAMETERS data, allowedeffects

with thisform.Expression1
    local s
    s = .SelText
```

```
if ( len(s) > 0 ) then
  allowedeffects = 1
  data.SetData( s, 1 )
endif
endwith
```

ExDataObjectFiles object

The ExDataObjectFiles contains a collection of filenames. The ExDataObjectFiles object is used in OLE Drag and drop events. In order to get the list of files used in drag and drop operations you have to use the [Files](#) property.

Name	Description
Add	Adds a filename to the Files collection
Clear	Removes all file names in the collection.
Count	Returns the number of file names in the collection.
Item	Returns an specific file name.
Remove	Removes an specific file name.

method `ExDataObjectFiles.Add (FileName as String)`

Adds a filename to the Files collection

Type	Description
FileName as String	A string expression that indicates a filename.

Use Add method to add your files to ExDataObject object. The [OLEStartDrag](#) event notifies the application that the user starts dragging data from the control. The [OLEDragDrop](#) event notifies your application that the user drags data to the control.

method `ExDataObjectFiles.Clear ()`

Removes all file names in the collection.

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

Use the `Clear` method to remove all filenames from the collection.

property ExDataObjectFiles.Count as Long

Returns the number of file names in the collection.

Type	Description
Long	A long value that indicates the count of elements into collection.

The Count property specifies the number of files in the Files object. In order to enable OLE drag and drop feature into control you have to check the [OLEDropMode](#) property. Use the [SelText](#) property to insert text at cursor position. The control fires the [OLEDragDrop](#) event when the user drags data to the control. Use the [Item](#) property to access a file giving its index. You can use "for each" statement to enumerate the files in the ExDataObjectFiles collection.

The following VB sample inserts the names of the files being dragged at the cursor position:

```
Private Sub Expression1_OLEDragDrop(ByVal Data As EXPRESSIONLibCtl.IExDataObject,
Effect As Long, ByVal Button As Integer, ByVal Shift As Integer, ByVal X As Single, ByVal Y As
Single)
    If (Data.GetFormat(exCFFiles)) Then
        With Data.Files
            Dim i As Long, s As String
            s = ""
            For i = 0 To .Count - 1
                s = s + .Item(i) + vbCrLf
            Next
            Expression1.SelText = s
        End With
    End If
End Sub
```

The following C++ sample inserts the names of the files being dragged at the cursor position:

```
void OnOLEDragDropExpression1(LPDISPATCH Data, long FAR* Effect, short Button, short
Shift, long X, long Y)
{
    if ( EXPRESSIONLib::IExDataObjectPtr spData( Data ) )
        if ( spData->GetFormat( EXPRESSIONLib::exCFFiles ) )
```

```

{
    EXPRESSIONLib::IExDataObjectFilesPtr spFiles = spData->Files;
    if ( spFiles != NULL )
    {
        CString s;
        for ( long i = 0; i < spFiles->Count; i++ )
            s += spFiles->Item[i];
        m_edit.SetSelText( s );
    }
}
}
}

```

the sample needs to include definitions for IExDataObject and IExDataObjectFiles objects, by using the `#import <expression.dll>` statement. The `#import <expression.dll>` includes the EXPRESSIONLib namespace where are defined all objects in the control's type library.

The following VB.NET sample inserts the names of the files being dragged at the cursor position:

```

Private Sub AxExpression1_OLEDragDrop(ByVal sender As Object, ByVal e As
AxEXPRESSIONLib._IExpressionEvents_OLEDragDropEvent) Handles
AxExpression1.OLEDragDrop
    If (e.data.GetFormat(EXPRESSIONLib.exClipboardFormatEnum.exCFFiles)) Then
        With e.data.Files
            Dim i As Long, s As String = ""
            For i = 0 To .Count - 1
                s = s + .Item(i) + vbCrLf
            Next
            AxExpression1.SelText = s
        End With
    End If
End Sub

```

The following C# sample inserts the names of the files being dragged at the cursor position:

```

private void axExpression1_OLEDragDrop(object sender,
AxEXPRESSIONLib._IExpressionEvents_OLEDragDropEvent e)
{
    if

```

```
(e.data.GetFormat(Convert.ToInt16(EXPRESSIONLib.exClipboardFormatEnum.exCFFiles)))
{
    EXPRESSIONLib.IExDataObjectFiles files = e.data.Files;
    if (files != null)
    {
        string s = "";
        for (int i = 0; i < files.Count; i++)
            s += files[i] + "\r\n";
        axExpression1.SelText = s;
    }
}
}
```

The following VFP sample inserts the names of the files being dragged at the cursor position:

```
*** ActiveX Control Event ***
LPARAMETERS data, effect, button, shift, x, y

if ( data.GetFormat( 15 ) ) then
    local i, s
    s = ""
    with data.Files
        for i = 0 to .Count - 1
            s = s + .Item(i) + chr(13) + chr(10)
        next
    thisform.Expression1.SelText = s
    endwith
endif
```

property ExDataObjectFiles.Item (Index as Long) as String

Returns a specific file name given its index.

Type	Description
Index as Long	A long expression that indicates the filename's index
String	A string value that indicates the filename

In order to enable OLE drag and drop feature into control you have to check the [OLEDropMode](#) property. Use the [SelText](#) property to insert text at cursor position. The control fires the [OLEDragDrop](#) event when the user drags data to the control. The [Count](#) property specifies the number of files being stored.

The following VB sample inserts the names of the files being dragged at the cursor position:

```
Private Sub Expression1_OLEDragDrop(ByVal Data As EXPRESSIONLibCtl.IExDataObject,
Effect As Long, ByVal Button As Integer, ByVal Shift As Integer, ByVal X As Single, ByVal Y As
Single)
    If (Data.GetFormat(exCFFiles)) Then
        With Data.Files
            Dim i As Long, s As String
            s = ""
            For i = 0 To .Count - 1
                s = s + .Item(i) + vbCrLf
            Next
            Expression1.SelText = s
        End With
    End If
End Sub
```

The following C++ sample inserts the names of the files being dragged at the cursor position:

```
void OnOLEDragDropExpression1(LPDISPATCH Data, long FAR* Effect, short Button, short
Shift, long X, long Y)
{
    if ( EXPRESSIONLib::IExDataObjectPtr spData( Data ) )
        if ( spData->GetFormat( EXPRESSIONLib::exCFFiles ) )
        {
```

```

EXPRESSONLib::IExDataObjectFilesPtr spFiles = spData->Files;
if ( spFiles != NULL )
{
    CString s;
    for ( long i = 0; i < spFiles->Count; i++ )
        s += spFiles->Item[i];
    m_edit.SetSelText( s );
}
}
}

```

the sample needs to include definitions for IExDataObject and IExDataObjectFiles objects, by using the #import <expression.dll> statement. The #import <expression.dll> includes the EXPRESSONLib namespace where are defined all objects in the control's type library.

The following VB.NET sample inserts the names of the files being dragged at the cursor position:

```

Private Sub AxExpression1_OLEDragDrop(ByVal sender As Object, ByVal e As
AxEXPRESSONLib._IExpressionEvents_OLEDragDropEvent) Handles
AxExpression1.OLEDragDrop
    If (e.data.GetFormat(EXPRESSONLib.exClipboardFormatEnum.exCFFiles)) Then
        With e.data.Files
            Dim i As Long, s As String = ""
            For i = 0 To .Count - 1
                s = s + .Item(i) + vbCrLf
            Next
            AxExpression1.SelText = s
        End With
    End If
End Sub

```

The following C# sample inserts the names of the files being dragged at the cursor position:

```

private void axExpression1_OLEDragDrop(object sender,
AxEXPRESSONLib._IExpressionEvents_OLEDragDropEvent e)
{
    if
(e.data.GetFormat(Convert.ToInt16(EXPRESSONLib.exClipboardFormatEnum.exCFFiles)))

```

```

{
    EXPRESSIONLib.IExDataObjectFiles files = e.data.Files;
    if (files != null)
    {
        string s = "";
        for (int i = 0; i < files.Count; i++)
            s += files[i] + "\r\n";
        axExpression1.SelText = s;
    }
}
}
}

```

The following VFP sample inserts the names of the files being dragged at the cursor position:

```

*** ActiveX Control Event ***
LPARAMETERS data, effect, button, shift, x, y

if ( data.GetFormat( 15 ) ) then
    local i, s
    s = ""
    with data.Files
        for i = 0 to .Count - 1
            s = s + .Item(i) + chr(13) + chr(10)
        next
        thisform.Expression1.SelText = s
    endwith
endif

```

method `ExDataObjectFiles.Remove (Index as Long)`

Removes a specific file name given its index into collection.

Type	Description
Index as Long	A long expression that indicates the index of filename into collection.

Use [Clear](#) method to remove all filenames.

Expression 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: {B33F5489-49AC-4155-98E7-9BBFC57FF019}. The object's program identifier is: "Exontrol.Expression". The /COM object module is: "Expression.dll"

The Exontrol's Expression component is a syntax-editor that helps you to define, view, edit and evaluate expressions. Most of our UI components support formatting based on arithmetic expressions. For instance, the Column.FormatColumn property specifies the format to display the column's content. In other words, if the Column.FormatColumn property is "currency(value)" the column displays values as currency, like \$1,234.00, instead 1234. The Expression objects supports the following properties and methods:

Name	Description
AddCustomOperator	Adds a custom operator.
AllowContextMenu	Specifies whether the control's default context menu is available.
AllowDefaultOperators	Specifies whether the expression supports default operators.
AllowFind	Specifies whether control displays a find dialog when user presses CTRL+F key.
AllowIncrementalSearch	Specifies whether the control allows incremental search.
AllowReplace	Specifies whether control displays a replace dialog when user presses CTRL+H key.
AllowSplitter	Specifies whether the control's splitter is visible or hidden.
AllowUndoRedo	Specifies whether the control allows undo/redo actions.
AllowValueKeyword	Specifies whether the expression supports the value keyword.
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	Retrieves or sets a value that indicates the control's background color.
Background	Returns or sets a value that indicates the background color for parts in the control.
CanRedo	Determines if the redo queue contains any actions.
CanUndo	Determines whether the last edit operation can be undone.
Caption	Specifies the caption for a field on the Find, Replace

	dialog or control's context menu.
ClearCustomOperators	Clears the custom operators.
CodeCompletion	Specifies whether the code completion feature is enabled or disabled.
ContextKey	Specifies the key combination that opens the control's context window.
ContextMenuItems	Specifies a list of items that are added to the control's context menu.
Cursor	Gets or sets the cursor that is displayed when the mouse pointer hovers the control.
Description	Specifies the description of the giving keyword.
DisplaySelection	Specifies whether the control displays the selection.
Enabled	Enables or disables the control.
Evaluate	Evaluates the current expression and returns the result.
EvaluateSelection	Specifies whether the control evaluates the selection when it is available.
EvaluationResult	Specifies the evaluation result, each line indicates a result value.
EvaluationText	Specifies the evaluation text, each line indicates a test value.
EvaluationTime	Indicates the time in ms to evaluate the expression.
EventParam	Retrieves or sets a value that indicates the current's event parameter.
ExecuteTemplate	Executes a template and returns the result.
Expression	Indicates the expression to be evaluated.
Find	Finds a string and selects the string if it is found.
FocusPane	Specifies the index of pane that has the focus.
Font	Retrieves or sets the control's font.
ForeColor	Retrieves or sets a value that indicates the control's foreground color, while the expression is valid.
ForeColorInvalid	Retrieves or sets a value that indicates the control's foreground color, while expression is invalid.
FormatABC	Formats the A,B,C values based on the giving expression and returns the result.

FormatDates	Specifies the HTML format that's applied to dates.
FormatNumbers	Specifies the HTML format that's applied to numbers.
FormatResult	Specifies the HTML format to show the result.
FormatStrings	Specifies the HTML format that's applied to strings.
HideSelection	Specifies whether the selection in the control is hidden when the control loses the focus.
hWnd	Gets the window's handle.
hWndPane	Gets the window handle of the control's pane.
Images	Sets the control's images list at runtime. The Handle should be a handle to an Image List control.
ImageSize	Retrieves or sets the size of icons the control displays..
IncrementalSearchError	Retrieves or sets a value that specifies the color to show the 'Incremental Search' field when the typing string is not found.
IndentOnTab	Specifies whether the multiple lines selection is indented when user presses the TAB key.
IsValid	Specifies whether the expression is valid (syntactically correct).
LineHeight	Specifies an expression that determines the height of the line within the editor.
Locked	Determines whether a control can be edited.
MultiLine	Specifies whether the control accepts multiple lines.
OLEDrag	Causes a component to initiate an OLE drag/drop operation.
OLEDropMode	Returns or sets how a target component handles drop operations
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
RClick	Specifies whether the control's cursor is placed at the mouse position when user right clicks the control.
Redo	Redoes the next action in the control's redo queue.
Refresh	Refreshes the control.

Replacelcon	Adds a new icon, replaces an icon or clears the control's image list.
ScrollBars	Specifies the type of scroll bars that control has.
ScrollButtonHeight	Specifies the height of the button in the vertical scrollbar.
ScrollButtonWidth	Specifies the width of the button in the horizontal scrollbar.
ScrollFont	Retrieves or sets the scrollbar's font.
ScrollHeight	Specifies the height of the horizontal scrollbar.
ScrollOrderParts	Specifies the order of the buttons in the scroll bar.
ScrollPartCaption	Specifies the caption being displayed on the specified scroll part.
ScrollPartCaptionAlignment	Specifies the alignment of the caption in the part of the scroll bar.
ScrollPartEnable	Indicates whether the specified scroll part is enabled or disabled.
ScrollPartVisible	Indicates whether the specified scroll part is visible or hidden.
ScrollThumbSize	Specifies the size of the thumb in the scrollbar.
ScrollToolTip	Specifies the tooltip being shown when the user moves the scroll box.
ScrollWidth	Specifies the width of the vertical scrollbar.
SelBackColor	Specifies the selection's background color.
SelBackColorHide	Specifies the selection's background color, when the control has no focus, and the HideSelection property is False.
SelForeColor	Specifies the selection's foreground color.
SelForeColorHide	Specifies the selection's foreground color, when the control has no focus, and the HideSelection property is False.
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.
ShowCaret	Specifies whether the control's caret is visible or hidden.

ShowImageList	Specifies whether the control's images panel dialog is visible or hidden.
ShowToolTip	Shows the specified tooltip at given position.
SplitPaneHeight	Specifies a value that indicates the height in pixels of the top pane(s) when splitting.
SplitPaneWidth	Specifies a value that indicates the width in pixels of the left pane(s) when splitting.
TabLength	Specifies the size of each tab stop, in units equal to the average character width.
Template	Specifies the control's template.
Text	Specifies the control's text.
ToolTipDelay	Specifies the time in ms that passes before the ToolTip appears.
ToolTipFont	Retrieves or sets the tooltip's font.
ToolTipOnTyping	Specifies a value that indicates whether the tooltip of the keyword is shown while typing.
ToolTipPopDelay	Specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control.
ToolTipWidth	Specifies a value that indicates the width of the tooltip window, in pixels.
Undo	Call this function to undo the last edit-control operation.
UseTabKey	Specifies whether the control uses the TAB key.
Version	Retrieves the control's version.
VisualAppearance	Retrieves the control's appearance.
WordFromPoint	Retrieves the word from the cursor.

method Expression.AddCustomOperator (Name as String, [Keyword as Variant], [ToolTip as Variant], [Type as Variant])

Adds a custom operator.

Type	Description
Name as String	A string expression that defines the name of the new operator to be added. The name should not contain any or # , " ' ` (). The name must not start with a digit character.
Keyword as Variant	A HTML expression that specifies how the operator will be displayed in the control's context menu. You can press the CTRL + Space key to display the control's drop down that lists all available operators.
ToolTip as Variant	A HTML expression that defines the operator's tooltip. For instance, the operator's tooltip may be displayed as soon as the cursor hovers the operator.
Type as Variant	A OperatorTypeEnum expression that specifies the type of the operator. If missing, the method adds a keyword (exKeywordOperator).

The AddCustomOperator method allows you to add custom operators. The newly operator is added only if the Name is not already included and valid. The operator is valid if it includes no # , " ' ` () characters, and not start with a digit character. No error occurs, if the AddCustomOperator method fails. The [ClearCustomOperators](#) method clears the custom operators. The [AllowDefaultOperators](#) property specifies whether the expression supports default operators. The [AllowValueKeyword](#) property specifies whether the expression supports the value keyword.

The following samples shows how you can define your own operators only:

VB

With Expression1

```
.AllowValueKeyword = False  
.AllowDefaultOperators = False  
.AddCustomOperator "+", "<b>+</b>", "This is a new binary-operator", 2  
.AddCustomOperator "xxx", "<b>xxx</b>", "This is a new keyword", 0  
.AddCustomOperator "yyy", "<b>yyy</b>", "This is a unary-operator", 1  
.AddCustomOperator "zzz", "<b>zzz</b>", "This is a binary-operator", 2  
.Text = "xxx + yyy(100 zzz 200)"
```

End With

VB.NET

With Expression1

```
.AllowValueKeyword = False  
.AllowDefaultOperators = False  
.AddCustomOperator("+", "<b>+</b>", "This is a new binary-operator",2)  
.AddCustomOperator("xxx", "<b>xxx</b>", "This is a new keyword",0)  
.AddCustomOperator("yyy", "<b>yyy</b>", "This is a unary-operator",1)  
.AddCustomOperator("zzz", "<b>zzz</b>", "This is a binary-operator",2)  
.Text = "xxx + yyy(100 zzz 200)"
```

End With

C++

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

```
    #import <Expression.dll>  
    using namespace EXPRESSIONLib;
```

```
*/  
EXPRESSIONLib::IExpressionPtr spExpression1 = GetDlgItem(IDC_EXPRESSION1)-  
>GetControlUnknown();  
spExpression1->PutAllowValueKeyword(VARIANT_FALSE);  
spExpression1->PutAllowDefaultOperators(VARIANT_FALSE);  
spExpression1->AddCustomOperator(L"+", "<b>+</b>", "This is a new binary-  
operator",long(2));  
spExpression1->AddCustomOperator(L"xxx", "<b>xxx</b>", "This is a new  
keyword",long(0));  
spExpression1->AddCustomOperator(L"yyy", "<b>yyy</b>", "This is a unary-  
operator",long(1));  
spExpression1->AddCustomOperator(L"zzz", "<b>zzz</b>", "This is a binary-  
operator",long(2));  
spExpression1->PutText(L"xxx + yyy(100 zzz 200)");
```

C++ Builder

```
Expression1->AllowValueKeyword = false;
Expression1->AllowDefaultOperators = false;
Expression1->AddCustomOperator(L"+",TVariant("<b>+</b>"),TVariant("This is a new
binary-operator"),TVariant(2));
Expression1->AddCustomOperator(L"xxx",TVariant("<b>xxx</b>"),TVariant("This is a
new keyword"),TVariant(0));
Expression1->AddCustomOperator(L"yyy",TVariant("<b>yyy</b>"),TVariant("This is a
unary-operator"),TVariant(1));
Expression1->AddCustomOperator(L"zzz",TVariant("<b>zzz</b>"),TVariant("This is a
binary-operator"),TVariant(2));
Expression1->Text = L"xxx + yyy(100 zzz 200)";
```

C#

```
expression1.AllowValueKeyword = false;
expression1.AllowDefaultOperators = false;
expression1.AddCustomOperator("+","<b>+</b>","This is a new binary-operator",2);
expression1.AddCustomOperator("xxx","<b>xxx</b>","This is a new keyword",0);
expression1.AddCustomOperator("yyy","<b>yyy</b>","This is a unary-operator",1);
expression1.AddCustomOperator("zzz","<b>zzz</b>","This is a binary-operator",2);
expression1.Text = "xxx + yyy(100 zzz 200)";
```

JScript/JavaScript

```
<BODY onload="Init()">
<OBJECT CLASSID="clsid:B33F5489-49AC-4155-98E7-9BBFC57FF019"
id="Expression1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    Expression1.AllowValueKeyword = false;
    Expression1.AllowDefaultOperators = false;
    Expression1.AddCustomOperator("+","<b>+</b>","This is a new binary-operator",2);
    Expression1.AddCustomOperator("xxx","<b>xxx</b>","This is a new keyword",0);
    Expression1.AddCustomOperator("yyy","<b>yyy</b>","This is a unary-operator",1);
    Expression1.AddCustomOperator("zzz","<b>zzz</b>","This is a binary-operator",2);
    Expression1.Text = "xxx + yyy(100 zzz 200)";
}
```

```
}  
</SCRIPT>  
</BODY>
```

VBScript

```
<BODY onload="Init()">  
<OBJECT CLASSID="clsid:B33F5489-49AC-4155-98E7-9BBFC57FF019"  
id="Expression1"> </OBJECT>  
  
<SCRIPT LANGUAGE="VBScript">  
Function Init()  
  With Expression1  
    .AllowValueKeyword = False  
    .AllowDefaultOperators = False  
    .AddCustomOperator "+", "<b>+</b>", "This is a new binary-operator",2  
    .AddCustomOperator "xxx", "<b>xxx</b>", "This is a new keyword",0  
    .AddCustomOperator "yyy", "<b>yyy</b>", "This is a unary-operator",1  
    .AddCustomOperator "zzz", "<b>zzz</b>", "This is a binary-operator",2  
    .Text = "xxx + yyy(100 zzz 200)"  
  End With  
End Function  
</SCRIPT>  
</BODY>
```

C# for /COM

```
axExpression1.AllowValueKeyword = false;  
axExpression1.AllowDefaultOperators = false;  
axExpression1.AddCustomOperator("+", "<b>+</b>", "This is a new binary-operator",2);  
axExpression1.AddCustomOperator("xxx", "<b>xxx</b>", "This is a new keyword",0);  
axExpression1.AddCustomOperator("yyy", "<b>yyy</b>", "This is a unary-operator",1);  
axExpression1.AddCustomOperator("zzz", "<b>zzz</b>", "This is a binary-operator",2);  
axExpression1.Text = "xxx + yyy(100 zzz 200)";
```

X++ (Dynamics Ax 2009)

```
public void init()
```

```

{
;

super();

expression1.AllowValueKeyword(false);
expression1.AllowDefaultOperators(false);
expression1.AddCustomOperator("+", "<b>+</b>", "This is a new binary-
operator", COMVariant::createFromInt(2));
expression1.AddCustomOperator("xxx", "<b>xxx</b>", "This is a new
keyword", COMVariant::createFromInt(0));
expression1.AddCustomOperator("yyy", "<b>yyy</b>", "This is a unary-
operator", COMVariant::createFromInt(1));
expression1.AddCustomOperator("zzz", "<b>zzz</b>", "This is a binary-
operator", COMVariant::createFromInt(2));
expression1.Text("xxx + yyy(100 zzz 200)");
}

```

VFP

```

with thisform.Expression1
.AllowValueKeyword = .F.
.AllowDefaultOperators = .F.
.AddCustomOperator("+", "<b>+</b>", "This is a new binary-operator", 2)
.AddCustomOperator("xxx", "<b>xxx</b>", "This is a new keyword", 0)
.AddCustomOperator("yyy", "<b>yyy</b>", "This is a unary-operator", 1)
.AddCustomOperator("zzz", "<b>zzz</b>", "This is a binary-operator", 2)
.Text = "xxx + yyy(100 zzz 200)"
endwith

```

dBASE Plus

local oExpression

oExpression = form.EXPRESSIONACTIVEXCONTROL1.nativeObject

oExpression.**AllowValueKeyword** = false

oExpression.**AllowDefaultOperators** = false

oExpression.**AddCustomOperator**("+", "+", "This is a new binary-operator", 2)

```
oExpression.AddCustomOperator("xxx", "<b>xxx</b>", "This is a new keyword",0)
oExpression.AddCustomOperator("yyy", "<b>yyy</b>", "This is a unary-operator",1)
oExpression.AddCustomOperator("zzz", "<b>zzz</b>", "This is a binary-operator",2)
oExpression.Text = "xxx + yyy(100 zzz 200)"
```

XBasic (Alpha Five)

```
Dim oExpression as P
```

```
oExpression = topparent:CONTROL_ACTIVEX1.activex
oExpression.AllowValueKeyword = .f
oExpression.AllowDefaultOperators = .f
oExpression.AddCustomOperator("+", "<b>+</b>", "This is a new binary-operator",2)
oExpression.AddCustomOperator("xxx", "<b>xxx</b>", "This is a new keyword",0)
oExpression.AddCustomOperator("yyy", "<b>yyy</b>", "This is a unary-operator",1)
oExpression.AddCustomOperator("zzz", "<b>zzz</b>", "This is a binary-operator",2)
oExpression.Text = "xxx + yyy(100 zzz 200)"
```

Delphi 8 (.NET only)

```
with AxExpression1 do
begin
  AllowValueKeyword := False;
  AllowDefaultOperators := False;
  AddCustomOperator('+', '<b>+</b>', 'This is a new binary-operator', TObject(2));
  AddCustomOperator('xxx', '<b>xxx</b>', 'This is a new keyword', TObject(0));
  AddCustomOperator('yyy', '<b>yyy</b>', 'This is a unary-operator', TObject(1));
  AddCustomOperator('zzz', '<b>zzz</b>', 'This is a binary-operator', TObject(2));
  Text := 'xxx + yyy(100 zzz 200)';
end
```

Delphi (standard)

```
with Expression1 do
begin
  AllowValueKeyword := False;
  AllowDefaultOperators := False;
  AddCustomOperator('+', '<b>+</b>', 'This is a new binary-operator', OleVariant(2));
```

```
AddCustomOperator('xxx', '<b>xxx</b>', 'This is a new keyword', OleVariant(0));  
AddCustomOperator('yyy', '<b>yyy</b>', 'This is a unary-operator', OleVariant(1));  
AddCustomOperator('zzz', '<b>zzz</b>', 'This is a binary-operator', OleVariant(2));  
Text := 'xxx + yyy(100 zzz 200)';  
end
```

Visual Objects

```
oDCOCX_Exontrol1:AllowValueKeyword := false  
oDCOCX_Exontrol1:AllowDefaultOperators := false  
oDCOCX_Exontrol1:AddCustomOperator("+", "<b>+</b>", "This is a new binary-  
operator", 2)  
oDCOCX_Exontrol1:AddCustomOperator("xxx", "<b>xxx</b>", "This is a new keyword", 0)  
oDCOCX_Exontrol1:AddCustomOperator("yyy", "<b>yyy</b>", "This is a unary-  
operator", 1)  
oDCOCX_Exontrol1:AddCustomOperator("zzz", "<b>zzz</b>", "This is a binary-  
operator", 2)  
oDCOCX_Exontrol1:Text := "xxx + yyy(100 zzz 200)"
```

PowerBuilder

OleObject oExpression

```
oExpression = ole_1.Object  
oExpression.AllowValueKeyword = false  
oExpression.AllowDefaultOperators = false  
oExpression.AddCustomOperator("+", "<b>+</b>", "This is a new binary-operator", 2)  
oExpression.AddCustomOperator("xxx", "<b>xxx</b>", "This is a new keyword", 0)  
oExpression.AddCustomOperator("yyy", "<b>yyy</b>", "This is a unary-operator", 1)  
oExpression.AddCustomOperator("zzz", "<b>zzz</b>", "This is a binary-operator", 2)  
oExpression.Text = "xxx + yyy(100 zzz 200)"
```

Visual DataFlex

Procedure OnCreate

Forward Send OnCreate

Set **ComAllowValueKeyword** to False

Set **ComAllowDefaultOperators** to False

```
Send ComAddCustomOperator "+" "<b>+</b>" "This is a new binary-operator" 2
Send ComAddCustomOperator "xxx" "<b>xxx</b>" "This is a new keyword" 0
Send ComAddCustomOperator "yyy" "<b>yyy</b>" "This is a unary-operator" 1
Send ComAddCustomOperator "zzz" "<b>zzz</b>" "This is a binary-operator" 2
Set ComText to "xxx + yyy(100 zzz 200)"
```

```
End_Procedure
```

Xbase++

```
#include "AppEvent.ch"
```

```
#include "ActiveX.ch"
```

```
PROCEDURE Main
```

```
LOCAL oForm
```

```
LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
```

```
LOCAL oExpression
```

```
oForm := XbpDialog():new( AppDesktop() )
```

```
oForm:drawingArea:clipChildren := .T.
```

```
oForm:create( ,, {100,100}, {640,480},,, .F. )
```

```
oForm:close := {|| PostAppEvent( xbeP_Quit )}
```

```
oExpression := XbpActiveXControl():new( oForm:drawingArea )
```

```
oExpression:CLSID := "Exontrol.Expression.1" /*{B33F5489-49AC-4155-98E7-
9BBFC57FF019}*/
```

```
oExpression:create(,, {10,60},{610,370} )
```

```
oExpression:AllowValueKeyword := .F.
```

```
oExpression:AllowDefaultOperators := .F.
```

```
oExpression:AddCustomOperator("+", "<b>+</b>", "This is a new binary-
operator",2)
```

```
oExpression:AddCustomOperator("xxx", "<b>xxx</b>", "This is a new keyword",0)
```

```
oExpression:AddCustomOperator("yyy", "<b>yyy</b>", "This is a unary-operator",1)
```

```
oExpression:AddCustomOperator("zzz", "<b>zzz</b>", "This is a binary-operator",2)
```

```
oExpression:Text := "xxx + yyy(100 zzz 200)"
```

```
oForm:Show()
```

```
DO WHILE nEvent != xbeP_Quit
```

```
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
```

```
    oXbp:handleEvent( nEvent, mp1, mp2 )
```

```
ENDDO
```

```
RETURN
```

property Expression.AllowContextMenu as Boolean

Specifies whether the control's default context menu is available.

Type	Description
Boolean	A boolean expression that indicates whether the control displays a popup menu when user right clicks the control.

The control's context menu is called if user right clicks the control's client area. By default, the AllowContextMenu property is True. Use the AllowContextMenu property to disable the default context menu. Use the [ContextMenuItems](#) property to add new entries to the control's context menu. Use the [Caption](#) property to change the captions in the control's default context menu.

The control's context menu contains by default the following items:

- Undo (Removes the last expression control operation).
- Redo (Redoes the next action in the control's redo queue).
- Cut (Delete (cuts) the current selection (if any) in the expression control and copies the deleted text to the clipboard).
- Copy (Copies the current selection (if any) in the expression control to the Clipboard).
- Paste (Inserts the data from the Clipboard into the expression control at the insertion point, the location of the caret).
- Delete (Deletes the current selection).
- Select All (Selects the entire document).

An item is disabled if the respective operation is not available in the context.

property Expression.AllowDefaultOperators as Boolean

Specifies whether the expression supports default operators.

Type	Description
Boolean	A Boolean expression that specifies whether the expression supports the default operators. The list of all default constants and operators are listed here .

By default, the AllowDefaultOperators property is True, which indicates that expression supports all default operators. The AllowDefaultOperators property specifies whether the expression supports default operators. For instance, you can disable all constants and operators by setting the AllowDefaultOperators property on False. The [AllowValueKeyword](#) property specifies whether the expression supports the value keyword. The [AddCustomOperator](#) method allows you to add custom operators. The [ClearCustomOperators](#) method clears the custom operators.

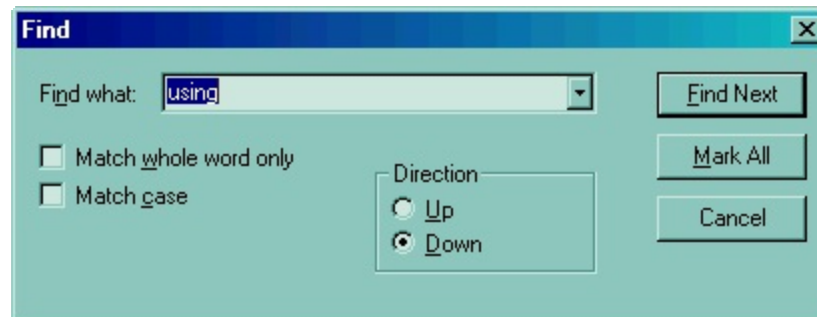
property Expression.AllowFind as Boolean

Specifies whether control displays a find dialog when user presses CTRL+F key.

Type	Description
Boolean	A boolean expression that indicates whether the control shows a Find dialog when user presses the CTRL + F key.

Use the AllowFind property to enable Find support into your control. By default the AllowFind property is True. Use the [AllowReplace](#) property to enable Replace support for your control. Use the [Find](#) method to find and highlight a string. Use the [Caption](#) property to change the captions for fields inside the control's Find dialog.

The following screen shot shows the Find dialog:



You can invoke any of the following commands: Delete, Copy, Cut, Find, Replace, FindNext, FindPrev, Paste, Select All, Undo, Redo, Incremental Search. Each command has a unique identifier like follows:

```
#define ID_EDIT_REPLACE          0xE129
#define ID_EDIT_DELETE          0xE120
#define ID_EDIT_COPY            0xE122
#define ID_EDIT_CUT             0xE123
#define ID_EDIT_FIND            0xE124
#define ID_EDIT_PASTE           0xE125
#define ID_EDIT_SELECT_ALL      0xE12A
#define ID_EDIT_UNDO            0xE12B
#define ID_EDIT_REDO            0xE12C
#define ID_EDIT_FINDNEXT        0xE12D
#define ID_EDIT_FINDPREV        0xE12E
#define ID_CODE_COMPLETION      0xEA01
#define ID_EDIT_INCREMENTALSEARCH 0xEA02
```

For instance, the following VB sample displays the Find dialog when user clicks a button:

```
Private Declare Function SendMessage Lib "user32" Alias "SendMessageA" (ByVal hwnd As Long, ByVal wParam As Long, ByVal lParam As Any) As Long
Private Const WM_COMMAND = &H111

Private Sub Command1_Click()
    Expression1.SetFocus
    SendMessage Expression1.hwnd, WM_COMMAND, &HE124 * 65536, 0
End Sub
```

The wParam parameter of SendMessage API function needs to be Command's Identifier * 65536.

The following C++ sample displays the Find dialog when the user clicks a button:

```
void OnButton1()
{
    m_expression.SetFocus();
    m_expression.SendMessage( WM_COMMAND, MAKEWPARAM(0, 0xE124), NULL );
}
```

The following VB.NET displays the Find dialog when user the clicks a button:

```
<System.Runtime.InteropServices.DllImport("user32.dll")> _
Public Shared Function SendMessage(ByVal hWnd As Int32, ByVal wParam As Int32, ByVal lParam As Int32) As Int32

End Function

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click
    With AxExpression1
        .Focus()
        SendMessage(hWnd, &H111, &HE124 << 16, 0)
    End With
End Sub
```

where the DllImport imports definition for SendMessage API function in VB.NET.

The following C# displays the Find dialog when user the clicks a button:

```
[System.Runtime.InteropServices.DllImport("user32.dll")]
static extern int SendMessage(int hWnd, int wParam, int lParam);

private void button1_Click(object sender, EventArgs e)
{
    axExpression1.Focus();
    SendMessage(axExpression1.hWnd, 0x111, 0xE124 << 16, 0);
}
```

where the DllImport imports definition for SendMessage API function in C#.

The following VFP displays the Find dialog when user the clicks a button:

```
DECLARE INTEGER SendMessage IN user32;
    INTEGER hWnd;;
    INTEGER Msg;;
    INTEGER wParam;;
    INTEGER lParam
```

with thisform.Expression1.Object

```
    SendMessage( .hWnd, 273, BITLSHIFT(57636, 16), 0)
endwith
```

property Expression.AllowIncrementalSearch as Boolean

Specifies whether the control allows incremental search.

Type	Description
Boolean	A boolean expression that indicates whether the control allows Incremental search.

By default, the AllowIncrementalSearch property is True. The AllowIncrementalSearch property specifies whether the control displays the "Incremental search" bar when the user presses the CTRL + I combination. If the "Incremental search" bar is visible, the control searches and selects the text as the user types characters. The "Incremental Search" bar is hidden as soon as the user presses ENTER key, or any arrow keys. Use the [AllowFind](#) property to allow Find dialog in the control. The control displays an "Incremental search" bar at the bottom side that indicates the string being searched, like in the following screen shot:

