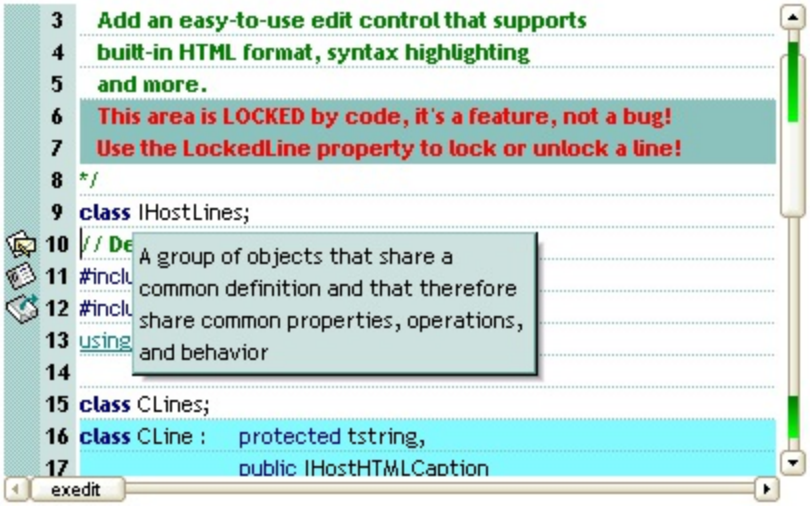
ab ExEdit

Add an easy-to-use edit control that supports built-in HTML format, and syntax highlighting as well. You can define your own keywords and expressions using built-in HTML expressions. The exEdit 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.

Features include:

- **Syntax highlighting** support
- **WYSWYG Template/Layout Editor** support
- **Skinnable Interface** support (ability to apply a skin to the any background part, including the scroll bars)
- **Incremental Search** support
- Locked-Unlocked lines support
- **Find** and Replace support
- Multi levels **Undo/Redo** support
- **Code completion** support
- Overstrike, Insert mode support
- Ability to assign multiple lines HTML **tooltip** to any keyword
- Ability to define **multiple lines expressions (blocks, multiple lines comments)**
- Ability to assign icons and colors to any line
- Ability to define the HTML format for numbers
- Ability to mark (on the control's vertical scroll bar) the positions for expressions, blocks or keywords
- Fast and fully customizable
- OLE **Drag and Drop** support.
- Context sensitive support
- Built-in **HTML support**
- Ability to load icons using the BASE64 encoded strings
- Bookmarks support
- Ability to display the control's text using built-in HTML tags
- Ability to define sensitive keywords and expressions using built-in HTML tags
- Ability to load or save files
- Ability to put a picture on control's background
- Ability to select a rectangular area using ALT key
- Autoindent and brackets automatic indenting
- Any font supported, no true type font required
- Ability to display a line number border
- Ability to insert lines, delete lines on the fly

- Ability to assign icons and colors for any line.
- ANSI and UNICODE versions available
- **IME** (Input Method Editor) support



Ž ExEdit 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

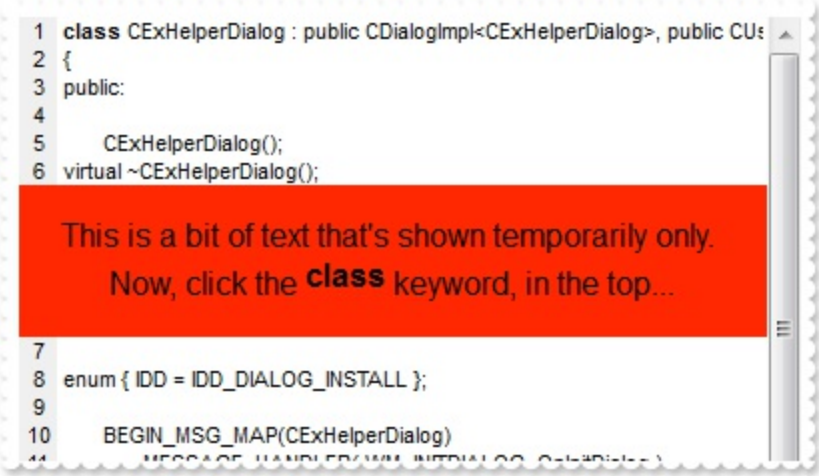
constants AlignmentEnum

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 AllowInlineToolTipEnum

The AllowInlineToolTipEnum type defines options to show the control's Inline-ToolTip feature. The Inline-ToolTip feature allows displaying inline tooltips of the keywords found at the current line as shown in the picture above. Use the AllowInlineToolTip property to enable or disable the control's inline-tooltip feature. For instance, you can use the Inline-ToolTip feature to display inside the control at current line, any information/html text you may need. The AddKeyword method adds a keyword to a sensitive control. The ToolTip/ToolTipTile parameters of the AddKeyword method defines the inline-tooltip for the specified keyword. The TempInlineToolTip property specifies a temporarily inline tooltip to be shown at the current line. The ExcludeInlineToolTip property specifies a list of keywords, separated by , (comma character), that specifies the keywords to be excluded from the inline tooltip support. The AllowInlineToolTipEnum supports the following values (combination of them):



Name	Value	Description
exDisableInlineToolTip	0	The inline tooltip support is disabled.
exInlineToolTip	1	The inline tooltip is shown as soon as the user clicks/types a keyword with a tooltip.
exInlineToolTipOnChange	2	The inline tooltip is displayed once the line is changed, and it contains keyword with a tooltip.
exInlineToolTipOver	256	The inline tooltip is shown over the lines, instead between lines.
exInlineToolTipWordWrap	512	The inline tooltip is supports word-wrap.
exInlineToolTipSingle	1024	The inline tooltip displays a single keyword (the first one), even if more are found on the same line.

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.
exContextForeColorUnmatch	4	Specifies the context's item foreground color when it does not match the typed characters.
		Indicates the visual appearance of the borders of the tooltips. Use the ToolTipPopDelay property specifies the period in ms of time the ToolTip

exToolTipAppearance	64	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.
exInlineToolTipBackColor	158	Specifies the visual appearance/background color of the inline tooltip. The AllowInlineToolTip property indicates that the current line shows inline tooltip for keywords found to the caret.
exInlineToolTipForeColor	159	Specifies the foreground color of the inline tooltip. The AllowInlineToolTip property indicates that the current line shows inline tooltip for keywords found to the caret.
exTempInlineToolTipBackColor	160	Specifies the visual appearance/background color of the temporarily inline tooltip. If not specified, the exInlineToolTipBackColor flag is used instead. The TempInlineToolTip property specifies a temporarily inline tooltip to be shown at the current line.
exTempInlineToolTipForeColor	161	Specifies the foreground color of the inline temporarily tooltip. If not specified, the exInlineToolTipForeColor flag is used instead. The TempInlineToolTip property specifies a temporarily inline tooltip to be shown at the current line.
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 .
		exScrollHoverAll. Indicates whether the hover-all

exScrollHoverAll	500	feature is by default (0), always on (-1) or disabled(1).
exVSThumbExt	503	The thumb-extension part in normal state.
exVSThumbExtP	504	The thumb-extension part when it is pressed.
exVSThumbExtD	505	The thumb-extension part when it is disabled.
exVSThumbExtH	506	The thumb-extension when the cursor hovers it.
exHSThumbExt	507	The thumb-extension in normal state.
exHSThumbExtP	508	The thumb-extension when it is pressed.
exHSThumbExtD	509	The thumb-extension when it is disabled.
exHSThumbExtH	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	The cursor is on the control's edit area.
exBookmarkArea	1	The cursor is on the control's bookmark bar.
exLineNumberArea	2	The cursor is on the control's line number bar.
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 CodeCompletionEnum

The CodeCompletionEnum type how the code completion feature works. The CodeCompletation property specifies whether the code completion feature is enabled or disabled. The CodeCompletionEnum type supports the following values:

Name	Value	Description
exCodeCompletionDisable	0	The control supports no code completion feature.
exCodeCompletionEnable	-1	The control supports code completion feature, and the context list panel is shown.
exCodeCompletionEnableNoUI	1	The control supports code completion feature, but the context list panel is not shown.

constants ContextOptionEnum

Specifies the options for a sensitive context list. Use the Options property to specify the options for a sensitive context list.

Name	Value	Description
exContextCaseSensitive	0	The selection of items to a context list is case sensitive.
exContextAllowChars	1	Specifies the list of characters that are allowed to a context list.
exContextAllowToolTip	2	Indicates if the sensitive context allows displaying tooltips for context items.
exContextItemToolTip	3	Specifies the tooltip for the latest added item, in the sensitive context.
exContextItemToolTipTitle	4	Specifies the title for the tooltip of the latest inserted item, in the sensitive context.
exContextAllowSpaceOnFront	5	Specifies that the sensitive context allows space characters to be typed, on front of the current context.
exContextInsertCaretPos	6	Returns or sets a numeric expression that determines the position to display the control's caret, once the user selects and inserts an item from the control's sensitive context. The value keyword indicates the item being inserted, or t??B
exContextDisableIncrementalSearchContains	7	exContextDisableIncrementalSearchContains. Specifies whether the Incremental Search Contains for control's sensitive context is disabled or enabled.

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 EditTypeEnum

Determines the control's edit type. Use the EditType property to change the control's edit type.

Name	Value	Description
exStandard	0	The control acts like a standard edit control
exSensitive	1	The control is sensitive to context, and highlights the keywords and expressions. Use the AddKeyword, AddExpression, DeleteKeyword, DeleteExpression, ClearKeywords, ClearExpressions to update the control's keywords and expressions collections.
exHTML	2	The control accepts HTML expressions that combines the following tags: • **bold** • *italic* • <s>~~strikeout~~</s> • <u>underline</u> • <fgcolor=RRGGBB>fgcolor</fgcolor> • <bgcolor=RRGGBB>bgcolor</bgcolor>

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	-16639	A 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 exOLEDropEffectEnum

Drop effect constants for OLE drag and drop events.

Name	Value	Description
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.
exOLEDropEffectScroll	-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 KeywordSensitiveEnum

Specifies if the keyword is case sensitive. Use the AddKeyword property or AddExpression method to add keywords or expressions to your control.

Name	Value	Description
exCaseSensitiveKeyword	0	The keyword is case-sensitive.
exNonCaseSensitiveKeyword	1	The keyword is not case-sensitive.
exReplaceKeyword	2	The keyword is always like it is defined.

constants PictureDisplayEnum

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.

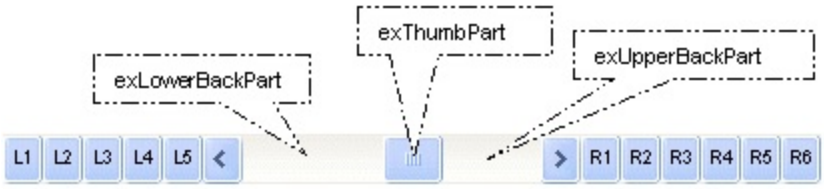
constants ScrollBarsEnum

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 forth 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 edit control. A splitter is shown only if the edit 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.
exBothSplitter	3	The control displays both splitters.

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)

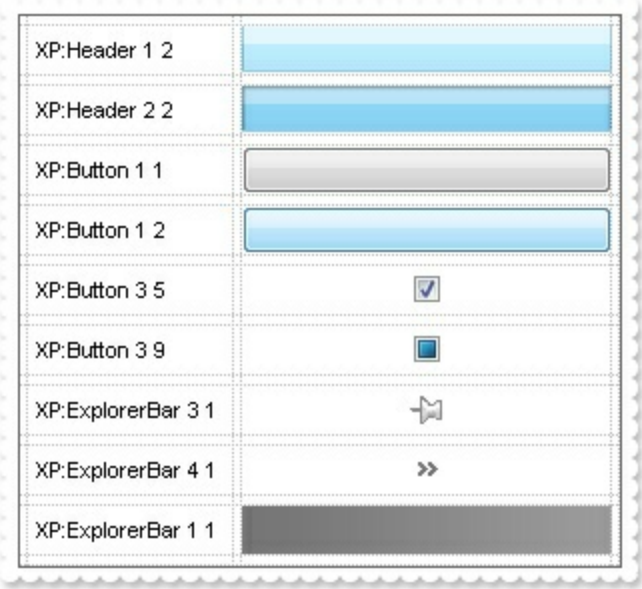
Adds or replaces a skin object to the control.

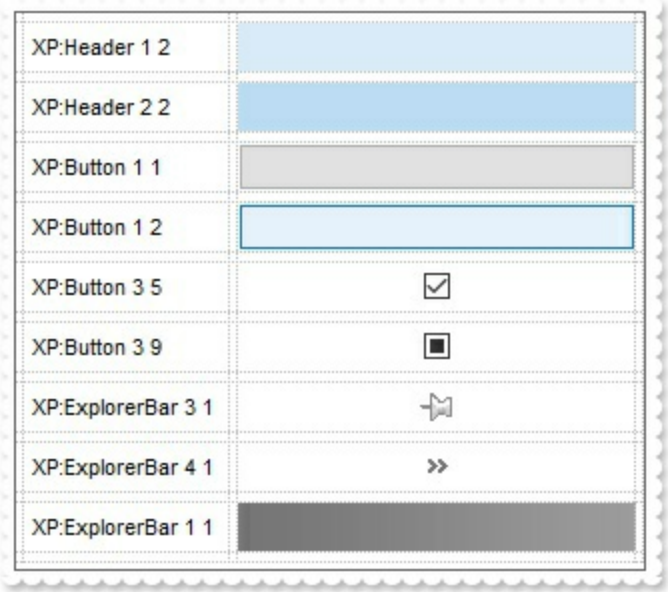
Type	Description
ID as Long	A Long expression that indicates the index of the skin being added or replaced. The value must be between 1 and 126, so Appearance collection should holds no more than 126 elements.
	The Skin parameter of the Add method can a STRING as explained bellow, a BYTE[] / safe arrays 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)) If the Skin parameter points to a string expression, it can be one of the following: • 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

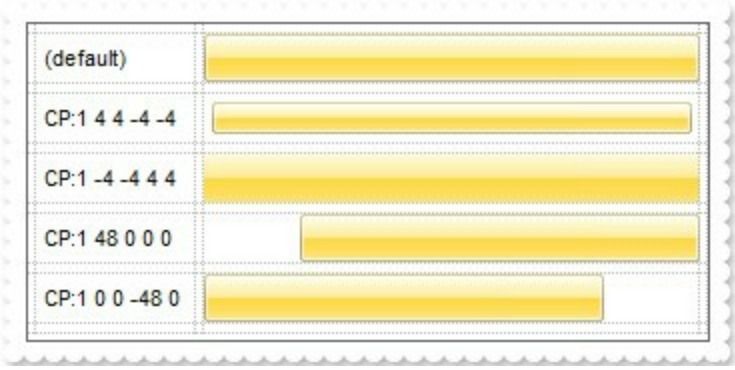




- A copy of another skin with different coordinates (position, size), if the Skin parameter starts with "**CP**:". Use this option, to display the EBN, using different coordinates (position, size). By default, the EBN skin object is rendered on the part's client area. Using this option, you can display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

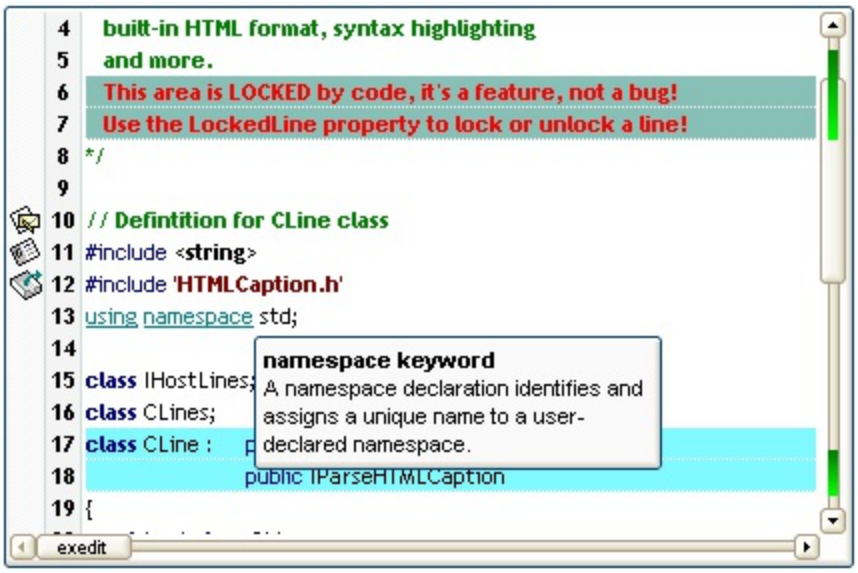
where the ID is the identifier of the EBN to be used (it is a number that specifies the ID parameter of the Add method), Left, Top, Right and Bottom parameters/numbers specifies the relative position to the part's client area, where the EBN should be rendered. The Left, Top, Right and Bottom parameters 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	Description
Boolean	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.