```
type(value) in (0,1) ? 'null' : ( dbl(value)<0 ? '<fgcolor=FF0000>'+ (value format '2,3,') : (dbl(value)>0 ? '<fgcolor=000
```

```
Incremental Search: value
```

```
value =
```

```
value = 1
```

```
value = _0 12
```

1
0 12

You can invoke any of the following commands: Delete, Copy, Cut, Find, Replace, FindNext, FindPrev, Paste, Select All, Undo, Redo, Incremental Search. Each command has an unique identifier like follows:

```
#define ID_EDIT_REPLACE          0xE129
#define ID_EDIT_DELETE          0xE120
#define ID_EDIT_COPY            0xE122
#define ID_EDIT_CUT             0xE123
#define ID_EDIT_FIND            0xE124
#define ID_EDIT_PASTE           0xE125
#define ID_EDIT_SELECT_ALL      0xE12A
#define ID_EDIT_UNDO            0xE12B
#define ID_EDIT_REDO            0xE12C
#define ID_EDIT_FINDNEXT       0xE12D
#define ID_EDIT_FINDPREV       0xE12E
#define ID_CODE_COMPLETION      0xEA01
#define ID_EDIT_INCREMENTALSEARCH 0xEA02
```

For instance, the following VB sample displays the Find dialog when user clicks a button:

```
Private Declare Function SendMessage Lib "user32" Alias "SendMessageA" (ByVal hwnd As Long, ByVal wParam As Long, ByVal lParam As Any) As Long
Private Const WM_COMMAND = &H111
```

```
Private Sub Command1_Click()
    Expression1.SetFocus
    SendMessage Expression1.hwnd, WM_COMMAND, &HEA02 * 65536, 0
End Sub
```

The wParam parameter of SendMessage API function needs to be Command's Identifier * 65536.

The following C++ sample displays the Find dialog when the user clicks a button:

```
void OnButton1()
{
    m_expression.SetFocus();
    m_expression.SendMessage( WM_COMMAND, MAKEWPARAM(0, 0xEA02), NULL );
}
```

The following VB.NET displays the Find dialog when user the clicks a button:

```
<System.Runtime.InteropServices.DllImport("user32.dll")> _
Public Shared Function SendMessage(ByVal hWnd As Int32, ByVal wParam As Int32, ByVal lParam As Int32) As Int32

End Function

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click
    With AxExpression1
        .Focus()
        SendMessage(hWnd, &H111, &HEA02 << 16, 0)
    End With
End Sub
```

where the DllImport imports definition for SendMessage API function in VB.NET.

The following C# displays the Find dialog when user the clicks a button:

```
[System.Runtime.InteropServices.DllImport("user32.dll")]
static extern int SendMessage(int hWnd, int wParam, int lParam);

private void button1_Click(object sender, EventArgs e)
{
    axExpression1.Focus();
    SendMessage(axExpression1.hWnd, 0x111, 0xEA02 << 16, 0);
}
```

where the DllImport imports definition for SendMessage API function in C#.

The following VFP displays the Find dialog when user the clicks a button:

```
DECLARE INTEGER SendMessage IN user32;
    INTEGER hWnd;;
    INTEGER Msg;;
    INTEGER wParam;;
    INTEGER lParam
```

with thisform.Expression1.Object

```
    SendMessage( .hWnd, 273, BITLSHIFT(59906, 16), 0)
endwith
```

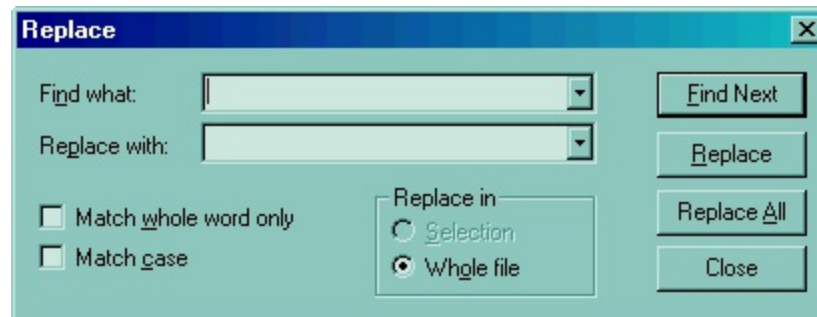
property Expression.AllowReplace as Boolean

Specifies whether control displays a replace dialog when user presses CTRL+H key.

Type	Description
Boolean	A boolean expression that indicates whether the control displays a Replace dialog when user presses CTRL + H key.

Use the AllowReplace property to enable Replace support for your control. Use the [AllowFind](#) property to enable Find support for your control. Use the [Caption](#) property to internationalize the Replace dialog.

The following screen show displays the Replace dialog:



You can invoke any of the following commands: Delete, Copy, Cut, Find, Replace, FindNext, FindPrev, Paste, Select All, Undo and Redo. Each command has a unique identifier like follows:

```
#define ID_EDIT_REPLACE          0xE129
#define ID_EDIT_DELETE          0xE120
#define ID_EDIT_COPY            0xE122
#define ID_EDIT_CUT             0xE123
#define ID_EDIT_FIND            0xE124
#define ID_EDIT_PASTE           0xE125
#define ID_EDIT_SELECT_ALL      0xE12A
#define ID_EDIT_UNDO            0xE12B
#define ID_EDIT_REDO            0xE12C
#define ID_EDIT_FINDNEXT       0xE12D
#define ID_EDIT_FINDPREV       0xE12E
#define ID_CODE_COMPLETION      0xEA01
#define ID_EDIT_INCREMENTALSEARCH 0xEA02
```

For instance, the following VB sample displays the Replace dialog when user clicks a

button:

```
Private Declare Function SendMessage Lib "user32" Alias "SendMessageA" (ByVal hwnd As Long, ByVal wParam As Long, ByVal lParam As Long, lParam As Any) As Long
Private Const WM_COMMAND = &H111

Private Sub Command1_Click()
    Expression1.SetFocus
    SendMessage Expression1.hwnd, WM_COMMAND, &HE129 * 65536, 0
End Sub
```

The wParam parameter of SendMessage API function needs to be Command's Identifier * 65536.

The following C++ sample displays the Replace dialog when the user clicks a button:

```
void OnButton1()
{
    m_expression.SetFocus();
    m_expression.SendMessage( WM_COMMAND, MAKEWPARAM(0, 0xE129), NULL );
}
```

The following VB.NET displays the Replace dialog when user the clicks a button:

```
<System.Runtime.InteropServices.DllImport("user32.dll")> _
Public Shared Function SendMessage(ByVal hWnd As Int32, ByVal wParam As Int32, ByVal lParam As Int32, ByVal lParam As Int32) As Int32

End Function

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click
    With AxExpression1
        .Focus()
        SendMessage(hWnd, &H111, &HE129 << 16, 0)
    End With
End Sub
```

where the DllImport imports definition for SendMessage API function in VB.NET.

The following C# displays the Replace dialog when user the clicks a button:

```
[System.Runtime.InteropServices.DllImport("user32.dll")]
static extern int SendMessage(int hWnd, int wParam, int lParam);

private void button1_Click(object sender, EventArgs e)
{
    axExpression1.Focus();
    SendMessage(axExpression1.hWnd, 0x111, 0xE129 << 16, 0);
}
```

where the DllImport imports definition for SendMessage API function in C#.

The following VFP displays the Replace dialog when user the clicks a button:

```
DECLARE INTEGER SendMessage IN user32;
    INTEGER hWnd;;
    INTEGER Msg;;
    INTEGER wParam;;
    INTEGER lParam
```

with thisform.Expression1.Object

```
    SendMessage( .hWnd, 273, BITLSHIFT(57641, 16), 0)
endwith
```

property Expression.AllowSplitter as SplitterEnum

Specifies whether the control's splitter is visible or hidden.

Type	Description
SplitterEnum	A SplitterEnum expression that indicates the splitters being visible.

By default, the control displays no splitters. Use the AllowSplitter property to add or remove the control's splitters. A splitter splits the expression control in panes. Use the [SplitPaneWidth](#) property to horizontally split the control. Use the [SplitPaneHeight](#) property to vertically split the control. The [SplitterChange](#) event notifies your application that the control's splitter is changed. Use the [FocusPane](#) property to specify the pane that has the focus.

property Expression.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. You can invoke any of the following commands: Delete, Copy, Cut, Find, Replace, Incremental Search, FindNext, FindPrev, Paste, Select All, Undo and Redo. Use the [AllowContextMenu](#) property to enable the control's context menu.

Each command has an unique identifier like follows:

```
#define ID_EDIT_REPLACE          0xE129
#define ID_EDIT_DELETE          0xE120
#define ID_EDIT_COPY            0xE122
#define ID_EDIT_CUT             0xE123
#define ID_EDIT_FIND            0xE124
#define ID_EDIT_PASTE           0xE125
#define ID_EDIT_SELECT_ALL      0xE12A
#define ID_EDIT_UNDO            0xE12B
#define ID_EDIT_REDO            0xE12C
#define ID_EDIT_FINDNEXT       0xE12D
#define ID_EDIT_FINDPREV       0xE12E
#define ID_CODE_COMPLETION      0xEA01
#define ID_EDIT_INCREMENTALSEARCH 0xEA02
```

The sample requires the definition for SendMessage API function. For instance, the following VB sample calls the Undo command when the user clicks a button.

```
Private Declare Function SendMessage Lib "user32" Alias "SendMessageA" (ByVal hwnd As Long, ByVal wParam As Long, ByVal lParam As Long, lParam As Any) As Long
Private Const WM_COMMAND = &H111

Private Sub Command1_Click()
    SendMessage Expression1.hwnd, WM_COMMAND, &HE12B * 65536, 0
End Sub
```

The `wParam` parameter of `SendMessage` API function needs to be Command's Identifier * 65536.

The following C++ sample calls the Redo command when user clicks a button:

```
void OnButtonRedo()
{
    m_expression.SendMessage( WM_COMMAND, MAKEWPARAM(0, 0xE12C), NULL );
}
```

The [CanUndo](#) property determines whether the last edit operation can be undone. The [CanRedo](#) property determines if the redo queue contains any actions.

The following VB.NET sample calls the Undo command when the user clicks a button:

```
<System.Runtime.InteropServices.DllImport("user32.dll")> _
Public Shared Function SendMessage(ByVal hWnd As Int32, ByVal wParam As Int32, ByVal lParam As Int32) As Int32

End Function

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click
    With AxExpression1
        .Focus()
        SendMessage(hWnd, &H111, &HE12B << 16, 0)
    End With
End Sub
```

The following C# sample calls the Undo command when the user clicks a button:

```
[System.Runtime.InteropServices.DllImport("user32.dll")]
static extern int SendMessage(int hWnd, int wParam, int lParam);

private void button1_Click(object sender, EventArgs e)
{
    axExpression1.Focus();
    SendMessage(axExpression1.hWnd, 0x111, 0xE12B << 16, 0);
}
```

}

The following VFP sample calls the Undo command when the user clicks a button:

```
DECLARE INTEGER SendMessage IN user32;  
  INTEGER hWnd;;  
  INTEGER Msg;;  
  INTEGER wParam;;  
  INTEGER lParam  
  
with thisform.Expression1.Object  
  SendMessage( .hWnd, 273, BITLSHIFT(57643, 16), 0)  
endwith
```

property Expression.AllowValueKeyword as Boolean

Specifies whether the expression supports the value keyword.

Type	Description
Boolean	A Boolean expression that specifies whether the expression supports the value keyword.

By default, the AllowValueKeyword property is True, which indicates that expression supports the **value** keyword. The AllowValueKeyword property specifies whether the expression supports the value keyword. The [AllowDefaultOperators](#) property specifies whether the expression supports default operators. The [AddCustomOperator](#) method allows you to add custom operators. The [ClearCustomOperators](#) method clears the custom operators.

property Expression.Appearance as AppearanceEnum

Retrieves or sets the control's appearance.

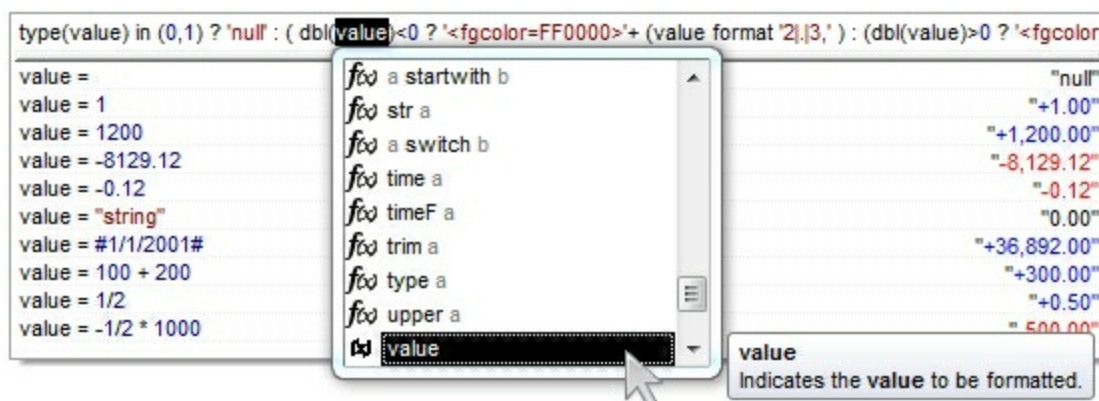
Type

Description

[AppearanceEnum](#)

An AppearanceEnum expression that indicates the control's appearance, or a color expression whose last 7 bits in the high significant byte of the value indicates the index of the skin in the [Appearance](#) collection, being displayed as control's borders. For instance, if the Appearance = 0x1000000, indicates that the first skin object in the Appearance collection defines the control's border. **The Client object in the skin, defines the client area of the control. The lines, scrollbars are always shown in the control's client area. The skin may contain transparent objects, and so you can define round corners. The [frame.ebn](#) file contains such of objects. Use the [exButton's Skin builder](#) to view or change this file**

Use the Appearance property to specify the control's border. Use the [Add](#) method to add new skins to the control. Use the [BackColor](#) property to specify the control's background color. Use the [Background\(exToolTipAppearance\)](#) property indicates the visual appearance of the borders of the tooltips.



The following VB sample changes the visual aspect of the borders of the control (please check the above picture for round corners):

With Expression1

```
.VisualAppearance.Add &H16, "c:\temp\frame.ebn"
```

```
.Appearance = &H16000000
```

```
.BackColor = RGB(250, 250, 250)
```

End With

The following VB.NET sample changes the visual aspect of the borders of the control:

```
With AxExpression1
    .VisualAppearance.Add(&H16, "c:\temp\frame.ebn")
    .Appearance = &H16000000
    .BackColor = Color.FromArgb(250, 250, 250)
End With
```

The following C# sample changes the visual aspect of the borders of the control:

```
axExpression1.VisualAppearance.Add(0x16, "c:\\temp\\frame.ebn");
axExpression1.Appearance = (EXPRESSIONLib.AppearanceEnum)0x16000000;
axExpression1.BackColor = Color.FromArgb(250, 250, 250);
```

The following C++ sample changes the visual aspect of the borders of the control:

```
m_expression.GetVisualAppearance().Add( 0x16, COleVariant( "c:\\temp\\frame.ebn" ) );
m_expression.SetAppearance( 0x16000000 );
m_expression.SetBackColor( RGB(250,250,250) );
```

The following VFP sample changes the visual aspect of the borders of the control:

```
with thisform.Expression1
    .VisualAppearance.Add(0x16, "c:\temp\frame.ebn")
    .Appearance = 0x16000000
    .BackColor = RGB(250, 250, 250)
endwith
```

method Expression.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 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 Expression.BackColor as Color

Retrieves or sets a value that indicates 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.

property Expression.Background(Part as BackgroundPartEnum) as Color

Returns or sets a value that indicates the background color for parts in the control.

Type	Description
Part as BackgroundPartEnum	A BackgroundPartEnum expression that indicates a part in the control.
Color	A Color expression that indicates the background color for a specified part. The last 7 bits in the high significant byte of the color to indicates the identifier of the skin being used. Use the Add method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the background's part.

The Background property specifies a background color or a visual appearance for specific parts in the control. If the Background property is 0, the control draws the part as default. Use the [Add](#) method to add new skins to the control. Use the [Remove](#) method to remove a specific skin from the control. Use the [Clear](#) method to remove all skins in the control. Use the [Refresh](#) method to refresh the control.

The following VB sample changes the visual appearance for the vertical split bar. The sample applies the skin "0" to vertical split bar:

```
With Expression1
  With .VisualAppearance
    .Add &H1, App.Path + "\split.ebn"
  End With
  .Background(exVSplitterApp) = &H1000000
End With
```

The following C++ sample changes the visual appearance for the "drop down" filter button:

```
#include "Appearance.h"
m_tree.GetVisualAppearance().Add( 0x01,
COleVariant(_T("D:\\Temp\\ExExpression.Help\\split.ebn")) );
m_tree.SetBackground( 1 /*exVSplitterApp*/, 0x1000000 );
```

The following VB.NET sample changes the visual appearance for the "drop down" filter

button:

```
With AxExpression1
  With .VisualAppearance
    .Add(&H1, "D:\Temp\ExExpression.Help\split.ebn")
  End With
  .set_Background(EXPRESSIONLib.BackgroundPartEnum.exVSpliterApp, &H1000000)
End With
```

The following C# sample changes the visual appearance for the "drop down" filter button:

```
axExpression1.VisualAppearance.Add(0x1, "D:\\Temp\\ExExpression.Help\\split.ebn");
axExpression1.set_Background(EXPRESSIONLib.BackgroundPartEnum.exVSpliterApp,
0x1000000);
```

The following VFP sample changes the visual appearance for the "drop down" filter button:

```
With thisform.Expression1
  With .VisualAppearance
    .Add(1, "D:\Temp\ExExpression.Help\split.ebn")
  EndWith
  .Object.Background(1) = 16777216
EndWith
```

The 16777216 value is the 0x1000000 value in hexadecimal.

property Expression.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.

Use the `CanRedo` and [CanUndo](#) properties to specify whether an undo/redo action is available. Use the [AllowUndoRedo](#) property to enable the undo/redo support. Use the [Redo](#) method to redo the next action in the control's redo queue.

property Expression.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.

You can use the `CanUndo` property just before doing an undo operation by code. Also, the `CanUndo` property can be used to update your menu, button. The [CanRedo](#) property determines if the redo queue contains any actions. Use the [AllowUndoRedo](#) property to enable undo/redo support. Use the [Undo](#) method to undo the last expression-control operation.

property Expression.Caption(Dialog as DialogEnum, Field as FieldDialogEnum) as String

Specifies the caption for a field on the dialog, or on the control's context menu.

Type	Description
Dialog as DialogEnum	A DialogEnum expression that indicates the dialog where the changes occurs.
Field as FieldDialogEnum	A FieldDialogEnum expression that indicates the field being changed.
String	A string expression that indicates the caption of the field in the dialog.

Use the Caption property of the control to internationalize the Find or Replace dialog of the control.

The following VB sample changes the descriptions of the fields in the 'Find' and 'Replace' dialogs for Romanian language:

With Expression1

```
.Caption(exFindDialog, exCaption) = "Cauta"  
.Caption(exFindDialog, exFieldFindWhat) = "Gaseste"  
.Caption(exFindDialog, exFieldMatchCase) = "Senzitiv la context"  
.Caption(exFindDialog, exFieldWordOnly) = "Doar intreg cuvintul"  
.Caption(exFindDialog, exFieldCancel) = "Abandon"  
.Caption(exFindDialog, exFieldDirection) = "Directie"  
.Caption(exFindDialog, exFieldDown) = "Jos"  
.Caption(exFindDialog, exFieldUp) = "Sus"  
.Caption(exFindDialog, exFieldFindNext) = "Urmatorul"  
.Caption(exFindDialog, exFieldMarkAll) = "Marcheaza tot"  
.Caption(exFindDialog, exErrorFindNext) = "Nu gasesc textul "  
.Caption(exFindDialog, exErrorTitle) = "Eroare"  
  
.Caption(exReplaceDialog, exCaption) = "Inlocuieste"  
.Caption(exReplaceDialog, exFieldFindWhat) = "Gaseste"  
.Caption(exReplaceDialog, exFieldCancel) = "Inchide"  
.Caption(exReplaceDialog, exFieldReplaceWith) = "Inlocuieste cu"  
.Caption(exReplaceDialog, exFieldMatchCase) = "Senzitiv la context"  
.Caption(exReplaceDialog, exFieldWordOnly) = "Doar intreg cuvintul"
```

```

.Caption(exReplaceDialog, exFieldReplaceln) = "Inlocuieste in"
.Caption(exReplaceDialog, exFieldSelection) = "Selectie"
.Caption(exReplaceDialog, exFieldWholeFile) = "In tot fisierul"
.Caption(exReplaceDialog, exFieldFindNext) = "Urmatorul"
.Caption(exReplaceDialog, exFieldReplace) = "Inlocuieste"
.Caption(exReplaceDialog, exFieldReplaceAll) = "Inlocuieste tot"
.Caption(exReplaceDialog, exReplaceDone) = "Sa terminat cautarea textului "
.Caption(exReplaceDialog, exErrorTitle) = "Eroare"
End With

```

The following VB sample changes the captions in the control's context menu:

With Expression1

```

.Caption(exContextMenu, exContextUndo) = "Revin"
.Caption(exContextMenu, exContextRedo) = "Refac"
.Caption(exContextMenu, exContextCut) = "Sterg si copiez"
.Caption(exContextMenu, exContextCopy) = "Copiez"
.Caption(exContextMenu, exContextPaste) = "Suprapun"
.Caption(exContextMenu, exContextDelete) = "Sterg"
.Caption(exContextMenu, exContextSelectAll) = "Selectez tot"
End With

```

The following C++ sample changes the captions in the control's context menu:

```

m_expression.SetCaption( 2 /*exContextMenu*/, 16384 /*exContextUndo*/, "Revin" );
m_expression.SetCaption( 2 /*exContextMenu*/, 16385 /*exContextRedo*/, "Refac" );
m_expression.SetCaption( 2 /*exContextMenu*/, 16387 /*exContextCut*/, "Sterg si copiez" );
m_expression.SetCaption( 2 /*exContextMenu*/, 16388 /*exContextCopy*/, "Copiez" );
m_expression.SetCaption( 2 /*exContextMenu*/, 16389 /*exContextPaste*/, "Suprapun" );
m_expression.SetCaption( 2 /*exContextMenu*/, 16390 /*exContextDelete*/, "Sterg" );
m_expression.SetCaption( 2 /*exContextMenu*/, 16392 /*exContextSelectAll*/, "Selectez tot" );

```

The following VB.NET sample changes the captions in the control's context menu:

With AxExpression1

```

.set_Caption(EXPRESSIONLib.DialogEnum.exContextMenu,
EXPRESSIONLib.FieldDialogEnum.exContextUndo, "Revin")

```

```
.set_Caption(EXPRESSIONLib.DialogEnum.exContextMenu,
EXPRESSIONLib.FieldDialogEnum.exContextRedo, "Refac")
.set_Caption(EXPRESSIONLib.DialogEnum.exContextMenu,
EXPRESSIONLib.FieldDialogEnum.exContextCut, "Sterg si copiez")
.set_Caption(EXPRESSIONLib.DialogEnum.exContextMenu,
EXPRESSIONLib.FieldDialogEnum.exContextCopy, "Copiez")
.set_Caption(EXPRESSIONLib.DialogEnum.exContextMenu,
EXPRESSIONLib.FieldDialogEnum.exContextPaste, "Suprapun")
.set_Caption(EXPRESSIONLib.DialogEnum.exContextMenu,
EXPRESSIONLib.FieldDialogEnum.exContextDelete, "Sterg")
.set_Caption(EXPRESSIONLib.DialogEnum.exContextMenu,
EXPRESSIONLib.FieldDialogEnum.exContextSelectAll, "Selectez tot")
End With
```

The following C# sample changes the captions in the control's context menu:

```
axExpression1.set_Caption(EXPRESSIONLib.DialogEnum.exContextMenu,
EXPRESSIONLib.FieldDialogEnum.exContextUndo, "Revin");
axExpression1.set_Caption(EXPRESSIONLib.DialogEnum.exContextMenu,
EXPRESSIONLib.FieldDialogEnum.exContextRedo, "Refac");
axExpression1.set_Caption(EXPRESSIONLib.DialogEnum.exContextMenu,
EXPRESSIONLib.FieldDialogEnum.exContextCut, "Sterg si copiez");
axExpression1.set_Caption(EXPRESSIONLib.DialogEnum.exContextMenu,
EXPRESSIONLib.FieldDialogEnum.exContextCopy, "Copiez");
axExpression1.set_Caption(EXPRESSIONLib.DialogEnum.exContextMenu,
EXPRESSIONLib.FieldDialogEnum.exContextPaste, "Suprapun");
axExpression1.set_Caption(EXPRESSIONLib.DialogEnum.exContextMenu,
EXPRESSIONLib.FieldDialogEnum.exContextDelete, "Sterg");
axExpression1.set_Caption(EXPRESSIONLib.DialogEnum.exContextMenu,
EXPRESSIONLib.FieldDialogEnum.exContextSelectAll, "Selectez tot");
```

The following VFP sample changes the captions in the control's context menu:

```
With thisform.Expression1.Object
.Caption( 2, 16384 ) = "Revin"
.Caption( 2, 16385 ) = "Refac"
.Caption( 2, 16387 ) = "Sterg si copiez"
.Caption( 2, 16388 ) = "Copiez"
```

.Caption(2, 16389) = "Suprapun"

.Caption(2, 16390) = "Sterg"

.Caption(2, 16392) = "Selectez tot"

endwith

method Expression.ClearCustomOperators ()

Clears the custom operators.

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

The ClearCustomOperators method clears the custom operators. The ClearCustomOperators method clears just operators being added previously by [AddCustomOperator](#) method. The [AllowDefaultOperators](#) property specifies whether the expression supports default operators. The [AddCustomOperator](#) method allows you to add custom operators. The [AllowValueKeyword](#) property specifies whether the expression supports the value keyword.

property Expression.CodeCompletion as Boolean

Specifies whether the code completion feature is enabled or disabled.

Type	Description
Boolean	A boolean expression that indicates whether the code completion feature is enabled.

The control provides code completion support. The user can invoke the control's context list by pressing CTRL+SPACE. combination.

property Expression.ContextKey as Long

Specifies the key combination that opens the control's context window.

Type	Description
Long	A long expression that defines the key combination to open the control's context window.

The ContextKey property specifies the key combination to display the control's context window. By default, the key combination to open the control's context window is CTRL + SPACE. The low byte of the ContextKey property specifies the code of the key, and the high byte of the lowest word of the ContextKey property is a bit combination of the CTRL, SHIFT and ALT keys state.

The following formula should be used to specify a key combination:

```
ContextKey = (KeyCode) + (256 * ( 1 * Ctrl + 2 * Alt + 4 * Shift) )
```

where:

- KeyCode specifies the ASCII code of the key
- Ctrl is 1 if the CTRL key is pressed, or 0 if the CTRL key is not pressed
- Alt is 1 if the ALT key is pressed, or 0 if the ALT key is not pressed
- Alt is 1 if the SHIFT key is pressed, or 0 if the SHIFT key is not pressed

For instance, if you need to change the key combination to ALT + SPACE you need to define the ContextKey property like bellow:

```
With Expression1  
    .ContextKey = vbKeySpace + 256 * 2  
End With
```

or

```
With Expression1  
    .ContextKey = 32 + 256 * 2  
End With
```

since the code for SPACE key is 32. By default, the ContextKey property is 288.

If you need to have a combination like SHIFT + CTRL + F1 you need to define the ContextKey property like follows:

```
With Expression1
```

```
    .ContextKey = vbKeyF1 + 256 * (1 + 4)
```

```
End With
```

How can I find the key code? The easiest way to find out the code for a key is to add a handler for [KeyDown](#) event like follows:

```
Private Sub Expression1_KeyDown(KeyCode As Integer, Shift As Integer)
```

```
    Debug.Print KeyCode
```

```
End Sub
```

Important to notice is that the control's context window is not the control's context menu. The control's context window is shown when user presses the ContextKey combination, instead the control's context menu is displayed when user does a right click

property Expression.ContextMenuItems as String

Specifies a list of items that are added to the control's context menu.

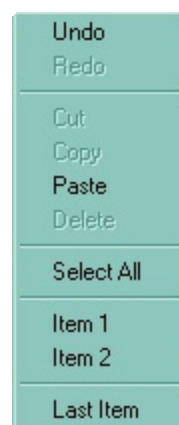
Type	Description
String	A string expression that indicates the list of items being added to the control's context menu.

The control's context menu is displayed if the user right clicks the control. Use the `ContextMenuItems` property to add new entries to the control's context menu. Use the [AllowContextMenu](#) property to enable the control's context menu. The items in the context menu are separated by #13 (carriage return, `vbCr`) character. The [ExecuteContextMenu](#) event occurs if the user selects a custom entry in the control's context menu.

The following sample adds three new entries to the control's context menu:

```
Private Sub Form_Load()  
    With Expression1  
        .AllowContextMenu = True  
        .ContextMenuItems = vbCr & "Item 1" & vbCr & "Item 2" & vbCr & vbCr & "Last  
Item"  
    End With  
End Sub
```

The separator item is represented by an empty item.



The following C++ sample adds some custom entries to the control's context menu:

```
m_expression.SetAllowContextMenu( TRUE );  
CString strCustom( "\\r" );  
strCustom += "Item 1";  
strCustom += "\\r\\r";
```

```
strCustom += "Item 2";  
strCustom += "\r";  
strCustom += "Item 3";  
m_expression.SetContextMenuItems( strCustom );
```

The following VB.NET sample adds some custom entries to the control's context menu:

```
With AxExpression1  
    .AllowContextMenu = True  
    .ContextMenuItems = vbCr & "Item 1" & vbCr & vbCr & "Item 2" & vbCr & "Item 3"  
End With
```

The following C# sample adds some custom entries to the control's context menu:

```
axExpression1.AllowContextMenu = true;  
axExpression1.ContextMenuItems = "\rItem1\r\rItem2\rItem3";
```

The following VFP sample adds some custom entries to the control's context menu:

```
with thisform.Expression1  
    .AllowContextMenu = .t.  
    .ContextMenuItems = chr(13) + "Item 1" + chr(13) + chr(13) + "Item 2" + chr(13) +  
"Item 3"  
endwith
```

property Expression.Cursor(Area as ClientAreaEnum) as Variant

Gets or sets the cursor that is displayed when the mouse pointer hovers the control.

Type	Description
Area as ClientAreaEnum	A ClientAreaEnum expression that indicates the control's zone where the mouse pointer is changed.
Variant	A string expression that indicates a predefined value listed below, a string expression that indicates the path to a cursor file, a long expression that indicates the handle of the cursor.

Use the Cursor property to specify the cursor that control displays when mouse pointer hovers the control's zone like edit area, bookmark bar and so on.

Here's the list of predefined values (string expressions):

- **"exDefault"** - (Default) Shape determined by the object.
- **"exArrow"** - Arrow.
- **"exCross"** - Cross (cross-hair pointer).
- **"exIBeam"** - I Beam.
- **"exIcon"** - Icon (small square within a square).
- **"exSize"** - Size (four-pointed arrow pointing north, south, east, and west).
- **"exSizeNESW"** - Size NE SW (double arrow pointing northeast and southwest).
- **"exSizeNS"** - Size N S (double arrow pointing north and south).
- **"exSizeNWSE"** - Size NW, SE.
- **"exSizeWE"** - Size W E (double arrow pointing west and east).
- **"exUpArrow"** - Up Arrow.
- **"exHourglass"** - Hourglass (wait).
- **"exNoDrop"** - No Drop.
- **"exArrowHourglass"** - Arrow and hourglass.
- **"exHelp"** - Arrow and question mark.
- **"exSizeAll"** - Size all.

If the cursor value is a string expression, the control looks first if it is not a predefined value like listed above, and if not, it tries to load the cursor from a file. If the Cursor property is a long expression it always indicates a handle to a cursor. The API functions like: LoadCursor or LoadCursorFromFile retrieves a handle to a cursor. In .NET framework, the Handle parameter of the Cursor object specifies the handle to the cursor. Use the Cursors object to access to the list of predefined cursors in the .NET framework.

The following VB sample changes the cursor while the mouse pointer hovers the control's line number bar:

```
With Expression1
```

```
    .Cursor(exLineNumberArea) = "exCross"
```

```
End With
```

Here's the VB.NET alternative:

```
AxExpression1.Ctlset_Cursor(EXPRESSIONLib.ClientAreaEnum.exLineNumberArea,  
"exCross")
```

The following VB sample loads a cursor from a file:

```
With Expression1
```

```
    .Cursor(exLineNumberArea) = "C:\WINNT\Cursors\metronom.ani"
```

```
End With
```

And here's the VB.NET alternative:

```
AxExpression1.Ctlset_Cursor(EXPRESSIONLib.ClientAreaEnum.exLineNumberArea,  
"C:\WINNT\Cursors\metronom.ani")
```

The following VB.NET sample changes the cursor with one that Cursors object defines (PanEast cursor):

```
AxExpression1.Ctlset_Cursor(EXPRESSIONLib.ClientAreaEnum.exLineNumberArea,  
Cursors.PanEast.Handle)
```

The following C++ sample loads the cursor from a file:

```
m_expression.SetCursor( 0 /*exExpressionArea*/,  
COleVariant("C:\\WINNT\\Cursors\\metronom.ani" ) );
```

The following C# sample loads the cursor from a file:

```
axExpression1.Ctlset_Cursor(EXPRESSIONLib.ClientAreaEnum.exExpressionArea,  
"C:\\WINNT\\Cursors\\metronom.ani");
```

The following VFP sample loads the cursor from a file:

```
with thisform.Expression1.Object  
    .Cursor(0) = "C:\WINNT\Cursors\metronom.ani"  
endwith
```


property Expression.Description(Keyword as KeywordEnum) as String

Specifies the description of the giving keyword.

Type	Description
Keyword as KeywordEnum	A KeywordEnum expression that specifies the keyword to change its description/tooltip.
String	A String expression that specifies the HTML description/tooltip of the keyword from the cursor.

Use the Description property to customize the description/tooltip for supported keywords. The Description supports the following HTML tags:

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

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

gradient-center

- **<out rrggbb;width> ... </out>** shows the text with outlined characters, where rr/gg/bb represents the red/green/blue values of the outline color, 808080 if missing as gray,

width indicates the size of the outline, 1 if missing. The text color or <fgcolor> defines the color to show the inside text. The HTML tag can be used to define the height of the font. For instance the "<out 000000>

<fgcolor=FFFFFF>outlined</fgcolor></out>" generates the following picture:

outlined

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

shadow

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

outline anti-aliasing

The following samples changes the tooltip that shows when cursor hovers a keyword:

VBA (MS Access, Excell...)

With Expression1

.Expression = "value"

.**Description**(0) = "This is a bit of text to be shown when cursor hovers the value keyword."

.**Refresh**

End With

VB6

With Expression1

.Expression = "value"

.**Description**(exKeywordValue) = "This is a bit of text to be shown when cursor hovers the value keyword."

.**Refresh**

End With

VB.NET

With Expression1

```
.Expression = "value"
```

```
.set_Description(exontrol.EXPRESSIONLib.KeywordEnum.exKeywordValue,"This is a bit of text to be shown when cursor hovers the <b>value</b> keyword.")
```

```
.Refresh()
```

End With

VB.NET for /COM

With AxExpression1

```
.Expression = "value"
```

```
.set_Description(EXPRESSIONLib.KeywordEnum.exKeywordValue,"This is a bit of text to be shown when cursor hovers the <b>value</b> keyword.")
```

```
.Refresh()
```

End With

C++

```
/*
```

```
Copy and paste the following directives to your header file as it defines the namespace 'EXPRESSIONLib' for the library: 'Expression 1.0 Control Library'
```

```
#import <Expression.dll>
```

```
using namespace EXPRESSIONLib;
```

```
*/
```

```
EXPRESSIONLib::IExpressionPtr spExpression1 = GetDlgItem(IDC_EXPRESSION1)->GetControlUnknown();
```

```
spExpression1->PutExpression(L"value");
```

```
spExpression1->PutDescription(EXPRESSIONLib::exKeywordValue,L"This is a bit of text to be shown when cursor hovers the <b>value</b> keyword.");
```

```
spExpression1->Refresh();
```

C++ Builder

```
Expression1->Expression = L"value";
```

```
Expression1->Description[Expressionlib_tlb::KeywordEnum::exKeywordValue] =  
L"This is a bit of text to be shown when cursor hovers the <b>value</b> keyword.";  
Expression1->Refresh();
```

C#

```
expression1.Expression = "value";  
expression1.set_Description(exontrol.EXPRESSIONLib.KeywordEnum.exKeywordValue,  
is a bit of text to be shown when cursor hovers the <b>value</b> keyword.);  
expression1.Refresh();
```

JScript/JavaScript

```
<BODY onload="Init()" >  
<OBJECT CLASSID="clsid:B33F5489-49AC-4155-98E7-9BBFC57FF019"  
id="Expression1"> </OBJECT>  
  
<SCRIPT LANGUAGE="JScript">  
function Init()  
{  
    Expression1.Expression = "value";  
    Expression1.Description(0) = "This is a bit of text to be shown when cursor hovers  
the <b>value</b> keyword.";  
    Expression1.Refresh();  
}  
</SCRIPT>  
</BODY>
```

VBScript

```
<BODY onload="Init()" >  
<OBJECT CLASSID="clsid:B33F5489-49AC-4155-98E7-9BBFC57FF019"  
id="Expression1"> </OBJECT>  
  
<SCRIPT LANGUAGE="VBScript">
```

```
Function Init()
```

```
  With Expression1
```

```
    .Expression = "value"
```

```
    .Description(0) = "This is a bit of text to be shown when cursor hovers the  
<b>value</b> keyword."
```

```
    .Refresh
```

```
  End With
```

```
End Function
```

```
</SCRIPT>
```

```
</BODY>
```

C# for /COM

```
axExpression1.Expression = "value";
```

```
axExpression1.set_Description(EXPRESSIONLib.KeywordEnum.exKeywordValue,"This  
is a bit of text to be shown when cursor hovers the <b>value</b> keyword.");
```

```
axExpression1.Refresh();
```

X++ (Dynamics Ax 2009)

```
public void init()
```

```
{
```

```
  ;
```

```
  super();
```

```
  expression1.Expression("value");
```

```
  expression1.Description(0/*exKeywordValue*/, "This is a bit of text to be shown  
when cursor hovers the <b>value</b> keyword.");
```

```
  expression1.Refresh();
```

```
}
```

Delphi 8 (.NET only)

```
with AxExpression1 do
```

```
begin
```

```
Expression := 'value';  
set_Description(EXPRESSIONLib.KeywordEnum.exKeywordValue, 'This is a bit of  
text to be shown when cursor hovers the <b>value</b> keyword.');
```

Refresh();
end

Delphi (standard)

```
with Expression1 do  
begin  
  Expression := 'value';  
  Description[EXPRESSIONLib_TLB.exKeywordValue] := 'This is a bit of text to be  
shown when cursor hovers the <b>value</b> keyword.';  
  Refresh();  
end
```

VFP

```
with thisform.Expression1  
  .Expression = "value"  
  .Object.Description(0) = "This is a bit of text to be shown when cursor hovers the  
<b>value</b> keyword."  
  .Refresh  
endwith
```

dBASE Plus

```
local oExpression  
  
oExpression = form.EXPRESSIONACTIVEXCONTROL1.nativeObject  
oExpression.Expression = "value"  
oExpression.Template = [Description(0) = "This is a bit of text to be shown when  
cursor hovers the <b>value</b> keyword."] // oExpression.Description(0) = "This is a  
bit of text to be shown when cursor hovers the <b>value</b> keyword."  
oExpression.Refresh()
```

XBasic (Alpha Five)

Dim oExpression as P

oExpression = topparent:CONTROL_ACTIVEX1.activex

oExpression.Expression = "value"

oExpression.Template = "Description(0) = `This is a bit of text to be shown when cursor hovers the value keyword.`" // oExpression.Description(0) = "This is a bit of text to be shown when cursor hovers the value keyword."

oExpression.**Refresh**()

Visual Objects

oDCOCX_Exontrol1.Expression := "value"

oDCOCX_Exontrol1:[Description,exKeywordValue] := "This is a bit of text to be shown when cursor hovers the value keyword."

oDCOCX_Exontrol1.**Refresh**()

PowerBuilder

OleObject oExpression

oExpression = ole_1.Object

oExpression.Expression = "value"

oExpression.**Description**(0, "This is a bit of text to be shown when cursor hovers the value keyword.")

oExpression.**Refresh**()

Visual DataFlex

Procedure OnCreate

Forward Send OnCreate

Set ComExpression to "value"

Set **ComDescription** OLEexKeywordValue to "This is a bit of text to be shown when cursor hovers the value keyword."

Send **ComRefresh**

XBase++

```

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
  LOCAL oForm
  LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
  LOCAL oExpression

  oForm := XbpDialog():new( AppDesktop() )
  oForm:drawingArea:clipChildren := .T.
  oForm:create( ,, {100,100}, {640,480},,, .F. )
  oForm:close := {|| PostAppEvent( xbeP_Quit )}

  oExpression := XbpActiveXControl():new( oForm:drawingArea )
  oExpression:CLSID := "Exontrol.Expression.1" /*{B33F5489-49AC-4155-98E7-
9BBFC57FF019}*/
  oExpression:create(,, {10,60},{610,370} )

  oExpression:Expression := "value"
  oExpression:SetProperty("Description",0/*exKeywordValue*/,"This is a bit of text
to be shown when cursor hovers the <b>value</b> keyword.")
  oExpression:Refresh()

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

```

property Expression.DisplaySelection as Boolean

Specifies whether the control displays the selection.

Type	Description
Boolean	A boolean expression that indicates whether the selection is visible or hidden.

Use the DisplaySelection property to hide the selection. Use the [HideSelection](#) property to hide 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. The [SelForeColorHide](#) property specifies the foreground color of the current selection when the control has no focus, and the [HideSelection](#) property is False. The [SelBackColorHide](#) property specifies the background color of the current selection when the control has no focus, and the [HideSelection](#) property is False.

property Expression.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.

The following VB sample disables the control:

```
Expression1.Enabled = False
```

The following C++ sample disables the control:

```
m_expression.SetEnabled( FALSE );
```

The following VB.NET sample disables the control:

```
AxExpression1.Enabled = False
```

The following C# sample disables the control:

```
axExpression1.Enabled = false;
```

The following VFP sample disables the control:

```
with thisform.Expression1.Object  
    .Enabled = .f.  
endwith
```

property Expression.Evaluate ([Variables as Variant]) as Variant

Evaluates the current expression and returns the result.

Type	Description
Variables as Variant	A list of variables and values to be evaluated within the current expression, separated by ; character,
Variant	A VARIANT expression that indicates the result of the evaluation.

Use the Evaluate property to get the result of evaluation of the control's expression. The [Expression/Text](#) property assigns a new expression to the control. The [IsValid](#) property indicates whether the control's expression is valid/syntactically correct. The [EvaluationResult](#) property specifies the result for each evaluation (line). The [EvaluationText](#) property specifies the values for variables in the expression to be evaluated (per line).

The following samples shows how you can programmatically evaluate the expression:

VBA (MS Access, Excell...)

```
With Expression1
    .Expression = "currency(value)"
    Debug.Print( .Evaluate("value=100") )
End With
```

VB6

```
With Expression1
    .Expression = "currency(value)"
    Debug.Print( .Evaluate("value=100") )
End With
```

VB.NET

```
With Expression1
    .Expression = "currency(value)"
    Debug.Print( .get_Evaluate("value=100") )
End With
```

VB.NET for /COM

With AxExpression1

```
.Expression = "currency(value)"
```

```
Debug.Print( get_Evaluate("value=100") )
```

End With

C++

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

```
#import <Expression.dll>  
using namespace EXPRESSIONLib;
```

```
*/  
EXPRESSIONLib::IExpressionPtr spExpression1 = GetDlgItem(IDC_EXPRESSION1)-  
>GetControlUnknown();  
spExpression1->PutExpression(L"currency(value)");  
OutputDebugStringW( _bstr_t(spExpression1->GetEvaluate("value=100")) );
```

C++ Builder

```
Expression1->Expression = L"currency(value)";  
OutputDebugString( PChar(Expression1->Evaluate[TVariant("value=100")] ) );
```

C#

```
expression1.Expression = "currency(value)";  
System.Diagnostics.Debug.Print( expression1.get_Evaluate("value=100").ToString() );
```

JScript/JavaScript

```
<BODY onload="Init()">  
<OBJECT CLASSID="clsid:B33F5489-49AC-4155-98E7-9BBFC57FF019"  
id="Expression1"></OBJECT>
```

```
<SCRIPT LANGUAGE="JScript">
function Init()
{
    Expression1.Expression = "currency(value)";
    alert( Expression1.Evaluate("value= 100") );
}
</SCRIPT>
</BODY>
```

VBScript

```
<BODY onload="Init()">
<OBJECT CLASSID="clsid:B33F5489-49AC-4155-98E7-9BBFC57FF019"
id="Expression1"></OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Expression1
        .Expression = "currency(value)"
        alert( .Evaluate("value= 100") )
    End With
End Function
</SCRIPT>
</BODY>
```

C# for /COM

```
axExpression1.Expression = "currency(value)";
System.Diagnostics.Debug.Print( axExpression1.get_Evaluate("value= 100").ToString()
);
```

X++ (Dynamics Ax 2009)

```
public void init()
{
```

```
;

super();

expression1.Expression("currency(value)");
print( expression1.Evaluate("value= 100") );
}
```

Delphi 8 (.NET only)

```
with AxExpression1 do
begin
  Expression := 'currency(value)';
  OutputDebugString( get_Evaluate('value= 100') );
end
```

Delphi (standard)

```
with Expression1 do
begin
  Expression := 'currency(value)';
  OutputDebugString( Evaluate['value= 100'] );
end
```

VFP

```
with thisform.Expression1
  .Expression = "currency(value)"
  DEBUGOUT( .Evaluate("value= 100") )
endwith
```

dBASE Plus

```
local oExpression

oExpression = form.EXPRESSIONACTIVEXCONTROL1.nativeObject
oExpression.Expression = "currency(value)"
? Str(oExpression.Evaluate("value= 100"))
```

XBasic (Alpha Five)

```
Dim oExpression as P
```

```
oExpression = topparent:CONTROL_ACTIVEX1.activex
```

```
oExpression.Expression = "currency(value)"
```

```
? oExpression.Evaluate("value=100")
```

Visual Objects

```
oDCOCX_Exontrol1:Expression := "currency(value)"
```

```
OutputDebugString(String2Psz( AsString(oDCOCX_Exontrol1:[Evaluate,"value=100"]  
))
```

PowerBuilder

```
OleObject oExpression
```

```
oExpression = ole_1.Object
```

```
oExpression.Expression = "currency(value)"
```

```
MessageBox("Information",string( String(oExpression.Evaluate("value=100")) ))
```

Visual DataFlex

```
Procedure OnCreate
```

```
Forward Send OnCreate
```

```
Set ComExpression to "currency(value)"
```

```
ShowIn (ComEvaluate(Self,"value=100"))
```

```
End_Procedure
```

XBase++

```
#include "AppEvent.ch"
```

```
#include "ActiveX.ch"
```

PROCEDURE Main

LOCAL oForm

LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL

LOCAL oExpression

oForm := XbpDialog():new(AppDesktop())

oForm:drawingArea:clipChildren := .T.

oForm:create(,, {100,100}, {640,480},,, .F.)

oForm:close := {|| PostAppEvent(xbeP_Quit)}

oExpression := XbpActiveXControl():new(oForm:drawingArea)

oExpression:CLSID := "Exontrol.Expression.1" /*{B33F5489-49AC-4155-98E7-9BBFC57FF019}*/

oExpression:create(,, {10,60},{610,370})

oExpression:Expression := "currency(value)"

DevOut(Transform(oExpression:Evaluate("value=100"), ""))

oForm:Show()

DO WHILE nEvent != xbeP_Quit

nEvent := AppEvent(@mp1, @mp2, @oXbp)

oXbp:handleEvent(nEvent, mp1, mp2)

ENDDO

RETURN

property Expression.EvaluateSelection as Boolean

Specifies whether the control evaluates the selection when it is available.

Type	Description
Boolean	A Boolean expression that specifies whether the control evaluates the selection (if available), instead the whole text.

By default, the EvaluateSelection property is True. While the EvaluateSelection property is True, the control evaluates the selection (if available), rather than entire text. If the EvaluateSelection property is False, the control evaluates the entire text. The [EvaluationText](#) property specifies the values for variables in the expression to be evaluated (per line). The [EvaluationResult](#) property specifies the result for each evaluation (line). The [IsValid](#) property indicates whether the control's expression is valid/syntactically correct. Use the [Evaluate](#) property to get the result of evaluation of the control's expression.

property Expression.EvaluationResult as String

Specifies the evaluation result, each line indicates a result value.

Type	Description
String	A String expression that indicates the result of evaluation for each line in the EvaluationText property

The EvaluationResult property specifies the result for each evaluation (line). The [EvaluationText](#) property specifies the values for variables in the expression to be evaluated (per line). The [IsValid](#) property indicates whether the control's expression is valid/syntactically correct. Use the [Evaluate](#) property to get the result of evaluation of the control's expression. The [Expression/Text](#) property assigns a new expression to the control.

property Expression.EvaluationText as String

Specifies the evaluation text, each line indicates a test value.

Type	Description
String	A String expression that specifies the values for variables to be evaluated (per line)

The EvaluationText property specifies the values for variables in the expression to be evaluated (per line). Known variables are value, %0, %1, and so on. Each line of the EvaluationText property indicates a set of variables to be evaluated. Each line includes assignments separated by ; character. Each assignment is separated by a = character. The [EvaluationResult](#) property specifies the result for each evaluation (line). The [IsValid](#) property indicates whether the control's expression is valid/syntactically correct. Use the [Evaluate](#) property to get the result of evaluation of the control's expression. The [Expression/Text](#) property assigns a new expression to the control.

By default, the EvaluationText property is:

```
value =  
value = 1  
value = -0.12  
value = "string"  
value = #1/1/2001#
```

Each line displays the result of evaluation on the right side of the second-panel.

The following samples assigns multiple values for variables %1, %2:

VBA (MS Access, Excell...)

```
With Expression1  
    .SplitPaneWidth = 196  
    .Expression = "currency(%1 + %2)"  
    .EvaluationText = "%1 = 100;%2 = 200"  
End With
```

VB6

```
With Expression1  
    .SplitPaneWidth = 196  
    .Expression = "currency(%1 + %2)"  
    .EvaluationText = "%1 = 100;%2 = 200"
```

End With

VB.NET

With Expression1

.SplitPaneWidth = 196

.Expression = "currency(%1 + %2)"

.EvaluationText = "%1 = 100;%2 = 200"

End With

VB.NET for /COM

With AxExpression1

.SplitPaneWidth = 196

.Expression = "currency(%1 + %2)"

.EvaluationText = "%1 = 100;%2 = 200"

End With

C++

/*

Copy and paste the following directives to your header file as it defines the namespace 'EXPRESSIONLib' for the library: 'Expression 1.0 Control Library'

#import <Expression.dll>

using namespace EXPRESSIONLib;

*/

EXPRESSIONLib::IExpressionPtr spExpression1 = GetDlgItem(IDC_EXPRESSION1)->GetControlUnknown();

spExpression1->PutSplitPaneWidth(196);

spExpression1->PutExpression(L"currency(%1 + %2)");

spExpression1->**PutEvaluationText**(L"%1 = 100;%2 = 200");

C++ Builder

Expression1->SplitPaneWidth = 196;

Expression1->Expression = L"currency(%1 + %2)";

```
Expression1->EvaluationText = L"%1 = 100;%2 = 200";
```

C#

```
expression1.SplitPaneWidth = 196;  
expression1.Expression = "currency(%1 + %2)";  
expression1.EvaluationText = "%1 = 100;%2 = 200";
```

JScript/JavaScript

```
<BODY onload="Init()">  
<OBJECT CLASSID="clsid:B33F5489-49AC-4155-98E7-9BBFC57FF019"  
id="Expression1"> </OBJECT>  
  
<SCRIPT LANGUAGE="JScript">  
function Init()  
{  
    Expression1.SplitPaneWidth = 196;  
    Expression1.Expression = "currency(%1 + %2)";  
    Expression1.EvaluationText = "%1 = 100;%2 = 200";  
}  
</SCRIPT>  
</BODY>
```

VBScript

```
<BODY onload="Init()">  
<OBJECT CLASSID="clsid:B33F5489-49AC-4155-98E7-9BBFC57FF019"  
id="Expression1"> </OBJECT>  
  
<SCRIPT LANGUAGE="VBScript">  
Function Init()  
    With Expression1  
        .SplitPaneWidth = 196  
        .Expression = "currency(%1 + %2)"  
    End With  
End Function  
</SCRIPT>
```

```
.EvaluationText = "%1 = 100;%2 = 200"
```

```
End With
```

```
End Function
```

```
</SCRIPT>
```

```
</BODY>
```

C# for /COM

```
axExpression1.SplitPaneWidth = 196;
```

```
axExpression1.Expression = "currency(%1 + %2)";
```

```
axExpression1.EvaluationText = "%1 = 100;%2 = 200";
```

X++ (Dynamics Ax 2009)

```
public void init()
```

```
{
```

```
;
```

```
super();
```

```
expression1.SplitPaneWidth(196);
```

```
expression1.Expression("currency(%1 + %2)");
```

```
expression1.EvaluationText("%1 = 100;%2 = 200");
```

```
}
```

Delphi 8 (.NET only)

```
with AxExpression1 do
```

```
begin
```

```
SplitPaneWidth := 196;
```

```
Expression := 'currency(%1 + %2)';
```

```
EvaluationText := '%1 = 100;%2 = 200';
```

```
end
```

Delphi (standard)

```
with Expression1 do
```

```
begin
  SplitPaneWidth := 196;
  Expression := 'currency(%1 + %2)';
  EvaluationText := '%1 = 100;%2 = 200';
end
```

VFP

```
with thisform.Expression1
  .SplitPaneWidth = 196
  .Expression = "currency(%1 + %2)"
  .EvaluationText = "%1 = 100;%2 = 200"
endwith
```

dBASE Plus

```
local oExpression

oExpression = form.EXPRESSIONACTIVEXCONTROL1.nativeObject
oExpression.SplitPaneWidth = 196
oExpression.Expression = "currency(%1 + %2)"
oExpression.EvaluationText = "%1 = 100;%2 = 200"
```

XBasic (Alpha Five)

```
Dim oExpression as P

oExpression = topparent:CONTROL_ACTIVEX1.activex
oExpression.SplitPaneWidth = 196
oExpression.Expression = "currency(%1 + %2)"
oExpression.EvaluationText = "%1 = 100;%2 = 200"
```

Visual Objects

```
oDCOCX_Exontrol1.SplitPaneWidth := 196
oDCOCX_Exontrol1.Expression := "currency(%1 + %2)"
```

```
oDCOCX_Exontrol1:EvaluationText := "%1 = 100;%2 = 200"
```

PowerBuilder

```
OleObject oExpression
```

```
oExpression = ole_1.Object
```

```
oExpression.SplitPaneWidth = 196
```

```
oExpression.Expression = "currency(%1 + %2)"
```

```
oExpression.EvaluationText = "%1 = 100;%2 = 200"
```

Visual DataFlex

```
Procedure OnCreate
```

```
Forward Send OnCreate
```

```
Set ComSplitPaneWidth to 196
```

```
Set ComExpression to "currency(%1 + %2)"
```

```
Set ComEvaluationText to "%1 = 100;%2 = 200"
```

```
End_Procedure
```

XBase++

```
#include "AppEvent.ch"
```

```
#include "ActiveX.ch"
```

```
PROCEDURE Main
```

```
LOCAL oForm
```

```
LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
```

```
LOCAL oExpression
```

```
oForm := XbpDialog():new( AppDesktop() )
```

```
oForm:drawingArea:clipChildren := .T.
```

```
oForm:create( ,, {100,100}, {640,480},,, .F. )
```

```
oForm:close := {|| PostAppEvent( xbeP_Quit )}
```

```
oExpression := XbpActiveXControl():new( oForm:drawingArea )
```

```
oExpression:CLSID := "Exontrol.Expression.1" /*{B33F5489-49AC-4155-98E7-9BBFC57FF019}*/
```

```
oExpression:create(, {10,60},{610,370} )
```

```
oExpression:SplitPaneWidth := 196
```

```
oExpression:Expression := "currency(%1 + %2)"
```

```
oExpression:EvaluationText := "%1 = 100;%2 = 200"
```

```
oForm:Show()
```

```
DO WHILE nEvent != xbeP_Quit
```

```
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
```

```
    oXbp:handleEvent( nEvent, mp1, mp2 )
```

```
ENDDO
```

```
RETURN
```

property Expression.EvaluationTime as Long

Indicates the time in ms to evaluate the expression.

Type	Description
Long	A Long expression that indicates the time in ms to evaluate the expression.

The EvaluationTime property indicates the time in ms to evaluate the expression.

property Expression.EventParam(Parameter as Long) as Variant

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

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

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

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

In most languages you will type something like:

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

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

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

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

reference.

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

method Expression.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 Variant expression that indicates the result after executing the Template.

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

For instance, the following sample retrieves the control's background color:

```
Debug.Print Expression1.ExecuteTemplate("BackColor")
```

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 Expression.Expression as String

Indicates the expression to be evaluated.

Type	Description
String	A String expression that specifies the text to be evaluated.

The Expression property returns the expression to be evaluated. The [EvaluateSelection](#) property specifies whether the control evaluates the selection when it is available. So, if the [EvaluateSelection](#) property the Expression property returns the selected text (if the [SelLength](#) property is greater than 0), or the entire text if the [SelLength](#) property is 0. The control fires the [SelChange](#) event when user changes the selection. Use the [SelfForeColor](#) and [SelBackColor](#) properties to specify the colors for the selected text. The [Text](#) property specifies the control's text/expression.

Most of our components support formatting the values. By formatting we mean that instead displaying a value we can display in the way we desire. Properties such as `Column.FormatColumn`, `Items.FormatCell` and so on support formatting. For instance, the `currency(100)` displays the value 100 as a currency, for instance in US format it will display \$100 while for German format will display 100 . The format expression supports operators, constants and values as described bellow. From case to case, there are few predefined keywords such as **value**, which indicates the value to be formatted, the **%0**, **%1**, **%2**, **%3**, ... indicates variables that could be: the value in a specified column, the value for a specified property of the bar, and so on. If the formatting is using in properties such as: `Items.ItemBar(exBarToolTip)`, `Items.ItemBar(exBarCaption)` or `Items.ItemBar(exBarExtraCaption)` the **%C0**, **%C1**, **%C2**, ... indicates the values in the columns.

For instance:

- the `currency(value)` displays the value using the current format for the currency ie, 1000 gets displayed as \$1,000.00, for US format
- the `longdate(date(value))` converts the value to a date and gets the long format to display the date in the column, ie #1/1/2001# displays instead Monday, January 01, 2001
- the `'' + ((0:=proper(value)) left 1) + '' + (:=0 mid 2)` converts the name to proper, so the first letter is capitalized, bolds the first character, and let unchanged the rest, ie a "mihai filimon" gets displayed "**M**ihai Filimon".
- the `value format '2|.3|,'` displays the value using 2 digits, . as decimal separator, grouping by 3 digits using the , as a grouping separator.
- the `type(value) in (0,1) ? 'null' : (dbl(value)<0 ? '<fgcolor=FF0000>' + (value format '2|.3|,') : (dbl(value)>0 ? '<fgcolor=0000FF>' + (value format '2|.3|,') : '0.00'))` displays the positive values in blue, being preceded by + sign, negative values in red

preceded by - sign, 0 as 0.00 while for null values is displays null. The numbers are displayed using 2 digits, . as decimal separator and grouping by 3 digits by , separator.

- the `((1:=int(0:= (value)))) != 0 ? (=:1 + ' day(s)' : ") + (=:1 ? ' ' : ") + ((1:=int(0:=((=:0 - =:1 + 1/24/60/60/2)*24))) != 0 ? =:1 + ' hour(s)' : ") + (=:1 ? ' ' : ") + ((1:=round((=:0 - =:1)*60)) != 0 ? =:1 + ' min(s)' : ")), displays the value in days, hours and minutes`

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 * dpi 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** (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 :

- 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 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 than 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. 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 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

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

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 .

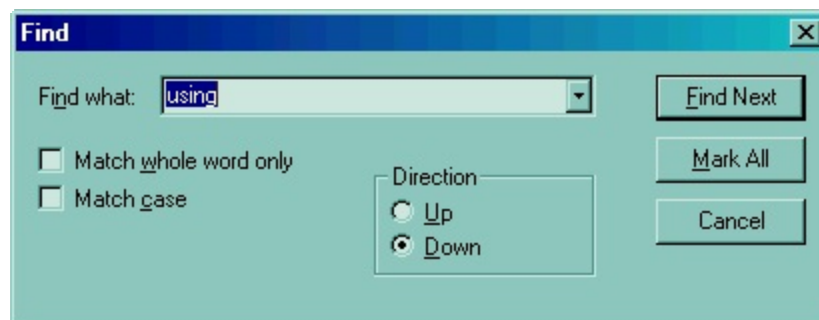
method Expression.Find (FindWhat as String, FindOption as FindOptionEnum)

Finds a string and selects the string if it is found.

Type	Description
FindWhat as String	A string expression that indicates the string being searched.
FindOption as FindOptionEnum	An OR combination of any of the FindOptionEnum values, that indicates how the string is searched.

Use the Find method to find and highlight a string by code. Use the [AllowFind](#) property to let users use the Find dialog implemented by the control. Use the [Caption](#) property to internationalize the Find dialog.

The following screen shot shows the Find dialog:



You can invoke any of the following commands: Delete, Copy, Cut, Find, Replace, FindNext, FindPrev, Paste, Select All, Undo and Redo. Each command has an unique identifier like follows:

```
#define ID_EDIT_REPLACE          0xE129
#define ID_EDIT_DELETE          0xE120
#define ID_EDIT_COPY            0xE122
#define ID_EDIT_CUT             0xE123
#define ID_EDIT_FIND            0xE124
#define ID_EDIT_PASTE           0xE125
#define ID_EDIT_SELECT_ALL      0xE12A
#define ID_EDIT_UNDO            0xE12B
#define ID_EDIT_REDO            0xE12C
#define ID_EDIT_FINDNEXT       0xE12D
#define ID_EDIT_FINDPREV       0xE12E
#define ID_CODE_COMPLETION      0xEA01
```

For instance, the following VB sample displays the Find dialog when user clicks a button:

```
Private Declare Function SendMessage Lib "user32" Alias "SendMessageA" (ByVal hwnd As Long, ByVal wParam As Long, ByVal lParam As Any) As Long
Private Const WM_COMMAND = &H111

Private Sub Command1_Click()
    Expression1.SetFocus
    SendMessage Expression1.hwnd, WM_COMMAND, &HE124 * 65536, 0
End Sub
```

The wParam parameter of SendMessage API function needs to be Command's Identifier * 65536.

The following C++ sample displays the Find dialog when the user clicks a button:

```
void OnButton1()
{
    m_expression.SetFocus();
    m_expression.SendMessage( WM_COMMAND, MAKEWPARAM(0, 0xE124), NULL );
}
```

The following VB.NET displays the Find dialog when user the clicks a button:

```
<System.Runtime.InteropServices.DllImport("user32.dll")> _
Public Shared Function SendMessage(ByVal hWnd As Int32, ByVal wParam As Int32, ByVal lParam As Int32) As Int32

End Function

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
    Handles Button1.Click
        With AxExpression1
            .Focus()
            SendMessage(hWnd, &H111, &HE124 << 16, 0)
        End With
    End Sub
```

where the DllImport imports definition for SendMessage API function in VB.NET.

The following C# displays the Find dialog when user the clicks a button:

```
[System.Runtime.InteropServices.DllImport("user32.dll")]
static extern int SendMessage(int hWnd, int wParam, int lParam);

private void button1_Click(object sender, EventArgs e)
{
    axExpression1.Focus();
    SendMessage(axExpression1.hWnd, 0x111, 0xE124 << 16, 0);
}
```

where the DllImport imports definition for SendMessage API function in C#.

The following VFP displays the Find dialog when user the clicks a button:

```
DECLARE INTEGER SendMessage IN user32;
    INTEGER hWnd;;
    INTEGER Msg;;
    INTEGER wParam;;
    INTEGER lParam

with thisform.Expression1.Object
    SendMessage( .hWnd, 273, BITLSHIFT(57636, 16), 0)
endwith
```

property Expression.FocusPane as Long

Specifies the index of pane that has the focus.

Type	Description
Long	A long expression that indicates the index of pane being focused.

The FocusPane property determines the index of pane that has the focus. Use the [AllowSplitter](#) property to enable splitters in the control. Use the [SplitPaneHeight](#) and [SplitPaneWidth](#) property to add panes in the control. Use the hWndPane property to get the handle of the pane's window.

property Expression.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 [Refresh](#) method to update the control's content. Use the Font property to assign a font to the control in design mode. The Font property adjusts the height of the line too.

The following template changes the font and refresh the control on ObjectStudio (Smalltalk):

```
Font
{
  Name = "Courier New"
  Size = 10
}
Refresh
```

The Refresh method refreshes the height of the line, as the Size of the Font property won't change it.

The following VB sample assigns by code a new font to the control:

```
With Expression1
  With .Font
    .Name = "Tahoma"
  End With
  .Refresh
End With
```

The following C++ sample assigns by code a new font to the control:

```
COleFont font = m_expression.GetFont();
font.SetName( "Tahoma" );
m_expression.Refresh();
```

the C++ sample requires definition of COleFont class (#include "Font.h")

The following VB.NET sample assigns by code a new font to the control:

With AxExpression1

```
Dim font As System.Drawing.Font = New System.Drawing.Font("Tahoma", 10,  
FontStyle.Regular, GraphicsUnit.Point)  
.Font = font  
.CtlRefresh()  
End With
```

The following C# sample assigns by code a new font to the control:

```
System.Drawing.Font font = new System.Drawing.Font("Tahoma", 10, FontStyle.Regular);  
axExpression1.Font = font;  
axExpression1.CtlRefresh();
```

The following VFP sample assigns by code a new font to the control:

```
with thisform.Expression1.Object  
.Font.Name = "Tahoma"  
.Refresh()  
endwith
```

The following Template sample assigns by code a new font to the control:

```
Font  
{  
Name = "Tahoma"  
}
```

property Expression.ForeColor as Color

Retrieves or sets a value that indicates 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 [ForeColorInvalid](#) property to specify a different color to show the control's expression while it is not valid.

property Expression.ForeColorInvalid as Color

Retrieves or sets a value that indicates the control's foreground color, while expression is invalid.

Type	Description
Color	A color expression that specifies the control's foreground color, while the expression is invalid

Use the ForeColorInvalid property to specify a different color to show the control's expression while it is not valid. The ForeColorInvalid property has no effect if it zero. Use the [ForeColor](#) property to specify the color to show the expression while it is valid. The [IsValid](#) property specifies whether the control's expression is valid/syntactically correct. The [Expression/Text](#) property assigns a new expression to the control. The [EvaluationText](#) property specifies the values for variables in the expression to be evaluated (per line). The [EvaluationResult](#) property specifies the result for each evaluation (line). Use the [Evaluate](#) property to get the result of evaluation of the control's expression.

The following samples shows how you can prevent changing the color when the expression is invalid:

VBA (MS Access, Excell...)

```
With Expression1
    .SplitPaneWidth = 196
    .Expression = "value 2"
    .ForeColorInvalid = RGB(0,0,0)
End With
```

VB6

```
With Expression1
    .SplitPaneWidth = 196
    .Expression = "value 2"
    .ForeColorInvalid = RGB(0,0,0)
End With
```

VB.NET

```
With Expression1
    .SplitPaneWidth = 196
    .Expression = "value 2"
```

```
.ForeColorInvalid = Color.FromArgb(0,0,0)
```

End With

VB.NET for /COM

```
With AxExpression1
```

```
.SplitPaneWidth = 196
```

```
.Expression = "value 2"
```

```
.ForeColorInvalid = RGB(0,0,0)
```

```
End With
```

C++

```
/*
```

Copy and paste the following directives to your header file as it defines the namespace 'EXPRESSIONLib' for the library: 'Expression 1.0 Control Library'

```
#import <Expression.dll>
```

```
using namespace EXPRESSIONLib;
```

```
*/
```

```
EXPRESSIONLib::IExpressionPtr spExpression1 = GetDlgItem(IDC_EXPRESSION1)->GetControlUnknown();
```

```
spExpression1->PutSplitPaneWidth(196);
```

```
spExpression1->PutExpression(L"value 2");
```

```
spExpression1->PutForeColorInvalid(RGB(0,0,0));
```

C++ Builder

```
Expression1->SplitPaneWidth = 196;
```

```
Expression1->Expression = L"value 2";
```

```
Expression1->ForeColorInvalid = RGB(0,0,0);
```

C#

```
expression1.SplitPaneWidth = 196;
```

```
expression1.Expression = "value 2";
```

```
expression1.ForeColorInvalid = Color.FromArgb(0,0,0);
```

JScript/JavaScript

```
<BODY onload="Init()">  
<OBJECT CLASSID="clsid:B33F5489-49AC-4155-98E7-9BBFC57FF019"  
id="Expression1"> </OBJECT>  
  
<SCRIPT LANGUAGE="JScript">  
function Init()  
{  
    Expression1.SplitPaneWidth = 196;  
    Expression1.Expression = "value 2";  
    Expression1.ForeColorInvalid = 0;  
}  
</SCRIPT>  
</BODY>
```

VBScript

```
<BODY onload="Init()">  
<OBJECT CLASSID="clsid:B33F5489-49AC-4155-98E7-9BBFC57FF019"  
id="Expression1"> </OBJECT>  
  
<SCRIPT LANGUAGE="VBScript">  
Function Init()  
    With Expression1  
        .SplitPaneWidth = 196  
        .Expression = "value 2"  
        .ForeColorInvalid = RGB(0,0,0)  
    End With  
End Function  
</SCRIPT>  
</BODY>
```

C# for /COM

```
axExpression1.SplitPaneWidth = 196;  
axExpression1.Expression = "value 2";  
axExpression1.ForeColorInvalid = Color.FromArgb(0,0,0);
```

X++ (Dynamics Ax 2009)

```
public void init()  
{  
    ;  
  
    super();  
  
    expression1.SplitPaneWidth(196);  
    expression1.Expression("value 2");  
    expression1.ForeColorInvalid(WinApi::RGB2int(0,0,0));  
}
```

Delphi 8 (.NET only)

```
with AxExpression1 do  
begin  
    SplitPaneWidth := 196;  
    Expression := 'value 2';  
    ForeColorInvalid := Color.FromArgb(0,0,0);  
end
```

Delphi (standard)

```
with Expression1 do  
begin  
    SplitPaneWidth := 196;  
    Expression := 'value 2';  
    ForeColorInvalid := RGB(0,0,0);  
end
```

VFP

```
with thisform.Expression1
    .SplitPaneWidth = 196
    .Expression = "value 2"
    .ForeColorInvalid = RGB(0,0,0)
endwith
```

dBASE Plus

```
local oExpression

oExpression = form.EXPRESSIONACTIVEXCONTROL1.nativeObject
oExpression.SplitPaneWidth = 196
oExpression.Expression = "value 2"
oExpression.ForeColorInvalid = 0x0
```

XBasic (Alpha Five)

```
Dim oExpression as P

oExpression = topparent:CONTROL_ACTIVEX1.activex
oExpression.SplitPaneWidth = 196
oExpression.Expression = "value 2"
oExpression.ForeColorInvalid = 0
```

Visual Objects

```
oDCOCX_Exontrol1.SplitPaneWidth := 196
oDCOCX_Exontrol1.Expression := "value 2"
oDCOCX_Exontrol1.ForeColorInvalid := RGB(0,0,0)
```

PowerBuilder

```
OleObject oExpression

oExpression = ole_1.Object
```

```
oExpression.SplitPaneWidth = 196
oExpression.Expression = "value 2"
oExpression.ForeColorInvalid = RGB(0,0,0)
```

Visual DataFlex

```
Procedure OnCreate
  Forward Send OnCreate
  Set ComSplitPaneWidth to 196
  Set ComExpression to "value 2"
  Set ComForeColorInvalid to (RGB(0,0,0))
End_Procedure
```

XBase++

```
#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
  LOCAL oForm
  LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
  LOCAL oExpression

  oForm := XbpDialog():new( AppDesktop() )
  oForm:drawingArea:clipChildren := .T.
  oForm:create( ,, {100,100}, {640,480},,, .F. )
  oForm:close := {|| PostAppEvent( xbeP_Quit )}

  oExpression := XbpActiveXControl():new( oForm:drawingArea )
  oExpression:CLSID := "Exontrol.Expression.1" /*{B33F5489-49AC-4155-98E7-
9BBFC57FF019}*/
  oExpression:create(,, {10,60},{610,370} )

  oExpression:SplitPaneWidth := 196
  oExpression:Expression := "value 2"
  oExpression:SetProperty("ForeColorInvalid",AutomationTranslateColor(
GraMakeRGBColor ( { 0,0,0 } ) , .F. ))
```

```
oForm:Show()
```

```
DO WHILE nEvent != xbeP_Quit
```

```
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
```

```
    oXbp:handleEvent( nEvent, mp1, mp2 )
```

```
ENDDO
```

```
RETURN
```

method Expression.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 Expression.

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 Expression of 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 Expression.FormatDates as String

Specifies the HTML format that's applied to dates.

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

By default, the FormatDates property is "<fgcolor=000080> </fgcolor>". By default the dates get colored in blue. Use the FormatDates to define the appearance for the dates in the control. If the FormatDates property is empty no format is applied to dates in the control.

You can use any of the following properties to format:

- numbers, [FormatNumbers](#) property, like 1.00
- dates, FormatDates property, like #1/1/2001#
- strings, [FormatStrings](#) property, like "string"
- result, [FormatResult](#) property

The FormatDates supports the following HTML tags:

- ** bold ** bolds a part of the caption.
- **<u> underline </u>** specifies that the portion should appear as underlined.
- **<s> strikethrough </s>** specifies that the portion should appear as strikethrough.
- **<i> italic </i>** specifies that the portion should appear as italic.
- **<fgcolor=FF0000>fgcolor</fgcolor>** changes the foreground color for a portion.
- **<bgcolor=FF0000>bgcolor</bgcolor>** changes the background color for a portion.
- **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.
- **&** glyph characters as **&** (&), **<** (<), **>** (>), **"** ("), **&#number**, For instance, the **€** displays the EUR character, in UNICODE configuration. The **&** ampersand is only recognized as markup when it is followed by a known letter or a # character and a digit. For instance if you want to display **bold** in HTML caption you can use **bold**;

The following sample shows how you can show the dates in bold:

VBA (MS Access, Excell...)