Edit 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: {39136531-DD0F-4281-B445-E36FC2CDDBC5}. The object's program identifier is: "Exontrol.Edit". The /COM object module is: "ExEdit.dll"

Add an easy-to-use edit control that supports built-in HTML format, and syntax highlighting as well. You can define your own keywords and expressions using built-in HTML expressions. The control is fully written in C++ using ATL. You should know that the control is written from scratch and **it does NOT use or subclass the RichEdit control or Windows Edit control**. The Edit control supports the following methods and properties:

Name	Description
ActiveContextItems	Specifies the key of the context to be open next.
AddExpression	Adds a single or multiple lines expression to a sensitive control.
AddKeyword	Adds a keyword to a sensitive control.
AddWild	Adds a wild characters expression to a sensitive control.
AllowContextMenu	Specifies whether the control's default context menu is available.
AllowFind	Specifies whether control displays a find dialog when user presses CTRL+F key.
AllowIncrementalSearch	Specifies whether the control allows incremental search.
AllowInlineToolTip	Indicates that the current line shows inline tooltip for keywords found to the caret.
AllowMark	Specifies a value that indicates whether the positions in text of expressions are marked on the vertical scroll bar.
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.
Appearance	Retrieves or sets the control's appearance.
ApplyColors	Specifies whether the control applies the HTML colors.
AttachTemplate	Attaches a script to the current object, including the events, from a string, file, a safe array of bytes.
AutoIndent	Specifies whether the control indents the line when ENTER key is pressed.
	Retrieves or sets a value that indicates the control's

BackColor	background color.
BackColorAlternate	Returns or sets the background color used to display alternate lines in the control.
BackColorCaretLine	Returns or sets the background color used to display the cursor line.
BackColorLine	Specifies the line's background color.
BackColorLockedLine	Retrieves or sets a value that indicates the line's background color when it is locked.
Background	Returns or sets a value that indicates the background color for parts in the control.
Bookmark	Adds or removes a bookmark.
BookMarkBackColor	Specifies the bookmark bar background's color.
BookMarkBackColor2	Specifies the bookmark bar background's color.
BookmarkBorderColor	Specifies the color for bookmark's border.
BookmarkImage	Specifies the index of image being displayed when the line has a bookmark.
BookmarkImageLine	Retrieves or sets a value that indicates the index of image being displayed on the bookmark header.
BookmarksList	Specifies the list of bookmarks..
BookmarkWidth	Specifies the bookmarks border's width.
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 or Replace dialog.
CaretLine	Specifies the line of caret.
CaretPos	Retrieves or sets a value that indicates the position of the caret in the line.
CaretPosX	Retrieves the x-coordinate of the caret in pixels, relative to the control's client area.
CaretPosY	Retrieves the y-coordinate of the caret, in pixels relative to the control's client area.
ChangeOnKey	Specifies the code of the last key that alters the control's text.
ClearBackColorLine	Clears the line's background color.

ClearBookmarks	Clears all bookmarks.
ClearExpressions	Clears the expressions collection into a sensitive control.
ClearForeColorLine	Clears the line's foreground color.
ClearKeywords	Clears the keywords collection into a sensitive control.
ClearWilds	Clears the wild characters expressions collection into a sensitive control.
ClientToCursor	Converts client coordinates to cursor coordinates.
CloseBrackets	Specifies the list of closed brackets.
CodeCompletion	Specifies whether the code completion feature is enabled or disabled.
Context	Gets the context's collection.
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.
Count	Counts the lines in the control.
Cursor	Gets or sets the cursor that is displayed when the mouse pointer hovers the control.
DeleteExpression	Deletes an expression from expressions collection.
DeleteKeyword	Deletes a keyword from keywords collection.
DeleteLine	Deletes the line.
DeleteWild	Deletes an entry from the wild characters expressions collection.
DisplaySelection	Specifies whether the control displays the selection.
DrawGridLines	Returns or sets a value that determines whether lines are drawn between rows, or unpopulated areas.
EditType	Specifies the control's type.
Enabled	Enables or disables the control.
EnsureVisibleLine	Ensures that specified line is visible.
EventParam	Retrieves or sets a value that indicates the current's event parameter.
ExcludeInlineToolTip	Specifies a list of keywords, separated by , (comma character), that specifies the keywords to be excluded from the inline tooltip support.

ExecuteTemplate	Executes a template and returns the result.
FileName	Specifies the name of the file being loaded or saved.
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.
ForeColorLine	Specifies the line's foreground color.
ForeColorLockedLine	Retrieves or sets a value that indicates the line's foreground color when it is locked.
FormatNumbers	Specifies the HTML format that's applied to numbers.
GetSelection	Retrieves the coordinates of the selection.
HideSelection	Specifies whether the selection in the control is hidden when the control loses the focus.
HTMLText	Specifies the control's HTML text.
hWnd	Gets the window's handle.
hWndPane	Gets the window handle of the control's pane.
IgnorePrefixInExpression	Specifies a value that indicates the prefix that's ignored in the expression.
Images	Sets the control's images list at runtime. The Handle should be a handle to an Image List control.
ImageSize	Specifies the size of the control' icons.
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.
IndentSel	Indents the selected text.
InsertText	Inserts text to control.
LineHeight	Specifies an expression that determines the height of the line within the editor.
LineNumberBackColor	Specifies the background color of the line number bar.
LineNumberBorderColor	Specifies the color for line number's border.

LineNumberFont	Retrieves or sets the font of the line numbers bar.
LineNumberForeColor	Specifies the foreground color of the line number bar.
LineNumberWidth	Specifies the width of the border where the that contains the line numbers.
Load	Loads a file.
LoadFromUnicode	Loads from an UNICODE file.
Locked	Determines whether a control can be edited.
LockedLine	Specifies whether a line is locked or unlocked.
Margin	Defines the distance between text and inner border (border, line-numbers or bookmarks).
MarkColor	Specifies a color being used to mark the position of the expression, on the control's vertical scroll bar.
MarkContinueBlocks	Marks continuously the blocks.
MultiLine	Specifies whether the control accepts multiple lines.
NextBookmark	Moves the cursor to the next bookmark.
OLEDrag	Causes a component to initiate an OLE drag/drop operation.
OLEDropMode	Returns or sets how a target component handles drop operations
OpenBrackets	Specifies the list of opened brackets.
Overtype	Specifies whether the control is running in overtype mode.
PagesContextItems	Specifies the pages (key[:caption]) of the control's senzitive context, separted by comma character.
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
PrevBookmark	Moves the cursor to the prior bookmark.
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.
ReplaceIcon	Adds a new icon, replaces an icon or clears the control's image list.
Save	Saves the control's text to a file.

SaveAsUnicode	Saves the control's text to file as UNICODE.
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.
SelectLine	Selects a line by code.
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.

SetSelection	Selects the text giving the start and end coordinates.
Show	Specifies the order and visibility of wild, keyword, expression and format-numbers within the control.
ShowCaret	Specifies whether the control's caret is visible or hidden.
ShowContext	Shows the control's context, as the user would press the CTRL + SPACE key.
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.
TempInlineToolTip	Specifies a temporarily inline tooltip to be shown on the current line.
Template	Specifies the control's template.
TemplateDef	Defines inside variables for the next Template/ExecuteTemplate call.
TemplatePut	Defines inside variables for the next Template/ExecuteTemplate call.
Text	Specifies the control's text.
TextLine	Specifies the line based on its index.
ToolTipDelay	Specifies the time in ms that passes before the ToolTip appears.
ToolTipFont	Retrieves or sets the tooltip's font.
ToolTipMargin	Defines the size of the control's tooltip margins.
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.

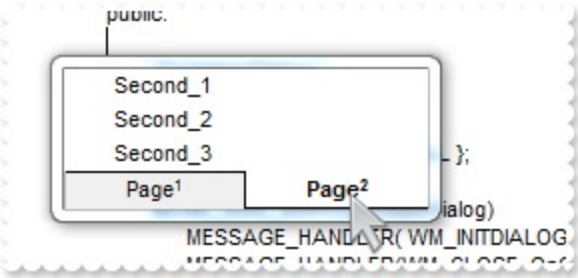
UseTabKey	Specifies whether the control uses the TAB key.
Version	Retrieves the control's version.
VisualAppearance	Retrieves the control's appearance.
WordFromPoint	Retrieves the word from the cursor.

property Edit.ActiveContextItems as String

Specifies the key of the context to be open next.

Type	Description
String	A String expression that specifies the key of the context to be open next (value of the Context parameter of the Context property).

By default, the ActiveContextItems property is empty. The ActiveContextItems property defines the key of the current context. The PagesContextItems property specifies the pages (key[:caption]) of the control's sensitive context, separated by comma character. Use the Context collection to customize the list of keywords when the code completion is enabled. The control can display multiple pages on the control's sensitive context, if multiple key-context are defined. Use the CodeCompletion property to disable the code completion support. By default, the user can display the control's context window by pressing the CTRL + SPACE key combination. Use the ContextKey property to define the key combination to open the control's context window. The ShowContext method shows programmatically the control's context, as the user would press the CTRL + SPACE key.



How can I display more pages on the control's sensitive context?

VBA (MS Access, Excell...)

```
With Edit1
   With .Context()
      .Add "First_1"
      .Add "First_2"
   End With
   With .Context("Second")
      .Add "Second_1"
      .Add "Second_2"
      .Add "Second_3"
   End With
   .ActiveContextItems = "Second"
```

```
   .PagesContextItems = ":Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>"
End With
```

VB6

```
With Edit1
   With .Context()
      .Add "First_1"
      .Add "First_2"
   End With
   With .Context("Second")
      .Add "Second_1"
      .Add "Second_2"
      .Add "Second_3"
   End With
   .ActiveContextItems = "Second"
   .PagesContextItems = ":Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>"
End With
```

VB.NET

```
With Exedit1
   With .get_Context()
      .Add("First_1")
      .Add("First_2")
   End With
   With .get_Context("Second")
      .Add("Second_1")
      .Add("Second_2")
      .Add("Second_3")
   End With
   .ActiveContextItems = "Second"
   .PagesContextItems = ":Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>"
End With
```

VB.NET for /COM

```
With AxEdit1
   With .get_Context()
      .Add("First_1")
      .Add("First_2")
   End With
   With .get_Context("Second")
      .Add("Second_1")
      .Add("Second_2")
      .Add("Second_3")
   End With
   .ActiveContextItems = "Second"
   .PagesContextItems = ":Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>"
End With
```

C++

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

    #import <ExEdit.dll>
    using namespace EXEDITLib;
*/
EXEDITLib::IEditPtr spEdit1 = GetDlgItem(IDC_EDIT1)->GetControlUnknown();
EXEDITLib::IEditContextPtr var_Context = spEdit1->GetContext(vtMissing);
   var_Context->Add(L"First_1",vtMissing,vtMissing,vtMissing);
   var_Context->Add(L"First_2",vtMissing,vtMissing,vtMissing);
EXEDITLib::IEditContextPtr var_Context1 = spEdit1->GetContext("Second");
   var_Context1->Add(L"Second_1",vtMissing,vtMissing,vtMissing);
   var_Context1->Add(L"Second_2",vtMissing,vtMissing,vtMissing);
   var_Context1->Add(L"Second_3",vtMissing,vtMissing,vtMissing);
spEdit1->PutActiveContextItems(L"Second");
spEdit1->PutPagesContextItems(L":Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>");
```

C++ Builder

```
Exeditlib_tlb::IEditContextPtr var_Context = Edit1->Context[TNoParam()];
   var_Context->Add(L"First_1",TNoParam(),TNoParam(),TNoParam());
   var_Context->Add(L"First_2",TNoParam(),TNoParam(),TNoParam());
Exeditlib_tlb::IEditContextPtr var_Context1 = Edit1->Context[TVariant("Second")];
   var_Context1->Add(L"Second_1",TNoParam(),TNoParam(),TNoParam());
   var_Context1->Add(L"Second_2",TNoParam(),TNoParam(),TNoParam());
   var_Context1->Add(L"Second_3",TNoParam(),TNoParam(),TNoParam());
Edit1->ActiveContextItems = L"Second";
Edit1->PagesContextItems = L":Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>";
```

C#

```
exontrol.EXEDITLib.Context var_Context = exedit1.get_Context(null);
   var_Context.Add("First_1",null,null,null);
   var_Context.Add("First_2",null,null,null);
exontrol.EXEDITLib.Context var_Context1 = exedit1.get_Context("Second");
   var_Context1.Add("Second_1",null,null,null);
   var_Context1.Add("Second_2",null,null,null);
   var_Context1.Add("Second_3",null,null,null);
exedit1.ActiveContextItems = "Second";
exedit1.PagesContextItems = ":Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>";
```

JScript/JavaScript

```
<BODY onload="Init()">
<OBJECT CLASSID="clsid:39136531-DD0F-4281-B445-E36FC2CDDBC5" id="Edit1">
</OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
   var var_Context = Edit1.Context(null);
```