```
.FormatDates = "<b> </b>"  
.Expression = "date(#1/1/2001# +1)"  
End With
```

VB6

```
With Expression1  
.FormatDates = "<b> </b>"  
.Expression = "date(#1/1/2001# +1)"  
End With
```

VB.NET

```
With Expression1  
.FormatDates = "<b> </b>"  
.Expression = "date(#1/1/2001# +1)"  
End With
```

VB.NET for /COM

```
With AxExpression1  
.FormatDates = "<b> </b>"  
.Expression = "date(#1/1/2001# +1)"  
End With
```

C++

```
/*  
Copy and paste the following directives to your header file as  
it defines the namespace 'EXPRESSIONLib' for the library: 'Expression 1.0 Control  
Library'  
  
#import <Expression.dll>  
using namespace EXPRESSIONLib;  
*/  
EXPRESSIONLib::IExpressionPtr spExpression1 = GetDlgItem(IDC_EXPRESSION1)-  
> GetControlUnknown();  
spExpression1->PutFormatDates(L"<b> </b>");  
spExpression1->PutExpression(L"date(#1/1/2001# +1)");
```

C++ Builder

```
Expression1->FormatDates = L"<b> </b>";  
Expression1->Expression = L"date(#1/1/2001# +1)";
```

C#

```
expression1.FormatDates = "<b> </b>";  
expression1.Expression = "date(#1/1/2001# +1)";
```

JScript/JavaScript

```
<BODY onload="Init()">  
<OBJECT CLASSID="clsid:B33F5489-49AC-4155-98E7-9BBFC57FF019"  
id="Expression1"> </OBJECT>  
  
<SCRIPT LANGUAGE="JScript">  
function Init()  
{  
    Expression1.FormatDates = "<b> </b>";  
    Expression1.Expression = "date(#1/1/2001# +1)";  
}  
</SCRIPT>  
</BODY>
```

VBScript

```
<BODY onload="Init()">  
<OBJECT CLASSID="clsid:B33F5489-49AC-4155-98E7-9BBFC57FF019"  
id="Expression1"> </OBJECT>  
  
<SCRIPT LANGUAGE="VBScript">  
Function Init()  
    With Expression1
```

```
.FormatDates = "<b> </b>"  
.Expression = "date(#1/1/2001# +1)"  
End With  
End Function  
</SCRIPT>  
</BODY>
```

C# for /COM

```
axExpression1.FormatDates = "<b> </b>";  
axExpression1.Expression = "date(#1/1/2001# +1)";
```

X++ (Dynamics Ax 2009)

```
public void init()  
{  
    ;  
  
    super();  
  
    expression1.FormatDates("<b> </b>");  
    expression1.Expression("date(#1/1/2001# +1)");  
}
```

Delphi 8 (.NET only)

```
with AxExpression1 do  
begin  
    FormatDates := '<b> </b>';  
    Expression := 'date(#1/1/2001# +1)';  
end
```

Delphi (standard)

```
with Expression1 do  
begin  
    FormatDates := '<b> </b>';
```

```
Expression := 'date(#1/1/2001# +1)';  
end
```

VFP

```
with thisform.Expression1  
  .FormatDates = "<b> </b>"  
  .Expression = "date(#1/1/2001# +1)"  
endwith
```

dBASE Plus

```
local oExpression  
  
oExpression = form.EXPRESSIONACTIVEXCONTROL1.nativeObject  
oExpression.FormatDates = "<b> </b>"  
oExpression.Expression = "date(#1/1/2001# +1)"
```

XBasic (Alpha Five)

```
Dim oExpression as P  
  
oExpression = topparent:CONTROL_ACTIVEX1.activex  
oExpression.FormatDates = "<b> </b>"  
oExpression.Expression = "date(#1/1/2001# +1)"
```

Visual Objects

```
oDCOCX_Exontrol1.FormatDates := "<b> </b>"  
oDCOCX_Exontrol1.Expression := "date(#1/1/2001# +1)"
```

PowerBuilder

```
OleObject oExpression
```

```
oExpression = ole_1.Object
oExpression.FormatDates = "<b> </b>"
oExpression.Expression = "date(#1/1/2001# +1)"
```

Visual DataFlex

```
Procedure OnCreate
  Forward Send OnCreate
  Set ComFormatDates to "<b> </b>"
  Set ComExpression to "date(#1/1/2001# +1)"
End_Procedure
```

XBase++

```
#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
  LOCAL oForm
  LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
  LOCAL oExpression

  oForm := XbpDialog():new( AppDesktop() )
  oForm:drawingArea:clipChildren := .T.
  oForm:create( ,, {100,100}, {640,480},,, .F. )
  oForm:close := {|| PostAppEvent( xbeP_Quit )}

  oExpression := XbpActiveXControl():new( oForm:drawingArea )
  oExpression:CLSID := "Exontrol.Expression.1" /*{B33F5489-49AC-4155-98E7-
9BBFC57FF019}*/
  oExpression:create(,, {10,60},{610,370} )

  oExpression:FormatDates := "<b> </b>"
  oExpression:Expression := "date(#1/1/2001# +1)"

  oForm:Show()
  DO WHILE nEvent != xbeP_Quit
```

```
nEvent := AppEvent( @mp1, @mp2, @oXbp )
```

```
oXbp:handleEvent( nEvent, mp1, mp2 )
```

```
ENDDO
```

```
RETURN
```

property Expression.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=000080> </fgcolor>". By default the numbers get colored in blue. 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.

You can use any of the following properties to format:

- numbers, FormatNumbers property, like 1.00
- dates, [FormatDates](#) property, like #1/1/2001#
- strings, [FormatStrings](#) property, like "string"
- result, [FormatResult](#) property

The FormatNumbers supports the following HTML tags:

- ** bold ** bolds a part of the caption.
- **<u> underline </u>** specifies that the portion should appear as underlined.
- **<s> strikethrough </s>** specifies that the portion should appear as strikethrough.
- **<i> italic </i>** specifies that the portion should appear as italic.
- **<fgcolor=FF0000>fgcolor</fgcolor>** changes the foreground color for a portion.
- **<bgcolor=FF0000>bgcolor</bgcolor>** changes the background color for a portion.
- **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.
- **&** glyph characters as **&** (&), **<** (<), **>** (>), **"** ("), **&#number**, For instance, the **€** displays the EUR character, in UNICODE configuration. The **&** ampersand is only recognized as markup when it is followed by a known letter or a # character and a digit. For instance if you want to display **bold** in HTML caption you can use **bold**;

The following sample shows how you can show the numbers in bold:

VBA (MS Access, Excell...)

```
.FormatNumbers = "<b> </b> "  
.Expression = "value + 100"  
End With
```

VB6

```
With Expression1  
.FormatNumbers = "<b> </b> "  
.Expression = "value + 100"  
End With
```

VB.NET

```
With Expression1  
.FormatNumbers = "<b> </b> "  
.Expression = "value + 100"  
End With
```

VB.NET for /COM

```
With AxExpression1  
.FormatNumbers = "<b> </b> "  
.Expression = "value + 100"  
End With
```

C++

```
/*  
Copy and paste the following directives to your header file as  
it defines the namespace 'EXPRESSIONLib' for the library: 'Expression 1.0 Control  
Library'  
  
#import <Expression.dll>  
using namespace EXPRESSIONLib;  
*/  
EXPRESSIONLib::IExpressionPtr spExpression1 = GetDlgItem(IDC_EXPRESSION1)-  
> GetControlUnknown();  
spExpression1->PutFormatNumbers(L"<b> </b>");  
spExpression1->PutExpression(L"value + 100");
```

C++ Builder

```
Expression1->FormatNumbers = L"<b> </b>";  
Expression1->Expression = L"value + 100";
```

C#

```
expression1.FormatNumbers = "<b> </b>";  
expression1.Expression = "value + 100";
```

JScript/JavaScript

```
<BODY onload="Init()">  
<OBJECT CLASSID="clsid:B33F5489-49AC-4155-98E7-9BBFC57FF019"  
id="Expression1"> </OBJECT>  
  
<SCRIPT LANGUAGE="JScript">  
function Init()  
{  
    Expression1.FormatNumbers = "<b> </b>";  
    Expression1.Expression = "value + 100";  
}  
</SCRIPT>  
</BODY>
```

VBScript

```
<BODY onload="Init()">  
<OBJECT CLASSID="clsid:B33F5489-49AC-4155-98E7-9BBFC57FF019"  
id="Expression1"> </OBJECT>  
  
<SCRIPT LANGUAGE="VBScript">  
Function Init()  
    With Expression1
```

```
.FormatNumbers = "<b> </b>"  
.Expression = "value + 100"  
End With  
End Function  
</SCRIPT>  
</BODY>
```

C# for /COM

```
axExpression1.FormatNumbers = "<b> </b>";  
axExpression1.Expression = "value + 100";
```

X++ (Dynamics Ax 2009)

```
public void init()  
{  
    ;  
  
    super();  
  
    expression1.FormatNumbers("<b> </b>");  
    expression1.Expression("value + 100");  
}
```

Delphi 8 (.NET only)

```
with AxExpression1 do  
begin  
    FormatNumbers := '<b> </b>';  
    Expression := 'value + 100';  
end
```

Delphi (standard)

```
with Expression1 do  
begin  
    FormatNumbers := '<b> </b>';
```

```
Expression := 'value + 100';  
end
```

VFP

```
with thisform.Expression1  
  .FormatNumbers = "<b> </b>"  
  .Expression = "value + 100"  
endwith
```

dBASE Plus

```
local oExpression  
  
oExpression = form.EXPRESSIONACTIVEXCONTROL1.nativeObject  
oExpression.FormatNumbers = "<b> </b>"  
oExpression.Expression = "value + 100"
```

XBasic (Alpha Five)

```
Dim oExpression as P  
  
oExpression = topparent:CONTROL_ACTIVEX1.activex  
oExpression.FormatNumbers = "<b> </b>"  
oExpression.Expression = "value + 100"
```

Visual Objects

```
oDCOCX_Exontrol1:FormatNumbers := "<b> </b>"  
oDCOCX_Exontrol1:Expression := "value + 100"
```

PowerBuilder

```
OleObject oExpression
```

```
oExpression = ole_1.Object
oExpression.FormatNumbers = "<b> </b>"
oExpression.Expression = "value + 100"
```

Visual DataFlex

```
Procedure OnCreate
  Forward Send OnCreate
  Set ComFormatNumbers to "<b> </b>"
  Set ComExpression to "value + 100"
End_Procedure
```

XBase++

```
#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
  LOCAL oForm
  LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
  LOCAL oExpression

  oForm := XbpDialog():new( AppDesktop() )
  oForm:drawingArea:clipChildren := .T.
  oForm:create( ,, {100,100}, {640,480},,, .F. )
  oForm:close := {|| PostAppEvent( xbeP_Quit )}

  oExpression := XbpActiveXControl():new( oForm:drawingArea )
  oExpression:CLSID := "Exontrol.Expression.1" /*{B33F5489-49AC-4155-98E7-
9BBFC57FF019}*/
  oExpression:create(,, {10,60},{610,370} )

  oExpression:FormatNumbers := "<b> </b>"
  oExpression:Expression := "value + 100"

  oForm:Show()
  DO WHILE nEvent != xbeP_Quit
```

```
nEvent := AppEvent( @mp1, @mp2, @oXbp )
```

```
oXbp:handleEvent( nEvent, mp1, mp2 )
```

```
ENDDO
```

```
RETURN
```

property Expression.FormatResult as String

Specifies the HTML format to show the result.

Type	Description
String	A string expression that defines the HTML format to display the result.

By default, the FormatResult property is "<fgcolor=000080> </fgcolor>". By default the numbers get colored in blue. 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.

You can use any of the following properties to format:

- numbers, [FormatNumbers](#) property, like 1.00
- dates, [FormatDates](#) property, like #1/1/2001#
- strings, [FormatStrings](#) property, like "string"
- result, FormatResult property

The FormatNumbers supports the following HTML tags:

- ** bold ** bolds a part of the caption.
- **<u> underline </u>** specifies that the portion should appear as underlined.
- **<s> strikethrough </s>** specifies that the portion should appear as strikethrough.
- **<i> italic </i>** specifies that the portion should appear as italic.
- **<fgcolor=FF0000>fgcolor</fgcolor>** changes the foreground color for a portion.
- **<bgcolor=FF0000>bgcolor</bgcolor>** changes the background color for a portion.
- **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.
- **&** glyph characters as **&** (&), **<** (<), **>** (>), **"** ("), **&#number**, For instance, the **€** displays the EUR character, in UNICODE configuration. The **&** ampersand is only recognized as markup when it is followed by a known letter or a # character and a digit. For instance if you want to display **bold** in HTML caption you can use **bold**;

The following sample shows how you can show the result in bold:

VBA (MS Access, Excell...)

```
.FormatResult = "<b> </b> "  
.Expression = "currency(value)"  
.AllowSplitter = 2  
.SplitPaneHeight = 196  
End With
```

VB6

```
With Expression1  
.FormatResult = "<b> </b> "  
.Expression = "currency(value)"  
.AllowSplitter = exVSplitter  
.SplitPaneHeight = 196  
End With
```

VB.NET

```
With Expression1  
.FormatResult = "<b> </b> "  
.Expression = "currency(value)"  
.AllowSplitter = exontrol.EXPRESSIONLib.SplitterEnum.exVSplitter  
.SplitPaneHeight = 196  
End With
```

VB.NET for /COM

```
With AxExpression1  
.FormatResult = "<b> </b> "  
.Expression = "currency(value)"  
.AllowSplitter = EXPRESSIONLib.SplitterEnum.exVSplitter  
.SplitPaneHeight = 196  
End With
```

C++

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

```
#import <Expression.dll>
using namespace EXPRESSIONLib;
*/
EXPRESSIONLib::IExpressionPtr spExpression1 = GetDlgItem(IDC_EXPRESSION1)-
> GetControlUnknown();
spExpression1->PutFormatResult(L"<b> </b>");
spExpression1->PutExpression(L"currency(value)");
spExpression1->PutAllowSplitter(EXPRESSIONLib::exVSplitter);
spExpression1->PutSplitPaneHeight(196);
```

C++ Builder

```
Expression1->FormatResult = L"<b> </b>";
Expression1->Expression = L"currency(value)";
Expression1->AllowSplitter = Expressionlib_tlb::SplitterEnum::exVSplitter;
Expression1->SplitPaneHeight = 196;
```

C#

```
expression1.FormatResult = "<b> </b>";
expression1.Expression = "currency(value)";
expression1.AllowSplitter = exontrol.EXPRESSIONLib.SplitterEnum.exVSplitter;
expression1.SplitPaneHeight = 196;
```

JScript/JavaScript

```
<BODY onload="Init()">
<OBJECT CLASSID="clsid:B33F5489-49AC-4155-98E7-9BBFC57FF019"
id="Expression1"> </OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
    Expression1.FormatResult = "<b> </b>";
```

```
Expression1.Expression = "currency(value)";
Expression1.AllowSplitter = 2;
Expression1.SplitPaneHeight = 196;
}
</SCRIPT>
</BODY>
```

VBScript

```
<BODY onload="Init()">
<OBJECT CLASSID="clsid:B33F5489-49AC-4155-98E7-9BBFC57FF019"
id="Expression1"></OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
  With Expression1
    .FormatResult = "<b> </b>"
    .Expression = "currency(value)"
    .AllowSplitter = 2
    .SplitPaneHeight = 196
  End With
End Function
</SCRIPT>
</BODY>
```

C# for /COM

```
axExpression1.FormatResult = "<b> </b>";
axExpression1.Expression = "currency(value)";
axExpression1.AllowSplitter = EXPRESSIONLib.SplitterEnum.exVSplitter;
axExpression1.SplitPaneHeight = 196;
```

X++ (Dynamics Ax 2009)

```
public void init()
```

```
{
;

super();

expression1.FormatResult("<b> </b>");
expression1.Expression("currency(value)");
expression1.AllowSplitter(2/*exVSplitter*/);
expression1.SplitPaneHeight(196);
}
```

Delphi 8 (.NET only)

```
with AxExpression1 do
begin
  FormatResult := '<b> </b>';
  Expression := 'currency(value)';
  AllowSplitter := EXPRESSIONLib.SplitterEnum.exVSplitter;
  SplitPaneHeight := 196;
end
```

Delphi (standard)

```
with Expression1 do
begin
  FormatResult := '<b> </b>';
  Expression := 'currency(value)';
  AllowSplitter := EXPRESSIONLib_TLB.exVSplitter;
  SplitPaneHeight := 196;
end
```

VFP

```
with thisform.Expression1
.FormatResult = "<b> </b>"
.Expression = "currency(value)"
.AllowSplitter = 2
.SplitPaneHeight = 196
```

endwith

dBASE Plus

local oExpression

oExpression = form.EXPRESSIONACTIVEXCONTROL1.nativeObject

oExpression.FormatResult = " "

oExpression.Expression = "currency(value)"

oExpression.AllowSplitter = 2

oExpression.SplitPaneHeight = 196

XBasic (Alpha Five)

Dim oExpression as P

oExpression = topparent:CONTROL_ACTIVEX1.activex

oExpression.FormatResult = " "

oExpression.Expression = "currency(value)"

oExpression.AllowSplitter = 2

oExpression.SplitPaneHeight = 196

Visual Objects

oDCOCX_Exontrol1.FormatResult := " "

oDCOCX_Exontrol1.Expression := "currency(value)"

oDCOCX_Exontrol1.AllowSplitter := exVSplitter

oDCOCX_Exontrol1.SplitPaneHeight := 196

PowerBuilder

OleObject oExpression

oExpression = ole_1.Object

oExpression.FormatResult = " "

```
oExpression.Expression = "currency(value)"
oExpression.AllowSplitter = 2
oExpression.SplitPaneHeight = 196
```

Visual DataFlex

```
Procedure OnCreate
  Forward Send OnCreate
  Set ComFormatResult to "<b> </b>"
  Set ComExpression to "currency(value)"
  Set ComAllowSplitter to OLEexVSplitter
  Set ComSplitPaneHeight to 196
End_Procedure
```

XBase++

```
#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
  LOCAL oForm
  LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
  LOCAL oExpression

  oForm := XbpDialog():new( AppDesktop() )
  oForm:drawingArea:clipChildren := .T.
  oForm:create( ,, {100,100}, {640,480},,, .F. )
  oForm:close := {|| PostAppEvent( xbeP_Quit )}

  oExpression := XbpActiveXControl():new( oForm:drawingArea )
  oExpression:CLSID := "Exontrol.Expression.1" /*{B33F5489-49AC-4155-98E7-
9BBFC57FF019}*/
  oExpression:create(,, {10,60},{610,370} )

  oExpression:FormatResult := "<b> </b>"
  oExpression:Expression := "currency(value)"
  oExpression:AllowSplitter := 2/*exVSplitter*/
```

```
oExpression:SplitPaneHeight := 196
```

```
oForm:Show()
```

```
DO WHILE nEvent != xbeP_Quit
```

```
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
```

```
    oXbp:handleEvent( nEvent, mp1, mp2 )
```

```
ENDDO
```

```
RETURN
```

property Expression.FormatStrings as String

Specifies the HTML format that's applied to strings.

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

By default, the FormatStrings property is "<fgcolor=800000> </fgcolor>". By default the strings get colored in red. Use the FormatStrings to define the appearance for the strings in the control. If the FormatStrings property is empty no format is applied to strings in the control.

You can use any of the following properties to format:

- numbers, [FormatNumbers](#) property, like 1.00
- dates, [FormatDates](#) property, like #1/1/2001#
- strings, FormatStrings property, like "string"
- result, [FormatResult](#) property

The FormatStrings supports the following HTML tags:

- ** bold ** bolds a part of the caption.
- **<u> underline </u>** specifies that the portion should appear as underlined.
- **<s> strikeout </s>** specifies that the portion should appear as strikeout.
- **<i> italic </i>** specifies that the portion should appear as italic.
- **<fgcolor=FF0000>fgcolor</fgcolor>** changes the foreground color for a portion.
- **<bgcolor=FF0000>bgcolor</bgcolor>** changes the background color for a portion.
- **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.
- **&** glyph characters as **&** (&), **<** (<), **>** (>), **&qout** ("), **&#number**, For instance, the **€** displays the EUR character, in UNICODE configuration. The **&** ampersand is only recognized as markup when it is followed by a known letter or a # character and a digit. For instance if you want to display **bold** in HTML caption you can use **bold**;

The following sample shows how you can show the strings in bold:

VBA (MS Access, Excell...)

```
.FormatStrings = "<b> </b> "  
.Expression = "value + ""aka""  
End With
```

VB6

```
With Expression1  
.FormatStrings = "<b> </b> "  
.Expression = "value + ""aka""  
End With
```

VB.NET

```
With Expression1  
.FormatStrings = "<b> </b> "  
.Expression = "value + ""aka""  
End With
```

VB.NET for /COM

```
With AxExpression1  
.FormatStrings = "<b> </b> "  
.Expression = "value + ""aka""  
End With
```

C++

```
/*  
Copy and paste the following directives to your header file as  
it defines the namespace 'EXPRESSIONLib' for the library: 'Expression 1.0 Control  
Library'  
  
#import <Expression.dll>  
using namespace EXPRESSIONLib;  
*/  
EXPRESSIONLib::IExpressionPtr spExpression1 = GetDlgItem(IDC_EXPRESSION1)-  
> GetControlUnknown();  
spExpression1->PutFormatStrings(L"<b> </b> ");  
spExpression1->PutExpression(L"value + \\"aka\\");
```

C++ Builder

```
Expression1->FormatStrings = L"<b> </b>";  
Expression1->Expression = L"value + \"aka\"";
```

C#

```
expression1.FormatStrings = "<b> </b>";  
expression1.Expression = "value + \"aka\"";
```

JScript/JavaScript

```
<BODY onload="Init()">  
<OBJECT CLASSID="clsid:B33F5489-49AC-4155-98E7-9BBFC57FF019"  
id="Expression1"> </OBJECT>  
  
<SCRIPT LANGUAGE="JScript">  
function Init()  
{  
    Expression1.FormatStrings = "<b> </b>";  
    Expression1.Expression = "value + \"aka\"";  
}  
</SCRIPT>  
</BODY>
```

VBScript

```
<BODY onload="Init()">  
<OBJECT CLASSID="clsid:B33F5489-49AC-4155-98E7-9BBFC57FF019"  
id="Expression1"> </OBJECT>  
  
<SCRIPT LANGUAGE="VBScript">  
Function Init()  
    With Expression1
```

```
.FormatStrings = "<b> </b> "  
.Expression = "value + ""aka""  
End With  
End Function  
</SCRIPT>  
</BODY>
```

C# for /COM

```
axExpression1.FormatStrings = "<b> </b>";  
axExpression1.Expression = "value + \"aka\"";
```

X++ (Dynamics Ax 2009)

```
public void init()  
{  
;  
  
super();  
  
expression1.FormatStrings("<b> </b>");  
expression1.Expression("value + \"aka\"");  
}
```

Delphi 8 (.NET only)

```
with AxExpression1 do  
begin  
FormatStrings := '<b> </b>';  
Expression := 'value + "aka";  
end
```

Delphi (standard)

```
with Expression1 do  
begin  
FormatStrings := '<b> </b>';
```

```
Expression := 'value + "aka";  
end
```

VFP

```
with thisform.Expression1  
.FormatStrings = "<b> </b>"  
.Expression = "value + " + chr(34) + "aka" + chr(34) + ""  
endwith
```

dBASE Plus

```
local oExpression  
  
oExpression = form.EXPRESSIONACTIVEXCONTROL1.nativeObject  
oExpression.FormatStrings = "<b> </b>"  
oExpression.Expression = "value + " + ["] + "aka" + ["] + ""
```

XBasic (Alpha Five)

```
Dim oExpression as P  
  
oExpression = topparent:CONTROL_ACTIVEX1.activex  
oExpression.FormatStrings = "<b> </b>"  
oExpression.Expression = "value + \"aka\""
```

Visual Objects

```
oDCOCX_Exontrol1.FormatStrings := "<b> </b>"  
oDCOCX_Exontrol1.Expression := "value + " + CHR(34) + "aka" + CHR(34) + ""
```

PowerBuilder

```
OleObject oExpression
```

```
oExpression = ole_1.Object
oExpression.FormatStrings = "<b> </b>"
oExpression.Expression = "value + " + CHAR(34) + "aka" + CHAR(34) + ""
```

Visual DataFlex

```
Procedure OnCreate
  Forward Send OnCreate
  Set ComFormatStrings to "<b> </b>"
  Set ComExpression to "value + "aka""
End_Procedure
```

XBase++

```
#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
  LOCAL oForm
  LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
  LOCAL oExpression

  oForm := XbpDialog():new( AppDesktop() )
  oForm:drawingArea:clipChildren := .T.
  oForm:create( ,, {100,100}, {640,480},,, .F. )
  oForm:close := {|| PostAppEvent( xbeP_Quit )}

  oExpression := XbpActiveXControl():new( oForm:drawingArea )
  oExpression:CLSID := "Exontrol.Expression.1" /*{B33F5489-49AC-4155-98E7-
9BBFC57FF019}*/
  oExpression:create(,, {10,60},{610,370} )

  oExpression:FormatStrings := "<b> </b>"
  oExpression:Expression := "value + " + CHR(34) + "aka" + CHR(34) + ""

  oForm:Show()
  DO WHILE nEvent != xbeP_Quit
```

```
nEvent := AppEvent( @mp1, @mp2, @oXbp )
```

```
oXbp:handleEvent( nEvent, mp1, mp2 )
```

```
ENDDO
```

```
RETURN
```

property Expression.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.

The property has no effect if the [DisplaySelection](#) property is False. Use the HideSelection property to hide the selection when control loses the focus. Use the [SelfForeColor](#) and [SelBackColor](#) properties to define the colors used to paint the selection. Use the [SelStart](#), [SelLength](#) and [SelText](#) properties to access the selection. The [SelfForeColorHide](#) property specifies the foreground color of the current selection when the control has no focus, and the [HideSelection](#) property is False. The [SelBackColorHide](#) property specifies the background color of the current selection when the control has no focus, and the [HideSelection](#) property is False.

property Expression.hWnd as Long

Gets the window's 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). Use the [hWndPane](#) property to retrieve the handle of the pane when control includes splitters.

property Expression.hWndPane (Pane as Long) as Long

Gets the window handle of the control's pane.

Type	Description
Pane as Long	A long expression that indicates the index of the pane being queried. The Pane parameter must be greater or equal with 0 and less than 4. The control supports maximum 4 panes.
Long	A long expression that

The Microsoft Windows operating environment identifies each form and control in an application by assigning it a handle, or hWnd. The hWnd property is used with Windows API calls. Many Windows operating environment functions require the hWnd of the active window as an argument. Use the hWndPane property to retrieve the handle of the pane when control displays splitters. Use the [FocusPane](#) property to get the index of the pane that has the focus. Use the [AllowSplitter](#) property to enable splitters in the control.

method Expression.Images (Handle as Variant)

Sets the control's images list at runtime. The Handle should be a handle to an Image List control.

Type

Description

The Handle parameter can be:

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

Handle as Variant

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

The control provides an image list window, that's displayed at design time. Use the [ShowImageList](#) property to hide the image list window, at design time. At design time, the user can add new icons to the control's Images collection, by dragging icon files, exe files, etc, to the images list window. At runtime, the user can use the Images and [Replacelcon](#) method to change the Images collection. The Images collection is 1 based. The control's context list may contain icons.

The following screen shot shows the control's images panel, where user can drag icon files in order to add new icons to the control at design mode:



property Expression.ImageSize as Long

Specifies the size of the control' icons.

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

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

property Expression.IncrementalSearchError as Color

Retrieves or sets a value that specifies the color to show the 'Incremental Search' field when the typing string is not found.

Type	Description
Color	A Color expression to show the incremental search field when it is not found (error).

By default, the IncrementalSearchError property is 0. The IncrementalSearchError property has no effect if the IncrementalSearchError property is the same as BackColor property. The IncrementalSearchError property retrieves or sets a value that specifies the color to show the 'Incremental Search' field when the typing string is not found. The [AllowIncrementalSearch](#) property specifies whether the control supports incremental searching.

property Expression.IndentOnTab as Boolean

Specifies whether the multiple lines selection is indented when user presses the TAB key.

Type	Description
Boolean	A boolean expression that indicates whether the selection (multiple lines) is indented when user presses the TAB key.

By default, the IndentOnTab property is True. Use the IndentOnTab property to disable indentation when user presses the TAB key. Use the [UseTabKey](#) property to specify whether the control handles the TAB key.

property Expression.IsValid as Boolean

Specifies whether the expression is valid (syntactically correct).

Type	Description
Boolean	A Boolean expression that specifies whether the control's expression is valid/syntactically correct.

The IsValid property indicates whether the control's expression is valid/syntactically correct. The [Expression/Text](#) property assigns a new expression to the control. The [EvaluationText](#) property specifies the values for variables in the expression to be evaluated (per line). The [EvaluationResult](#) property specifies the result for each evaluation (line). Use the [Evaluate](#) property to get the result of evaluation of the control's expression. Use the [ForeColorInvalid](#) property to specify a different color to show the control's expression while it is not valid.

property Expression.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.

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** (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 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 than 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 Expression.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 scrollbars are enabled, and the user can select or moves the caret. Use the [Enabled](#) property to disable the control. If the control is disabled, the scrollbars are disabled, and the user can't select or move the caret. Use the [ShowCaret](#) property to hide the control's caret. If the Locked property is True, the [UseTabKey](#) property is automatically False.

property Expression.MultiLine as Boolean

Specifies whether the control accepts multiple lines.

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

Use the MultiLine property to allow multiple lines in the control's text. By default, the MultiLine property is True. If the MultiLine property is False, the control contains a single line. Use the [Text](#) property to access the control's text.

method **Expression.OLEDrag ()**

Causes a component to initiate an OLE drag/drop operation.

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

Only for internal use.

property Expression.OLEDropMode as exOLEDropModeEnum

Returns or sets how a target component handles drop operations

Type	Description
exOLEDropModeEnum	An exOLEDropModeEnum expression that indicates the OLE Drag and Drop mode.

The ExExpression control supports manual or automatic OLE Drag and Drop operation. The control fires the [OLEStartDrag](#) when the user starts OLE drag and drop operation.

The following VB sample puts the selected text to the clipboard when the drag and drop operation starts (The OLEDropMode property is exOLEDropAutomatic):

```
With Expression1
    .OLEDropMode = exOLEDropManual
End With
```

```
Private Sub Expression1_OLEStartDrag(ByVal Data As EXPRESSIONLibCtl.IExDataObject,
AllowedEffects As Long)
    Dim s As String
    s = Expression1.SelText
    If (Len(s) > 0) Then
        AllowedEffects = 1
        Data.SetData s, 1
    End If
End Sub
```

The following C++ sample puts the selected text to the clipboard when the drag and drop operation starts:

```
m_expression.SetOLEDropMode( 1 );
```

```
void OnOLEStartDragExpression1(LPDISPATCH Data, long FAR* AllowedEffects)
{
    CString s( m_expression.GetSelText() );
    if ( s.GetLength() > 0 )
    {
        *AllowedEffects = 1; /*exOLEDropEffectCopy*/
        if ( EXPRESSIONLib::IExDataObjectPtr spData( Data ) )
```

```
spData->SetData( COleVariant( s ), COleVariant( long(EXPRESSIONLib::exCFText) ) );
```

```
}  
}
```

The C++ sample uses the `#import <expression.dll>` statement to include definitions for the `IExDataObject` and `IExDataObjectFiles` objects.

The following VB.NET sample puts the selected text to the clipboard when the drag and drop operation starts:

```
With AxExpression1  
    .OLEDropMode = EXPRESSIONLib.exOLEDropModeEnum.exOLEDropManual  
End With
```

```
Private Sub AxExpression1_OLEStartDrag(ByVal sender As Object, ByVal e As  
AxEXPRESSIONLib._IExpressionEvents_OLEStartDragEvent) Handles  
AxExpression1.OLEStartDrag  
    Dim s As String = AxExpression1.SelText  
    If (Len(s) > 0) Then  
        e.allowedEffects = 1  
        e.data.SetData(s, 1)  
    End If  
End Sub
```

The following C# sample puts the selected text to the clipboard when the drag and drop operation starts:

```
axExpression1.OLEDropMode =  
EXPRESSIONLib.exOLEDropModeEnum.exOLEDropManual;
```

```
private void axExpression1_OLEStartDrag(object sender,  
AxEXPRESSIONLib._IExpressionEvents_OLEStartDragEvent e)  
{  
    string s = axExpression1.SelText;  
    if (s.Length > 0)  
    {  
        e.allowedEffects = 1;  
        e.data.SetData(s, EXPRESSIONLib.exClipboardFormatEnum.exCFText );  
    }  
}
```

```
}  
}
```

The following VFP sample puts the selected text to the clipboard when the drag and drop operation starts:

```
with thisform.Expression1  
    .OLEDropMode = 1  
endwith
```

```
*** ActiveX Control Event ***  
LPARAMETERS data, allowedeffects  
  
with thisform.Expression1  
    local s  
    s = .SelText  
    if ( len(s) > 0 ) then  
        allowedeffects = 1  
        data.SetData( s, 1 )  
    endif  
endwith
```

property Expression.Overtyping as Boolean

Specifies whether the control is running in overtyping mode.

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

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

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

```
Private Sub Expression1_KeyDown(KeyCode As Integer, Shift As Integer)
    If (KeyCode = vbKeyInsert) Then
        KeyCode = 0
    End If
End Sub
```

property Expression.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 Expression.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.

property Expression.RClick as Boolean

Specifies whether the control's cursor is placed at the mouse position when user right clicks the control.

Type	Description
Boolean	A boolean expression that indicates whether the cursor's position is changed when user right clicks the control.

By default, the RClick property is False. Use the RClick property to change the position of the cursor when the user right clicks the control. By default, the user changes the cursor position when left click the control. Use the [AllowContextMenu](#) property to specify whether the control displays a context menu when user does a right click.

method `Expression.Redo ()`

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

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

Use the `Redo` method to redo the last expression-control operation. Use the [CanRedo](#) property to specify whether an redo operation is available. Use the [Undo](#) method to undo the last expression-control operation. Use the [AllowContextMenu](#) property to specify whether the control displays a context menu when user right clicks the control.

method Expression.Refresh ()

Refreshes the control.

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

The Refresh method repaints the control.

The following VB sample updates the control:

```
Expression1.Refresh
```

The following C++ sample updates the control:

```
m_expression.Refresh();
```

The following VB.NET sample updates the control:

```
AxExpression1.CtlRefresh()
```

The following C# sample updates the control:

```
axExpression1.CtlRefresh();
```

The following VFP sample updates the control:

```
With thisform.Expression1  
    .Object.Refresh  
EndWith
```

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

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

Type	Description
Icon as Variant	<p>A Variant expression that specifies the icon to add or insert, as one of the following options:</p> <ul style="list-style-type: none">• a long expression that specifies the handle of the icon (HICON)• a string expression that indicates the path to the picture file• a string expression that defines the picture's content encoded as BASE64 strings using the eXImages tool• a Picture reference, which is an object that holds image data. It is often used in controls like PictureBox, Image, or in custom controls (e.g., IPicture, IPictureDisp) <p>If the Icon parameter is 0, it specifies that the icon at the given Index is removed. Furthermore, setting the Index parameter to -1 removes all icons.</p> <p>By default, if the Icon parameter is not specified or is missing, a value of 0 is used.</p>
Index as Variant	<p>A long expression that defines the index of the icon to insert or remove, as follows:</p> <ul style="list-style-type: none">• A zero or positive value specifies the index of the icon to insert (when Icon is non-zero) or to remove (when the Icon parameter is zero)• A negative value clears all icons when the Icon parameter is zero <p>By default, if the Index parameter is not specified or is missing, a value of -1 is used.</p>
Return	Description
Long	A long expression that indicates the index of the icon in the images collection.

Use the Replacelcon property to add, remove or replace an icon in the control's images

collection. Also, the `Replacelcon` property can clear the images collection. Use the [Images](#) method to attach an image list to the control.

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

`i = Expression1.Replacelcon(LoadPicture("d:\icons\help.ico").Handle)`, where `i` is the index to insert the icon

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

`i = Expression1.Replacelcon(LoadPicture("d:\icons\help.ico").Handle, 0)`, in this case the `i` is zero, because the first icon was replaced.

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

`Expression1.Replacelcon 0, i`, in this case the `i` must be the index of the icon that follows to be removed

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

`Expression1.Replacelcon 0, -1`

property Expression.ScrollBars as ScrollBarsEnum

Specifies the type of scroll bars that control has.

Type	Description
ScrollBarsEnum	A ScrollBarsEnum expression that indicates the control's scroll bars.

Use the ScrollBars property to specifies the scrollbars used by the control. By default, the control's ScrollBars property is exBoth. The control displays the scroll bars only when they are required. Use the [Font](#) property to specify the control's font. The Font property indicates the height of the line too. For instance, if the control has no lines inside, the control displays no scroll bars, even if the property ScrollBars is exBoth.

property Expression.ScrollButtonHeight as Long

Specifies the height of the button in the vertical scrollbar.

Type	Description
Long	A long expression that defines the height of the button in the vertical scroll bar.

By default, the ScrollButtonHeight property is -1. If the ScrollButtonHeight property is -1, the control uses the default height (from the system) for the buttons in the vertical scroll bar. Use the [ScrollButtonWidth](#) property to specify the width of the buttons in the horizontal scroll bar. Use the [ScrollWidth](#) property to specify the width of the vertical scroll bar. Use the [ScrollBars](#) property to specify which scroll bar is visible or hidden in the control. Use the [ScrollHeight](#) property to specify the height of the horizontal scroll bar. Use the [ScrollPartVisible](#) property to specify the visible parts in the control's scroll bar. Use the [ScrollThumbSize](#) property to define a fixed size for the scrollbar's thumb

property Expression.ScrollButtonWidth as Long

Specifies the width of the button in the horizontal scrollbar.

Type	Description
Long	A long expression that defines the width of the button in the horizontal scroll bar.

By default, the ScrollButtonWidth property is -1. If the ScrollButtonWidth property is -1, the control uses the default width (from the system) for the buttons in the horizontal scroll bar. Use the [ScrollButtonHeight](#) property to specify the height of the buttons in the vertical scroll bar. Use the [ScrollWidth](#) property to specify the width of the vertical scroll bar. Use the [ScrollBars](#) property to specify which scroll bar is visible or hidden in the control. Use the [ScrollHeight](#) property to specify the height of the horizontal scroll bar. Use the [ScrollPartVisible](#) property to specify the visible parts in the control's scroll bar. Use the [ScrollThumbSize](#) property to define a fixed size for the scrollbar's thumb.

property Expression.ScrollFont (ScrollBar as ScrollBarEnum) as IFontDisp

Retrieves or sets the scrollbar's font.

Type	Description
ScrollBar as ScrollBarEnum	A ScrollBarEnum expression that indicates the vertical or the horizontal scroll bar.
IFontDisp	A Font object

Use the ScrollFont property to specify the font in the control's scroll bar. Use the [ScrollPartCaption](#) property to specify the caption of the scroll's part. Use the [ScrollPartVisible](#) property to add or remove buttons/parts in the control's scrollbar. By default, when a part becomes visible, the [ScrollPartEnable](#) property is automatically called, so the parts becomes enabled. Use the [ScrollPartEnable](#) property to specify enable or disable parts in the control's scrollbar. Use the [ScrollBars](#) property to specify the visible scrollbars in the control. The control fires the [ScrollButtonClick](#) event when the user clicks a part of the scroll bar.

property Expression.ScrollHeight as Long

Specifies the height of the horizontal scrollbar.

Type	Description
Long	A long expression that defines the height of the horizontal scroll bar.

By default, the ScrollHeight property is -1. If the ScrollHeight property is -1, the control uses the default height of the horizontal scroll bar from the system. Use the ScrollHeight property to specify the height of the horizontal scroll bar. Use the [ScrollBars](#) property to specify which scroll bar is visible or hidden in the control. Use the [ScrollButtonWidth](#) property to specify the width of the buttons in the horizontal scroll bar. Use the [ScrollWidth](#) property to specify the width of the vertical scroll bar. Use the [ScrollButtonHeight](#) property to specify the height of the buttons in the vertical scroll bar. Use the [ScrollPartVisible](#) property to specify the visible parts in the control's scroll bar. Use the [ScrollThumbSize](#) property to define a fixed size for the scrollbar's thumb.

property Expression.ScrollOrderParts(ScrollBar as ScrollBarEnum) as String

Specifies the order of the buttons in the scroll bar.

Type	Description
ScrollBar as ScrollBarEnum	A ScrollBar expression that indicates the scrollbar where the order of buttons is displayed.
String	A String expression that indicates the order of the buttons in the scroll bar. The list includes expressions like l, l1, ..., l5, t, r, r1, ..., r6 separated by comma, each expression indicating a part of the scroll bar, and its position indicating the displaying order.

Use the ScrollOrderParts to customize the order of the buttons in the scroll bar. By default, the ScrollOrderParts property is empty. If the ScrollOrderParts property is empty the default order of the buttons in the scroll bar are displayed like follows:



so, the order of the parts is: l1, l2, l3, l4, l5, l, t, r, r1, r2, r3, r4, r5 and r6. Use the [ScrollPartVisible](#) to specify whether a button in the scrollbar is visible or hidden. Use the [ScrollPartEnable](#) property to enable or disable a button in the scroll bar. Use the [ScrollPartCaption](#) property to assign a caption to a button in the scroll bar.

Use the ScrollOrderParts property to change the order of the buttons in the scroll bar. For instance, "l,r,t,l1,r1" puts the left and right buttons to the left of the thumb area, and the l1 and r1 buttons right after the thumb area. If the parts are not specified in the ScrollOrderParts property, automatically they are added to the end.



The list of supported literals in the ScrollOrderParts property is:

- **l** for exLeftBPart, (<) The left or top button.
- **l1** for exLeftB1Part, (L1) The first additional button, in the left or top area.
- **l2** for exLeftB2Part, (L2) The second additional button, in the left or top area.
- **l3** for exLeftB3Part, (L3) The third additional button, in the left or top area.
- **l4** for exLeftB4Part, (L4) The fourth additional button, in the left or top area.
- **l5** for exLeftB5Part, (L5) The fifth additional button, in the left or top area.
- **t** for exLowerBackPart, exThumbPart and exUpperBackPart, The union between the exLowerBackPart and the exUpperBackPart parts.
- **r** for exRightBPart, (>) The right or down button.

- **r1** for exRightB1Part, (R1) The first additional button in the right or down side.
- **r2** for exRightB2Part, (R2) The second additional button in the right or down side.
- **r3** for exRightB3Part, (R3) The third additional button in the right or down side.
- **r4** for exRightB4Part, (R4) The fourth additional button in the right or down side.
- **r5** for exRightB5Part, (R5) The fifth additional button in the right or down side.
- **r6** for exRightB6Part, (R6) The sixth additional button in the right or down side.

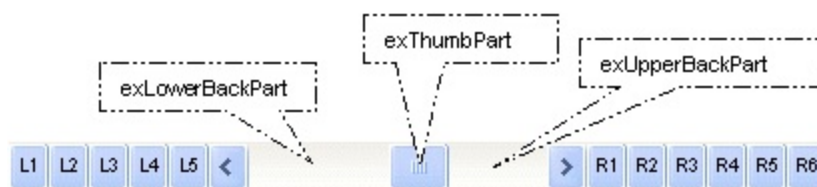
Any other literal between commas is ignored. If duplicate literals are found, the second is ignored, and so on. For instance, "t,l,r" indicates that the left/top and right/bottom buttons are displayed right/bottom after the thumb area.

Property Expression.ScrollPartCaption(ScrollBar as ScrollBarEnum, Part as ScrollPartEnum) as String

Specifies the caption being displayed on the specified scroll part.

Type	Description
ScrollBar as ScrollBarEnum	A ScrollBar expression that indicates the scrollbar where the caption is displayed.
Part as ScrollPartEnum	A ScrollPartEnum expression that specifies the parts of the scroll where the text is displayed
String	A String expression that specifies the caption being displayed on the part of the scroll bar.

Use the ScrollPartCaption property to specify the caption of the scroll's part. Use the [ScrollPartVisible](#) property to add or remove buttons/parts in the control's scrollbar. By default, when a part becomes visible, the [ScrollPartEnable](#) property is automatically called, so the parts becomes enabled. Use the [ScrollPartEnable](#) property to specify enable or disable parts in the control's scrollbar. Use the [ScrollBars](#) property to specify the visible scrollbars in the control. The control fires the [ScrollButtonClick](#) event when the user clicks a part of the scroll bar. Use the [ScrollFont](#) property to specify the font in the control's scroll bar. Use the [ScrollOrderParts](#) property to customize the order of the buttons in the scroll bar.



By default, the following parts are shown:

- exLeftBPart (the left or up button of the control)
- exLowerBackPart (the part between the left/up button and the thumb part of the control)
- exThumbPart (the thumb/scrollbox part)
- exUpperBackPart (the part between the the thumb and the right/down button of the control)
- exRightBPart (the right or down button of the control)

The following VB sample adds up and down additional buttons to the control's vertical scroll bar :

With Expression1

```
.ScrollPartVisible(exVScroll, exLeftB1Part Or exRightB1Part) = True
```

```
.ScrollPartCaption(exVScroll, exLeftB1Part) = "1"
```

```
.ScrollPartCaption(exVScroll, exRightB1Part) = "2"
```

```
End With
```

The following VB.NET sample adds up and down additional buttons to the control's vertical scroll bar :

```
With AxExpression1
```

```
    .set_ScrollPartVisible(EXPRESSIONLib.ScrollBarEnum.exVScroll,  
EXPRESSIONLib.ScrollPartEnum.exLeftB1Part Or  
EXPRESSIONLib.ScrollPartEnum.exRightB1Part, True)
```

```
    .set_ScrollPartCaption(EXPRESSIONLib.ScrollBarEnum.exVScroll,  
EXPRESSIONLib.ScrollPartEnum.exLeftB1Part, "1")
```

```
    .set_ScrollPartCaption(EXPRESSIONLib.ScrollBarEnum.exVScroll,  
EXPRESSIONLib.ScrollPartEnum.exRightB1Part, "2")
```

```
End With
```

The following C# sample adds up and down additional buttons to the control's vertical scroll bar :

```
axExpression1.set_ScrollPartVisible(EXPRESSIONLib.ScrollBarEnum.exVScroll,  
EXPRESSIONLib.ScrollPartEnum.exLeftB1Part |  
EXPRESSIONLib.ScrollPartEnum.exRightB1Part, true);
```

```
axExpression1.set_ScrollPartCaption(EXPRESSIONLib.ScrollBarEnum.exVScroll,  
EXPRESSIONLib.ScrollPartEnum.exLeftB1Part , "1");
```

```
axExpression1.set_ScrollPartCaption(EXPRESSIONLib.ScrollBarEnum.exVScroll,  
EXPRESSIONLib.ScrollPartEnum.exRightB1Part, "2");
```

The following C++ sample adds up and down additional buttons to the control's vertical scroll bar :

```
m_expression.SetScrollPartVisible( 0 /*exVScroll*/, 32768 /*exLeftB1Part*/ | 32  
/*exRightB1Part*/, TRUE );
```

```
m_expression.SetScrollPartCaption( 0 /*exVScroll*/, 32768 /*exLeftB1Part*/ , _T("1") );
```

```
m_expression.SetScrollPartCaption( 0 /*exVScroll*/, 32 /*exRightB1Part*/ , _T("2") );
```

The following VFP sample adds up and down additional buttons to the control's vertical scroll bar :

```
With thisform.Expression1
```

```
.ScrollPartVisible(0, bitor(32768,32)) = .t.
```

```
.ScrollPartCaption(0,32768) = "1"
```

```
.ScrollPartCaption(0, 32) = "2"
```

```
EndWith
```

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

```
LPARAMETERS scrollpart
```

```
wait window nowait ltrim(str(scrollpart))
```

property Expression.ScrollPartCaptionAlignment(ScrollBar as ScrollBarEnum, Part as ScrollPartEnum) as AlignmentEnum

Specifies the alignment of the caption in the part of the scroll bar.

Type	Description
ScrollBar as ScrollBarEnum	A ScrollBar expression that indicates the scrollbar where the caption is displayed.
Part as ScrollPartEnum	A ScrollPartEnum expression that specifies the parts of the scroll where the text is displayed.
AlignmentEnum	An AlignmentEnum expression that specifies the alignment of the caption in the part of the scrollbar.

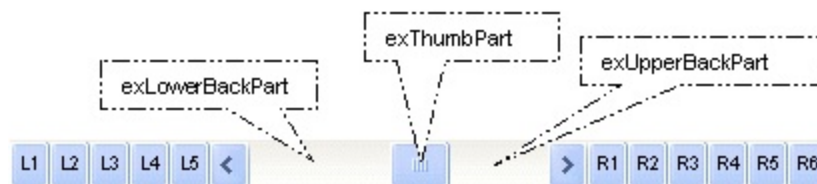
The ScrollPartCaptionAlignment property specifies the alignment of the caption in the part of the scroll bar. By default, the caption is centered. Use the [ScrollPartCaption](#) property to specify the caption being displayed on specified part of the scroll bar. Use the [ScrollPartVisible](#) property to add or remove buttons/parts in the control's scrollbar.

property Expression.ScrollPartEnable(ScrollBar as ScrollBarEnum, Part as ScrollPartEnum) as Boolean

Indicates whether the specified scroll part is enabled or disabled.

Type	Description
ScrollBar as ScrollBarEnum	A ScrollBar expression that indicates the scrollbar where the part is enabled or disabled.
Part as ScrollPartEnum	A ScrollPartEnum expression that specifies the parts of the scroll bar being enabled or disabled.
Boolean	A Boolean expression that specifies whether the scrollbar's part is enabled or disabled.

By default, when a part becomes visible, the [ScrollPartEnable](#) property is automatically called, so the parts becomes enabled. Use the [ScrollPartVisible](#) property to add or remove buttons/parts in the control's scrollbar. Use the [ScrollPartEnable](#) property to specify enable or disable parts in the control's scrollbar. Use the [ScrollBars](#) property to specify the visible scrollbars in the control. Use the [ScrollPartCaption](#) property to specify the caption of the scroll's part. The control fires the [ScrollButtonClick](#) event when the user clicks a part of the scroll bar. Use the [ScrollOrderParts](#) property to customize the order of the buttons in the scroll bar.

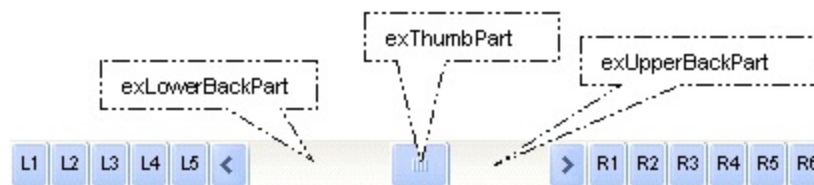


property Expression.ScrollPartVisible(ScrollBar as ScrollBarEnum, Part as ScrollPartEnum) as Boolean

Indicates whether the specified scroll part is visible or hidden.

Type	Description
ScrollBar as ScrollBarEnum	A ScrollBar expression that indicates the scrollbar where the part is visible or hidden.
Part as ScrollPartEnum	A ScrollPartEnum expression that specifies the parts of the scroll bar being visible
Boolean	A Boolean expression that specifies whether the scrollbar's part is visible or hidden.

Use the ScrollPartVisible property to add or remove buttons/parts in the control's scrollbar. By default, when a part becomes visible, the [ScrollPartEnable](#) property is automatically called, so the parts becomes enabled. Use the [ScrollPartEnable](#) property to specify enable or disable parts in the control's scrollbar. Use the [ScrollBars](#) property to specify the visible scrollbars in the control. Use the [ScrollPartCaption](#) property to specify the caption of the scroll's part. The control fires the [ScrollButtonClick](#) event when the user clicks a part of the scroll bar. Use the [Background](#) property to change the visual appearance for any part in the control's scroll bar. Use the [ScrollOrderParts](#) property to customize the order of the buttons in the scroll bar.



By default, the following parts are shown:

- exLeftBPart (the left or up button of the control)
- exLowerBackPart (the part between the left/up button and the thumb part of the control)
- exThumbPart (the thumb/scrollbox part)
- exUpperBackPart (the part between the the thumb and the right/down button of the control)
- exRightBPart (the right or down button of the control)

The following VB sample adds up and down additional buttons to the control's vertical scroll bar :

With Expression1

```
.ScrollPartVisible(exVScroll, exLeftB1Part Or exRightB1Part) = True
```

```
.ScrollPartCaption(exVScroll, exLeftB1Part) = "1"
```

```
.ScrollPartCaption(exVScroll, exRightB1Part) = "2"
```

End With

The following VB.NET sample adds up and down additional buttons to the control's vertical scroll bar :

With AxExpression1

```
.set_ScrollPartVisible(EXPRESSIONLib.ScrollBarEnum.exVScroll,  
EXPRESSIONLib.ScrollPartEnum.exLeftB1Part Or  
EXPRESSIONLib.ScrollPartEnum.exRightB1Part, True)
```

```
.set_ScrollPartCaption(EXPRESSIONLib.ScrollBarEnum.exVScroll,  
EXPRESSIONLib.ScrollPartEnum.exLeftB1Part, "1")
```

```
.set_ScrollPartCaption(EXPRESSIONLib.ScrollBarEnum.exVScroll,  
EXPRESSIONLib.ScrollPartEnum.exRightB1Part, "2")
```

End With

The following C# sample adds up and down additional buttons to the control's vertical scroll bar :

```
axExpression1.set_ScrollPartVisible(EXPRESSIONLib.ScrollBarEnum.exVScroll,  
EXPRESSIONLib.ScrollPartEnum.exLeftB1Part |
```

```
EXPRESSIONLib.ScrollPartEnum.exRightB1Part, true);
```

```
axExpression1.set_ScrollPartCaption(EXPRESSIONLib.ScrollBarEnum.exVScroll,  
EXPRESSIONLib.ScrollPartEnum.exLeftB1Part , "1");
```

```
axExpression1.set_ScrollPartCaption(EXPRESSIONLib.ScrollBarEnum.exVScroll,  
EXPRESSIONLib.ScrollPartEnum.exRightB1Part, "2");
```

The following C++ sample adds up and down additional buttons to the control's vertical scroll bar :

```
m_expression.SetScrollPartVisible( 0 /*exVScroll*/, 32768 /*exLeftB1Part*/ | 32  
/*exRightB1Part*/, TRUE );
```

```
m_expression.SetScrollPartCaption( 0 /*exVScroll*/, 32768 /*exLeftB1Part*/ , _T("1") );
```

```
m_expression.SetScrollPartCaption( 0 /*exVScroll*/, 32 /*exRightB1Part*/ , _T("2") );
```

The following VFP sample adds up and down additional buttons to the control's vertical

scroll bar :

```
With thisform.Expression1
```

```
  .ScrollPartVisible(0, bitor(32768,32)) = .t.
```

```
  .ScrollPartCaption(0,32768) = "1"
```

```
  .ScrollPartCaption(0, 32) = "2"
```

```
EndWith
```

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

```
LPARAMETERS scrollpart
```

```
wait window nowait ltrim(str(scrollpart))
```

property Expression.ScrollThumbSize(ScrollBar as ScrollBarEnum) as Long

Specifies the size of the thumb in the scrollbar.

Type	Description
ScrollBar as ScrollBarEnum	A ScrollBarEnum expression that indicates the vertical or the horizontal scroll bar.
Long	A long expression that defines the size of the scrollbar's thumb.

use the ScrollThumbSize property to define a fixed size for the scrollbar's thumb. By default, the ScrollThumbSize property is -1, that makes the control computes automatically the size of the thumb based on the scrollbar's range. If case, use the fixed size for your thumb when you change its visual appearance using the [Background\(exVSTThumb\)](#) or [Background\(exHSTThumb\)](#) property. Use the [ScrollWidth](#) property to specify the width of the vertical scroll bar. Use the [ScrollButtonWidth](#) property to specify the width of the buttons in the horizontal scroll bar. Use the [ScrollHeight](#) property to specify the height of the horizontal scroll bar. Use the [ScrollButtonHeight](#) property to specify the height of the buttons in the vertical scroll bar. Use the [ScrollPartVisible](#) property to specify the visible parts in the control's scroll bar.

property Expression.ScrollToolTip(ScrollBar as ScrollBarEnum) as String

Specifies the tooltip being shown when the user moves the scroll box.

Type	Description
ScrollBar as ScrollBarEnum	A ScrollBarEnum expression that indicates the vertical scroll bar or the horizontal scroll bar.
String	A string expression being shown when the user clicks and moves the scrollbar's thumb.

Use the ScrollToolTip property to specify whether the control displays a tooltip when the user clicks and moves the scrollbar's thumb. By default, the ScrollToolTip property is empty. If the ScrollToolTip property is empty, the tooltip is not shown when the user clicks and moves the thumb of the scroll bar. Use the [SortPartVisible](#) property to specify the parts being visible in the control's scroll bar. Use the [ScrollBars](#) property to specify the visible scrollbars in the control.

property Expression.ScrollWidth as Long

Specifies the width of the vertical scrollbar.

Type	Description
Long	A long expression that defines the width of the vertical scroll bar.

By default, the ScrollWidth property is -1. If the ScrollWidth property is -1, the control uses the default width of the vertical scroll bar from the system. Use the ScrollWidth property to specify the width of the vertical scroll bar. Use the [ScrollBars](#) property to specify which scroll bar is visible or hidden in the control. Use the [ScrollButtonWidth](#) property to specify the width of the buttons in the horizontal scroll bar. Use the [ScrollHeight](#) property to specify the height of the horizontal scroll bar. Use the [ScrollButtonHeight](#) property to specify the height of the buttons in the vertical scroll bar. Use the [ScrollPartVisible](#) property to specify the visible parts in the control's scroll bar. Use the [ScrollThumbSize](#) property to define a fixed size for the scrollbar's thumb.

property Expression.SelBackColor as Color

Specifies the selection's background color.

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

Use the [SelForeColor](#) and [SelBackColor](#) properties to define the colors used to paint the selection, when the control has the focus. Use the [HideSelection](#) property to specify whether the control hides the selection when the control loses the focus. The [SelForeColorHide](#) property specifies the foreground color of the current selection when the control has no focus, and the [HideSelection](#) property is False. The [SelBackColorHide](#) property specifies the background color of the current selection when the control has no focus, and the [HideSelection](#) property is False. Use the [DisplaySelection](#) to hide the selection. Use the [SelStart](#), [SelLength](#) and [SelText](#) properties to access the selection. The control fires the [SelChange](#) event when user changes the selection.

property Expression.SelBackColorHide as Color

Specifies the selection's background color, when the control has no focus, and the [HideSelection](#) property is False.

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

The [SelBackColorHide](#) property specifies the background color of the current selection when the control has no focus, and the [HideSelection](#) property is False. Use the [HideSelection](#) property to specify whether the control hides the selection when the control loses the focus. The [SelForeColorHide](#) property specifies the foreground color of the current selection when the control has no focus, and the [HideSelection](#) property is False.

property Expression.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, when the control has the focus. Use the [HideSelection](#) property to specify whether the control hides the selection when the control loses the focus. The [SelForeColorHide](#) property specifies the foreground color of the current selection when the control has no focus, and the [HideSelection](#) property is `False`. The [SelBackColorHide](#) property specifies the background color of the current selection when the control has no focus, and the [HideSelection](#) property is `False`. Use the [DisplaySelection](#) to hide the selection. Use the [SelStart](#), [SelLength](#) and [SelText](#) properties to access the selection. The control fires the [SelChange](#) event when user changes the selection.

property Expression.**SelfForeColorHide** as Color

Specifies the selection's foreground color, when the control has no focus, and the [HideSelection](#) property is False.

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

The [SelfForeColorHide](#) property specifies the foreground color of the current selection when the control has no focus, and the [HideSelection](#) property is False. Use the [HideSelection](#) property to specify whether the control hides the selection when the control loses the focus. The [SelBackColorHide](#) property specifies the background color of the current selection when the control has no focus, and the [HideSelection](#) property is False.

property Expression.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 control fires the [SelChange](#) event when user changes the selection. Use the [SelForeColor](#) and [SelBackColor](#) properties to specify the colors for the selected text.

property Expression.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 [SelLength](#) property to get the selection's length.

property Expression.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 control fires the [SelChange](#) event when the user changes the selection.

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

```
Private Sub Expression1_SelChange()  
    If Not Expression1.SelText = "" Then  
        Debug.Print Expression1.SelText  
    End If  
End Sub
```

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

```
void OnSelChangeExpression1()  
{  
    OutputDebugString( m_expression.GetSelText() );  
}
```

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

```
Private Sub AxExpression1_SelChange(ByVal sender As Object, ByVal e As  
System.EventArgs) Handles AxExpression1.SelChange  
    With AxExpression1  
        Debug.WriteLine(.SelText)  
    End With  
End Sub
```

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

```
private void axExpression1_SelChange(object sender, EventArgs e)
```

```
{  
  System.Diagnostics.Debug.WriteLine(axExpression1.SelText);  
}
```

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

```
*** ActiveX Control Event ***  
  
with thisform.Expression1  
  wait window nowait .SelText  
endwith
```

property Expression.ShowCaret as Boolean

Specifies whether the control's caret is visible or hidden.

Type	Description
Boolean	A boolean expression that indicates whether the control's caret is shown or hidden.

Use the ShowCaret property to hide the control's caret. By default, the ShowCaret property is True. The control hides the caret when the control is disabled. Use the [Enabled](#) property to disable the control. Use the [Locked](#) property to lock the control.

property Expression.ShowImageList as Boolean

Specifies whether the control's images panel dialog is visible or hidden.

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

The property has effect only at design time. Use the [Images](#) method to assign a list of icons to the control, at run time. Use the [RepaceIcon](#) method to add, remove or clear icons in the control's images collection.

The following screen shot shows the control's images panel, where user can drag icon files in order to add new icons to the control at design mode:



method Expression.ShowToolTip (ToolTip as String, [Title as Variant], [Alignment as Variant], [X as Variant], [Y as Variant])

Shows the specified tooltip at given position.

Type	Description
ToolTip as String	<p>The ToolTip parameter can be any of the following:</p> <ul style="list-style-type: none">• NULL(BSTR) or "<null>"(string) to indicate that the tooltip for the object being hovered is not changed• A String expression that indicates the description of the tooltip, that supports built-in HTML format (adds, replaces or changes the object's tooltip)
Title as Variant	<p>The Title parameter can be any of the following:</p> <ul style="list-style-type: none">• missing (VT_EMPTY, VT_ERROR type) or "<null>" (string) the title for the object being hovered is not changed.• A String expression that indicates the title of the tooltip (no built-in HTML format) (adds, replaces or changes the object's title)
Alignment as Variant	<p>A long expression that indicates the alignment of the tooltip relative to the position of the cursor. If missing (VT_EMPTY, VT_ERROR) the alignment of the tooltip for the object being hovered is not changed.</p> <p>The Alignment parameter can be one of the following:</p> <ul style="list-style-type: none">• 0 - exTopLeft• 1 - exTopRight• 2 - exBottomLeft• 3 - exBottomRight• 0x10 - exCenter• 0x11 - exCenterLeft• 0x12 - exCenterRight• 0x13 - exCenterTop• 0x14 - exCenterBottom <p>By default, the tooltip is aligned relative to the top-left corner (0 - exTopLeft).</p>

Specifies the horizontal position to display the tooltip as one of the following:

- missing (VT_EMPTY, VT_ERROR type), indicates that the tooltip is shown on its default position / current cursor position (ignored)
- -1, indicates the current horizontal position of the cursor (current x-position)
- a numeric expression that indicates the horizontal screen position to show the tooltip (fixed screen x-position)
- a string expression that indicates the horizontal displacement relative to default position to show the tooltip (moved)

X as Variant

Specifies the vertical position to display the tooltip as one of the following:

- missing (VT_EMPTY, VT_ERROR type), indicates that the tooltip is shown on its default position / current cursor position (ignored)
- -1, indicates the current vertical position of the cursor (current y-position)
- a numeric expression that indicates the vertical screen position to show the tooltip (fixed screen y-position)
- a string expression that indicates the vertical displacement relative to default position to show the tooltip (displacement)

Y as Variant

Use the ShowToolTip method to display a custom tooltip at specified position or to update the object's tooltip, title or position. You can call the ShowToolTip method during the [MouseMove](#) event. Use the [ToolTipPopDelay](#) property specifies the period in ms of time the ToolTip remains visible if the mouse pointer is stationary within a control. The [ToolTipDelay](#) property specifies the time in ms that passes before the ToolTip appears. Use the [ToolTipWidth](#) property to specify the width of the tooltip window. Use the [ToolTipFont](#) property to change the tooltip's font. Use the [Background\(exToolTipAppearance\)](#) property indicates the visual appearance of the borders of the tooltips. Use the [Background\(exToolTipBackColor\)](#) property indicates the tooltip's background color. Use the [Background\(exToolTipForeColor\)](#) property indicates the tooltip's foreground color.

For instance:

- [ShowToolTip](#)(`<null>`,`<null>`,`+8`,`+8`), shows the tooltip of the object moved relative

to its default position

- `ShowToolTip(<null>, 'new title')`, adds, changes or replaces the title of the object's tooltip
- `ShowToolTip('new content')`, adds, changes or replaces the object's tooltip
- `ShowToolTip('new content', 'new title')`, shows the tooltip and title at current position
- `ShowToolTip('new content', 'new title', '+8', '+8')`, shows the tooltip and title moved relative to the current position
- `ShowToolTip('new content', '', 128, 128)`, displays the tooltip at a fixed position
- `ShowToolTip('', '')`, hides the tooltip

The ToolTip parameter supports the built-in HTML format like follows:

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

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

gradient-center

- `<out rrggbb;width> ... </out>` shows the text with outlined characters, where rr/gg/bb represents the red/green/blue values of the outline color, 808080 if missing as gray, width indicates the size of the outline, 1 if missing. The text color or `<fgcolor>` defines the color to show the inside text. The `` HTML tag can be used to define the

height of the font. For instance the "`<out 000000>`

`<fgcolor=FFFFFF>outlined</fgcolor></out>`" generates the following picture:

outlined

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

shadow

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

outline anti-aliasing

property Expression.SplitPaneHeight as Long

Specifies a value that indicates the height in pixels of the top pane(s) when splitting.

Type	Description
Long	A long expression that indicates the width of the left pane(s) when splitting.

By default, the SplitPaneHeight property is 0. Use the SplitPaneHeight property to split vertically the expression control. Use the [SplitPaneWidth](#) property to split horizontally the expression control. Use the [AllowSplitter](#) property to show or hide certain splitters. Use the [FocusPane](#) property to specify the index of the pane that has the focus.

property Expression.SplitPaneWidth as Long

Specifies a value that indicates the width in pixels of the left pane(s) when splitting.

Type	Description
Long	A long expression that indicates the width in pixels of the left pane(s) when splitting.

By default, the SplitPaneWidth property is 0. Use the SplitPaneWidth property to split horizontally the expression control. Use the [SplitPaneHeight](#) property to split vertically the expression control. Use the [AllowSplitter](#) property to show or hide certain splitters. The [SplitterChange](#) event notifies your application when user splits the control.

property Expression.TabLength as Long

Specifies the size of each tab stop, in units equal to the average character width.

Type	Description
Long	A long expression that indicates the size of each tab stop, in units equal to the average character width.

Use the TabLength property to defines the size for each tab character. By default, the TabLength property is 4.

property Expression.Template as String

Specifies the control's template.

Type	Description
String	A string expression that indicates the template script.

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

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

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

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

The Template 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 Expression.Text as String

Specifies the control's text.

Type	Description
String	A string expression that defines the control's text/expression.

The Text property specifies the control's text/expression. The [Expression](#) property returns the expression to be evaluated. The [EvaluateSelection](#) property specifies whether the control evaluates the selection when it is available. So, if the [EvaluateSelection](#) property the Expression property returns the selected text (if the [SelLength](#) property is greater than 0), or the entire text if the [SelLength](#) property is 0. The control fires the [SelChange](#) event when user changes the selection. Use the [SelForeColor](#) and [SelBackColor](#) properties to specify the colors for the selected text.

For instance:

- the *currency(value)* displays the value using the current format for the currency ie, 1000 gets displayed as \$1,000.00, for US format
- the *longdate(date(value))* converts the value to a date and gets the long format to display the date in the column, ie #1/1/2001# displays instead Monday, January 01, 2001
- the *'' + ((0:=proper(value)) left 1) + '' + (=:0 mid 2)* converts the name to proper, so the first letter is capitalized, bolds the first character, and let unchanged the rest, ie a "mihai filimon" gets displayed "**M**ihai Filimon".
- the *value format '2|.3|,'* displays the value using 2 digits, . as decimal separator, grouping by 3 digits using the , as a grouping separator.
- the *type(value) in (0,1) ? 'null' : (dbl(value)<0 ? '<fgcolor=FF0000>' + (value format '2|.3|,') : (dbl(value)>0 ? '<fgcolor=0000FF>' + (value format '2|.3|,') : '0.00')* displays the positive values in blue, being preceded by + sign, negative values in red preceded by - sign, 0 as 0.00 while for null values is displays null. The numbers are displayed using 2 digits, . as decimal separator and grouping by 3 digits by , separator.
- the *((1:=int(0:= (value))) != 0 ? (=:1 + ' day(s)' : ") + (=:1 ? ' ' : ") + ((1:=int(0:=((=:0 - =:1 + 1/24/60/60/2)*24))) != 0 ? =:1 + ' hour(s)' : ") + (=:1 ? ' ' : ") + ((1:=round((=:0 - =:1)*60)) != 0 ? =:1 + ' min(s)' : ")*, displays the value in days, hours and minutes

Most of our components support formatting the values. By formatting we mean that instead displaying a value we can display in the way we desire. Properties such as Column.FormatColumn, Items.FormatCell and so on support formatting. For instance, the *currency(100)* displays the value 100 as a currency, for instance in US format it will display \$100 while for German format will display 100 . The format expression supports operators, constants and values as described bellow. From case to case, there are few predefined

keywords such as **value**, which indicates the value to be formatted, the **%0**, **%1**, **%2**, **%3**, ... indicates variables that could be: the value in a specified column, the value for a specified property of the bar, and so on. If the formatting is using in properties such as: `Items.ItemBar(exBarToolTip)`, `Items.ItemBar(exBarCaption)` or `Items.ItemBar(exBarExtraCaption)` the **%C0**, **%C1**, **%C2**, ... indicates the captions of the cells, ...

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

property Expression.ToolTipDelay as Long

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

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

By default, the ToolTipDelay property is 500 ms. If the ToolTipDelay property is 0, the control displays no tooltips for any keyword. Use the ToolTipDelay and [ToolTipPopDelay](#) properties to define the time before showing a tooltip and the period of the time that tooltip remains visible if the mouse pointer is stationary within a control. Use the [ToolTipOnTyping](#) property to display the keyword's tooltip while typing in the control. Use the [ToolTipFont](#) property to assign a font for the control's tooltip. Use the [Background\(exToolTipAppearance\)](#) property indicates the visual appearance of the borders of the tooltips. Use the [Background\(exToolTipBackColor\)](#) property indicates the tooltip's background color. Use the [Background\(exToolTipForeColor\)](#) property indicates the tooltip's foreground color.

property Expression.ToolTipFont as IFontDisp

Retrieves or sets the tooltip's font.

Type	Description
IFontDisp	A Font object being used to display the tooltip.

Use the `ToolTipFont` property to assign a font for the control's tooltip. Use the `ToolTipDelay` and [ToolTipPopDelay](#) properties to define the time before showing a tooltip and the period of the time that tooltip remains visible if the mouse pointer is stationary within a control. Use the [ToolTipOnTyping](#) property to display the keyword's tooltip while typing in the control. Use the [Background\(exToolTipAppearance\)](#) property indicates the visual appearance of the borders of the tooltips. Use the [Background\(exToolTipBackColor\)](#) property indicates the tooltip's background color. Use the [Background\(exToolTipForeColor\)](#) property indicates the tooltip's foreground color.

property Expression.ToolTipOnTyping as Boolean

Specifies a value that indicates whether the tooltip of the keyword is shown while typing.

Type	Description
Boolean	A boolean expression that indicates whether the control shows the tooltip of the keyword being typed.

By default, the `ToolTipOnTyping` property is `True`. Use the `ToolTipOnTyping` property to display the keyword's tooltip while typing in the control. Use the `ToolTip` parameter of the `AddKeyword` method to assign a tooltip to a keyword. By default, the keyword's tooltip is shown when the cursor hovers the keyword. Use the [ToolTipDelay](#) and [ToolTipPopDelay](#) properties to define the time before showing a tooltip and the period of the time that tooltip remains visible. The [ToolTipWidth](#) property specifies the width of the tooltip's window. Use the [ToolTipFont](#) property to assign a font for the control's tooltip.

property Expression.ToolTipPopDelay as Long

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

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

By default, the ToolTipPopDelay property is 5000 ms. If the ToolTipPopDelay property is 0, the control displays no tooltips for any keyword. Use the [ToolTipDelay](#) and ToolTipPopDelay properties to define the time before showing a tooltip and the period of the time that tooltip remains visible if the mouse pointer is stationary within a control. Use the [ToolTipOnTyping](#) property to display the keyword's tooltip while typing in the control. Use the [ToolTipFont](#) property to assign a font for the control's tooltip. Use the [Background\(exToolTipAppearance\)](#) property indicates the visual appearance of the borders of the tooltips. Use the [Background\(exToolTipBackColor\)](#) property indicates the tooltip's background color. Use the [Background\(exToolTipForeColor\)](#) property indicates the tooltip's foreground color.

property Expression.ToolTipWidth as Long

Specifies a value that indicates the width of the tooltip window, in pixels.

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

Use the `ToolTipWidth` property to specify the width of the tooltip window. By default, the `ToolTipWidth` property is 196 pixels. The height of the tooltip window is computed based on the tooltip's description. Use the [ToolTipOnTyping](#) property to display the keyword's tooltip while typing in the control. Use the [ToolTipFont](#) property to assign a font for the control's tooltip. Use the [Background\(exToolTipAppearance\)](#) property indicates the visual appearance of the borders of the tooltips. Use the [Background\(exToolTipBackColor\)](#) property indicates the tooltip's background color. Use the [Background\(exToolTipForeColor\)](#) property indicates the tooltip's foreground color.

method **Expression.Undo ()**

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

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

Use the Undo method to undo the last expression-control operation. Use the [CanUndo](#) property to specify whether an undo operation is available. Use the [Redo](#) method to redo the last expression-control operation. Use the [AllowContextMenu](#) property to specify whether the control displays a context menu when user right clicks the control.

property Expression.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 True. 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 [IndentOnTab](#) property is True. 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 Expression.Version as String

Retrieves the control's version.

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

The Version property specifies the control's version

property Expression.VisualAppearance as Appearance

Retrieves the control's appearance.

Type	Description
Appearance	An Appearance object that holds a collection of skins.

Use the [Add](#) method to add or replace skins to the control. The skin method, in its simplest form, uses a single graphic file (*.ebn) assigned to a part of the control. By using a collection of objects laid over the graphic, it is possible to define which sections of the graphic will be used as borders, corners and other possible elements, fixing them to their proper position regardless of the size of the part. Use the [Background](#) property to change the visual appearance for different parts of the control.

property Expression.WordFromPoint (X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS, [Reserved as Variant]) as String

Retrieves the word from the cursor.

Type	Description
X as OLE_XPOS_PIXELS	A single expression that indicates the X position in client coordinate.
Y as OLE_YPOS_PIXELS	A single expression that indicates the Y position in client coordinate.
Reserved as Variant	Not used.
String	A long expression that indicates word from point (X,Y).

The WordFromPoint property retrieves the word from the specified point. Use the WordFromPoint property to get the word from the point specified by the {X,Y}. The X and Y coordinates are expressed in client coordinates, so a conversion must be done in case your coordinates are relative to the screen or to other window. **If the X parameter is -1 and Y parameter is -1 the WordFromPoint property determines the word from the cursor.** The [ShowToolTip](#) method can be used to show a custom tooltip at specified location.

ExPression 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: {B33F5489-49AC-4155-98E7-9BBFC57FF019}. The object's program identifier is: "Exontrol.Expression". The /COM object module is: "Expression.dll"

The ExExpression 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 list control.
DbClick	Occurs when the user double clicks the left mouse button over an object.
ExecuteContextMenu	Occurs when the user selects an user item from the control's context menu.
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.
OLECompleteDrag	Occurs when a source component is dropped onto a target component, informing the source component that a drag action was either performed or canceled
OLEDragDrop	Occurs when a source component is dropped onto a target component when the source component determines that a drop can occur.
OLEDragOver	Occurs when one component is dragged over another.
OLEGiveFeedback	Allows the drag source to specify the type of OLE drag-and-drop operation and the visual feedback.
OLESetData	Occurs on a drag source when a drop target calls the GetData method and there is no data in a specified format in the OLE drag-and-drop DataObject.
OLEStartDrag	Occurs when the OLEDrag method is called.
	Occurs when the user invokes the control's context

[OnContext](#)

window.

[ScrollButtonClick](#)

Occurs when the user clicks a button in the scrollbar.

[SelChange](#)

Occurs when the user selects text in the control.

[SplitterChange](#)

Occurs when the user splits the control.

event Change ()

Indicates that the control's text have changed.

Type

Description

Use the Change event to notify you application that the user changes the text/expression in the control. Use the [Text](#) property to access the control's text/expression. The [IsValid](#) property indicates whether the control's expression is valid/syntactically correct.

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;
```

```
Powe... 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(oExpression)  
RETURN
```

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

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

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

```
Visual  
Data... Procedure OnComChange  
Forward Send OnComChange  
End_Procedure
```

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

```
X++ void onEvent_Change()  
{
```

```
}
```

XBasic

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

dBASE

```
function nativeObject_Change()  
return
```

The following sample shows how you can check if the expression is valid/syntactically correct.

VBA (MS Access, Excell...)

```
' Change event - Indicates that the control's text has changed.
```

```
Private Sub Expression1_Change()
```

```
    With Expression1
```

```
        Debug.Print( "Valid: " )
```

```
        Debug.Print( .IsValid )
```

```
    End With
```

```
End Sub
```

```
With Expression1
```

```
    .Expression = "value"
```

```
    .SplitPaneHeight = 196
```

```
    .Background(1) = RGB(240,240,240)
```

```
    .AllowSplitter = 2
```

```
End With
```

VB6

```
' Change event - Indicates that the control's text has changed.
```

```
Private Sub Expression1_Change()
```

```
    With Expression1
```

```
        Debug.Print( "Valid: " )
```

```
        Debug.Print( .IsValid )
```

```
    End With
```

```
End Sub
```

```
With Expression1
```

```
.Expression = "value"
```

```
.SplitPaneHeight = 196
```

```
.Background(exVSplitterApp) = RGB(240,240,240)
```

```
.AllowSplitter = exVSplitter
```

```
End With
```

VB.NET

```
' Change event - Indicates that the control's text has changed.
```

```
Private Sub Expression1_Change(ByVal sender As System.Object) Handles
```

```
Expression1.Change
```

```
With Expression1
```

```
Debug.Print( "Valid: " )
```

```
Debug.Print( .IsValid )
```

```
End With
```

```
End Sub
```

```
With Expression1
```

```
.Expression = "value"
```

```
.SplitPaneHeight = 196
```

```
.set_Background(exontrol.EXPRESSIONLib.BackgroundPartEnum.exVSplitterApp,Color.F
```

```
.AllowSplitter = exontrol.EXPRESSIONLib.SplitterEnum.exVSplitter
```

```
End With
```

VB.NET for /COM

```
' Change event - Indicates that the control's text has changed.
```

```
Private Sub AxExpression1_Change(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles AxExpression1.Change
```

```
With AxExpression1
```

```
Debug.Print( "Valid: " )
```

```
Debug.Print( .IsValid )
```

```
End With
```

```
End Sub
```

With AxExpression1

```
.Expression = "value"
```

```
.SplitPaneHeight = 196
```

```
.set_Background(EXPRESSIONLib.BackgroundPartEnum.exVSplitterApp,15790320)
```

```
.AllowSplitter = EXPRESSIONLib.SplitterEnum.exVSplitter
```

End With

C++

```
// Change event - Indicates that the control's text has changed.
```

```
void OnChangeExpression1()
```

```
{
```

```
/*
```

```
Copy and paste the following directives to your header file as
```

```
it defines the namespace 'EXPRESSIONLib' for the library: 'Expression 1.0 Control
```

```
Library'
```

```
#import <Expression.dll>
```

```
using namespace EXPRESSIONLib;
```

```
*/
```

```
EXPRESSIONLib::IExpressionPtr spExpression1 = GetDlgItem(IDC_EXPRESSION1)-  
>GetControlUnknown();
```

```
OutputDebugStringW( L"Valid: " );
```

```
OutputDebugStringW( _bstr_t(spExpression1->GetIsValid()) );
```

```
}
```

```
EXPRESSIONLib::IExpressionPtr spExpression1 = GetDlgItem(IDC_EXPRESSION1)-  
>GetControlUnknown();
```

```
spExpression1->PutExpression(L"value");
```

```
spExpression1->PutSplitPaneHeight(196);
```

```
spExpression1->PutBackground(EXPRESSIONLib::exVSplitterApp,RGB(240,240,240));
```

```
spExpression1->PutAllowSplitter(EXPRESSIONLib::exVSplitter);
```

C++ Builder

```
// Change event - Indicates that the control's text has changed.
```

```
void __fastcall TForm1::Expression1Change(TObject *Sender)
```

```
{
```

```

OutputDebugString( L"Valid: " );
OutputDebugString( PChar(Expression1->IsValid) );
}

Expression1->Expression = L"value";
Expression1->SplitPaneHeight = 196;
Expression1->Background[Expressionlib_tlb::BackgroundPartEnum::exVSplitterApp] =
RGB(240,240,240);
Expression1->AllowSplitter = Expressionlib_tlb::SplitterEnum::exVSplitter;

```

C#

```

// Change event - Indicates that the control's text has changed.
private void expression1_Change(object sender)
{
    System.Diagnostics.Debug.Print( "Valid: " );
    System.Diagnostics.Debug.Print( expression1.IsValid.ToString() );
}

//this.expression1.Change += new
exontrol.EXPRESSIONLib.exg2antt.ChangeEventHandler(this.expression1_Change);

expression1.Expression = "value";
expression1.SplitPaneHeight = 196;
expression1.set_Background(exontrol.EXPRESSIONLib.BackgroundPartEnum.exVSplitte

expression1.AllowSplitter = exontrol.EXPRESSIONLib.SplitterEnum.exVSplitter;

```

JScript/JavaScript

```

<BODY onload="Init()" >
<SCRIPT FOR="Expression1" EVENT="Change()" LANGUAGE="JScript" >
    alert( "Valid: " );
    alert( Expression1.IsValid );
</SCRIPT>

<OBJECT CLASSID="clsid:B33F5489-49AC-4155-98E7-9BBFC57FF019"

```

```
id="Expression1"></OBJECT>
```

```
<SCRIPT LANGUAGE="JScript">
```

```
function Init()
```

```
{  
    Expression1.Expression = "value";  
    Expression1.SplitPaneHeight = 196;  
    Expression1.Background(1) = 15790320;  
    Expression1.AllowSplitter = 2;
```

```
}  
</SCRIPT>
```

```
</BODY>
```

VBScript

```
<BODY onload="Init()">
```

```
<SCRIPT LANGUAGE="VBScript">
```

```
Function Expression1_Change()
```

```
    With Expression1
```

```
        alert( "Valid: " )
```

```
        alert( .IsValid )
```

```
    End With
```

```
End Function
```

```
</SCRIPT>
```

```
<OBJECT CLASSID="clsid:B33F5489-49AC-4155-98E7-9BBFC57FF019"
```

```
id="Expression1"></OBJECT>
```

```
<SCRIPT LANGUAGE="VBScript">
```

```
Function Init()
```

```
    With Expression1
```

```
        .Expression = "value"
```

```
        .SplitPaneHeight = 196
```

```
        .Background(1) = RGB(240,240,240)
```

```
        .AllowSplitter = 2
```

```
    End With
```

End Function

</SCRIPT>

</BODY>

C# for /COM

```
// Change event - Indicates that the control's text has changed.
private void axExpression1_Change(object sender, EventArgs e)
{
    System.Diagnostics.Debug.Print( "Valid: " );
    System.Diagnostics.Debug.Print( axExpression1.IsValid.ToString() );
}
//this.axExpression1.Change += new EventHandler(this.axExpression1_Change);

axExpression1.Expression = "value";
axExpression1.SplitPaneHeight = 196;
axExpression1.set_Background(EXPRESSIONLib.BackgroundPartEnum.exVSplitterApp,
(uint)ColorTranslator.ToWin32(Color.FromArgb(240,240,240)));
axExpression1.AllowSplitter = EXPRESSIONLib.SplitterEnum.exVSplitter;
```

X++ (Dynamics Ax 2009)

```
// Change event - Indicates that the control's text has changed.
void onEvent_Change()
{
    ;
    print( "Valid: " );
    print( expression1.IsValid() );
}

public void init()
{
    ;

    super();
}
```

```
expression1.Expression("value");
expression1.SplitPaneHeight(196);
expression1.Background(1/*exVSplitterApp*/,WinApi::RGB2int(240,240,240));
expression1.AllowSplitter(2/*exVSplitter*/);
}
```

Delphi 8 (.NET only)

```
// Change event - Indicates that the control's text has changed.
procedure TForm1.AxExpression1_Change(sender: System.Object; e:
System.EventArgs);
begin
  with AxExpression1 do
  begin
    OutputDebugString( 'Valid: ' );
    OutputDebugString( IsValid );
  end
end;

with AxExpression1 do
begin
  Expression := 'value';
  SplitPaneHeight := 196;
  set_Background(EXPRESSIONLib.BackgroundPartEnum.exVSplitterApp,$f0f0f0);
  AllowSplitter := EXPRESSIONLib.SplitterEnum.exVSplitter;
end
```

Delphi (standard)

```
// Change event - Indicates that the control's text has changed.
procedure TForm1.Expression1Change(ASender: TObject; );
begin
  with Expression1 do
  begin
    OutputDebugString( 'Valid: ' );
    OutputDebugString( IsValid );
  end
end;
```

```

with Expression1 do
begin
    Expression := 'value';
    SplitPaneHeight := 196;
    Background[EXPRESSIONLib_TLB.exVSplitterApp] := $f0f0f0;
    AllowSplitter := EXPRESSIONLib_TLB.exVSplitter;
end

```

VFP

*** Change event - Indicates that the control's text has changed. ***

```

LPARAMETERS nop
with thisform.Expression1
    DEBUGOUT( "Valid: " )
    DEBUGOUT( .IsValid )
endwith

with thisform.Expression1
    .Expression = "value"
    .SplitPaneHeight = 196
    .Object.Background(1) = RGB(240,240,240)
    .AllowSplitter = 2
endwith

```

dBASE Plus

```

/*
with (this.EXPRESSIONACTIVEXCONTROL1.nativeObject)
    Change = class::nativeObject_Change
endwith
*/
// Indicates that the control's text has changed.
function nativeObject_Change()
    local oExpression
    oExpression = form.EXPRESSIONACTIVEXCONTROL1.nativeObject
    ? "Valid: "
    ? Str(oExpression.IsValid)

```

```
return
```

```
local oExpression
```

```
oExpression = form.EXPRESSIONACTIVEXCONTROL1.nativeObject
```

```
oExpression.Expression = "value"
```

```
oExpression.SplitPaneHeight = 196
```

```
oExpression.Template = [Background(1) = 15790320] // oExpression.Background(1) =  
0xf0f0f0
```

```
oExpression.AllowSplitter = 2
```

XBasic (Alpha Five)

```
' Indicates that the control's text has changed.
```

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

```
    Dim oExpression as P
```

```
    oExpression = topparent:CONTROL_ACTIVEX1.activex
```

```
    ? "Valid: "
```

```
    ? oExpression.IsValid
```

```
end function
```

```
Dim oExpression as P
```

```
oExpression = topparent:CONTROL_ACTIVEX1.activex
```

```
oExpression.Expression = "value"
```

```
oExpression.SplitPaneHeight = 196
```

```
oExpression.Template = "Background(1) = 15790320" // oExpression.Background(1)  
= 15790320
```

```
oExpression.AllowSplitter = 2
```

Visual Objects

```
METHOD OCX_Exontrol1Change() CLASS MainDialog
```

```
    // Change event - Indicates that the control's text has changed.
```

```
    OutputDebugString(String2Psz( "Valid: " ))
```

```
    OutputDebugString(String2Psz( AsString(oDCOCX_Exontrol1:IsValid) ))
```

```
RETURN NIL
```

```
oDCOCX_Exontrol1:Expression := "value"  
oDCOCX_Exontrol1:SplitPaneHeight := 196  
oDCOCX_Exontrol1:[Background,exVSplitterApp] := RGB(240,240,240)  
oDCOCX_Exontrol1:AllowSplitter := exVSplitter
```

PowerBuilder

```
/*begin event Change() - Indicates that the control's text has changed.*/  
/*  
    OleObject oExpression  
    oExpression = ole_1.Object  
    MessageBox("Information",string( "Valid: " ))  
    MessageBox("Information",string( String(oExpression.IsValid) ))  
*/  
/*end event Change*/
```

```
OleObject oExpression  
  
oExpression = ole_1.Object  
oExpression.Expression = "value"  
oExpression.SplitPaneHeight = 196  
oExpression.Background(1,RGB(240,240,240))  
oExpression.AllowSplitter = 2
```

Visual DataFlex

```
// Indicates that the control's text has changed.  
Procedure OnComChange  
    Forward Send OnComChange  
    ShowIn "Valid: " (ComIsValid(Self))  
End_Procedure  
  
Procedure OnCreate
```

```
Forward Send OnCreate
Set ComExpression to "value"
Set ComSplitPaneHeight to 196
Set ComBackground OLEexVSplitterApp to (RGB(240,240,240))
Set ComAllowSplitter to OLEexVSplitter
End_Procedure
```

XBase++

```
PROCEDURE OnChange(oExpression)
  DevOut( "Valid: " )
  DevOut( Transform(oExpression:IsValid(),"") )
RETURN

#include "AppEvent.ch"
#include "ActiveX.ch"

PROCEDURE Main
  LOCAL oForm
  LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
  LOCAL oExpression

  oForm := XbpDialog():new( AppDesktop() )
  oForm:drawingArea:clipChildren := .T.
  oForm:create( ,, {100,100}, {640,480},,, .F. )
  oForm:close := {|| PostAppEvent( xbeP_Quit )}

  oExpression := XbpActiveXControl():new( oForm:drawingArea )
  oExpression:CLSID := "Exontrol.Expression.1" /*{B33F5489-49AC-4155-98E7-
9BBFC57FF019}*/
  oExpression:create(,, {10,60},{610,370} )

  oExpression:Change := {|| OnChange(oExpression)} /*Indicates that the control's
text has changed.*/*

  oExpression:Expression := "value"
  oExpression:SplitPaneHeight := 196
```

```
oExpression:SetProperty("Background",1/*exVSplitterApp*/,AutomationTranslateColor  
GraMakeRGBColor ( { 240,240,240 } ) , .F. ))  
    oExpression:AllowSplitter := 2/*exVSplitter*/  
  
oForm:Show()  
DO WHILE nEvent != xbeP_Quit  
    nEvent := AppEvent( @mp1, @mp2, @oXbp )  
    oXbp:handleEvent( nEvent, mp1, mp2 )  
ENDDO  
RETURN
```

event Click ()

Occurs when the user presses and then releases the left mouse button over the list 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. The [WordFromCursor\(-1,-1\)](#) property indicates the word from the cursor.

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;
```

Powe...
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(oExpression)
RETURN

Syntax for Click event, **ICOM** 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
```

The following samples determines the word from the cursor:

VBA (MS Access, Excell...)

' Click event - Occurs when the user presses and then releases the left mouse button over the list control.

```
Private Sub Expression1_Click()  
    With Expression1  
        Debug.Print( "Word: " )  
        Debug.Print( .WordFromPoint(-1,-1) )  
    End With  
End Sub
```

```
With Expression1  
    .Expression = "value"  
    .SplitPaneHeight = 196  
    .Background(1) = RGB(240,240,240)  
    .AllowSplitter = 2  
End With
```

VB6

' Click event - Occurs when the user presses and then releases the left mouse button

over the list control.

```
Private Sub Expression1_Click()
```

```
    With Expression1
```

```
        Debug.Print( "Word: " )
```

```
        Debug.Print( .WordFromPoint(-1,-1) )
```

```
    End With
```

```
End Sub
```

```
With Expression1
```

```
    .Expression = "value"
```

```
    .SplitPaneHeight = 196
```

```
    .Background(exVSplitterApp) = RGB(240,240,240)
```

```
    .AllowSplitter = exVSplitter
```

```
End With
```

VB.NET

' Click event - Occurs when the user presses and then releases the left mouse button over the list control.

```
Private Sub Expression1_Click(ByVal sender As System.Object) Handles
```

```
Expression1.Click
```

```
    With Expression1
```

```
        Debug.Print( "Word: " )
```

```
        Debug.Print( .get_WordFromPoint(-1,-1) )
```

```
    End With
```

```
End Sub
```

```
With Expression1
```

```
    .Expression = "value"
```

```
    .SplitPaneHeight = 196
```

```
    .set_Background(exontrol.EXPRESSIONLib.BackgroundPartEnum.exVSplitterApp,Color.F
```

```
    .AllowSplitter = exontrol.EXPRESSIONLib.SplitterEnum.exVSplitter
```

```
End With
```

VB.NET for /COM

' Click event - Occurs when the user presses and then releases the left mouse button over the list control.

```
Private Sub AxExpression1_ClickEvent(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles AxExpression1.ClickEvent
```

```
    With AxExpression1
```

```
        Debug.Print( "Word: " )
```

```
        Debug.Print( .get_WordFromPoint(-1,-1) )
```

```
    End With
```

```
End Sub
```

```
With AxExpression1
```

```
    .Expression = "value"
```

```
    .SplitPaneHeight = 196
```

```
    .set_Background(EXPRESSIONLib.BackgroundPartEnum.exVSplitterApp,15790320)
```

```
    .AllowSplitter = EXPRESSIONLib.SplitterEnum.exVSplitter
```

```
End With
```

C++

// Click event - Occurs when the user presses and then releases the left mouse button over the list control.

```
void OnClickExpression1()
```

```
{
```

```
    /*
```

```
        Copy and paste the following directives to your header file as
```

```
        it defines the namespace 'EXPRESSIONLib' for the library: 'Expression 1.0 Control
```

```
Library'
```

```
        #import <Expression.dll>
```

```
        using namespace EXPRESSIONLib;
```

```
    */
```

```
    EXPRESSIONLib::IExpressionPtr spExpression1 = GetDlgItem(IDC_EXPRESSION1)->GetControlUnknown();
```

```
    OutputDebugStringW( L"Word: " );
```

```
    OutputDebugStringW( spExpression1->GetWordFromPoint(-1,-1,vtMissing) );
```

```
}
```

```
EXPRESSIONLib::IExpressionPtr spExpression1 = GetDlgItem(IDC_EXPRESSION1)-
```

```
> GetControlUnknown();
spExpression1->PutExpression(L"value");
spExpression1->PutSplitPaneHeight(196);
spExpression1->PutBackground(EXPRESSIONLib::exVSplitterApp,RGB(240,240,240));
spExpression1->PutAllowSplitter(EXPRESSIONLib::exVSplitter);
```

C++ Builder

// Click event - Occurs when the user presses and then releases the left mouse button over the list control.

```
void __fastcall TForm1::Expression1Click(TObject *Sender)
```

```
{
    OutputDebugString( L"Word: " );
    OutputDebugString( Expression1->WordFromPoint[-1,-1,TNoParam()] );
}
```

```
Expression1->Expression = L"value";
```

```
Expression1->SplitPaneHeight = 196;
```

```
Expression1->Background[Expressionlib_tlb::BackgroundPartEnum::exVSplitterApp] =
RGB(240,240,240);
```

```
Expression1->AllowSplitter = Expressionlib_tlb::SplitterEnum::exVSplitter;
```

C#

// Click event - Occurs when the user presses and then releases the left mouse button over the list control.

```
private void expression1_Click(object sender)
```

```
{
    System.Diagnostics.Debug.Print( "Word: " );
    System.Diagnostics.Debug.Print( expression1.get_WordFromPoint(-1,-1,null) );
}
```

```
//this.expression1.Click += new
```

```
exontrol.EXPRESSIONLib.exg2antt.ClickEventHandler(this.expression1_Click);
```

```
expression1.Expression = "value";
```

```
expression1.SplitPaneHeight = 196;
```

```
expression1.set_Background(exontrol.EXPRESSIONLib.BackgroundPartEnum.exVSplitter
```

```
expression1.AllowSplitter = exontrol.EXPRESSIONLib.SplitterEnum.exVSplitter;
```

JScript/JavaScript

```
<BODY onload="Init()">  
<SCRIPT FOR="Expression1" EVENT="Click()" LANGUAGE="JScript">  
  alert( "Word: " );  
  alert( Expression1.WordFromPoint(-1,-1,null) );  
</SCRIPT>  
  
<OBJECT CLASSID="clsid:B33F5489-49AC-4155-98E7-9BBFC57FF019"  
id="Expression1"></OBJECT>  
  
<SCRIPT LANGUAGE="JScript">  
function Init()  
{  
  Expression1.Expression = "value";  
  Expression1.SplitPaneHeight = 196;  
  Expression1.Background(1) = 15790320;  
  Expression1.AllowSplitter = 2;  
}  
</SCRIPT>  
</BODY>
```

VBScript

```
<BODY onload="Init()">  
<SCRIPT LANGUAGE="VBScript">  
Function Expression1_Click()  
  With Expression1  
    alert( "Word: " )  
    alert( .WordFromPoint(-1,-1) )  
  End With  
End Function
```

```
</SCRIPT>
```

```
<OBJECT CLASSID="clsid:B33F5489-49AC-4155-98E7-9BBFC57FF019"  
id="Expression1"></OBJECT>
```

```
<SCRIPT LANGUAGE="VBScript">
```

```
Function Init()
```

```
With Expression1
```

```
.Expression = "value"
```

```
.SplitPaneHeight = 196
```

```
.Background(1) = RGB(240,240,240)
```

```
.AllowSplitter = 2
```

```
End With
```

```
End Function
```

```
</SCRIPT>
```

```
</BODY>
```

C# for /COM

```
// Click event - Occurs when the user presses and then releases the left mouse button  
over the list control.
```

```
private void axExpression1_ClickEvent(object sender, EventArgs e)
```

```
{
```

```
System.Diagnostics.Debug.Print( "Word: " );
```

```
System.Diagnostics.Debug.Print( axExpression1.get_WordFromPoint(-1,-1,null) );
```

```
}
```

```
//this.axExpression1.ClickEvent += new EventHandler(this.axExpression1_ClickEvent);
```

```
axExpression1.Expression = "value";
```

```
axExpression1.SplitPaneHeight = 196;
```

```
axExpression1.set_Background(EXPRESSIONLib.BackgroundPartEnum.exVSplitterApp,  
(uint)ColorTranslator.ToWin32(Color.FromArgb(240,240,240)));
```

```
axExpression1.AllowSplitter = EXPRESSIONLib.SplitterEnum.exVSplitter;
```

X++ (Dynamics Ax 2009)

```

// Click event - Occurs when the user presses and then releases the left mouse button
over the list control.
void onEvent_Click()
{
    ;
    print( "Word: " );
    print( expression1.WordFromPoint(-1,-1) );
}

public void init()
{
    ;

    super();

    expression1.Expression("value");
    expression1.SplitPaneHeight(196);
    expression1.Background(1/*exVSplitterApp*/,WinApi::RGB2int(240,240,240));
    expression1.AllowSplitter(2/*exVSplitter*/);
}

```

Delphi 8 (.NET only)

```

// Click event - Occurs when the user presses and then releases the left mouse button
over the list control.
procedure TWinForm1.AxExpression1_ClickEvent(sender: System.Object; e:
System.EventArgs);
begin
    with AxExpression1 do
        begin
            OutputDebugString( 'Word: ' );
            OutputDebugString( get_WordFromPoint(-1,-1,Nil) );
        end
    end;

    with AxExpression1 do
        begin

```

```

Expression := 'value';
SplitPaneHeight := 196;
set_Background(EXPRESSIONLib.BackgroundPartEnum.exVSplitterApp,$f0f0f0);
AllowSplitter := EXPRESSIONLib.SplitterEnum.exVSplitter;
end

```

Delphi (standard)

// Click event - Occurs when the user presses and then releases the left mouse button over the list control.

```

procedure TForm1.Expression1Click(ASender: TObject; );
begin
  with Expression1 do
  begin
    OutputDebugString( 'Word: ' );
    OutputDebugString( WordFromPoint[-1,-1,Null] );
  end
end;

with Expression1 do
begin
  Expression := 'value';
  SplitPaneHeight := 196;
  Background[EXPRESSIONLib_TLB.exVSplitterApp] := $f0f0f0;
  AllowSplitter := EXPRESSIONLib_TLB.exVSplitter;
end

```

VFP

*** Click event - Occurs when the user presses and then releases the left mouse button over the list control. ***

```

LPARAMETERS nop
  with thisform.Expression1
    DEBUGOUT( "Word: " )
    DEBUGOUT( WordFromPoint(-1,-1) )
  endwith

with thisform.Expression1

```

```
.Expression = "value"  
.SplitPaneHeight = 196  
.Object.Background(1) = RGB(240,240,240)  
.AllowSplitter = 2  
endwith
```

dBASE Plus

```
/*  
with (this.EXPRESSIONACTIVEXCONTROL1.nativeObject)  
    Click = class::nativeObject_Click  
endwith  
*/  
// Occurs when the user presses and then releases the left mouse button over the list  
control.  
function nativeObject_Click()  
    local oExpression  

```

XBasic (Alpha Five)

```
' Occurs when the user presses and then releases the left mouse button over the list  
control.  
function Click as v ()  
    Dim oExpression as P
```

```
oExpression = topparent:CONTROL_ACTIVEX1.activex
? "Word: "
? oExpression.WordFromPoint(-1,-1)
end function
```

Dim oExpression as P

```
oExpression = topparent:CONTROL_ACTIVEX1.activex
oExpression.Expression = "value"
oExpression.SplitPaneHeight = 196
oExpression.Template = "Background(1) = 15790320" // oExpression.Background(1)
= 15790320
oExpression.AllowSplitter = 2
```

Visual Objects

```
METHOD OCX_Exontrol1Click() CLASS MainDialog
  // Click event - Occurs when the user presses and then releases the left mouse
  // button over the list control.
  OutputDebugString(String2Psz( "Word: " ))
  OutputDebugString(String2Psz( oDCOCX_Exontrol1:[WordFromPoint,-1,-1,nil] ))
RETURN NIL
```

```
oDCOCX_Exontrol1.Expression := "value"
oDCOCX_Exontrol1.SplitPaneHeight := 196
oDCOCX_Exontrol1:[Background,exVSplitterApp] := RGB(240,240,240)
oDCOCX_Exontrol1.AllowSplitter := exVSplitter
```

PowerBuilder

```
/*begin event Click() - Occurs when the user presses and then releases the left mouse
button over the list control.*/
/*
  OleObject oExpression
  oExpression = ole_1.Object
```

```
    MessageBox("Information",string( "Word: " ))
    MessageBox("Information",string( oExpression.WordFromPoint(-1,-1) ))
*/
/*end event Click*/
```

OleObject oExpression

```
oExpression = ole_1.Object
oExpression.Expression = "value"
oExpression.SplitPaneHeight = 196
oExpression.Background(1,RGB(240,240,240))
oExpression.AllowSplitter = 2
```

Visual DataFlex

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

```
Procedure OnComClick
```

```
    Forward Send OnComClick
```

```
    ShowLn "Word: " (ComWordFromPoint(Self,-1,-1,Nothing))
```

```
End_Procedure
```

```
Procedure OnCreate
```

```
    Forward Send OnCreate
```

```
    Set ComExpression to "value"
```

```
    Set ComSplitPaneHeight to 196
```

```
    Set ComBackground OLEexVSplitterApp to (RGB(240,240,240))
```

```
    Set ComAllowSplitter to OLEexVSplitter
```

```
End_Procedure
```

XBase++

```
PROCEDURE OnClick(oExpression)
```

```
    DevOut( "Word: " )
```

```
    DevOut( oExpression:WordFromPoint(-1,-1) )
```

```
RETURN
```

```
#include "AppEvent.ch"
```

```
#include "ActiveX.ch"
```

```
PROCEDURE Main
```

```
    LOCAL oForm
```

```
    LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
```

```
    LOCAL oExpression
```

```
    oForm := XbpDialog():new( AppDesktop() )
```

```
    oForm:drawingArea:clipChildren := .T.
```

```
    oForm:create( ,, {100,100}, {640,480},,, .F. )
```

```
    oForm:close := {|| PostAppEvent( xbeP_Quit )}
```

```
    oExpression := XbpActiveXControl():new( oForm:drawingArea )
```

```
    oExpression:CLSID := "Exontrol.Expression.1" /*{B33F5489-49AC-4155-98E7-9BBFC57FF019}*/
```

```
    oExpression:create(,, {10,60},{610,370} )
```

```
        oExpression:Click := {|| OnClick(oExpression)} /*Occurs when the user presses and then releases the left mouse button over the list control.*/*
```

```
        oExpression:Expression := "value"
```

```
        oExpression:SplitPaneHeight := 196
```

```
oExpression:SetProperty("Background",1/*exVSplitterApp*/,AutomationTranslateColor  
GraMakeRGBColor ( { 240,240,240 } ) , .F. ))
```

```
    oExpression:AllowSplitter := 2/*exVSplitter*/
```

```
oForm:Show()
```

```
DO WHILE nEvent != xbeP_Quit
```

```
    nEvent := AppEvent( @mp1, @mp2, @oXbp )
```

```
    oXbp:handleEvent( nEvent, mp1, mp2 )
```

```
ENDDO
```

```
RETURN
```

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

Occurs when the user double clicks 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 DbClick 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. The [WordFromCursor\(-1,-1\)](#) property indicates the word from the cursor.