```
      var_Context.Add("First_1",null,null,null);
      var_Context.Add("First_2",null,null,null);
   var var_Context1 = Edit1.Context("Second");
      var_Context1.Add("Second_1",null,null,null);
      var_Context1.Add("Second_2",null,null,null);
      var_Context1.Add("Second_3",null,null,null);
   Edit1.ActiveContextItems = "Second";
   Edit1.PagesContextItems = ":Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>";
}
</SCRIPT>
</BODY>
```

VBScript

```
<BODY onload="Init()">
<OBJECT CLASSID="clsid:39136531-DD0F-4281-B445-E36FC2CDDBC5" id="Edit1">
</OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
   With Edit1
      With .Context()
         .Add "First_1"
         .Add "First_2"
      End With
      With .Context("Second")
         .Add "Second_1"
         .Add "Second_2"
         .Add "Second_3"
      End With
      .ActiveContextItems = "Second"
      .PagesContextItems = ":Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>"
   End With
End Function
```

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

C# for /COM

```
EXEDITLib.Context var_Context = axEdit1.get_Context(null);
   var_Context.Add("First_1",null,null,null);
   var_Context.Add("First_2",null,null,null);
EXEDITLib.Context var_Context1 = axEdit1.get_Context("Second");
   var_Context1.Add("Second_1",null,null,null);
   var_Context1.Add("Second_2",null,null,null);
   var_Context1.Add("Second_3",null,null,null);
axEdit1.ActiveContextItems = "Second";
axEdit1.PagesContextItems = ":Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>";
```

X++ (Dynamics Ax 2009)

```
public void init()
{
   COM com_Context,com_Context1;
   anytype var_Context,var_Context1;
   ;

   super();

   var_Context = exedit1.Context(); com_Context = var_Context;
      com_Context.Add("First_1");
      com_Context.Add("First_2");
   var_Context1 = exedit1.Context("Second"); com_Context1 = var_Context1;
      com_Context1.Add("Second_1");
      com_Context1.Add("Second_2");
      com_Context1.Add("Second_3");
   exedit1.ActiveContextItems("Second");
   exedit1.PagesContextItems(":Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>");
```

```
}
```

Delphi 8 (.NET only)

```
with AxEdit1 do
begin
   with get_Context(Nil) do
   begin
      Add('First_1',Nil,Nil,Nil);
      Add('First_2',Nil,Nil,Nil);
   end;
   with get_Context('Second') do
   begin
      Add('Second_1',Nil,Nil,Nil);
      Add('Second_2',Nil,Nil,Nil);
      Add('Second_3',Nil,Nil,Nil);
   end;
   ActiveContextItems := 'Second';
   PagesContextItems := ':Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>';
end
```

Delphi (standard)

```
with Edit1 do
begin
   with Context[Null] do
   begin
      Add('First_1',Null,Null,Null);
      Add('First_2',Null,Null,Null);
   end;
   with Context['Second'] do
   begin
      Add('Second_1',Null,Null,Null);
      Add('Second_2',Null,Null,Null);
      Add('Second_3',Null,Null,Null);
   end;
   ActiveContextItems := 'Second';
```

```
   PagesContextItems := ':Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>';
end
```

VFP

```
with thisform.Edit1
   with .Context()
      .Add("First_1")
      .Add("First_2")
   endwith
   with .Context("Second")
      .Add("Second_1")
      .Add("Second_2")
      .Add("Second_3")
   endwith
   .ActiveContextItems = "Second"
   .PagesContextItems = ":Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>"
endwith
```

dBASE Plus

```
local oEdit,var_Context,var_Context1

oEdit = form.EXEDITACTIVEXCONTROL1.nativeObject
var_Context = oEdit.Context()
   var_Context.Add("First_1")
   var_Context.Add("First_2")
var_Context1 = oEdit.Context("Second")
   var_Context1.Add("Second_1")
   var_Context1.Add("Second_2")
   var_Context1.Add("Second_3")
oEdit.ActiveContextItems = "Second"
oEdit.PagesContextItems = ":Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>"
```

XBasic (Alpha Five)

```
Dim oEdit as P
Dim var_Context as P
Dim var_Context1 as P

oEdit = topparent:CONTROL_ACTIVEX1.activex
var_Context = oEdit.Context()
   var_Context.Add("First_1")
   var_Context.Add("First_2")
var_Context1 = oEdit.Context("Second")
   var_Context1.Add("Second_1")
   var_Context1.Add("Second_2")
   var_Context1.Add("Second_3")
oEdit.ActiveContextItems = "Second"
oEdit.PagesContextItems = ":Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>"
```

Visual Objects

```
local var_Context,var_Context1 as IEditContext

var_Context := oDCOCX_Exontrol1:[Context,nil]
   var_Context:Add("First_1",nil,nil,nil)
   var_Context:Add("First_2",nil,nil,nil)
var_Context1 := oDCOCX_Exontrol1:[Context,"Second"]
   var_Context1:Add("Second_1",nil,nil,nil)
   var_Context1:Add("Second_2",nil,nil,nil)
   var_Context1:Add("Second_3",nil,nil,nil)
oDCOCX_Exontrol1:ActiveContextItems := "Second"
oDCOCX_Exontrol1:PagesContextItems := ":Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>"
```

PowerBuilder

```
OleObject oEdit,var_Context,var_Context1
```

```
oEdit = ole_1.Object
var_Context = oEdit.Context()
   var_Context.Add("First_1")
   var_Context.Add("First_2")
var_Context1 = oEdit.Context("Second")
   var_Context1.Add("Second_1")
   var_Context1.Add("Second_2")
   var_Context1.Add("Second_3")
oEdit.ActiveContextItems = "Second"
oEdit.PagesContextItems = ":Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>"
```

Visual DataFlex

```
Procedure OnCreate
   Forward Send OnCreate
   Variant voContext
   Get ComContext Nothing to voContext
   Handle hoContext
   Get Create (RefClass(cComContext)) to hoContext
   Set pvComObject of hoContext to voContext
      Send ComAdd of hoContext "First_1" Nothing Nothing Nothing
      Send ComAdd of hoContext "First_2" Nothing Nothing Nothing
   Send Destroy to hoContext
   Variant voContext1
   Get ComContext "Second" to voContext1
   Handle hoContext1
   Get Create (RefClass(cComContext)) to hoContext1
   Set pvComObject of hoContext1 to voContext1
      Send ComAdd of hoContext1 "Second_1" Nothing Nothing Nothing
      Send ComAdd of hoContext1 "Second_2" Nothing Nothing Nothing
      Send ComAdd of hoContext1 "Second_3" Nothing Nothing Nothing
   Send Destroy to hoContext1
   Set ComActiveContextItems to "Second"
   Set ComPagesContextItems to ":Page<font ;6><off -4>1</off>
```

```
</font>,Second:Page<font ;6><off -4>2</off></font>"
End_Procedure
```

XBase++

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

PROCEDURE Main
   LOCAL oForm
   LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
   LOCAL oEdit
   LOCAL oContext,oContext1

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

   oEdit := XbpActiveXControl():new( oForm:drawingArea )
   oEdit:CLSID  := "Exontrol.Edit.1" /*{39136531-DD0F-4281-B445-
E36FC2CDDBC5}*/
   oEdit:create(,, {10,60},{610,370} )

      oContext := oEdit:Context()
         oContext:Add("First_1")
         oContext:Add("First_2")
      oContext1 := oEdit:Context("Second")
         oContext1:Add("Second_1")
         oContext1:Add("Second_2")
         oContext1:Add("Second_3")
      oEdit:ActiveContextItems := "Second"
      oEdit:PagesContextItems := ":Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>"

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

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

method Edit.AddExpression (Start as String, Expression as String, End as String, [MultipleLines as Variant], [CaseSensitive as Variant])

Adds a single or multiple lines expression to a sensitive control.

Type	Description
Start as String	A string expression that defines the start of an expression. The Start parameter is a combination of HTML tags. The HTML tags are used to paint the starting of an expression. The text in the expression is used to define the beginning of an expression. The Start parameter should be a non empty string in order to define an expression. *If the first character in the Start parameter is ^, it indicates that the expression should be at the beginning of the line, else it is ignored*. For instance, the AddExpression("^<fgcolor=008000>*</fgcolor>", " <fgcolor=008000> </fgcolor>", ""), makes green the lines that starts with *. For instance, AddExpression("^", " <fgcolor=FF0000> ", ":"), highlights the start of the line until : character is found.
Expression as String	A string expression that defines the HTML tags used to paint the expression. If the parameter is empty the Start HTML tags are used to paint the rest of the expression. The Expression parameter defines the HTML tags used to paint the expression. The text of the Expression is ignored.
End as String	A string expression that defines the end of an expression. The End parameter is a combination of HTML tags that defines the ending string of an expression, and the HTML tags used to paint the end of an expression. If End parameter is Empty the expression ends to the end of line.
MultipleLines as Variant	Optional. A boolean expression that indicates whether the Start expression affects multiple lines or a single line. By default, the MultipleLines parameter is False.
CaseSensitive as Variant	Optional. The CaseSensitive parameter is of KeywordSensitiveEnum type. A long expression that indicates the type of the start and end expressions. The valid values are 0 - exCaseSensitiveKeyword, 1 - exNonCaseSensitiveKeyword and 2 - exReplaceKeyword. By default, the CaseSensitive parameter is 0

(exCaseSensitiveKeyword)

The AddExpression method adds a new expression to control's expressions collection. The method has effect if the control's EditType property is exSensitive. An expression begins with some text and may end with another text, or none. Use the AddKeyword method to add new keywords to the control. Use the IgnorePrefixInExpression property to define the prefixes being ignored by an expression. Use the MarkColor property to mark (on the control's vertical scroll bar) the positions of the expressions in the control's text. Use the FormatNumbers property to specify the format of the numbers in the control.

The list of supported built-in HTML tags is:

- **bold**
- *****italic*
- **<s>**~~strikeout~~</s>
- **<u>**underline</u>
- **<fgcolor=RRGGBB>**fgcolor</fgcolor>
- **<bgcolor=RRGGBB>**bgcolor</bgcolor>
- ****text displays portions of text with a different font and/or different size. For instance, the bit draws the bit text using the Tahoma font, on size 12 pt. If the name of the font is missing, and instead size is present, the current font is used with a different size. For instance, bit displays the bit text using the current font, but with a different size.

The AddExpression adds a new expression only if the text into the Start HTML expression is not empty. For instance if you define an expression like follows: "" no expression will be added. Instead if you use "<fgcolor=008000>//</fgcolor>" it defines the "//" expression.

Here's few samples how to define expressions and their explications:

- **AddExpression("<fgcolor=008000>//</fgcolor>", "", "")**. The statement defines an expression that starts with "//" and ends at the end of line. So, if in the control's text we have lines that contains "//" the control will use the defined expression to paint the text that starts at "//" to the end of the line. For instance. The "just a text // just another text", becomes "just a text **// just another text**"

- **AddExpression("<fgcolor=FFFFFF><bgcolor=FF0000>//</bgcolor></fgcolor> ", "<fgcolor=0000FF><u> </u></fgcolor>", "")**. The statement defines an expression that starts with "//" and ends at the end of line. The expression also defines the HTML tags that will be used to paint the expression excluding the starting of it. So, if in the control's text we have lines that contains "//" the control will use the Start HTML expression to paint the "//", and the for the rest of line, the Expression HTML

expression will be used.. For instance. The "just a text // just another text", becomes "just a text // just another text"

- **AddExpression("<fgcolor=FFFFFF><bgcolor=800000>/*</bgcolor></fgcolor>", "<fgcolor=000080><u> </u></fgcolor>", "<fgcolor=FFFFFF><bgcolor=800000>/*</bgcolor></fgcolor>")**. The statement defines an expression that starts with "/*" and end ends with "*/". So, if in the control's text we have lines that contains "/*" and ends with "*/" the control will use the Start HTML expression to paint the "/*", will use the Expression HTML expression to pain the middle of the expression, and will use the End HTML expression to paint the "*/" . For instance. The "just a text /* just another text */ just another text", becomes "just a text /* just another text */ just another text"

- **.AddExpression "<fgcolor=008F00>/*</fgcolor>", "<fgcolor=007F00></fgcolor>", "<fgcolor=008F00>*/</fgcolor>", True** adds multiple lines expression to the context. Useful to define multiple lines comments like /* and */.

- **.AddExpression "<fgcolor=008000>// <u>TODO</u></fgcolor>", "<fgcolor=0090F0> </fgcolor>", ""**. Adds a // TODO expression, that uses multiple HTML tags at the beginning of the expression like follows: '// TODO **bla bla** '

- **.AddExpression "<fgcolor=008000>// <u>TODO</u></fgcolor>", "<fgcolor=0090F0> </fgcolor>", "", , exReplaceKeyword**. Adds a // TODO expression, that uses multiple HTML tags at the beginning of the expression like follows: '// TODO **bla bla** '. For instance, if the user types // Todo, the control automatically replaces it with // TODO

The Start parameter of the AddExpression method defines the name of the expression that should be used in order to delete an expression. Use DeleteExpression method to delete an expression. For instance, if the Start parameter is "START" the name of the expression is "START" and "START" should be used for accessing the expression. The name of the expression is the text of Start parameter after excluding the HTML tags.

Call the Refresh method to reflect the changes in the control. The AddExpression, AddKeyword, DeleteExpression, DeleteKeyword, ClearKeywords and ClearExpressions do not reflect automatically the changes in the control's visible area until Refresh method is called.

```
2
3  /*
4    Add an easy-to-use edit control that supports
5    built-in HTML format, syntax highlighting
6  */
7
8  // Defintition for CLine class
9  #include <string>
10 #include 'HTMLCaption.h'
11 using namespace std;
```

The following VB sample adds and a new expression to highlight the text between /* and */:

```
With Edit1
   .AddExpression "<b><fgcolor=008000>/*</fgcolor></b>", "<b><fgcolor=008000>
</fgcolor></b>", "<b><fgcolor=008000>*/</fgcolor></b>", True
End With
```

The following C++ sample adds and a new expression to highlight the text between /* and */:

```
COleVariant vtMissing; vtMissing.vt = VT_ERROR;
m_edit.AddExpression("<b><fgcolor=008000>/*</fgcolor></b>", "<b>
<fgcolor=008000> </fgcolor></b>", "<b><fgcolor=008000>*/</fgcolor></b>",
COleVariant( VARIANT_TRUE ), vtMissing );
```

The following VB.NET sample adds and a new expression to highlight the text between /* and */:

```
With AxEdit1
   .AddExpression("<b><fgcolor=008000>/*</fgcolor></b>", "<b><fgcolor=008000>
</fgcolor></b>", "<b><fgcolor=008000>*/</fgcolor></b>", True, Nothing)
End With
```

The following C# sample adds and a new expression to highlight the text between /* and */:

```
axEdit1.AddExpression("<b><fgcolor=008000>/*</fgcolor></b>", "<b>
<fgcolor=008000> </fgcolor></b>", "<b><fgcolor=008000>*/</fgcolor></b>", true,
null);
```

The following VFP sample adds and a new expression to highlight the text between /* and */:

```
With thisform.Edit1
```

```
   .AddExpression("<b><fgcolor=008000>/*</fgcolor></b>", "<b><fgcolor=008000>
</fgcolor></b>", "<b><fgcolor=008000>*/</fgcolor></b>", .t.)
EndWith
```

method Edit.AddKeyword (Keyword as String, [Tooltip as Variant], [TooltipTitle as Variant], [CaseSensitive as Variant])

Adds a keyword to a sensitive control.

Type	Description
Keyword as String	A string expression that defines the keyword being added. The Keyword parameter may include built-in HTML tags. *If the first character in the Keyword parameter is ^, it indicates that the keyword should be at the beginning of the line, else it is ignored*.
Tooltip as Variant	Optional. A string expression that indicates the keyword's description. The tooltip is displayed when cursor is over the keyword. The Tooltip supports HTML format too. You can use the QueryContext / QueryContextToolTip events to display a different tooltip for the same keyword.
TooltipTitle as Variant	Optional. A string expression that indicates the title of keyword's tooltip. You can use the QueryContext / QueryContextToolTip events to display a different tooltip for the same keyword.
CaseSensitive as Variant	Optional. The CaseSensitive parameter is of KeywordSensitiveEnum type. A long expression that indicates the type of the start and end expressions. The valid values are 0 - exCaseSensitiveKeyword, 1 - exNonCaseSensitiveKeyword and 2 - exReplaceKeyword. By default, the CaseSensitive parameter is 0 (exCaseSensitiveKeyword)

Use the AddKeyword method to add keywords to a sensitive control. The AddKeyword method has effect if the control's EditType property is exSensitive. Use the AddExpression method to add expression to your sensitive control (an expression starts with something and ends with something). Use the AddWild method to define expression that contains wild characters like: * or ?. Use the MarkColor property to mark (on the control's vertical scroll bar) the positions of the keywords in the control's text. Use the FormatNumbers property to specify the format of the numbers in the control. Use the ToolTipOnTyping property to display the keyword's tooltip while typing in the control. The AllowInlineToolTip property indicates that the current line shows inline tooltip for keywords found to the caret.

The list of supported built-in HTML tags are:

- **bold**
- *****italic*

- **<s>**~~strikeout~~</s>
- **<u>**underline</u>
- **<fgcolor=RRGGBB>**fgcolor</fgcolor>
- **<bgcolor=RRGGBB>**bgcolor</bgcolor>
- ****text displays portions of text with a different font and/or different size. For instance, the bit draws the bit text using the Tahoma font, on size 12 pt. If the name of the font is missing, and instead size is present, the current font is used with a different size. For instance, bit displays the bit text using the current font, but with a different size.

Use the DeleteKeyword method to remove a particular keyword. If the Keyword parameter contains only HTML tags and no text, the control will not add the keyword.

The Tooltip parameter supports the following built-in HTML tags:

- **bold**
- *****italic*
- **<s>**~~strikeout~~</s>
- **<u>**underline</u>
- **<fgcolor=RRGGBB>**fgcolor</fgcolor>
- **<bgcolor=RRGGBB>**bgcolor</bgcolor>
- **<solidline>** draws a solid line
- **<dotline>** draws a dotted line
- **<upline>** draws the line to the top of the text line
- **<r>** aligns the rest of the text line to the right side
- **<img>**number[:width]</img> inserts an icon inside the cell's caption. The number indicates the index of the icon being inserted.
- ****text displays portions of text with a different font and/or different size. For instance, the bit draws the bit text using the Tahoma font, on size 12 pt. If the name of the font is missing, and instead size is present, the current font is used with a different size. For instance, bit displays the bit text using the current font, but with a different size.

For instance the following sequence: AddKeyword("") wont add any new keyword. Instead AddKeyword("public") will add the "public" keyword.

Here's few samples how to define keywords:

- **AddKeyword("<fgcolor=0000FF>public</fgcolor>").** If user types 'public' in the control, it displays: '**public**'
- **AddKeyword("<fgcolor=FF0000>Dim</fgcolor>").** If user types 'Dim' in the control, it displays 'Dim'
- **AddKeyword("<bgcolor=000000><fgcolor=FFFFFF>word</fgcolor></bgcolor>").**

If user types 'word' in the control, it displays'word'

- **AddKeyword "<u>c</u>lass", "The definition for 'class' keyword contains multiple HTML tags.
<upline><dotline><r><u>c</u>lass keyword".** The syntax defines a keyword using multiple HTML tags. If user types 'class' the control displays '**class**'.

If you are adding a temporary keyword, the Refresh method should be called, to let control reflects the changes in the visible area.

The following VB sample adds a case sensitive keyword (the exCaseSensitiveKeyword is not required, because the AddKeyword adds by default a case sensitive keyword) :

```
With Edit1
    .AddKeyword "<fgcolor=0000FF><b>class</b></fgcolor>", "Defining a case sensitive keyword.", "Case Sensitive", exCaseSensitiveKeyword
End With
```

In this case, if you are tying 'class', 'ClaSs', 'clasS' and so on, it will be highlighted as '**class**', 'ClaSs', 'clasS', and so on. The exCaseSensitiveKeyword constant is 0

The following VB sample adds a non case sensitive keyword:

```
With Edit1
    .AddKeyword "<fgcolor=0000FF><b>class</b></fgcolor>", "Defining a non case sensitive keyword.", "Non Case Sensitive", exNonCaseSensitiveKeyword
End With
```

In this case, if you are tying 'class', 'ClaSs', 'clasS' and so on, it will be highlighted as '**class**', '**ClaSs**', '**clasS**', and so on. The exNonCaseSensitiveKeyword constant is 1

The following VB sample adds a non case sensitive keyword, but any occurrence of the keyword in the text will be displayed as it is defined:

```
With Edit1
    .AddKeyword "<fgcolor=0000FF><b>class</b></fgcolor>", "Defining a non case sensitive keyword.", "Non Case Sensitive", exReplaceKeyword
End With
```

In this case, if you are tying 'class', 'ClaSs', 'clasS' and so on, it will be highlighted as '**class**', '**class**', '**class**', and so on. The exReplaceKeyword constant is 2

```
9  using namespace std;
10
11 class IHostLines;
12 class CLines;
13 class CLine :    protected tstring,
14                 public IHostHTMLCaption
```

The following C++ sample adds a case sensitive keyword:

```
m_edit.AddKeyword("<fgcolor=0000FF><b>class</b></fgcolor>", COleVariant( "Defining a case sensitive keyword." ), COleVariant( "Case Sensitive") , COleVariant( (long)0 /*exCaseSensitiveKeyword*/ ) );
```

The following VB.NET sample adds a case sensitive keyword:

```
With AxEdit1
   .AddKeyword("<fgcolor=0000FF><b>class</b></fgcolor>", "Defining a case sensitive keyword.", "Case Sensitive", EXEDITLib.KeywordSensitiveEnum.exCaseSensitiveKeyword)
End With
```

The following C# sample adds a case sensitive keyword:

```
axEdit1.AddKeyword("<fgcolor=0000FF><b>class</b></fgcolor>", "Defining a case sensitive keyword.", "Case Sensitive", EXEDITLib.KeywordSensitiveEnum.exCaseSensitiveKeyword);
```

The following VFP sample adds a case sensitive keyword:

```
With thisform.Edit1
      .AddKeyword("<fgcolor=0000FF><b>class</b></fgcolor>", "Defining a case sensitive keyword.", "Case Sensitive", 0) && exCaseSensitiveKeyword
endwith
```

method Edit.AddWild (Expression as String)

Adds a wild characters expression to a sensitive control.

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

The AddWild method adds an expression that may contain wild characters like '*' or '?'. Use the FormatNumbers property to specify the format of the numbers in the control. Use the AddKeyword method to add new keywords to the control. Use the AddExpression method to add new expressions to the control.

The following VB sample shows in bold all till : character, shows in gray till ; and let the rest unchanged:

```
With Edit1
    .AddWild "<b>*</b>:*"
    .AddWild "<b>*</b>;<fgcolor=808080>*</fgcolor>:*"
End With
```

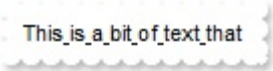
and shows as:

instead of:

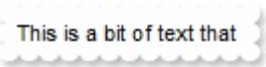
For instance the following VB sample underlines all ' ' (space character) in the text:

```
With Edit1
    .AddWild ("<u> </u>")
End With
```

and shows as:



instead of:

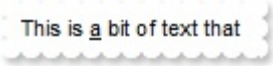


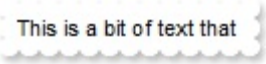
Note that if you are adding a keyword like:

```
With Edit1
    .AddKeyword ("<u>a</u>")
End With
```

only 'a' characters that is a separated word will be underlined. The wild characters expression may contain '*' that means any string of characters and '?' that means a single character.

and it shows as:

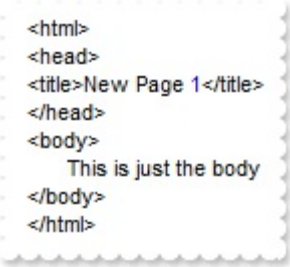


instead of:


For instance the following sample shows in bold all expressions that starts with '<' and ends with the '>' and it has at least once character inside.

```
With Edit1
    .AddWild ("<fgcolor=0000FF><?*></fgcolor>")
End With
```

like shown in the following picture:

instead of:


Note that if you are adding an expression that starts with '<' and ends with '>' the expression will be highlighted even there is no character between them:

```
With Edit1
    .AddExpression "<b><</b>", "<b> </b>", "<b>></b>"
End With
```

If you want to highlight only the part between '<' and '>' then you should use a sample like follows:

```
With Edit1
    .AddExpression "<", "<b> </b>", ">"
End With
```

The following C++ sample underlines all ' ' (space character) in the text:

```
m_edit.AddWild("<u> </u>");
```

The following VB.NET sample underlines all ' ' (space character) in the text:

```
With AxEdit1
   .AddWild("<u> </u>")
End With
```

The following C# sample underlines all ' ' (space character) in the text:

```
axEdit1.AddWild("<u> </u>");
```

The following VFP sample underlines all ' ' (space character) in the text:

```
With thisform.Edit1
   .AddWild("<u> </u>")
endwith
```

property Edit.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 meu 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 edit control operation).
- Redo (Redoes the next action in the control's redo queue).
- Cut (Delete (cuts) the current selection (if any) in the edit control and copies the deleted text to the clipboard).
- Copy (Copies the current selection (if any) in the edit control to the Clipboard).
- Paste (Inserts the data from the Clipboard into the edit 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.

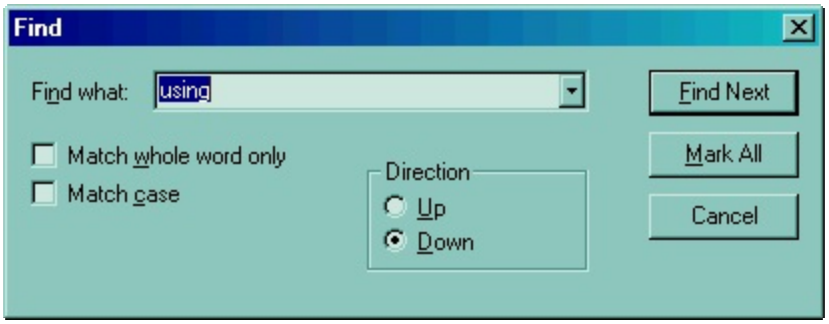
property Edit.AllowFind as Boolean

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 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 wMsg As Long, ByVal wParam As Long, lParam As Any) As Long
Private Const WM_COMMAND = &H111

Private Sub Command1_Click()
   Edit1.SetFocus
   SendMessage Edit1.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_edit.SetFocus();
   m_edit.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 wMsg 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 AxEdit1
      .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 wMsg, int wParam, int lParam);

private void button1_Click(object sender, EventArgs e)
{
   axEdit1.Focus();
   SendMessage(axEdit1.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.Edit1.Object
   SendMessage( .hWnd, 273, BITLSHIFT(57636, 16), 0)
endwith
```

property Edit.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:

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 wMsg As Long, ByVal wParam As Long, lParam As Any) As Long
Private Const WM_COMMAND = &H111

Private Sub Command1_Click()
   Edit1.SetFocus
   SendMessage Edit1.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_edit.SetFocus();
   m_edit.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 wMsg 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 AxEdit1
      .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 wMsg, int wParam, int lParam);

private void button1_Click(object sender, EventArgs e)
{
   axEdit1.Focus();
   SendMessage(axEdit1.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:

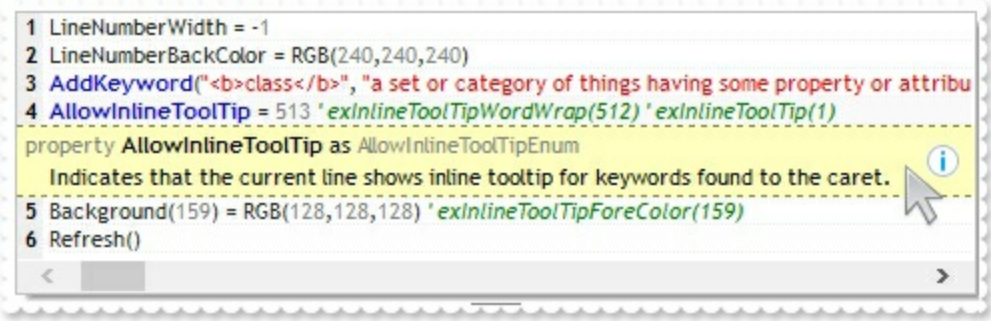
```
DECLARE INTEGER SendMessage IN user32;
   INTEGER hWnd,;
   INTEGER Msg,;
   INTEGER wParam,;
   INTEGER lParam

with thisform.Edit1.Object
   SendMessage( .hWnd, 273, BITLSHIFT(59906, 16), 0)
endwith
```

property Edit.AllowInlineToolTip as AllowInlineToolTipEnum

Indicates that the current line shows inline tooltip for keywords found to the caret.

Type	Description
AllowInlineToolTipEnum	An AllowInlineToolTipEnum expression that specifies whether the control's inline-tooltip feature is enabled or disabled. For instance, exInlineToolTip + exInlineToolTipOnChange indicates that the inline-tooltip is shown when the user changes the current line.