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

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

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

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

```
C# private void DbClick(object sender,
AxEXPRESSIONLib._IExpressionEvents_DbClickEvent e)
{
}
```

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

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

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

```
Delphi 8 (.NET only) procedure DblClick(sender: System.Object; e:
AxEXPRESSIONLib._IExpressionEvents_DblClickEvent);
begin
end;
```

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

```
VB.NET Private Sub DblClick(ByVal sender As System.Object, ByVal e As
AxEXPRESSIONLib._IExpressionEvents_DblClickEvent) Handles DblClick
End Sub
```

```
VB6 Private Sub DblClick(Shift As Integer,X As Single,Y As Single)
End Sub
```

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

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

```
Xbase PROCEDURE OnDblClick(oExpression,Shift,X,Y)
RETURN
```

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

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

```
VBSc... <SCRIPT LANGUAGE="VBScript">
```

```
Function DblClick(Shift,X,Y)
End Function
</SCRIPT>
```

Visual
Data...

```
Procedure 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.Expression.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.Expression.1::OLE_YPOS_PIXELS)
end function
```

dBASE

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

The following samples get the word from the cursor when the user double clicks the control:

VBA (MS Access, Excell...)

```
' DblClick event - Occurs when the user double clicks the left mouse button over an
object.
Private Sub Expression1_DblClick(ByVal Shift As Integer,ByVal X As Long,ByVal Y As
Long)
    With Expression1
        Debug.Print( "Word: " )
        Debug.Print( .WordFromPoint(-1,-1) )
    End With
End Sub
```

```
With Expression1
    .Expression = "value"
    .AllowSplitter = 2
    .SplitPaneHeight = 196
End With
```

VB6

' DblClick event - Occurs when the user double clicks the left mouse button over an object.

```
Private Sub Expression1_DblClick(Shift As Integer,X As Single,Y As Single)
    With Expression1
        Debug.Print( "Word: " )
        Debug.Print( .WordFromPoint(-1,-1) )
    End With
End Sub
```

```
With Expression1
    .Expression = "value"
    .AllowSplitter = exVSplitter
    .SplitPaneHeight = 196
End With
```

VB.NET

' DblClick event - Occurs when the user double clicks the left mouse button over an object.

```
Private Sub Expression1_DblClick(ByVal sender As System.Object,ByVal Shift As Short,ByVal X As Integer,ByVal Y As Integer) Handles Expression1.DblClick
    With Expression1
        Debug.Print( "Word: " )
        Debug.Print( .get_WordFromPoint(-1,-1) )
    End With
End Sub
```

```
With Expression1
    .Expression = "value"
    .AllowSplitter = exontrol.EXPRESSIONLib.SplitterEnum.exVSplitter
```

```
.SplitPaneHeight = 196  
End With
```

VB.NET for /COM

' DbClick event - Occurs when the user double clicks the left mouse button over an object.

```
Private Sub AxExpression1_DblClick(ByVal sender As System.Object, ByVal e As  
AxEXPRESSIONLib._IExpressionEvents_DblClickEvent) Handles AxExpression1.DblClick  
    With AxExpression1  
        Debug.Print( "Word: " )  
        Debug.Print( .get_WordFromPoint(-1,-1) )  
    End With  
End Sub
```

```
With AxExpression1  
    .Expression = "value"  
    .AllowSplitter = EXPRESSIONLib.SplitterEnum.exVSplitter  
    .SplitPaneHeight = 196  
End With
```

C++

```
// DbClick event - Occurs when the user double clicks the left mouse button over an  
object.  
void OnDbClickExpression1(short Shift,long X,long Y)  
{  
    /*  
        Copy and paste the following directives to your header file as  
        it defines the namespace 'EXPRESSIONLib' for the library: 'Expression 1.0 Control  
Library'  
        #import <Expression.dll>  
        using namespace EXPRESSIONLib;  
    */  
    EXPRESSIONLib::IExpressionPtr spExpression1 = GetDlgItem(IDC_EXPRESSION1)-  
>GetControlUnknown();  
    OutputDebugStringW( L"Word: " );  
    OutputDebugStringW( spExpression1->GetWordFromPoint(-1,-1,vtMissing) );  
}
```

```
}
```

```
EXPRESSIONLib::IExpressionPtr spExpression1 = GetDlgItem(IDC_EXPRESSION1)-  
> GetControlUnknown();  
spExpression1->PutExpression(L"value");  
spExpression1->PutAllowSplitter(EXPRESSIONLib::exVSplitter);  
spExpression1->PutSplitPaneHeight(196);
```

C++ Builder

```
// DbClick event - Occurs when the user double clicks the left mouse button over an  
object.
```

```
void __fastcall TForm1::Expression1DbClick(TObject *Sender,short Shift,int X,int Y)  
{  
    OutputDebugString( L"Word: " );  
    OutputDebugString( Expression1->WordFromPoint[-1,-1,TNoParam()] );  
}
```

```
Expression1->Expression = L"value";  
Expression1->AllowSplitter = Expressionlib_tlb::SplitterEnum::exVSplitter;  
Expression1->SplitPaneHeight = 196;
```

C#

```
// DbClick event - Occurs when the user double clicks the left mouse button over an  
object.
```

```
private void expression1_DbClick(object sender,short Shift,int X,int Y)  
{  
    System.Diagnostics.Debug.Print( "Word: " );  
    System.Diagnostics.Debug.Print( expression1.get_WordFromPoint(-1,-1,null) );  
}  
//this.expression1.DbClick += new  
exontrol.EXPRESSIONLib.exg2antt.DbClickEventHandler(this.expression1_DbClick);
```

```
expression1.Expression = "value";  
expression1.AllowSplitter = exontrol.EXPRESSIONLib.SplitterEnum.exVSplitter;
```

```
expression1.SplitPaneHeight = 196;
```

JScript/JavaScript

```
<BODY onload="Init()">  
<SCRIPT FOR="Expression1" EVENT="DbClick(Shift,X,Y)" LANGUAGE="JScript">  
  alert( "Word: " );  
  alert( Expression1.WordFromPoint(-1,-1,null) );  
</SCRIPT>  
  
<OBJECT CLASSID="clsid:B33F5489-49AC-4155-98E7-9BBFC57FF019"  
id="Expression1"></OBJECT>  
  
<SCRIPT LANGUAGE="JScript">  
function Init()  
{  
  Expression1.Expression = "value";  
  Expression1.AllowSplitter = 2;  
  Expression1.SplitPaneHeight = 196;  
}  
</SCRIPT>  
</BODY>
```

VBScript

```
<BODY onload="Init()">  
<SCRIPT LANGUAGE="VBScript">  
Function Expression1_DbClick(Shift,X,Y)  
  With Expression1  
    alert( "Word: " )  
    alert( .WordFromPoint(-1,-1) )  
  End With  
End Function  
</SCRIPT>  
  
<OBJECT CLASSID="clsid:B33F5489-49AC-4155-98E7-9BBFC57FF019"
```

```
id="Expression1"></OBJECT>
```

```
<SCRIPT LANGUAGE="VBScript">
```

```
Function Init()
```

```
    With Expression1
```

```
        .Expression = "value"
```

```
        .AllowSplitter = 2
```

```
        .SplitPaneHeight = 196
```

```
    End With
```

```
End Function
```

```
</SCRIPT>
```

```
</BODY>
```

C# for /COM

```
// DbClick event - Occurs when the user double clicks the left mouse button over an object.
```

```
private void axExpression1_DblClick(object sender,  
AxEXPRESSIONLib._IExpressionEvents_DblClickEvent e)
```

```
{
```

```
    System.Diagnostics.Debug.Print( "Word: " );
```

```
    System.Diagnostics.Debug.Print( axExpression1.get_WordFromPoint(-1,-1,null) );
```

```
}
```

```
//this.axExpression1.DblClick += new
```

```
AxEXPRESSIONLib._IExpressionEvents_DblClickEventHandler(this.axExpression1_DblClick
```

```
axExpression1.Expression = "value";
```

```
axExpression1.AllowSplitter = EXPRESSIONLib.SplitterEnum.exVSplitter;
```

```
axExpression1.SplitPaneHeight = 196;
```

X++ (Dynamics Ax 2009)

```
// DbClick event - Occurs when the user double clicks the left mouse button over an object.
```

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

```

{
;
print( "Word: " );
print( expression1.WordFromPoint(-1,-1) );
}

public void init()
{
;

super();

expression1.Expression("value");
expression1.AllowSplitter(2/*exVSplitter*/);
expression1.SplitPaneHeight(196);
}

```

Delphi 8 (.NET only)

```

// DbClick event - Occurs when the user double clicks the left mouse button over an
// object.
procedure TForm1.AxExpression1_DblClick(sender: System.Object; e:
AxEXPRESSIONLib._IExpressionEvents_DblClickEvent);
begin
with AxExpression1 do
begin
OutputDebugString( 'Word: ' );
OutputDebugString( get_WordFromPoint(-1,-1,Nil) );
end
end;

with AxExpression1 do
begin
Expression := 'value';
AllowSplitter := EXPRESSIONLib.SplitterEnum.exVSplitter;
SplitPaneHeight := 196;
end

```

Delphi (standard)

```
// DbClick event - Occurs when the user double clicks the left mouse button over an
object.
procedure TForm1.Expression1DbClick(ASender: TObject; Shift : Smallint;X : Integer;Y
: Integer);
begin
  with Expression1 do
  begin
    OutputDebugString( 'Word: ' );
    OutputDebugString( WordFromPoint[-1,-1,Null] );
  end
end;

with Expression1 do
begin
  Expression := 'value';
  AllowSplitter := EXPRESSIONLib_TLB.exVSplitter;
  SplitPaneHeight := 196;
end
```

VFP

```
*** DbClick event - Occurs when the user double clicks the left mouse button over an
object. ***
LPARAMETERS Shift,X,Y
  with thisform.Expression1
    DEBUGOUT( "Word: " )
    DEBUGOUT( WordFromPoint(-1,-1) )
  endwith

with thisform.Expression1
  .Expression = "value"
  .AllowSplitter = 2
  .SplitPaneHeight = 196
endwith
```

dBASE Plus

```

/*
with (this.EXPRESSIONACTIVEXCONTROL1.nativeObject)
    DbClick = class::nativeObject_DbClick
endwith
*/
// Occurs when the user double clicks the left mouse button over an object.
function nativeObject_DbClick(Shift,X,Y)
    local oExpression
    oExpression = form.EXPRESSIONACTIVEXCONTROL1.nativeObject
    ? "Word: "
    ? oExpression.WordFromPoint(-1,-1)
return

local oExpression

oExpression = form.EXPRESSIONACTIVEXCONTROL1.nativeObject
oExpression.Expression = "value"
oExpression.AllowSplitter = 2
oExpression.SplitPaneHeight = 196

```

XBasic (Alpha Five)

```

' Occurs when the user double clicks the left mouse button over an object.
function DbClick as v (Shift as N,X as OLE::Exontrol.Expression.1::OLE_XPOS_PIXELS,Y
as OLE::Exontrol.Expression.1::OLE_YPOS_PIXELS)
    Dim oExpression as P
    oExpression = topparent:CONTROL_ACTIVEX1.activex
    ? "Word: "
    ? oExpression.WordFromPoint(-1,-1)
end function

Dim oExpression as P

oExpression = topparent:CONTROL_ACTIVEX1.activex
oExpression.Expression = "value"
oExpression.AllowSplitter = 2

```

```
oExpression.SplitPaneHeight = 196
```

Visual Objects

```
METHOD OCX_Exontrol1DbfClick(Shift,X,Y) CLASS MainDialog
  // DbfClick event - Occurs when the user double clicks the left mouse button over
  an object.
  OutputDebugString(String2Psz( "Word: " ))
  OutputDebugString(String2Psz( oDCOCX_Exontrol1:[WordFromPoint,-1,-1,nil] ))
RETURN NIL

oDCOCX_Exontrol1:Expression := "value"
oDCOCX_Exontrol1:AllowSplitter := exVSplitter
oDCOCX_Exontrol1:SplitPaneHeight := 196
```

PowerBuilder

```
/*begin event DbfClick(integer Shift,long X,long Y) - Occurs when the user double
clicks the left mouse button over an object.*/
/*
  OleObject oExpression
  oExpression = ole_1.Object
  MessageBox("Information",string( "Word: " ))
  MessageBox("Information",string( oExpression.WordFromPoint(-1,-1) ))
*/
/*end event DbfClick*/

OleObject oExpression

oExpression = ole_1.Object
oExpression.Expression = "value"
oExpression.AllowSplitter = 2
oExpression.SplitPaneHeight = 196
```

Visual DataFlex

```
// Occurs when the user double clicks the left mouse button over an object.
```

```
Procedure OnComDbfClick Short IIShift OLE_XPOS_PIXELS IIX OLE_YPOS_PIXELS IIY  
  Forward Send OnComDbfClick IIShift IIX IIY  
  ShowIn "Word: " (ComWordFromPoint(Self,-1,-1,Nothing))  
End_Procedure
```

```
Procedure OnCreate  
  Forward Send OnCreate  
  Set ComExpression to "value"  
  Set ComAllowSplitter to OLEexVSplitter  
  Set ComSplitPaneHeight to 196  
End_Procedure
```

XBase++

```
PROCEDURE OnDbfClick(oExpression,Shift,X,Y)  
  DevOut( "Word: " )  
  DevOut( oExpression:WordFromPoint(-1,-1) )  
RETURN
```

```
#include "AppEvent.ch"  
#include "ActiveX.ch"
```

```
PROCEDURE Main  
  LOCAL oForm  
  LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL  
  LOCAL oExpression
```

```
oForm := XbpDialog():new( AppDesktop() )  
oForm:drawingArea:clipChildren := .T.  
oForm:create( ,,{100,100}, {640,480},,, .F. )  
oForm:close := {|| PostAppEvent( xbeP_Quit )}
```

```
oExpression := XbpActiveXControl():new( oForm:drawingArea )  
oExpression:CLSID := "Exontrol.Expression.1" /*{B33F5489-49AC-4155-98E7-  
9BBFC57FF019}*/
```

```
oExpression:create(,, {10,60},{610,370} )
```

```
oExpression:DbIcIck := { |Shift,X,Y| OnDbIcIck(oExpression,Shift,X,Y) } /*Occurs  
when the user double clicks the left mouse button over an object.* /
```

```
oExpression:Expression := "value"
```

```
oExpression:AllowSplitter := 2/*exVSplitter*/
```

```
oExpression:SplitPaneHeight := 196
```

```
oForm:Show()
```

```
DO WHILE nEvent != xbeP_Quit
```

```
  nEvent := AppEvent( @mp1, @mp2, @oXbp )
```

```
  oXbp:handleEvent( nEvent, mp1, mp2 )
```

```
ENDDO
```

```
RETURN
```

event ExecuteContextMenu (Item as String, Position as Long)

Occurs when the user selects an user item from the control's context menu.

Type	Description
Item as String	A string expression that indicates the selected item.
Position as Long	A long expression that indicates the position of the selected item. The Position is 1 based, and the separator items do not count.

The ExecuteContextMenu event notifies your application that the user selects a custom context menu item. Use the [ContextMenuItems](#) property to add custom items to the control's context menu. Use the [AllowContextMenu](#) property to allow the control's context menu. The ExecuteContextMenu event is not fired if the user selects one of the default entries in the control's context menu such as: Undo, Copy, and so on.

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

```
C# private void ExecuteContextMenu(object sender,string Item,int Position)
{
}
```

```
VB Private Sub ExecuteContextMenu(ByVal sender As System.Object,ByVal Item As
String,ByVal Position As Integer) Handles ExecuteContextMenu
End Sub
```

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

```
C# private void ExecuteContextMenu(object sender,
AxEXPRESSIONLib._IExpressionEvents_ExecuteContextMenuEvent e)
{
}
```

```
C++ void OnExecuteContextMenu(LPCTSTR Item,long Position)
{
}
```

```
C++ Builder void __fastcall ExecuteContextMenu(TObject *Sender,BSTR Item,long Position)
{
}
```

```
Delphi procedure ExecuteContextMenu(ASender: TObject; Item : WideString;Position : Integer);  
begin  
end;
```

```
Delphi 8 (.NET only) procedure ExecuteContextMenu(sender: System.Object; e: AxEXPRESSIONLib._IExpressionEvents_ExecuteContextMenuEvent);  
begin  
end;
```

```
Powe... begin event ExecuteContextMenu(string Item,long Position)  
end event ExecuteContextMenu
```

```
VB.NET Private Sub ExecuteContextMenu(ByVal sender As System.Object, ByVal e As AxEXPRESSIONLib._IExpressionEvents_ExecuteContextMenuEvent) Handles ExecuteContextMenu  
End Sub
```

```
VB6 Private Sub ExecuteContextMenu(ByVal Item As String,ByVal Position As Long)  
End Sub
```

```
VBA Private Sub ExecuteContextMenu(ByVal Item As String,ByVal Position As Long)  
End Sub
```

```
VFP LPARAMETERS Item,Position
```

```
Xbas... PROCEDURE OnExecuteContextMenu(oExpression,Item,Position)  
RETURN
```

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

```
Java... <SCRIPT EVENT="ExecuteContextMenu(Item,Position)" LANGUAGE="JScript">  
</SCRIPT>
```

```
VBSc... <SCRIPT LANGUAGE="VBScript">  
Function ExecuteContextMenu(Item,Position)  
End Function
```

```
</SCRIPT>
```

```
Visual  
Data... Procedure OnComExecuteContextMenu String IItem Integer IPosition  
Forward Send OnComExecuteContextMenu IItem IPosition  
End_Procedure
```

```
Visual  
Objects METHOD OCX_ExecuteContextMenu(Item,Position) CLASS MainDialog  
RETURN NIL
```

```
X++ void onEvent_ExecuteContextMenu(str _Item,int _Position)  
{  
}
```

```
XBasic function ExecuteContextMenu as v (Item as C,Position as N)  
end function
```

```
dBASE function nativeObject_ExecuteContextMenu(Item,Position)  
return
```

The following VB sample adds some custom entries to the control's context menu, and prints the selected item, when the user selects it from the control's context menu:

```
Private Sub Form_Load()  
With Expression1  
.AllowContextMenu = True  
.ContextMenuItems = vbCr & "Item 1" & vbCr & vbCr & "Item 2" & vbCr & "Item 3"  
End With  
End Sub
```

```
Private Sub Expression1_ExecuteContextMenu(ByVal Item As String, ByVal Position As  
Long)  
Debug.Print "The user selects the '" & Item & "' item. The item's position is '" & Position  
& "'."  
End Sub
```

The following C++ sample adds some custom entries to the control's context menu, and prints the selected item, when the user selects it from the control's context menu:

```
m_edit.SetAllowContextMenu( TRUE );
CString strCustom( "\\r" );
strCustom += "Item 1";
strCustom += "\\r\\r";
strCustom += "Item 2";
strCustom += "\\r";
strCustom += "Item 3";
m_edit.SetContextMenuItems( strCustom );
```

```
void OnExecuteContextMenuExpression1(LPCTSTR Item, long Position)
{
    OutputDebugString( Item );
}
```

The following VB.NET sample adds some custom entries to the control's context menu, and prints the selected item, when the user selects it from the control's context menu:

```
With AxExpression1
    .AllowContextMenu = True
    .ContextMenuItems = vbCr & "Item 1" & vbCr & vbCr & "Item 2" & vbCr & "Item 3"
End With
```

```
Private Sub AxExpression1_ExecuteContextMenu(ByVal sender As Object, ByVal e As
AxEXPRESSIONLib._IExpressionEvents_ExecuteContextMenuEvent) Handles
AxExpression1.ExecuteContextMenu
    Debug.WriteLine(e.item)
End Sub
```

The following C# sample adds some custom entries to the control's context menu, and prints the selected item, when the user selects it from the control's context menu:

```
axExpression1.AllowContextMenu = true;
axExpression1.ContextMenuItems = "\\rItem1\\r\\rItem2\\rItem3";
```

```
private void axExpression1_ExecuteContextMenu(object sender,
AxEXPRESSIONLib._IExpressionEvents_ExecuteContextMenuEvent e)
{
    System.Diagnostics.Debug.WriteLine(e.item);
}
```

The following VFP sample adds some custom entries to the control's context menu, and prints the selected item, when the user selects it from the control's context menu:

```
with thisform.Expression1
  .AllowContextMenu = .t.
  .ContextMenuItem = chr(13) + "Item 1" + chr(13) + chr(13) + "Item 2" + chr(13) +
  "Item 3"
endwith
```

```
*** ActiveX Control Event ***
LPARAMETERS item, position

wait window nowait item
```

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 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,
AxEXPRESSIONLib._IExpressionEvents_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:
AxEXPRESSIONLib._IExpressionEvents_KeyDownEvent);
begin
end;
```

Power...

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

VB.NET

```
Private Sub KeyDownEvent(ByVal sender As System.Object, ByVal e As
AxEXPRESSIONLib._IExpressionEvents_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(oExpression,KeyCode,Shift)
RETURN
```

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

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

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

```
Visual Data... Procedure OnComKeyDown Short I(KeyCode) Short I(Shift)
Forward Send OnComKeyDown I(KeyCode) I(Shift)
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,
AxEXPRESSIONLib._IExpressionEvents_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:  
AxEXPRESSIONLib._IExpressionEvents_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  
AxEXPRESSIONLib._IExpressionEvents_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(oExpression,KeyAscii)  
RETURN
```

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

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

```
VBS... <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)
{
}
```

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

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

```
C# private void KeyUpEvent(object sender,
AxEXPRESSIONLib._IExpressionEvents_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:
AxEXPRESSIONLib._IExpressionEvents_KeyUpEvent);
begin
end;
```

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

```
VB.NET Private Sub KeyUpEvent(ByVal sender As System.Object, ByVal e As
AxEXPRESSIONLib._IExpressionEvents_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(oExpression,KeyCode,Shift)
RETURN
```

Syntax for KeyUp event, **ICOM** 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 [MouseDown](#) 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. The [WordFromCursor\(-1,-1\)](#) property indicates the word from the cursor.

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,
AxEXPRESSIONLib._IExpressionEvents_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: AxEXPRESSIONLib._IExpressionEvents_MouseDownEvent);
begin
end;
```

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

```
VB.NET Private Sub MouseDownEvent(ByVal sender As System.Object, ByVal e As AxEXPRESSIONLib._IExpressionEvents_MouseDownEvent) Handles MouseDownEvent
End Sub
```

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

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

```
VFP LPARAMETERS Button,Shift,X,Y
```

Xbas...

```
PROCEDURE OnMouseDown(oExpression,Button,Shift,X,Y)
RETURN
```

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

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

```
VBS... <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.Expression.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.Expression.1::OLE_YPOS_PIXELS)
end function
```

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

event MouseEventArgs (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 MouseEventArgs event is generated continually as the mouse pointer moves across objects. Unless another object has captured the mouse, an object recognizes a MouseEventArgs event whenever the mouse position is within its borders. The [WordFromCursor\(-1,-1\)](#) property indicates the word from the cursor.

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

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

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

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

```
C# private void MouseEventArgs(object sender,
AxEXPRESSIONLib._IExpressionEvents_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:
AxEXPRESSIONLib._IExpressionEvents_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
AxEXPRESSIONLib._IExpressionEvents_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(oExpression,Button,Shift,X,Y)
```

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

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

```
VBS... <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.Expression.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.Expression.1::OLE_YPOS_PIXELS)
end function
```

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

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

Occurs when the user releases a mouse button.

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

Use a [MouseDown](#) or MouseUp event procedure to specify actions that will occur when a mouse button is pressed or released. Unlike the [Click](#) and [DbClick](#) events, MouseDown and MouseUp events lets you distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers. The [WordFromCursor\(-1,-1\)](#) property indicates the word from the cursor.

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,
AxEXPRESSIONLib._IExpressionEvents_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:  
AxEXPRESSIONLib._IExpressionEvents_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  
AxEXPRESSIONLib._IExpressionEvents_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(oExpression,Button,Shift,X,Y)
```

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

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

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

```
Visual Data... Procedure OnComMouseUp Short IButton Short IShift OLE_XPOS_PIXELS IIX
OLE_YPOS_PIXELS IY
    Forward Send OnComMouseUp IButton IShift IIX IY
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.Expression.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.Expression.1::OLE_YPOS_PIXELS)
end function
```

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

event **OLECompleteDrag** (Effect as Long)

Occurs when a source component is dropped onto a target component, informing the source component that a drag action was either performed or canceled

Type	Description
Effect as Long	A long set by the source object identifying the action that has been performed, thus allowing the source to take appropriate action if the component was moved (such as the source deleting data if it is moved from one component to another.

The **OLECompleteDrag** event is the final event to be called in an OLE drag/drop operation. This event informs the source component of the action that was performed when the object was dropped onto the target component. The target sets this value through the effect parameter of the [OLEDragDrop](#) event. Based on this, the source can then determine the appropriate action it needs to take. For example, if the object was moved into the target (`exDropEffectMove`), the source needs to delete the object from itself after the move. The control supports manual or automatic OLE drag and drop events. Use the [OLEDropMode](#) property to enable the OLE drag and drop support.

The settings for Effect are:

- `exOLEDropEffectNone` (0), Drop target cannot accept the data, or the drop operation was cancelled
- `exOLEDropEffectCopy` (1), Drop results in a copy of data from the source to the target. The original data is unaltered by the drag operation.
- `exOLEDropEffectMove` (2), Drop results in data being moved from drag source to drop source. The drag source should remove the data from itself after the move.

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

```
C# // OLECompleteDrag event is not supported. Use the  
DragEnter,DragLeave,DragOver, DragDrop ... events.
```

```
VB // OLECompleteDrag event is not supported. Use the  
DragEnter,DragLeave,DragOver, DragDrop ... events.
```

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

```
C# private void OLECompleteDrag(object sender,  
AxEXPRESSIONLib._IExpressionEvents_OLECompleteDragEvent e)  
{
```

```
}
```

```
C++ void OnOLECompleteDrag(long Effect)
```

```
{  
}
```

```
C++ Builder void __fastcall OLECompleteDrag(TObject *Sender,long Effect)
```

```
{  
}
```

```
Delphi procedure OLECompleteDrag(ASender: TObject; Effect : Integer);  
begin  
end;
```

```
Delphi 8  
(.NET  
only) procedure OLECompleteDrag(sender: System.Object; e:  
AxEXPRESSIONLib._IExpressionEvents_OLECompleteDragEvent);  
begin  
end;
```

```
Powe... begin event OLECompleteDrag(long Effect)  
end event OLECompleteDrag
```

```
VB.NET Private Sub OLECompleteDrag(ByVal sender As System.Object, ByVal e As  
AxEXPRESSIONLib._IExpressionEvents_OLECompleteDragEvent) Handles  
OLECompleteDrag  
End Sub
```

```
VB6 Private Sub OLECompleteDrag(ByVal Effect As Long)  
End Sub
```

```
VBA Private Sub OLECompleteDrag(ByVal Effect As Long)  
End Sub
```

```
VFP LPARAMETERS Effect
```

```
Xbas... PROCEDURE OnOLECompleteDrag(oExpression,Effect)  
RETURN
```

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

```
Java... <SCRIPT EVENT="OLECompleteDrag(Effect)" LANGUAGE="JScript">  
</SCRIPT>
```

```
VBS... <SCRIPT LANGUAGE="VBScript">  
Function OLECompleteDrag(Effect)  
End Function  
</SCRIPT>
```

```
Visual  
Data... Procedure OnComOLECompleteDrag Integer IEffect  
Forward Send OnComOLECompleteDrag IEffect  
End_Procedure
```

```
Visual  
Objects METHOD OCX_OLECompleteDrag(Effect) CLASS MainDialog  
RETURN NIL
```

```
X++ // OLECompleteDrag event is not supported. Use the  
DragEnter,DragLeave,DragOver, DragDrop ... events.
```

```
XBasic function OLECompleteDrag as v (Effect as N)  
end function
```

```
dBASE function nativeObject_OLECompleteDrag(Effect)  
return
```

event **OLEDragDrop** (Data as **ExDataObject**, Effect as **Long**, Button as **Integer**, Shift as **Integer**, X as **OLE_XPOS_PIXELS**, Y as **OLE_YPOS_PIXELS**)

Occurs when a source component is dropped onto a target component when the source component determines that a drop can occur.

Type	Description
Data as ExDataObject	An ExDataObject object containing formats that the source will provide and, in addition, possibly the data for those formats. If no data is contained in the ExDataObject , it is provided when the control calls the GetData method. The SetData and Clear methods cannot be used here.
Effect as Long	A Long set by the target component identifying the action that has been performed (if any), thus allowing the source to take appropriate action if the component was moved (such as the source deleting the data). The possible values are listed in bellow.
Button as Integer	An integer which acts as a bit field corresponding to the state of a mouse button when it is depressed. The left button is bit 0, the right button is bit 1, and the middle button is bit 2. These bits correspond to the values 1, 2, and 4, respectively. It indicates the state of the mouse buttons; some, all, or none of these three bits can be set, indicating that some, all, or none of the buttons are depressed
Shift as Integer	An integer which acts as a bit field corresponding to the state of the SHIFT , CTRL , and ALT keys when they are depressed. The SHIFT key is bit 0, the CTRL key is bit 1, and the ALT key is bit 2. These bits correspond to the values 1, 2, and 4, respectively. The shift parameter indicates the state of these keys; some, all, or none of the bits can be set, indicating that some, all, or none of the keys are depressed. For example, if both the CTRL and ALT keys were depressed, the value of shift would be 6.
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

The `OLEDragDrop` event is fired when the user has dropped files or clipboard information into control. In order to enable OLE drag and drop feature into control you have to check the [OLEDropMode](#) property. The idea of drag and drop in the control is the same as in the other controls. To start accepting drag and drop sources the control should have the `OLEDropMode` to `exOLEDropManual`. Once that is set, the controls starts accepting any drag and drop sources. Use the [SelText](#) property to insert text at cursor position.

The settings for Effect are:

- `exOLEDropEffectNone` (0), Drop target cannot accept the data, or the drop operation was cancelled
- `exOLEDropEffectCopy` (1), Drop results in a copy of data from the source to the target. The original data is unaltered by the drag operation.
- `exOLEDropEffectMove` (2), Drop results in data being moved from drag source to drop source. The drag source should remove the data from itself after the move.

Syntax for `OLEDragDrop` event, **/.NET** version, on:

```
C# // OLEDragDrop event is not supported. Use the DragEnter,DragLeave,DragOver,
DragDrop ... events.
```

```
VB // OLEDragDrop event is not supported. Use the DragEnter,DragLeave,DragOver,
DragDrop ... events.
```

Syntax for `OLEDragDrop` event, **/COM** version, on:

```
C# private void OLEDragDrop(object sender,
AxEXPRESSIONLib._IExpressionEvents_OLEDragDropEvent e)
{
}
```

```
C++ void OnOLEDragDrop(LPDISPATCH Data,long FAR* Effect,short Button,short
Shift,long X,long Y)
{
}
```

```
C++ Builder void __fastcall OLEDragDrop(TObject *Sender,Expressionlib_tlb::IExDataObject
*Data,long * Effect,short Button,short Shift,int X,int Y)
{
}
```

```
Delphi procedure OLEDragDrop(ASender: TObject; Data : IExDataObject;var Effect : Integer;Button : Smallint;Shift : Smallint;X : Integer;Y : Integer);  
begin  
end;
```

```
Delphi 8 (.NET only) procedure OLEDragDrop(sender: System.Object; e: AxEXPRESSIONLib._IExpressionEvents_OLEDragDropEvent);  
begin  
end;
```

```
Powe... begin event OLEDragDrop(oleobject Data,long Effect,integer Button,integer Shift,long X,long Y)  
end event OLEDragDrop
```

```
VB.NET Private Sub OLEDragDrop(ByVal sender As System.Object, ByVal e As AxEXPRESSIONLib._IExpressionEvents_OLEDragDropEvent) Handles OLEDragDrop  
End Sub
```

```
VB6 Private Sub OLEDragDrop(ByVal Data As EXPRESSIONLibCtl.IExDataObject,Effect As Long,ByVal Button As Integer,ByVal Shift As Integer,ByVal X As Single,ByVal Y As Single)  
End Sub
```

```
VBA Private Sub OLEDragDrop(ByVal Data As Object,Effect As Long,ByVal Button As Integer,ByVal Shift As Integer,ByVal X As Long,ByVal Y As Long)  
End Sub
```

```
VFP LPARAMETERS Data,Effect,Button,Shift,X,Y
```

```
Xbas... PROCEDURE OnOLEDragDrop(oExpression,Data,Effect,Button,Shift,X,Y)  
RETURN
```

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

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

```
VBScri... <SCRIPT LANGUAGE="VBScript" >  
Function OLEDragDrop(Data,Effect,Button,Shift,X,Y)  
End Function  
</SCRIPT>
```

```
Visual  
Data... Procedure OnComOLEDragDrop Variant IIData Integer IIEffect Short IIButton  
Short IIShift OLE_XPOS_PIXELS IIX OLE_YPOS_PIXELS IIY  
    Forward Send OnComOLEDragDrop IIData IIEffect IIButton IIShift IIX IIY  
End_Procedure
```

```
Visual  
Objects METHOD OCX_OLEDragDrop(Data,Effect,Button,Shift,X,Y) CLASS MainDialog  
RETURN NIL
```

```
X++ // OLEDragDrop event is not supported. Use the DragEnter,DragLeave,DragOver,  
DragDrop ... events.
```

```
XBasic function OLEDragDrop as v (Data as  
OLE::Exontrol.Expression.1::IExDataObject,Effect as N,Button as N,Shift as N,X as  
OLE::Exontrol.Expression.1::OLE_XPOS_PIXELS,Y as  
OLE::Exontrol.Expression.1::OLE_YPOS_PIXELS)  
end function
```

```
dBASE function nativeObject_OLEDragDrop(Data,Effect,Button,Shift,X,Y)  
return
```

The following VB sample inserts the names of the files being dragged at the cursor position:

```
Private Sub Expression1_OLEDragDrop(ByVal Data As EXPRESSIONLibCtl.IExDataObject,  
Effect As Long, ByVal Button As Integer, ByVal Shift As Integer, ByVal X As Single, ByVal Y As  
Single)  
    If (Data.GetFormat(exCFFiles)) Then  
        With Data.Files  
            Dim i As Long, s As String  
            s = ""  
            For i = 0 To .Count - 1  
                s = s + .Item(i) + vbCrLf  
            Next
```

```
Expression1.SelText = s
End With
End If
End Sub
```

The following C++ sample inserts the names of the files being dragged at the cursor position:

```
void OnOLEDragDropExpression1(LPDISPATCH Data, long FAR* Effect, short Button, short Shift, long X, long Y)
{
    if ( EXPRESSIONLib::IExDataObjectPtr spData( Data ) )
        if ( spData->GetFormat( EXPRESSIONLib::exCFFiles ) )
            {
                EXPRESSIONLib::IExDataObjectFilesPtr spFiles = spData->Files;
                if ( spFiles != NULL )
                    {
                        CString s;
                        for ( long i = 0; i < spFiles->Count; i++ )
                            s += spFiles->Item[i];
                        m_edit.SetSelText( s );
                    }
            }
}
```

the sample needs to include definitions for IExDataObject and IExDataObjectFiles objects, by using the #import <expression.dll> statement. The #import <expression.dll> includes the EXPRESSIONLib namespace where are defined all objects in the control's type library.

The following VB.NET sample inserts the names of the files being dragged at the cursor position:

```
Private Sub AxExpression1_OLEDragDrop(ByVal sender As Object, ByVal e As AxEXPRESSIONLib._IExpressionEvents_OLEDragDropEvent) Handles AxExpression1.OLEDragDrop
    If (e.data.GetFormat(EXPRESSIONLib.exClipboardFormatEnum.exCFFiles)) Then
        With e.data.Files
            Dim i As Long, s As String = ""
            For i = 0 To .Count - 1
```

```

        s = s + .Item(i) + vbCrLf
    Next
    AxExpression1.SelText = s
End With
End If
End Sub

```

The following C# sample inserts the names of the files being dragged at the cursor position:

```

private void axExpression1_OLEDragDrop(object sender,
AxEXPRESSIONLib._IExpressionEvents_OLEDragDropEvent e)
{
    if
(e.data.GetFormat(Convert.ToInt16(EXPRESSIONLib.exClipboardFormatEnum.exCFFiles)))
    {
        EXPRESSIONLib.IExDataObjectFiles files = e.data.Files;
        if (files != null)
        {
            string s = "";
            for (int i = 0; i < files.Count; i++)
                s += files[i] + "\r\n";
            axExpression1.SelText = s;
        }
    }
}

```

The following VFP sample inserts the names of the files being dragged at the cursor position:

```

*** ActiveX Control Event ***
LPARAMETERS data, effect, button, shift, x, y

if ( data.GetFormat( 15 ) ) then
    local i, s
    s = ""
    with data.Files
        for i = 0 to .Count - 1
            s = s + .Item(i) + chr(13) + chr(10)

```

```
next  
thisform.Expression1.SelText = s  
endwith  
endif
```

event **OLEDragOver** (Data as **ExDataObject**, Effect as **Long**, Button as **Integer**, Shift as **Integer**, X as **OLE_XPOS_PIXELS**, Y as **OLE_YPOS_PIXELS**, State as **Integer**)

Occurs when one component is dragged over another.

Type	Description
Data as ExDataObject	An ExDataObject object containing formats that the source will provide and, in addition, possibly the data for those formats. If no data is contained in the ExDataObject , it is provided when the control calls the GetData method. The SetData and Clear methods cannot be used here
Effect as Long	A Long set by the target component identifying the action that has been performed (if any), thus allowing the source to take appropriate action if the component was moved (such as the source deleting the data). The possible values are listed below.
Button as Integer	An integer which acts as a bit field corresponding to the state of a mouse button when it is depressed. The left button is bit 0, the right button is bit 1, and the middle button is bit 2. These bits correspond to the values 1, 2, and 4, respectively. It indicates the state of the mouse buttons; some, all, or none of these three bits can be set, indicating that some, all, or none of the buttons are depressed.
Shift as Integer	These bits correspond to the values 1, 2, and 4, respectively. The shift parameter indicates the state of these keys; some, all, or none of the bits can be set, indicating that some, all, or none of the keys are depressed. For example, if both the CTRL and ALT keys were depressed, the value of shift would be 6.
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.
State as Integer	An integer that corresponds to the transition state of the control being dragged in relation to a target form or control. The possible values are listed below.

The settings for effect are:

- `exOLEDropEffectNone` (0), Drop target cannot accept the data, or the drop operation was cancelled
- `exOLEDropEffectCopy` (1), Drop results in a copy of data from the source to the target. The original data is unaltered by the drag operation.
- `exOLEDropEffectMove` (2), Drop results in data being moved from drag source to drop source. The drag source should remove the data from itself after the move.

The settings for state are:

- `exOLEDragEnter` (0), Source component is being dragged within the range of a target.
- `exOLEDragLeave` (1), Source component is being dragged out of the range of a target.
- `exOLEOLEDragOver` (2), Source component has moved from one position in the target to another

Note If the state parameter is 1, indicating that the mouse pointer has left the target, then the x and y parameters will contain zeros.

The source component should always mask values from the effect parameter to ensure compatibility with future implementations of ActiveX components. As a precaution against future problems, drag sources and drop targets should mask these values appropriately before performing any comparisons.

For example, a source component should not compare an effect against, say, `exOLEDropEffectCopy`, such as in this manner:

If `Effect = exOLEDropEffectCopy...`

Instead, the source component should mask for the value or values being sought, such as this:

If `Effect And exOLEDropEffectCopy = exOLEDropEffectCopy...`

-or-

If `(Effect And exOLEDropEffectCopy)...`

This allows for the definition of new drop effects in future versions while preserving backwards compatibility with your existing code.

Syntax for `OLEDragOver` event, **/.NET** version, on:

```
C# // OLEDragOver event is not supported. Use the DragEnter,DragLeave,DragOver,  
DragDrop ... events.
```

```
VB // OLEDragOver event is not supported. Use the DragEnter,DragLeave,DragOver,  
DragDrop ... events.
```

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

```
C# private void OLEDragOver(object sender,
AxEXPRESSIONLib._IExpressionEvents_OLEDragOverEvent e)
{
}
```

```
C++ void OnOLEDragOver(LPDISPATCH Data,long FAR* Effect,short Button,short
Shift,long X,long Y,short State)
{
}
```

```
C++ Builder void __fastcall OLEDragOver(TObject *Sender,Expressionlib_tlb::IExDataObject
*Data,long * Effect,short Button,short Shift,int X,int Y,short State)
{
}
```

```
Delphi procedure OLEDragOver(ASender: TObject; Data : IExDataObject;var Effect :
Integer;Button : Smallint;Shift : Smallint;X : Integer;Y : Integer;State : Smallint);
begin
end;
```

```
Delphi 8
(.NET
only) procedure OLEDragOver(sender: System.Object; e:
AxEXPRESSIONLib._IExpressionEvents_OLEDragOverEvent);
begin
end;
```

```
Powe... begin event OLEDragOver(oleobject Data,long Effect,integer Button,integer
Shift,long X,long Y,integer State)
end event OLEDragOver
```

```
VB.NET Private Sub OLEDragOver(ByVal sender As System.Object, ByVal e As
AxEXPRESSIONLib._IExpressionEvents_OLEDragOverEvent) Handles OLEDragOver
End Sub
```

```
VB6 Private Sub OLEDragOver(ByVal Data As EXPRESSIONLibCtl.IExDataObject,Effect
As Long,ByVal Button As Integer,ByVal Shift As Integer,ByVal X As Single,ByVal Y As
Single,ByVal State As Integer)
```



```
OLE::Exontrol.Expression.1::OLE_YPOS_PIXELS,State as N)
```

```
end function
```

dBASE

```
function nativeObject_OLEDragOver(Data,Effect,Button,Shift,X,Y,State)
```

```
return
```

event OLEGiveFeedback (Effect as Long, DefaultCursors as Boolean)

Allows the drag source to specify the type of OLE drag-and-drop operation and the visual feedback.

Type	Description
Effect as Long	A long integer set by the target component in the OLEDragOver event specifying the action to be performed if the user drops the selection on it. This allows the source to take the appropriate action (such as giving visual feedback). The possible values are listed below.
DefaultCursors as Boolean	Boolean value that determines whether to use the default mouse cursor, or to use a user-defined mouse cursor. True (default) = use default mouse cursor. False = do not use default cursor. Mouse cursor must be set with the MousePointer property of the Screen object

The settings for Effect are:

- exOLEDropEffectNone (0), Drop target cannot accept the data, or the drop operation was cancelled
- exOLEDropEffectCopy (1), Drop results in a copy of data from the source to the target. The original data is unaltered by the drag operation.
- exOLEDropEffectMove (2), Drop results in data being moved from drag source to drop source. The drag source should remove the data from itself after the move

If there is no code in the OLEGiveFeedback event, or if the defaultcursors parameter is set to True, the mouse cursor will be set to the default cursor provided by the control. The source component should always mask values from the effect parameter to ensure compatibility with future implementations of ActiveX components. As a precaution against future problems, drag sources and drop targets should mask these values appropriately before performing any comparisons.

For example, a source component should not compare an effect against, say, exOLEDropEffectCopy, such as in this manner:

If Effect = exOLEDropEffectCopy...

Instead, the source component should mask for the value or values being sought, such as this:

If Effect And exOLEDropEffectCopy = exOLEDropEffectCopy...

-or-

If (Effect And exOLEDropEffectCopy)...

This allows for the definition of new drop effects in future versions while preserving backwards compatibility with your existing code.

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

```
C# // OLEGiveFeedback event is not supported. Use the  
DragEnter,DragLeave,DragOver, DragDrop ... events.
```

```
VB // OLEGiveFeedback event is not supported. Use the  
DragEnter,DragLeave,DragOver, DragDrop ... events.
```

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

```
C# private void OLEGiveFeedback(object sender,  
AxEXPRESSIONLib._IExpressionEvents_OLEGiveFeedbackEvent e)  
{  
}
```

```
C++ void OnOLEGiveFeedback(long Effect,BOOL FAR* DefaultCursors)  
{  
}
```

```
C++ Builder void __fastcall OLEGiveFeedback(TObject *Sender,long Effect,VARIANT_BOOL *  
DefaultCursors)  
{  
}
```

```
Delphi procedure OLEGiveFeedback(ASender: TObject; Effect : Integer;var DefaultCursors  
: WordBool);  
begin  
end;
```

```
Delphi 8  
(.NET  
only) procedure OLEGiveFeedback(sender: System.Object; e:  
AxEXPRESSIONLib._IExpressionEvents_OLEGiveFeedbackEvent);  
begin  
end;
```

```
Powe... begin event OLEGiveFeedback(long Effect,boolean DefaultCursors)  
end event OLEGiveFeedback
```

```
VB.NET Private Sub OLEGiveFeedback(ByVal sender As System.Object, ByVal e As  
AxEXPRESSIONLib._IExpressionEvents_OLEGiveFeedbackEvent) Handles
```

```
OLEGiveFeedback  
End Sub
```

```
VB6 Private Sub OLEGiveFeedback(ByVal Effect As Long,DefaultCursors As Boolean)  
End Sub
```

```
VBA Private Sub OLEGiveFeedback(ByVal Effect As Long,DefaultCursors As Boolean)  
End Sub
```

```
VFP LPARAMETERS Effect,DefaultCursors
```

```
Xbas... PROCEDURE OnOLEGiveFeedback(oExpression,Effect,DefaultCursors)  
RETURN
```

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

```
Java... <SCRIPT EVENT="OLEGiveFeedback(Effect,DefaultCursors)"  
LANGUAGE="JScript">  
</SCRIPT>
```

```
VBS... <SCRIPT LANGUAGE="VBScript">  
Function OLEGiveFeedback(Effect,DefaultCursors)  
End Function  
</SCRIPT>
```

```
Visual  
Data... Procedure OnComOLEGiveFeedback Integer IEffect Boolean IIDefaultCursors  
Forward Send OnComOLEGiveFeedback IEffect IIDefaultCursors  
End_Procedure
```

```
Visual  
Objects METHOD OCX_OLEGiveFeedback(Effect,DefaultCursors) CLASS MainDialog  
RETURN NIL
```

```
X++ // OLEGiveFeedback event is not supported. Use the  
DragEnter,DragLeave,DragOver, DragDrop ... events.
```

```
XBasic function OLEGiveFeedback as v (Effect as N,DefaultCursors as L)  
end function
```

dBASE

```
function nativeObject_OLEGiveFeedback(Effect,DefaultCursors)  
return
```

event **OLESetData** (Data as **ExDataObject**, Format as **Integer**)

Occurs on a drag source when a drop target calls the `GetData` method and there is no data in a specified format in the OLE drag-and-drop `DataObject`.

Type	Description
Data as ExDataObject	An <code>ExDataObject</code> object in which to place the requested data. The component calls the <code>SetData</code> method to load the requested format.
Format as Integer	An integer specifying the format of the data that the target component is requesting. The source component uses this value to determine what to load into the <code>ExDataObject</code> object.

The `OLESetData` is not implemented

Syntax for `OLESetData` event, **/NET** version, on:

```
C# // OLESetData event is not supported. Use the DragEnter,DragLeave,DragOver,  
DragDrop ... events.
```

```
VB // OLESetData event is not supported. Use the DragEnter,DragLeave,DragOver,  
DragDrop ... events.
```

Syntax for `OLESetData` event, **/COM** version, on:

```
C# private void OLESetData(object sender,  
AxEXPRESSIONLib._IExpressionEvents_OLESetDataEvent e)  
{  
}
```

```
C++ void OnOLESetData(LPDISPATCH Data,short Format)  
{  
}
```

```
C++  
Builder void __fastcall OLESetData(TObject *Sender,Expressionlib_tlb::IExDataObject  
*Data,short Format)  
{  
}
```

Delphi

```
procedure OLESetData(ASender: TObject; Data : IExDataObject;Format : Smallint);  
begin  
end;
```

Delphi 8
(.NET
only)

```
procedure OLESetData(sender: System.Object; e:  
AxEXPRESSIONLib._IExpressionEvents_OLESetDataEvent);  
begin  
end;
```

Power...

```
begin event OLESetData(oleobject Data,integer Format)  
end event OLESetData
```

VB.NET

```
Private Sub OLESetData(ByVal sender As System.Object, ByVal e As  
AxEXPRESSIONLib._IExpressionEvents_OLESetDataEvent) Handles OLESetData  
End Sub
```

VB6

```
Private Sub OLESetData(ByVal Data As EXPRESSIONLibCtl.IExDataObject,ByVal  
Format As Integer)  
End Sub
```

VBA

```
Private Sub OLESetData(ByVal Data As Object,ByVal Format As Integer)  
End Sub
```

VFP

```
LPARAMETERS Data,Format
```

Xbas...

```
PROCEDURE OnOLESetData(oExpression,Data,Format)  
RETURN
```

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

Java...

```
<SCRIPT EVENT="OLESetData(Data,Format)" LANGUAGE="JScript">  
</SCRIPT>
```

VBSc...

```
<SCRIPT LANGUAGE="VBScript">  
Function OLESetData(Data,Format)  
End Function
```

```
</SCRIPT>
```

Visual
Data...

```
Procedure OnComOLESetData Variant IIData Short IIFormat  
    Forward Send OnComOLESetData IIData IIFormat  
End_Procedure
```

Visual
Objects

```
METHOD OCX_OLESetData(Data,Format) CLASS MainDialog  
RETURN NIL
```

X++

```
// OLESetData event is not supported. Use the DragEnter,DragLeave,DragOver,  
DragDrop ... events.
```

XBasic

```
function OLESetData as v (Data as  
OLE::Exontrol.Expression.1::IExDataObject,Format as N)  
end function
```

dBASE

```
function nativeObject_OLESetData(Data,Format)  
return
```

event **OLEStartDrag** (Data as **ExDataObject**, AllowedEffects as **Long**)

Occurs when the **OLEDrag** method is called.

Type	Description
Data as ExDataObject	An ExDataObject object containing formats that the source will provide and, optionally, the data for those formats. If no data is contained in the ExDataObject , it is provided when the control calls the GetData method. The programmer should provide the values for this parameter in this event. The SetData and Clear methods cannot be used here.
AllowedEffects as Long	A long containing the effects that the source component supports. The possible values are listed in Settings . The programmer should provide the values for this parameter in this event.

The settings for **AllowEffects** are:

- **exOLEDropEffectNone** (0), Drop target cannot accept the data, or the drop operation was cancelled
- **exOLEDropEffectCopy** (1), Drop results in a copy of data from the source to the target. The original data is unaltered by the drag operation.
- **exOLEDropEffectMove** (2), Drop results in data being moved from drag source to drop source. The drag source should remove the data from itself after the move

The source component should logically Or together the supported values and places the result in the **AllowedEffects** parameter. The target component can use this value to determine the appropriate action (and what the appropriate user feedback should be). You may wish to defer putting data into the **ExDataObject** object until the target component requests it. This allows the source component to save time. If the user does not load any formats into the **ExDataObject**, then the drag/drop operation is canceled.

To start accepting drag and drop sources the **exGrid** control should have the [OLEDropMode](#) to **exOLEDropManual** or **exOLEDropAutomatic**.

Syntax for **OLEStartDrag** event, **!NET** version, on:

```
C# // OLEStartDrag event is not supported. Use the DragEnter,DragLeave,DragOver,  
DragDrop ... events.
```

```
VB // OLEStartDrag event is not supported. Use the DragEnter,DragLeave,DragOver,  
DragDrop ... events.
```

Syntax for OLEStartDrag event, **ICOM** version, on:

```
C# private void OLEStartDrag(object sender,
AxEXPRESSIONLib._IExpressionEvents_OLEStartDragEvent e)
{
}
```

```
C++ void OnOLEStartDrag(LPDISPATCH Data,long FAR* AllowedEffects)
{
}
```

```
C++ Builder void __fastcall OLEStartDrag(TObject *Sender,Expressionlib_tlb::IExDataObject
*Data,long * AllowedEffects)
{
}
```

```
Delphi procedure OLEStartDrag(ASender: TObject; Data : IExDataObject;var
AllowedEffects : Integer);
begin
end;
```

```
Delphi 8 (.NET only) procedure OLEStartDrag(sender: System.Object; e:
AxEXPRESSIONLib._IExpressionEvents_OLEStartDragEvent);
begin
end;
```

```
Powe... begin event OLEStartDrag(oleobject Data,long AllowedEffects)
end event OLEStartDrag
```

```
VB.NET Private Sub OLEStartDrag(ByVal sender As System.Object, ByVal e As
AxEXPRESSIONLib._IExpressionEvents_OLEStartDragEvent) Handles OLEStartDrag
End Sub
```

```
VB6 Private Sub OLEStartDrag(ByVal Data As
EXPRESSIONLibCtl.IExDataObject,AllowedEffects As Long)
End Sub
```

```
VBA Private Sub OLEStartDrag(ByVal Data As Object,AllowedEffects As Long)
```

```
End Sub
```

```
VFP LPARAMETERS Data,AllowedEffects
```

```
Xbas... PROCEDURE OnOLEStartDrag(oExpression,Data,AllowedEffects)  
RETURN
```

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

```
Java... <SCRIPT EVENT="OLEStartDrag(Data,AllowedEffects)" LANGUAGE="JScript" >  
</SCRIPT>
```

```
VBS... <SCRIPT LANGUAGE="VBScript" >  
Function OLEStartDrag(Data,AllowedEffects)  
End Function  
</SCRIPT>
```

```
Visual  
Data... Procedure OnComOLEStartDrag Variant IIData Integer IIAIAllowedEffects  
Forward Send OnComOLEStartDrag IIData IIAIAllowedEffects  
End_Procedure
```

```
Visual  
Objects METHOD OCX_OLEStartDrag(Data,AllowedEffects) CLASS MainDialog  
RETURN NIL
```

```
X++ // OLEStartDrag event is not supported. Use the DragEnter,DragLeave,DragOver,  
DragDrop ... events.
```

```
XBasic function OLEStartDrag as v (Data as  
OLE::Exontrol.Expression.1::IExDataObject,AllowedEffects as N)  
end function
```

```
dBASE function nativeObject_OLEStartDrag(Data,AllowedEffects)  
return
```

The following VB sample puts the selected text to the clipboard when the drag and drop operation starts (The OLEDropMode property is exOLEDropManual):

```

Private Sub Expression1_OLEStartDrag(ByVal Data As EXPRESSIONLibCtl.IExDataObject,
AllowedEffects As Long)
    Dim s As String
    s = Expression1.SelText
    If (Len(s) > 0) Then
        AllowedEffects = 1
        Data.SetData s, 1
    End If
End Sub

```

The following C++ sample puts the selected text to the clipboard when the drag and drop operation starts:

```

void OnOLEStartDragExpression1(LPDISPATCH Data, long FAR* AllowedEffects)
{
    CString s( m_edit.GetSelText() );
    if ( s.GetLength() > 0 )
    {
        *AllowedEffects = 1; /*exOLEDropEffectCopy*/
        if ( EXPRESSIONLib::IExDataObjectPtr spData( Data ) )
            spData->SetData( COleVariant( s ), COleVariant( long(EXPRESSIONLib::exCFTText) ) );
    }
}

```

The C++ sample uses the `#import <expression.dll>` statement to include definitions for the `IExDataObject` and `IExDataObjectFiles` objects.

The following VB.NET sample puts the selected text to the clipboard when the drag and drop operation starts:

```

Private Sub AxExpression1_OLEStartDrag(ByVal sender As Object, ByVal e As
AxEXPRESSIONLib._IExpressionEvents_OLEStartDragEvent) Handles
AxExpression1.OLEStartDrag
    Dim s As String = AxExpression1.SelText
    If (Len(s) > 0) Then
        e.allowedEffects = 1
        e.data.SetData(s, 1)
    End If

```

The following C# sample puts the selected text to the clipboard when the drag and drop operation starts:

```
private void axExpression1_OLEStartDrag(object sender,
AxEXPRESSIONLib._IExpressionEvents_OLEStartDragEvent e)
{
    string s = axExpression1.SelText;
    if (s.Length > 0)
    {
        e.allowedEffects = 1;
        e.data.SetData(s, EXPRESSIONLib.exClipboardFormatEnum.exCFText );
    }
}
```

The following VFP sample puts the selected text to the clipboard when the drag and drop operation starts:

```
*** ActiveX Control Event ***
LPARAMETERS data, allowedeffects

with thisform.Expression1
    local s
    s = .SelText
    if ( len(s) > 0 ) then
        allowedeffects = 1
        data.SetData( s, 1 )
    endif
endwith
```

event OnContext (Start as Long, Context as String)

Occurs when the user invokes the control's context window.

Type	Description
Start as Long	A long expression that indicates the starting of the context
Context as String	A string expression that defines the context.

The OnContext event notifies your application that user invokes the control's context window. By default, the control displays the context window if the user presses the CTRL + SPACE combination. Use the [ContextKey](#) property to define the key combination that's used to display the control's context window. The version 1.0.3.4 introduces the OnContext event that's fired if the control is not read only, and the user opens the control's context menu (pressing the CTRL + SPACE key combination). The OnContext event passes the context string and the position where the context begins. Use the OnContext event to notify your application that the user asks for a context.

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

```
C# private void OnContext(object sender,int Start,string Context)
{
}
```

```
VB Private Sub OnContext(ByVal sender As System.Object,ByVal Start As Integer,ByVal
Context As String) Handles OnContext
End Sub
```

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

```
C# private void OnContext(object sender,
AxEXPRESSIONLib._IExpressionEvents_OnContextEvent e)
{
}
```

```
C++ void OnOnContext(long Start,LPCTSTR Context)
{
}
```

```
C++
Builder void __fastcall OnContext(TObject *Sender,long Start,BSTR Context)
{
```

```
}
```

```
Delphi procedure OnContext(ASender: TObject; Start : Integer;Context : WideString);  
begin  
end;
```

```
Delphi 8 (.NET only) procedure OnContext(sender: System.Object; e:  
AxEXPRESSIONLib._IExpressionEvents_OnContextEvent);  
begin  
end;
```

```
Power... begin event OnContext(long Start,string Context)  
end event OnContext
```

```
VB.NET Private Sub OnContext(ByVal sender As System.Object, ByVal e As  
AxEXPRESSIONLib._IExpressionEvents_OnContextEvent) Handles OnContext  
End Sub
```

```
VB6 Private Sub OnContext(ByVal Start As Long,ByVal Context As String)  
End Sub
```

```
VBA Private Sub OnContext(ByVal Start As Long,ByVal Context As String)  
End Sub
```

```
VFP LPARAMETERS Start,Context
```

```
Xbas... PROCEDURE OnOnContext(oExpression,Start,Context)  
RETURN
```

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

```
Java... <SCRIPT EVENT="OnContext(Start,Context)" LANGUAGE="JScript">  
</SCRIPT>
```

```
VBSc... <SCRIPT LANGUAGE="VBScript">  
Function OnContext(Start,Context)  
End Function
```

```
</SCRIPT>
```

Visual
Data...

```
Procedure OnComOnContext Integer IStart String IContext  
    Forward Send OnComOnContext IStart IContext  
End_Procedure
```

Visual
Objects

```
METHOD OCX_OnContext(Start,Context) CLASS MainDialog  
RETURN NIL
```

X++

```
void onEvent_OnContext(int _Start,str _Context)  
{  
}
```

XBasic

```
function OnContext as v (Start as N,Context as C)  
end function
```

dBASE

```
function nativeObject_OnContext(Start,Context)  
return
```

event ScrollButtonClick (ScrollBar as ScrollBarEnum, ScrollPart as ScrollPartEnum)

Occurs when the user clicks a button in the scrollbar.

Type	Description
ScrollBar as ScrollBarEnum	A ScrollBarEnum expression that specifies the scrollbar being clicked.
ScrollPart as ScrollPartEnum	A ScrollPartEnum expression that indicates the part of the scroll being clicked.

Use the ScrollButtonClick event to notify your application that the user clicks a button in the control's scrollbar. The ScrollButtonClick event is fired when the user clicks and releases the mouse over an enabled part of the scroll bar. Use the [ScrollBars](#) property to specify the visible scrollbars in the control. Use the [ScrollPartVisible](#) property to add or remove buttons/parts in the control's scrollbar. Use the [ScrollPartEnable](#) property to specify enable or disable parts in the control's scrollbar. Use the [ScrollPartCaption](#) property to specify the caption of the scroll's part. Use the [Background](#) property to change the visual appearance for any part in the control's scroll bar.

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

```
C# private void ScrollButtonClick(object sender,exontrol.EXPRESSIONLib.ScrollBarEnum ScrollBar,exontrol.EXPRESSIONLib.ScrollPartEnum ScrollPart)
{
}
```

```
VB Private Sub ScrollButtonClick(ByVal sender As System.Object,ByVal ScrollBar As exontrol.EXPRESSIONLib.ScrollBarEnum,ByVal ScrollPart As exontrol.EXPRESSIONLib.ScrollPartEnum) Handles ScrollButtonClick
End Sub
```

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

```
C# private void ScrollButtonClick(object sender,
AxEXPRESSIONLib._IExpressionEvents_ScrollButtonClickEvent e)
{
}
```

```
C++ void OnScrollButtonClick(long ScrollBar,long ScrollPart)
```

```
{  
}
```

**C++
Builder**

```
void __fastcall ScrollButtonClick(TObject *Sender,Expressionlib_tlb::ScrollBarEnum  
ScrollBar,Expressionlib_tlb::ScrollPartEnum ScrollPart)  
{  
}
```

Delphi

```
procedure ScrollButtonClick(ASender: TObject; ScrollBar :  
ScrollBarEnum;ScrollPart : ScrollPartEnum);  
begin  
end;
```

**Delphi 8
(.NET
only)**

```
procedure ScrollButtonClick(sender: System.Object; e:  
AxEXPRESSIONLib._IExpressionEvents_ScrollButtonClickEvent);  
begin  
end;
```

Powe...

```
begin event ScrollButtonClick(long ScrollBar,long ScrollPart)  
end event ScrollButtonClick
```

VB.NET

```
Private Sub ScrollButtonClick(ByVal sender As System.Object, ByVal e As  
AxEXPRESSIONLib._IExpressionEvents_ScrollButtonClickEvent) Handles  
ScrollButtonClick  
End Sub
```

VB6

```
Private Sub ScrollButtonClick(ByVal ScrollBar As  
EXPRESSIONLibCtl.ScrollBarEnum,ByVal ScrollPart As  
EXPRESSIONLibCtl.ScrollPartEnum)  
End Sub
```

VBA

```
Private Sub ScrollButtonClick(ByVal ScrollBar As Long,ByVal ScrollPart As Long)  
End Sub
```

VFP

```
LPARAMETERS ScrollBar,ScrollPart
```

Xbas...

```
PROCEDURE OnScrollButtonClick(oExpression,ScrollBar,ScrollPart)
```

RETURN

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

```
Java... <SCRIPT EVENT="ScrollButtonClick(ScrollBar,ScrollPart)" LANGUAGE="JScript">
</SCRIPT>
```

```
VBS... <SCRIPT LANGUAGE="VBScript">
Function ScrollButtonClick(ScrollBar,ScrollPart)
End Function
</SCRIPT>
```

```
Visual Data... Procedure OnComScrollButtonClick OLEScrollBarEnum IScrollBar
OLEScrollPartEnum IScrollPart
    Forward Send OnComScrollButtonClick IScrollBar IScrollPart
End_Procedure
```

```
Visual Objects METHOD OCX_ScrollButtonClick(ScrollBar,ScrollPart) CLASS MainDialog
RETURN NIL
```

```
X++ void onEvent_ScrollButtonClick(int _ScrollBar,int _ScrollPart)
{
}
```

```
XBasic function ScrollButtonClick as v (ScrollBar as
OLE::Exontrol.Expression.1::ScrollBarEnum,ScrollPart as
OLE::Exontrol.Expression.1::ScrollPartEnum)
end function
```

```
dBASE function nativeObject_ScrollButtonClick(ScrollBar,ScrollPart)
return
```

The following VB sample displays the identifier of the scroll's button being clicked:

With Expression1

```
.ScrollPartVisible(exVScroll, exLeftB1Part Or exRightB1Part) = True
```

```
.ScrollPartCaption(exVScroll, exLeftB1Part) = "1"
```

```
.ScrollPartCaption(exVScroll, exRightB1Part) = "2"
```

```
End With
```

```
Private Sub Expression1_ScrollButtonClick(ByVal ScrollPart As  
EXPRESSIONLibCtl.ScrollPartEnum)  
    MsgBox (ScrollPart)  
End Sub
```

The following VB.NET sample displays the identifier of the scroll's button being clicked:

```
With AxExpression1  
    .set_ScrollPartVisible(EXPRESSIONLib.ScrollBarEnum.exVScroll,  
EXPRESSIONLib.ScrollPartEnum.exLeftB1Part Or  
EXPRESSIONLib.ScrollPartEnum.exRightB1Part, True)  
    .set_ScrollPartCaption(EXPRESSIONLib.ScrollBarEnum.exVScroll,  
EXPRESSIONLib.ScrollPartEnum.exLeftB1Part, "1")  
    .set_ScrollPartCaption(EXPRESSIONLib.ScrollBarEnum.exVScroll,  
EXPRESSIONLib.ScrollPartEnum.exRightB1Part, "2")  
End With
```

```
Private Sub AxExpression1_ScrollButtonClick(ByVal sender As System.Object, ByVal e As  
AxEXPRESSIONLib._IExpressionEvents_ScrollButtonClickEvent) Handles  
AxExpression1.ScrollButtonClick  
    MessageBox.Show( e.scrollPart.ToString())  
End Sub
```

The following C# sample displays the identifier of the scroll's button being clicked:

```
axExpression1.set_ScrollPartVisible(EXPRESSIONLib.ScrollBarEnum.exVScroll,  
EXPRESSIONLib.ScrollPartEnum.exLeftB1Part |  
EXPRESSIONLib.ScrollPartEnum.exRightB1Part, true);  
axExpression1.set_ScrollPartCaption(EXPRESSIONLib.ScrollBarEnum.exVScroll,  
EXPRESSIONLib.ScrollPartEnum.exLeftB1Part , "1");  
axExpression1.set_ScrollPartCaption(EXPRESSIONLib.ScrollBarEnum.exVScroll,  
EXPRESSIONLib.ScrollPartEnum.exRightB1Part, "2");
```

```
private void axExpression1_ScrollButtonClick(object sender,  
AxEXPRESSIONLib._IExpressionEvents_ScrollButtonClickEvent e)
```

```
{  
    MessageBox.Show(e.scrollPart.ToString());  
}
```

The following C++ sample displays the identifier of the scroll's button being clicked:

```
m_edit.SetScrollPartVisible( 0 /*exVScroll*/, 32768 /*exLeftB1Part*/ | 32  
/*exRightB1Part*/, TRUE );  
m_edit.SetScrollPartCaption( 0 /*exVScroll*/, 32768 /*exLeftB1Part*/ , _T("1") );  
m_edit.SetScrollPartCaption( 0 /*exVScroll*/, 32 /*exRightB1Part*/ , _T("2") );
```

```
void OnScrollButtonClickExpression1(long ScrollPart)  
{  
    CString strFormat;  
    strFormat.Format( _T("%i"), ScrollPart );  
    MessageBox( strFormat );  
}
```

The following VFP sample displays the identifier of the scroll's button being clicked:

```
With thisform.Expression1  
    .ScrollPartVisible(0, bitor(32768,32)) = .t.  
    .ScrollPartCaption(0,32768) = "1"  
    .ScrollPartCaption(0, 32) = "2"  
EndWith
```

event SelChange ()

Occurs when the user selects text in the control.

Type

Description

Use the SelChange event to notify your application that the selected text was changed. The SelChange event is called even if the selection is empty and user moves the cursor. The SelChange event is called anytime when the position of the cursor was changed. Use the [SelText](#) property to get the selected text. The [SelStart](#) and [SelLength](#) properties determine the position of the selected text. Use the [SelForeColor](#) and [SelBackColor](#) properties to specify the colors for the selected text.

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(oExpression)
RETURN

Syntax for SelChange event, **ICOM** 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 Expression1_SelChange()  
    If Not Expression1.SelText = "" Then  
        Debug.Print Expression1.SelText  
    End If  
End Sub
```

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

```
void OnSelChangeExpression1()  
{  
    OutputDebugString( m_edit.GetSelText() );  
}
```

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

```
Private Sub AxExpression1_SelChange(ByVal sender As Object, ByVal e As  
System.EventArgs) Handles AxExpression1.SelChange  
    With AxExpression1  
        Debug.WriteLine(.SelText)  
    End With  
End Sub
```

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

```
private void axExpression1_SelChange(object sender, EventArgs e)
{
    System.Diagnostics.Debug.WriteLine(axExpression1.SelText);
}
```

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

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

with thisform.Expression1
    wait window nowait .SelText
endwith
```

event SplitterChange ()

Occurs when the user splits the control.

Type

Description

Use the SplitterChange event to notify your application that user splits the control. The SplitterChange event is Use the [SplitPaneWidth](#) and [SplitPaneHeight](#) properties to control the size of the panes inside the edit control. The [AllowSplitter](#) property shows or hides the splitters in the edit control. Use the [FocusPane](#) property to specify the pane that has the focus. Use the [SplitPaneWidth](#) property to horizontally split the control. Use the [SplitPaneHeight](#) property to vertically split the control.

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

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

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

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

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

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

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

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

Delphi 8
(.NET
only)

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

Powe...
begin event SplitterChange()
end event SplitterChange

VB.NET
Private Sub SplitterChange(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles SplitterChange
End Sub

VB6
Private Sub SplitterChange()
End Sub

VBA
Private Sub SplitterChange()
End Sub

VFP
LPARAMETERS nop

Xbas...
PROCEDURE OnSplitterChange(oExpression)
RETURN

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

Java...
<SCRIPT EVENT="SplitterChange()" LANGUAGE="JScript">
</SCRIPT>

VBSc...
<SCRIPT LANGUAGE="VBScript">
Function SplitterChange()
End Function
</SCRIPT>

Visual
Data...
Procedure OnComSplitterChange
Forward Send OnComSplitterChange

End_Procedure

Visual
Objects

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

X++

```
void onEvent_SplitterChange()  
{  
}
```

XBasic

```
function SplitterChange as v ()  
end function
```

dBASE

```
function nativeObject_SplitterChange()  
return
```

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 *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 than 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. 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 .