By default, the AllowInlineToolTip property is exDisableInlineToolTip, which indicates that the control supports no inline-tooltip feature. The Inline-ToolTip feature allows displaying inline tooltips of the keywords found at the current line as shown in the picture above. Use the AllowInlineToolTip property to enable or disable the control's inline-tooltip feature. For instance, you can use the Inline-ToolTip feature to display inside the control at current line, any information/html text you may need. The AddKeyword method adds a keyword to a sensitive control. The ToolTip/ToolTipTile parameters of the AddKeyword method defines the inline-tooltip for the specified keyword. The TempInlineToolTip property specifies a temporarily inline tooltip to be shown at the current line. The ExcludeInlineToolTip property specifies a list of keywords, separated by , (comma character), that specifies the keywords to be excluded from the inline tooltip support. You can use the QueryContext / QueryContextToolTip events to display a different tooltip for the same keyword.



The Background(exInlineToolTipForeColor) / Background(exInlineToolTipBackColor) property define the foreground / background color / visual appearance of the inline-tooltip panel.

How do I enable the inline tooltip support?

VBA (MS Access, Excell...)

```
With Edit1
   .LineNumberWidth = -1
   .LineNumberBackColor = RGB(240,240,240)
```

```
    .AddKeyword "<b>class</b>","a set or category of things having some property
or attribute in common and differentiated from others by kind, type, or qualit" & _
"y."
    .AllowInlineToolTip = 513 ' AllowInlineToolTipEnum.exInlineToolTipWordWrap Or
AllowInlineToolTipEnum.exInlineToolTip
    .Background(159) = RGB(128,128,128)
    .Refresh
End With
```

VB6

```
With Edit1
    .LineNumberWidth = -1
    .LineNumberBackColor = RGB(240,240,240)
    .AddKeyword "<b>class</b>","a set or category of things having some property
or attribute in common and differentiated from others by kind, type, or qualit" & _
"y."
    .AllowInlineToolTip = AllowInlineToolTipEnum.exInlineToolTipWordWrap Or
AllowInlineToolTipEnum.exInlineToolTip
    .Background(exInlineToolTipForeColor) = RGB(128,128,128)
    .Refresh
End With
```

VB.NET

```
With Exedit1
    .LineNumberWidth = -1
    .LineNumberBackColor = Color.FromArgb(240,240,240)
    .AddKeyword("<b>class</b>","a set or category of things having some property
or attribute in common and differentiated from others by kind, type, or qualit" & _
"y.")
    .AllowInlineToolTip =
exontrol.EXEDITLib.AllowInlineToolTipEnum.exInlineToolTipWordWrap Or
exontrol.EXEDITLib.AllowInlineToolTipEnum.exInlineToolTip

.set_Background(exontrol.EXEDITLib.BackgroundPartEnum.exInlineToolTipForeColor,Col

    .Refresh()
```

```
End With
```

VB.NET for /COM

```
With AxEdit1
   .LineNumberWidth = -1
   .LineNumberBackColor = RGB(240,240,240)
   .AddKeyword("<b>class</b>","a set or category of things having some property
or attribute in common and differentiated from others by kind, type, or qualit" & _
"y.")
   .AllowInlineToolTip =
EXEDITLib.AllowInlineToolTipEnum.exInlineToolTipWordWrap Or
EXEDITLib.AllowInlineToolTipEnum.exInlineToolTip

.set_Background(EXEDITLib.BackgroundPartEnum.exInlineToolTipForeColor,8421504)
   .Refresh()
End With
```

C++

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

   #import <ExEdit.dll>
   using namespace EXEDITLib;
*/
EXEDITLib::IEditPtr spEdit1 = GetDlgItem(IDC_EDIT1)->GetControlUnknown();
spEdit1->PutLineNumberWidth(-1);
spEdit1->PutLineNumberBackColor(RGB(240,240,240));
spEdit1->AddKeyword(L"<b>class</b>",_bstr_t("a set or category of things having
some property or attribute in common and differentiated from others by kind, type,
or qualit") +
"y.",vtMissing,vtMissing);
spEdit1-
>PutAllowInlineToolTip(EXEDITLib::AllowInlineToolTipEnum(EXEDITLib::exInlineToolTi
| EXEDITLib::exInlineToolTip));
spEdit1->PutBackground(EXEDITLib::exInlineToolTipForeColor,RGB(128,128,128));
```

```
spEdit1->Refresh();
```

C++ Builder

```
Edit1->LineNumberWidth = -1;
Edit1->LineNumberBackColor = RGB(240,240,240);
Edit1->AddKeyword(L"<b>class</b>",TVariant(String("a set or category of things having some property or attribute in common and differentiated from others by kind, type, or qualit") +
"y."),TNoParam(),TNoParam());
Edit1->AllowInlineToolTip =
Exeditlib_tlb::AllowInlineToolTipEnum::exInlineToolTipWordWrap |
Exeditlib_tlb::AllowInlineToolTipEnum::exInlineToolTip;
Edit1->Background[Exeditlib_tlb::BackgroundPartEnum::exInlineToolTipForeColor] =
RGB(128,128,128);
Edit1->Refresh();
```

C#

```
exedit1.LineNumberWidth = -1;
exedit1.LineNumberBackColor = Color.FromArgb(240,240,240);
exedit1.AddKeyword("<b>class</b>","a set or category of things having some property or attribute in common and differentiated from others by kind, type, or qualit" +
"y.",null,null);
exedit1.AllowInlineToolTip =
exontrol.EXEDITLib.AllowInlineToolTipEnum.exInlineToolTipWordWrap |
exontrol.EXEDITLib.AllowInlineToolTipEnum.exInlineToolTip;
exedit1.set_Background(exontrol.EXEDITLib.BackgroundPartEnum.exInlineToolTipForeC

exedit1.Refresh();
```

JScript/JavaScript

```
<BODY onload="Init()">
```

```
<OBJECT CLASSID="clsid:39136531-DD0F-4281-B445-E36FC2CDDBC5" id="Edit1">
</OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
   Edit1.LineNumberWidth = -1;
   Edit1.LineNumberBackColor = 15790320;
   Edit1.AddKeyword("<b>class</b>","a set or category of things having some
property or attribute in common and differentiated from others by kind, type, or
qualit" +
   "y.",null,null);
   Edit1.AllowInlineToolTip = 513;
   Edit1.Background(159) = 8421504;
   Edit1.Refresh();
}
</SCRIPT>
</BODY>
```

VBScript

```
<BODY onload="Init()">
<OBJECT CLASSID="clsid:39136531-DD0F-4281-B445-E36FC2CDDBC5" id="Edit1">
</OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
   With Edit1
      .LineNumberWidth = -1
      .LineNumberBackColor = RGB(240,240,240)
      .AddKeyword "<b>class</b>","a set or category of things having some
property or attribute in common and differentiated from others by kind, type, or
qualit" & _
   "y."
      .AllowInlineToolTip = 513 ' AllowInlineToolTipEnum.exInlineToolTipWordWrap
Or AllowInlineToolTipEnum.exInlineToolTip
```

```
      .Background(159) = RGB(128,128,128)
      .Refresh
   End With
End Function
</SCRIPT>
</BODY>
```

C# for /COM

```
axEdit1.LineNumberWidth = -1;
axEdit1.LineNumberBackColor = Color.FromArgb(240,240,240);
axEdit1.AddKeyword("<b>class</b>","a set or category of things having some property or attribute in common and differentiated from others by kind, type, or qualit" +
"y.",null,null);
axEdit1.AllowInlineToolTip =
EXEDITLib.AllowInlineToolTipEnum.exInlineToolTipWordWrap |
EXEDITLib.AllowInlineToolTipEnum.exInlineToolTip;
axEdit1.set_Background(EXEDITLib.BackgroundPartEnum.exInlineToolTipForeColor,
(uint)ColorTranslator.ToWin32(Color.FromArgb(128,128,128)));
axEdit1.Refresh();
```

X++ (Dynamics Ax 2009)

```
public void init()
{
   str var_s;
   ;

   super();

   exedit1.LineNumberWidth(-1);
   exedit1.LineNumberBackColor(WinApi::RGB2int(240,240,240));
   var_s = "a set or category of things having some property or attribute in common and differentiated from others by kind, type, or quality";
   var_s = var_s + ".";
```

```
    exedit1.AddKeyword("<b>class</b>",COMVariant::createFromStr(var_s));
    exedit1.AllowInlineToolTip(513/*exInlineToolTipWordWrap | exInlineToolTip*/);

exedit1.Background(159/*exInlineToolTipForeColor*/,WinApi::RGB2int(128,128,128));
    exedit1.Refresh();
}
```

Delphi 8 (.NET only)

```
with AxEdit1 do
begin
    LineNumberWidth := -1;
    LineNumberBackColor := Color.FromArgb(240,240,240);
    AddKeyword('<b>class</b>','a set or category of things having some property
or attribute in common and differentiated from others by kind, type, or quality' +
    '.',Nil,Nil);
    AllowInlineToolTip :=
Integer(EXEDITLib.AllowInlineToolTipEnum.exInlineToolTipWordWrap) Or
Integer(EXEDITLib.AllowInlineToolTipEnum.exInlineToolTip);

set_Background(EXEDITLib.BackgroundPartEnum.exInlineToolTipForeColor,$808080);
    Refresh();
end
```

Delphi (standard)

```
with Edit1 do
begin
    LineNumberWidth := -1;
    LineNumberBackColor := RGB(240,240,240);
    AddKeyword('<b>class</b>','a set or category of things having some property
or attribute in common and differentiated from others by kind, type, or quality' +
    '.',Null,Null);
    AllowInlineToolTip := Integer(EXEDITLib_TLB.exInlineToolTipWordWrap) Or
Integer(EXEDITLib_TLB.exInlineToolTip);
    Background[EXEDITLib_TLB.exInlineToolTipForeColor] := $808080;
    Refresh();
end
```

VFP

```
with thisform.Edit1
   .LineNumberWidth = -1
   .LineNumberBackColor = RGB(240,240,240)
   var_s = "a set or category of things having some property or attribute in common and differentiated from others by kind, type, or quality"
   var_s = var_s + "."
   .AddKeyword("<b>class</b>",var_s)
   .AllowInlineToolTip = 513 && AllowInlineToolTipEnum.exInlineToolTipWordWrap Or AllowInlineToolTipEnum.exInlineToolTip
   .Object.Background(159) = RGB(128,128,128)
   .Refresh
endwith
```

dBASE Plus

```
local oEdit

oEdit = form.EXEDITACTIVEXCONTROL1.nativeObject
oEdit.LineNumberWidth = -1
oEdit.LineNumberBackColor = 0xf0f0f0
oEdit.AddKeyword("<b>class</b>","a set or category of things having some property or attribute in common and differentiated from others by kind, type, or quality.")
oEdit.AllowInlineToolTip = 513 /*exInlineToolTipWordWrap | exInlineToolTip*/
oEdit.Template = [Background(159) = 8421504] // oEdit.Background(159) = 0x808080
oEdit.Refresh()
```

XBasic (Alpha Five)

```
Dim oEdit as P

oEdit = topparent:CONTROL_ACTIVEX1.activex
oEdit.LineNumberWidth = -1
oEdit.LineNumberBackColor = 15790320
```

```
oEdit.AddKeyword("<b>class</b>","a set or category of things having some property or attribute in common and differentiated from others by kind, type, or quality.")
oEdit.AllowInlineToolTip = 513 'exInlineToolTipWordWrap + exInlineToolTip
oEdit.Template = "Background(159) = 8421504" // oEdit.Background(159) = 8421504
oEdit.Refresh()
```

Visual Objects

```
oDCOCX_Exontrol1:LineNumberWidth := -1
oDCOCX_Exontrol1:LineNumberBackColor := RGB(240,240,240)
oDCOCX_Exontrol1:AddKeyword("<b>class</b>","a set or category of things having some property or attribute in common and differentiated from others by kind, type, or quality.",nil,nil)
oDCOCX_Exontrol1:AllowInlineToolTip := exInlineToolTipWordWrap | exInlineToolTip
oDCOCX_Exontrol1:[Background,exInlineToolTipForeColor] := RGB(128,128,128)
oDCOCX_Exontrol1:Refresh()
```

PowerBuilder

```
OleObject oEdit

oEdit = ole_1.Object
oEdit.LineNumberWidth = -1
oEdit.LineNumberBackColor = RGB(240,240,240)
oEdit.AddKeyword("<b>class</b>","a set or category of things having some property or attribute in common and differentiated from others by kind, type, or quality.")
oEdit.AllowInlineToolTip = 513 /*exInlineToolTipWordWrap | exInlineToolTip*/
oEdit.Background(159,RGB(128,128,128))
oEdit.Refresh()
```

Visual DataFlex

```
Procedure OnCreate
   Forward Send OnCreate
   Set ComLineNumberWidth to -1
   Set ComLineNumberBackColor to (RGB(240,240,240))
   Send ComAddKeyword "<b>class</b>" "a set or category of things having some
property or attribute in common and differentiated from others by kind, type, or
quality." Nothing Nothing
   Set ComAllowInlineToolTip to (OLEexInlineToolTipWordWrap +
OLEexInlineToolTip)
   Set ComBackground OLEexInlineToolTipForeColor to (RGB(128,128,128))
   Send ComRefresh
End_Procedure
```

XBase++

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

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

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

   oEdit := XbpActiveXControl():new( oForm:drawingArea )
   oEdit:CLSID  := "Exontrol.Edit.1" /*{39136531-DD0F-4281-B445-
E36FC2CDDBC5}*/
   oEdit:create(,, {10,60},{610,370} )

      oEdit:LineNumberWidth := -1
      oEdit:SetProperty("LineNumberBackColor",AutomationTranslateColor(
GraMakeRGBColor  ( { 240,240,240 } )  , .F. ))
      oEdit:AddKeyword("<b>class</b>","a set or category of things having some
```

```
property or attribute in common and differentiated from others by kind, type, or
quality.")
      oEdit:AllowInlineToolTip := 513/*exInlineToolTipWordWrap+exInlineToolTip*/

oEdit:SetProperty("Background",159/*exInlineToolTipForeColor*/,AutomationTranslate(
 GraMakeRGBColor  ( { 128,128,128 } )  , .F. ))
      oEdit:Refresh()

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

property Edit.AllowMark as Boolean

Specifies a value that indicates whether the positions in text of expressions are marked on the vertical scroll bar.

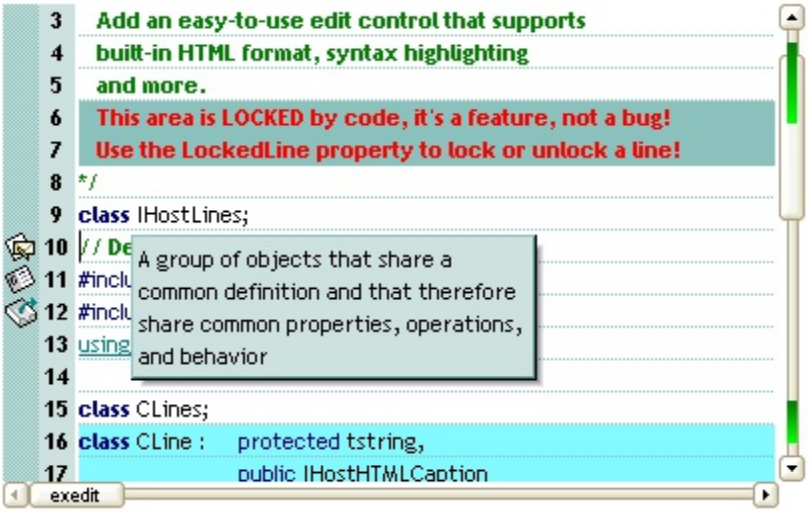
Type	Description
Boolean	A boolean expression that indicates whether the control displays on the control's vertical scroll bar a line for each keyword/expression defined.

Use the AllowMark property to show on the control's vertical scroll bar the positions of expressions in the text. Use the MarkColor property to specify a marking color for an expression. Use the AddKeyword to add new keywords to the control's context. Use the AddExpression to add new expressions to the control. By default, the AllowMark property is False. If the AllowMark property is True, the control could load the text slowly, because each line requires to be parsed in order to find the expression to be highlighted on the control's vertical scroll bar, so we would suggest to use this option only before you check the time to load the file using this option. Use the MarkContinueBlocks property to mark multiple lines block. Use the Refresh method to refresh the control's visible area, when adding expressions on the fly.

The following VB sample defines the // TODO expressions, and marks on the control's vertical scroll bar each position of the // TODO expression in the control's text:

```
With Edit1
   .AllowMark = True
   .AddExpression "<fgcolor=000080>// TODO</fgcolor>", "<b><fgcolor=000080>
</fgcolor></b>", ""
   .MarkColor("// TODO") = RGB(0, 0, &H80)
   .Refresh
End With
```

As you can see from the following screen shot, there are lines (on the control's vertical scroll bar) that marks the position for each // TODO expression in the text.



The following C++ sample marks the position of the // TODO expressions on the control's vertical scroll bar:

```
COleVariant vtMissing; vtMissing.vt = VT_ERROR;
m_edit.SetAllowMark( TRUE );
m_edit.AddExpression("<fgcolor=000080>// TODO</fgcolor>", "<fgcolor=000080>
</fgcolor>", "", vtMissing, vtMissing );
m_edit.SetMarkColor("// TODO", RGB(0, 0, 0x80) );
m_edit.Refresh();
```

The following VB.NET sample marks the position of the // TODO expressions on the control's vertical scroll bar:

```
With AxEdit1
   .AllowMark = True
   .LineNumberWidth = -1
   .AddExpression("<fgcolor=000080>// TODO</fgcolor>", "<b><fgcolor=000080>
</fgcolor></b>", "")
   .set_MarkColor("// TODO", ToUInt32(Color.FromArgb(0, 0, &H80)))
   .CtlRefresh()
End With
```

where the ToUInt32 function converts a Color expression to an OLE_COLOR expressions:

```
Shared Function ToUInt32(ByVal c As Color) As UInt32
   Dim i As Long
   i = c.R
   i = i + 256 * c.G
   i = i + 256 * 256 * c.B
```

```
    ToUInt32 = Convert.ToUInt32(i)
End Function
```

The following C# sample marks the position of the // TODO expressions on the control's vertical scroll bar:

```
axEdit1.AllowMark = true;
axEdit1.AddExpression("<fgcolor=000080>// TODO</fgcolor>", "<b>
<fgcolor=000080> </fgcolor></b>", "");
axEdit1.set_MarkColor("// TODO", ToUInt32(Color.FromArgb(0, 0, 0x80)));
axEdit1.CtlRefresh();
```

where the ToUInt32 function converts a Color expression to an OLE_COLOR expressions:

```
private UInt32 ToUInt32(Color c)
{
    long i;
    i = c.R;
    i = i + 256 * c.G;
    i = i + 256 * 256 * c.B;
    return Convert.ToUInt32(i);
}
```

The following VFP sample marks the position of the // TODO expressions on the control's vertical scroll bar:

```
With thisform.Edit1.Object
    .AllowMark = .t.
    .AddExpression("<fgcolor=000080>// TODO</fgcolor>", "<b><fgcolor=000080>
</fgcolor></b>", "")
    .MarkColor("// TODO") = RGB(0, 0, 128)
    .Refresh
EndWith
```

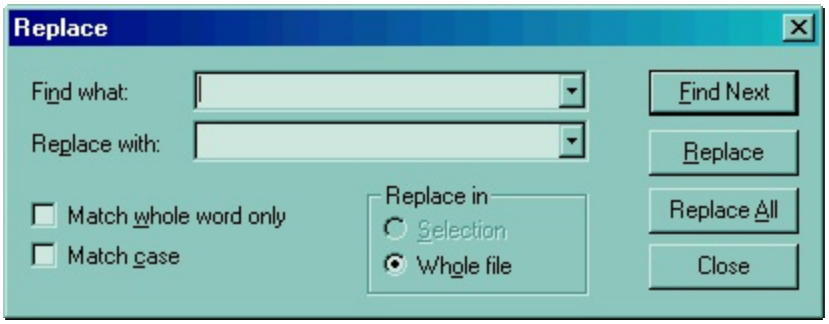
property Edit.AllowReplace as Boolean

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 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 Replace dialog when user clicks a

button:

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

Private Sub Command1_Click()
   Edit1.SetFocus
   SendMessage Edit1.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_edit.SetFocus();
   m_edit.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 wMsg 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 AxEdit1
      .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 wMsg, int wParam, int lParam);

private void button1_Click(object sender, EventArgs e)
{
   axEdit1.Focus();
   SendMessage(axEdit1.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:

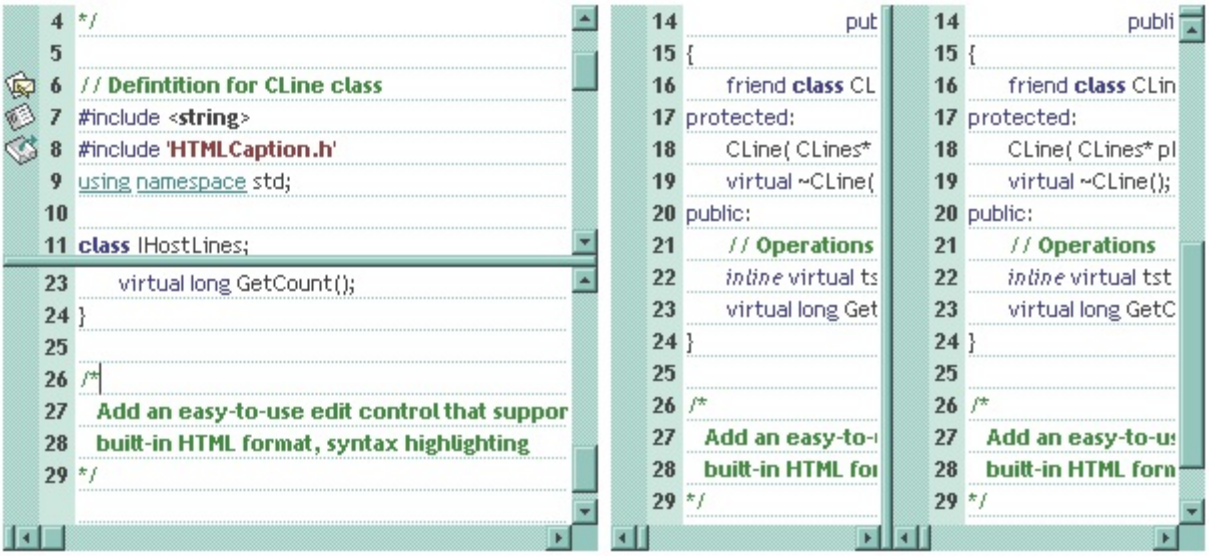
```
DECLARE INTEGER SendMessage IN user32;
   INTEGER hWnd,;
   INTEGER Msg,;
   INTEGER wParam,;
   INTEGER lParam

with thisform.Edit1.Object
   SendMessage( .hWnd, 273, BITLSHIFT(57641, 16), 0)
endwith
```

property Edit.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 edit 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 Edit.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 wMsg As Long, ByVal wParam As Long, lParam As Any) As Long
Private Const WM_COMMAND = &H111

Private Sub Command1_Click()
   SendMessage Edit1.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_edit.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 wMsg 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 AxEdit1
      .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 wMsg, int wParam, int lParam);

private void button1_Click(object sender, EventArgs e)
{
   axEdit1.Focus();
   SendMessage(axEdit1.hWnd, 0x111, 0xE12B << 16, 0);
```

```
}
```

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

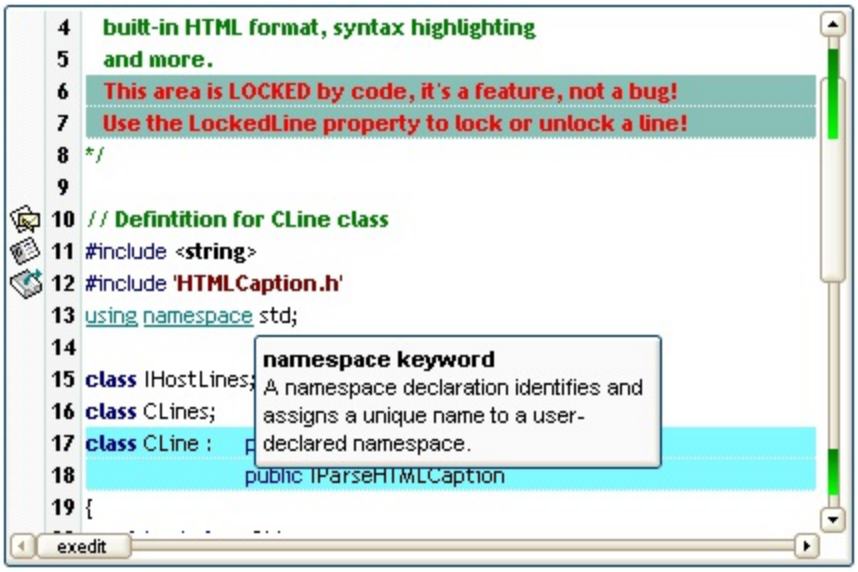
```
DECLARE INTEGER SendMessage IN user32;
   INTEGER hWnd,;
   INTEGER Msg,;
   INTEGER wParam,;
   INTEGER lParam

with thisform.Edit1.Object
   SendMessage( .hWnd, 273, BITLSHIFT(57643, 16), 0)
endwith
```

property Edit.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 Edit1
    .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 AxEdit1
  .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:

```
axEdit1.VisualAppearance.Add(0x16, "c:\\temp\\frame.ebn");
axEdit1.Appearance = (EXEDITLib.AppearanceEnum)0x16000000;
axEdit1.BackColor = Color.FromArgb(250, 250, 250);
```

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

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

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

```
with thisform.Edit1
    .VisualAppearance.Add(0x16, "c:\temp\frame.ebn")
    .Appearance = 0x16000000
    .BackColor = RGB(250, 250, 250)
endwith
```

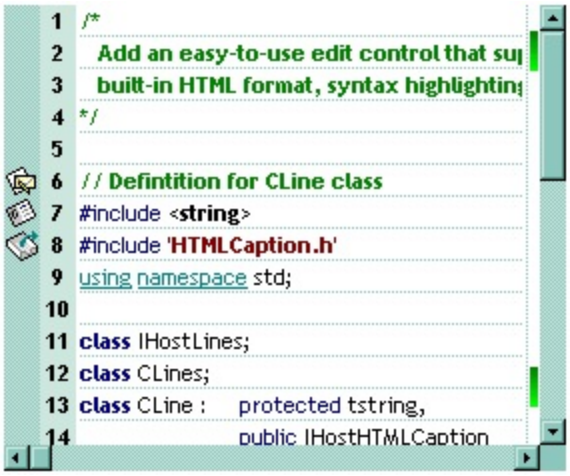
property Edit.ApplyColors as Boolean

Specifies whether the control applies the HTML colors.

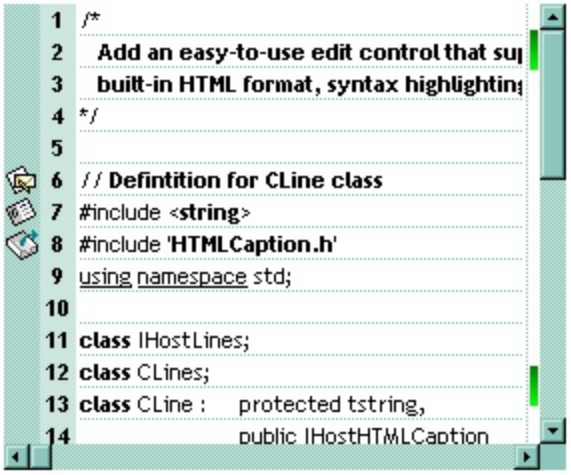
Type	Description
Boolean	A boolean expression that indicates whether the control uses the HTML tags like <fgcolor> and <bgcolor> to paint the control's content.

The ApplyColors property specifies whether HTML tags like <fgcolor> and <bgcolor> are used to paint the control's content. By default, the AllowColors property is True. Use the ApplyColors property to disable displaying colors into a sensitive control. The AllowColors property has effect if the control's EditType property is exSensitive or exHTML. The AllowColors property has no effect if the control's EditType property is exStandard.

The following screen shot views the control's content when ApplyColors property is True:



The following screen shot displays the control's content when ApplyColors property is False:



method Edit.AttachTemplate (Template as Variant)

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

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

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

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

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

This script is equivalent with the following VB code:

```
Private Sub Edit1_Click()
   With CreateObject("internetexplorer.application")
      .Visible = True
      .Navigate ("https://www.exontrol.com")
   End With
End Sub
```

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

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

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

where:

<identifier> indicates an identifier of the variable, property, method or event, and should start with a letter.
<type> indicates the type the CreateObject function creates, as a progID for /COM version or the assembly-qualified name of the type to create for /NET or /WPF version
<text> any string of characters

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

The advantage of the AttachTemplate relative to Template / ExecuteTemplate is that the AttachTemplate can add handlers to the control events.

property Edit.AutoIndent as Boolean

Specifies whether the control indents the line when ENTER key is pressed.

Type	Description
Boolean	A boolean expression that indicates whether the auto indent feature is enabled or disabled.

Use the AutoIndent to disable auto indent feature. By default, the AutoIndent property is True. If the AutoIndent property is True, and the user presses the Enter key, the new line added will be automatically indented.

property Edit.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 BackColorAlternate property to specify the background color for alternate lines. Use the BackColorLine property to specify the line's background color. Use the ForeColorLine property to specify the line's foreground color. Use the Picture property to assign a picture on the control's background. If the control is read-only (Locked property is True), the BackColorLockedLine property indicates the control's background color, if it is not zero. If the control is read-only (Locked property is True), the ForeColorLockedLine property indicates the control's foreground color, if it is not zero. Use the ApplyColors property to hide the colors inside the control.

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

```
With Edit1
   .BackColor = ColorConstants.vbWhite
End With
```

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

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

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

```
With AxEdit1
   .BackColor = Color.White
End With
```

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

```
axEdit1.BackColor = Color.White;
```

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

```
With thisform.Edit1.Object
   .BackColor = RGB(255,255,255)
```

```
endwith
```

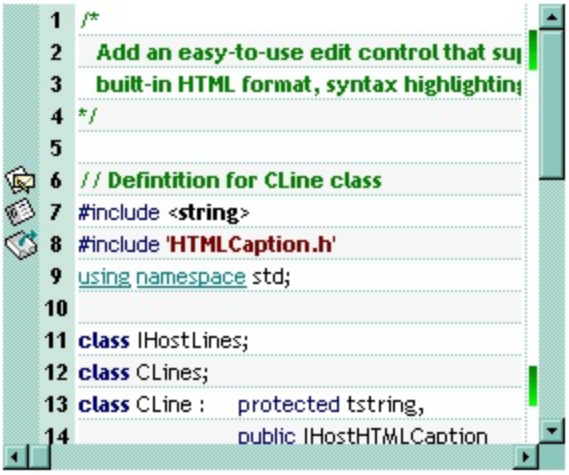
property Edit.BackColorAlternate as Color

Returns or sets the background color used to display alternate lines in the control.

Type	Description
Color	A color expression that indicates the alternate color

Use the BackColorAlternate property to display alternate lines in the control. Use the BackColor property to specify the control's background color. By default, the BackColorAlternate is 0. If the BackColorAlternate is 0 it has no visual effect.

The following screen shot displays the control when BackColorAlternats property is RGB(&HF0, &HF0, &HF0):



property Edit.BackColorCaretLine as Color

Returns or sets the background color used to display the cursor line.

Type	Description
Color	A color expression that indicates the background color for the control's header. 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 current line displays a different background color if the BackColorCaretLine property is not 0. By default, the BackColorCaretLine property is 0. If the BackColorCaretLine property is 0 it has no visual effect. Use the BackColorALternate to display alternate lines in the control. Use the BackColor property to specify the control's background color. Use the CaretLine property to specify the line where the caret is.

property Edit.BackColorLine(Index as Long) as Color

Specifies the line's background color.

Type	Description
Index as Long	A long expression that indicates the index of line being colored.
Color	A color expression that indicates the line's background color.

Use the BackColorLine property to change the line's background color. Use the ClearBackColorLine method to clear the line's background color. Use the ForeColorLine property to specify the line's foreground color. Use the BackColor property to specify the control's background color. Use the ForeColor property to specify the control's foreground color. The BackColorCaretLine property returns or sets the background color used to display the cursor line.

property Edit.BackColorLockedLine as Color

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

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

Use the BackColorLockedLine property to specify the background color for locked lines. Use the LockedLine property to lock a specific line. Use the ForeColorLockedLine property to specify the foreground color for locked lines. Use the Locked property to make read-only the entire control.

property Edit.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 "" to vertical split bar:

```
With Edit1
   With .VisualAppearance
      .Add &H1, App.Path + "\split.ebn"
   End With
   .Background(exVSpliterApp) = &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\\ExEdit.Help\\split.ebn")) );
m_tree.SetBackground( 1 /*exVSpliterApp*/, 0x1000000 );
```

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

```
With AxEdit1
   With .VisualAppearance
      .Add(&H1, "D:\Temp\ExEdit.Help\split.ebn")
   End With
   .set_Background(EXEDITLib.BackgroundPartEnum.exVSpliterApp, &H1000000)
End With
```

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

```
axEdit1.VisualAppearance.Add(0x1, "D:\\Temp\\ExEdit.Help\\split.ebn");
axEdit1.set_Background(EXEDITLib.BackgroundPartEnum.exVSpliterApp, 0x1000000);
```

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

```
With thisform.Edit1
   With .VisualAppearance
      .Add(1, "D:\Temp\ExEdit.Help\split.ebn")
   EndWith
   .Object.Background(1) = 16777216
EndWith
```

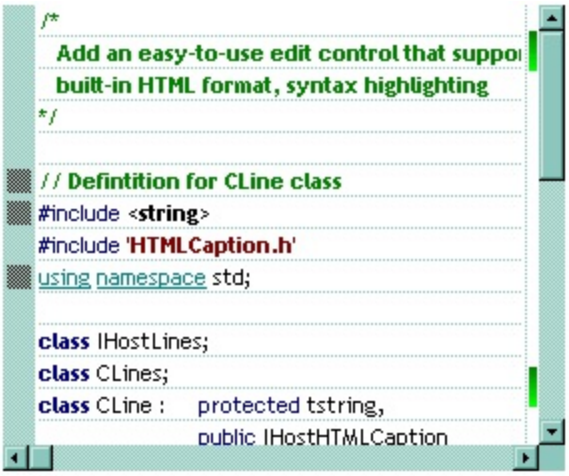
The 16777216 value is the 0x1000000 value in hexadecimal.

property Edit.Bookmark(Index as Long) as Boolean

Adds or removes a bookmark.

Type	Description
Index as Long	A long expression that indicates the index of line being bookmarked. The Index parameter is 1 based. The first line in the control is 1.
Boolean	A boolean expression that indicates the state of the line's bookmark.

Use the Bookmark property to define programmatically bookmarks for a specified line. The BookmarkChange event occurs when user changes the bookmarks for lines. The CTRL + F2 key places or erases the current line's bookmark. The F2 key moves the current line to the next bookmarked line. The Shift + F2 key moves the current line to the previous bookmarked line. Use the BookmarkWidth property to show the control's bookmarks header bar. Use the ClearBookmarks method to clear all bookmarks in the control. Use the BookmarksList property to retrieve the list of line that has a bookmark. Use the BookMarkBackColor property to specify the bookmark's header background color. Use the NextBookmark method to move the cursor to the next bookmark. Use the PrevBookmark method to move the cursor to the prior bookmark.



The following VB sample bookmarks or un-bookmarks the current line:

```
With Edit1
    .Bookmark(.CaretLine) = Not .Bookmark(.CaretLine)
End With
```

The following C++ sample bookmarks or un-bookmarks the current line:

```
m_edit.SetBookmark( m_edit.GetCaretLine(), !m_edit.GetBookmark(m_edit.GetCaretLine() )
```

```
);
```

The following VB.NET sample bookmarks or un-bookmarks the current line:

```
With AxEdit1
    .set_Bookmark(.CaretLine, Not .get_Bookmark(.CaretLine))
End With
```

The following C# sample bookmarks or un-bookmarks the current line:

```
axEdit1.set_Bookmark( axEdit1.CaretLine, !axEdit1.get_Bookmark(axEdit1.CaretLine ) );
```

The following VFP sample bookmarks or un-bookmarks the current line:

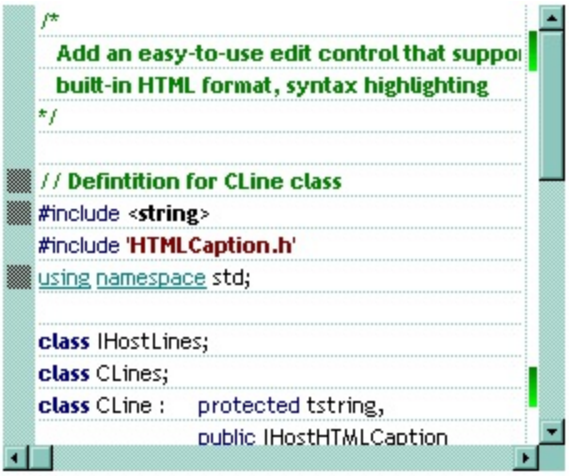
```
With thisform.Edit1.Object
    .Bookmark(.CaretLine) = !.Bookmark(.CaretLine)
endwith
```

property Edit.BookMarkBackColor as Color

Specifies the bookmark bar background's color.

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

Use the BookMarkBackColor and BookMarkBackColor2 properties to customize colors of the control's bookmark bar. Use the Bookmark property to bookmark a line. Use the BookmarksList property to retrieve the list of line that has a bookmark. Use the ClearBookmarks method to clear all bookmarks in the control.



property Edit.BookMarkBackColor2 as Color

Specifies the bookmark bar background's color.

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

Use the BookMarkBackColor and BookMarkBackColor2 properties to customize colors of the control's bookmark bar. The BookMarkBackColor2 property determines the color of the control's grid lines, when the DrawGridLines property is True. Use the Bookmark property to bookmark a line.

property Edit.BookmarkBorderColor as Color

Specifies the color for bookmark's border.

Type	Description
Color	A color expression that indicates the color for bookmark's border.

Use the BookmarkWidth property to show the control's bookmarks header bar. Use the BookMarkBackColor and BookMarkBackColor2 properties to specify the background color for the control's bookmark border.

property Edit.BookmarkImage as Long

Specifies the index of image being displayed when the line has a bookmark.

Type	Description
Long	A long expression that indicates the index of icon being displayed in the control's bookmarks header bar when a line is bookmarked.

Use the BookmarkImage property to specify the icon being displayed when a line is bookmarked. Use the BookmarkWidth property to display the control's bookmarks header bar. Use the Images method to assign a list of icons to the control. Use the ReplaceIcon method to add, remove or replace an icon in the control's images list. By default, the BookmarkImage property is -1. If the BookmarkImage property is -1, the control displays the default bookmark. Use the BookmarkImageLine property to display an icon on the bookmarks header bar.

property Edit.BookmarkImageLine(Index as Long) as Long

Retrieves or sets a value that indicates the index of image being displayed on the bookmark header.

Type	Description
Index as Long	A long expression that indicates the index of line where the icon is displayed.
Long	A long expression that indicates the index of line where the icon is displayed.

Use the BookmarkImageLine property to assign an icon to a line. The line's icon is displayed in the control's bookmarks header bar. Use the BookmarkWidth property to display the control's bookmarks header bar. Use the BookmarkImage property to change the default bookmark picture. By default, the BookmarkImageLine property is -1. If the BookmarkImageLine property is -1, no icon is displayed. Use the Images method to assign a list of icons to the control, at run time. Use the ReplaceIcon method to add, remove or change an icon in the control's images list. Use the Bookmark property to bookmark a line.

property Edit.BookmarksList as Variant

Specifies the list of bookmarks..

Type	Description
Variant	An array of long values that specifies the list of bookmarks in the list.

Use the BookmarkList property to access all bookmarks in the control. Use the Bookmark property to bookmark a specified line. Use the BookmarkWidth property to display the control's bookmark border. The BookmarkChange event is fired when user adds or removes bookmarks to the control. Use the BookmarkImage property to change the icon being displayed as a bookmark. Use the NextBookmark method to move the cursor to the next bookmark. Use the PrevBookmark method to move the cursor to the prior bookmark.

The following VB sample bookmarks the first, third and fifth line in the control:

```
With Edit1

    .BookmarkWidth = -1
    .LineNumberWidth = -1

    Dim list(3) As Long
    list(0) = 1
    list(1) = 3
    list(2) = 5
    .BookmarksList = list

End With
```

The following VB sample displays the lines with bookmarks:

```
Private Sub Edit1_BookmarkChange(ByVal Index As Long)
  With Edit1
    Dim i As Long, b As Variant
    b = .BookmarksList
    For i = LBound(b) To UBound(b)
      Debug.Print .TextLine(b(i))
    Next
  End With
```

```
End Sub
```

The following C++ sample displays the lines with bookmarks:

```
void OnBookmarkChangeEdit1(long Index)
{
   COleVariant vtBookmarks = m_edit.GetBookmarksList();
   SAFEARRAY* pBookmarks = V_ARRAY( &vtBookmarks );
   long nDim = 0;
   for ( long i = pBookmarks->rgsabound[nDim].lLbound; i < (long)pBookmarks-
>rgsabound[nDim].cElements; i++ )
   {
      long nLine = 0;
      SafeArrayGetElement( pBookmarks, &i, &nLine );
      OutputDebugString( m_edit.GetTextLine( nLine ) );
   }
}
```

The following VB.NET sample displays the lines with bookmarks:

```
Private Sub AxEdit1_BookmarkChange(ByVal sender As Object, ByVal e As
AxEXEDITLib._IEditEvents_BookmarkChangeEvent) Handles AxEdit1.BookmarkChange
   With AxEdit1
      Dim i As Integer, b As Integer() = .BookmarksList
      For Each i In b
         Debug.WriteLine(.get_TextLine(i))
      Next
   End With
End Sub
```

or

```
Private Sub AxEdit1_BookmarkChange(ByVal sender As Object, ByVal e As
AxEXEDITLib._IEditEvents_BookmarkChangeEvent) Handles AxEdit1.BookmarkChange
   With AxEdit1
      Dim i As Long, b As Object
      b = .BookmarksList
      For i = LBound(b) To UBound(b)
         Debug.WriteLine(.get_TextLine(b(i)))
```

```
        Next
    End With
End Sub
```

The following C# sample displays the lines with bookmarks:

```
private void axEdit1_BookmarkChange(object sender,
AxEXEDITLib._IEditEvents_BookmarkChangeEvent e)
{
    int[] b = axEdit1.BookmarksList as int[];
    foreach (int i in b)
        System.Diagnostics.Debug.WriteLine(axEdit1.get_TextLine(i));
}
```

The following VFP sample displays the lines with bookmarks:

```
*** ActiveX Control Event ***
LPARAMETERS index

with thisform.Edit1
    local b, i
    b = .BookmarksList
    for i = 1 to alen(b)
        wait window .TextLine(b[i])
    next
endwith
```

property Edit.BookmarkWidth as Long

Specifies the bookmarks border's width.

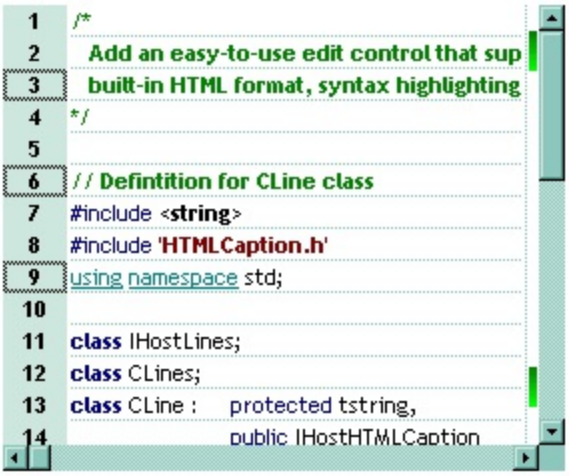
Type	Description
Long	A long expression that defines the bookmark border's width. The value is in pixels.

Use the BookmarkWidth property to show the control's bookmark. By default, the BookmarkWidth property is 0. If the BookmarkWidth property is 0 the control's bookmark border is hidden. If the BookmarkWidth property is less than 0, and the line numbers border is displayed, it displays the line bookmarks too. Use the LineNumberWidth property to show the control's line number border. Use the BookmarkImage property to assign a custom icon for bookmarks. Use the BookMarkBackColor property to specify the bookmark's header background color.

The following VB sample puts the line numbers and bookmarks on the same header:

```
With Edit1
    .LineNumberWidth = 32
    .BookmarkWidth = -1
End With
```

The following screen show shows the control's bookmarks header bar (on the left side):



The following C++ sample puts the line numbers and bookmarks on the same header:

```
m_edit.SetLineNumberWidth( 32 );
m_edit.SetBookmarkWidth( -1 );
```

The following VB.NET sample puts the line numbers and bookmarks on the same header:

```
With AxEdit1
   .LineNumberWidth = 32
   .BookmarkWidth = -1
End With
```

The following C# sample puts the line numbers and bookmarks on the same header:

```
axEdit1.LineNumberWidth = 32;
axEdit1.BookmarkWidth = -1;
```

The following VFP sample puts the line numbers and bookmarks on the same header:

```
With thisform.Edit1.Object
   .LineNumberWidth = 32
   .BookmarkWidth = -1
endwith
```

property Edit.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 Edit.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 edit-control operation.

property Edit.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 Edit1
  .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, exFieldReplaceIn) = "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 Edit1
    .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_edit.SetCaption( 2 /*exContextMenu*/, 16384 /*exContextUndo*/, "Revin" );
m_edit.SetCaption( 2 /*exContextMenu*/, 16385 /*exContextRedo*/, "Refac" );
m_edit.SetCaption( 2 /*exContextMenu*/, 16387 /*exContextCut*/, "Sterg si copiez" );
m_edit.SetCaption( 2 /*exContextMenu*/, 16388 /*exContextCopy*/, "Copiez" );
m_edit.SetCaption( 2 /*exContextMenu*/, 16389 /*exContextPaste*/, "Suprapun" );
m_edit.SetCaption( 2 /*exContextMenu*/, 16390 /*exContextDelete*/, "Sterg" );
m_edit.SetCaption( 2 /*exContextMenu*/, 16392 /*exContextSelectAll*/, "Selectez tot" );
```

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

```
With AxEdit1
    .set_Caption(EXEDITLib.DialogEnum.exContextMenu,
EXEDITLib.FieldDialogEnum.exContextUndo, "Revin")
    .set_Caption(EXEDITLib.DialogEnum.exContextMenu,
EXEDITLib.FieldDialogEnum.exContextRedo, "Refac")
```

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

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

```
axEdit1.set_Caption(EXEDITLib.DialogEnum.exContextMenu,
EXEDITLib.FieldDialogEnum.exContextUndo, "Revin");
axEdit1.set_Caption(EXEDITLib.DialogEnum.exContextMenu,
EXEDITLib.FieldDialogEnum.exContextRedo, "Refac");
axEdit1.set_Caption(EXEDITLib.DialogEnum.exContextMenu,
EXEDITLib.FieldDialogEnum.exContextCut, "Sterg si copiez");
axEdit1.set_Caption(EXEDITLib.DialogEnum.exContextMenu,
EXEDITLib.FieldDialogEnum.exContextCopy, "Copiez");
axEdit1.set_Caption(EXEDITLib.DialogEnum.exContextMenu,
EXEDITLib.FieldDialogEnum.exContextPaste, "Suprapun");
axEdit1.set_Caption(EXEDITLib.DialogEnum.exContextMenu,
EXEDITLib.FieldDialogEnum.exContextDelete, "Sterg");
axEdit1.set_Caption(EXEDITLib.DialogEnum.exContextMenu,
EXEDITLib.FieldDialogEnum.exContextSelectAll, "Selectez tot");
```

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

```
With thisform.Edit1.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
```

property Edit.CaretLine as Long

Specifies the line of caret.

Type	Description
Long	A long expression that defines the caret's line.

Use the CaretLine and CaretPos properties to determine the caret's position. The CaretLine property is 1-based. The index for the first line in the control's text is 1. Use the TextLine property to get the line based on its index. Use the DeleteLine method to delete a line given its index. Use the CaretPosX and CaretPosY properties to get the pixels coordinates of the control's caret relative to the control's client area. The BackColorCaretLine property returns or sets the background color used to display the cursor line. Use the ClientToCursor property to determine the line from cursor. Use the Bookmark property to bookmark a line.

property Edit.CaretPos as Long

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

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

Use the CaretLine and CaretPos properties to determine the caret's position. The CaretPos property is 0-based. The first character in a line is 0. Use the TextLine property to get the line based on its index. Use the DeleteLine method to delete a line given its index. Use the CaretPosX and CaretPosY properties to get the pixels coordinates of the control's caret relative to the control's client area. Use the ClientToCursor property to determine the line from cursor. Use the CaretPos to ensure that specified character fits the control's client area. Use the EnsureVisibleLine method to ensure that a line fits the control's client area.

property Edit.CaretPosX as Long

Retrieves the x-coordinate of the caret in pixels, relative to the control's client area.

Type	Description
Long	A long expression that indicates the x-coordinate of the control's caret, in pixels relative to the control's client area.

Use the CaretPosX and CaretPosY properties to get the pixels coordinates of the control's caret relative to the control's client area. The CaretPos property retrieves or sets a value that indicates the position of the caret in the line. The CaretLine property determines the line of the caret. Use the FocusPane property to determine the index of the pane that has the focus, when control includes splitters.

The following VB sample converts the caret client coordinates to screen coordinates:

```
Private Type POINTAPI
    x As Long
    y As Long
End Type
Private Declare Function ClientToScreen Lib "user32" (ByVal hwnd As Long, lpPoint As
POINTAPI) As Long

Private Function client2screen(ByVal e As EXEDITLibCtl.Edit, ByVal xClient As Long, ByVal
yClient As Long) As POINTAPI
  Dim p As POINTAPI
  With e
    p.x = xClient
    p.y = yClient
    ClientToScreen .hWndPane(.FocusPane), p
  End With
  client2screen = p
End Function
```

The following C++ sample converts client coordinates to screen coordinates:

```
POINT ptCursor = { m_edit.GetCaretPosX(), m_edit.GetCaretPosY() };
m_edit.ClientToScreen( &ptCursor );
```

The following VB.NET sample converts client coordinates to screen coordinates:

```
With AxEdit1
   Dim ptCursor As Point = New Point(.CaretPosX, .CaretPosY)
   ptCursor = .PointToScreen(ptCursor)
End With
```

The following C# sample converts client coordinates to screen coordinates:

```
Point ptCursor = new Point( axEdit1.CaretPosX, axEdit1.CaretPosY );
ptCursor = axEdit1.PointToScreen(ptCursor);
```

The following VFP sample converts client coordinates to screen coordinates:

```
DECLARE integer ClientToScreen IN WIN32API as "ClientToScreen" integer hWnd,string
@lpPoint

with thisform.Edit1.Object
   local ptCursor
   ptCursor = Replicate(Chr(0),8)

   ClientToScreen( .hWnd, @ptCursor )
endwith
```

property Edit.CaretPosY as Long

Retrieves the y-coordinate of the caret, in pixels relative to the control's client area.

Type	Description
Long	A long expression that indicates the y-coordinate of the caret, in pixels relative to the control's client area.

Use the CaretPosX and CaretPosY properties to get the pixels coordinates of the control's caret relative to the control's client area. The CaretPos property retrieves or sets a value that indicates the position of the caret in the line. The CaretLine property determines the line of the caret. Use the FocusPane property to determine the index of the pane that has the focus, when control includes splitters.

The following VB sample converts the caret client coordinates to screen coordinates:

```
Private Type POINTAPI
      x As Long
      y As Long
End Type
Private Declare Function ClientToScreen Lib "user32" (ByVal hwnd As Long, lpPoint As
POINTAPI) As Long

Private Function client2screen(ByVal e As EXEDITLibCtl.Edit, ByVal xClient As Long, ByVal
yClient As Long) As POINTAPI
   Dim p As POINTAPI
   With e
      p.x = xClient
      p.y = yClient
      ClientToScreen .hWndPane(.FocusPane), p
   End With
   client2screen = p
End Function
```

The following C++ sample converts client coordinates to screen coordinates:

```
POINT ptCursor = { m_edit.GetCaretPosX(), m_edit.GetCaretPosY() };
m_edit.ClientToScreen( &ptCursor );
```

The following VB.NET sample converts client coordinates to screen coordinates:

```
With AxEdit1
   Dim ptCursor As Point = New Point(.CaretPosX, .CaretPosY)
   ptCursor = .PointToScreen(ptCursor)
End With
```

The following C# sample converts client coordinates to screen coordinates:

```
Point ptCursor = new Point( axEdit1.CaretPosX, axEdit1.CaretPosY );
ptCursor = axEdit1.PointToScreen(ptCursor);
```

The following VFP sample converts client coordinates to screen coordinates:

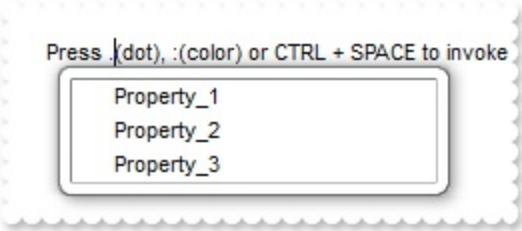
```
DECLARE integer ClientToScreen IN WIN32API as "ClientToScreen" integer hWnd,string
@lpPoint

with thisform.Edit1.Object
   local ptCursor
   ptCursor = Replicate(Chr(0),8)

   ClientToScreen( .hWnd, @ptCursor )
endwith
```

property Edit.ChangeOnKey as Integer

Specifies the code of the last key that alters the control's text.

Type	Description
Integer	A Long expression that defines the code of the last key that alters the control's text.

The ChangeOnKey property specifies the code of the last key that alters the control's text. Use the Change event to notify you application that the user changes the text in the control. Use the Text property to access the control's text. Use the CodeCompletion property to disable the code completion support. By default, the user can display the control's context window by pressing the CTRL + SPACE key combination. Use the ContextKey property to define the key combination to open the control's context window.



How can I provide different sensitive context?

VBA (MS Access, Excell...)

```
' Change event - Indicates that the control's text have changed.
Private Sub Edit1_Change()
   With Edit1
      .ShowContext .ChangeOnKey
   End With
End Sub

With Edit1
   .Text = ""
   .InsertText "Press .(dot), :(colon) or CTRL + SPACE to invoke the control's context"
   With .Context()
      .Add "General_1"
      .Add "General_2"
   End With
   With .Context("46")
      .Add "Property_1"
```

```
        .Add "Property_2"
        .Add "Property_3"
    End With
    With .Context("58")
        .Add "Method_1"
        .Add "Method_2"
        .Add "Method_3"
    End With
End With
```

VB6

```
' Change event - Indicates that the control's text have changed.
Private Sub Edit1_Change()
    With Edit1
        .ShowContext .ChangeOnKey
    End With
End Sub

With Edit1
    .Text = ""
    .InsertText "Press .(dot), :(colon) or CTRL + SPACE to invoke the control's context"
    With .Context()
        .Add "General_1"
        .Add "General_2"
    End With
    With .Context("46")
        .Add "Property_1"
        .Add "Property_2"
        .Add "Property_3"
    End With
    With .Context("58")
        .Add "Method_1"
        .Add "Method_2"
        .Add "Method_3"
    End With
End With
```

VB.NET

```
' Change event - Indicates that the control's text have changed.
Private Sub Exedit1_Change(ByVal sender As System.Object) Handles Exedit1.Change
    With Exedit1
        .ShowContext(.ChangeOnKey)
    End With
End Sub

With Exedit1
    .Text = ""
    .InsertText("Press .(dot), :(colon) or CTRL + SPACE to invoke the control's context")
    With .get_Context()
        .Add("General_1")
        .Add("General_2")
    End With
    With .get_Context("46")
        .Add("Property_1")
        .Add("Property_2")
        .Add("Property_3")
    End With
    With .get_Context("58")
        .Add("Method_1")
        .Add("Method_2")
        .Add("Method_3")
    End With
End With
```

VB.NET for /COM

```
' Change event - Indicates that the control's text have changed.
Private Sub AxEdit1_Change(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles AxEdit1.Change
    With AxEdit1
        .ShowContext(.ChangeOnKey)
    End With
End Sub
```

```
With AxEdit1
    .Text = ""
    .InsertText("Press .(dot), :(colon) or CTRL + SPACE to invoke the control's context")
    With .get_Context()
        .Add("General_1")
        .Add("General_2")
    End With
    With .get_Context("46")
        .Add("Property_1")
        .Add("Property_2")
        .Add("Property_3")
    End With
    With .get_Context("58")
        .Add("Method_1")
        .Add("Method_2")
        .Add("Method_3")
    End With
End With
```

C++

```
// Change event - Indicates that the control's text have changed.
void OnChangeEdit1()
{
    /*
        Copy and paste the following directives to your header file as
        it defines the namespace 'EXEDITLib' for the library: 'ExEdit 1.0 Control Library'
        #import <ExEdit.dll>
        using namespace EXEDITLib;
     */
    EXEDITLib::IEditPtr spEdit1 = GetDlgItem(IDC_EDIT1)->GetControlUnknown();
    spEdit1->ShowContext(spEdit1->GetChangeOnKey());
}

EXEDITLib::IEditPtr spEdit1 = GetDlgItem(IDC_EDIT1)->GetControlUnknown();
spEdit1->PutText(L"");
spEdit1->InsertText(L"Press .(dot), :(colon) or CTRL + SPACE to invoke the control's
```

```
context",vtMissing);
EXEDITLib::IEditContextPtr var_Context = spEdit1->GetContext(vtMissing);
   var_Context->Add(L"General_1",vtMissing,vtMissing,vtMissing);
   var_Context->Add(L"General_2",vtMissing,vtMissing,vtMissing);
EXEDITLib::IEditContextPtr var_Context1 = spEdit1->GetContext("46");
   var_Context1->Add(L"Property_1",vtMissing,vtMissing,vtMissing);
   var_Context1->Add(L"Property_2",vtMissing,vtMissing,vtMissing);
   var_Context1->Add(L"Property_3",vtMissing,vtMissing,vtMissing);
EXEDITLib::IEditContextPtr var_Context2 = spEdit1->GetContext("58");
   var_Context2->Add(L"Method_1",vtMissing,vtMissing,vtMissing);
   var_Context2->Add(L"Method_2",vtMissing,vtMissing,vtMissing);
   var_Context2->Add(L"Method_3",vtMissing,vtMissing,vtMissing);
```

C++ Builder

```
// Change event - Indicates that the control's text have changed.
void __fastcall TForm1::Edit1Change(TObject *Sender)
{
   Edit1->ShowContext(TVariant(Edit1->ChangeOnKey));
}

Edit1->Text = L"";
Edit1->InsertText(L"Press .(dot), :(colon) or CTRL + SPACE to invoke the control's
context",TNoParam());
Exeditlib_tlb::IEditContextPtr var_Context = Edit1->Context[TNoParam()];
   var_Context->Add(L"General_1",TNoParam(),TNoParam(),TNoParam());
   var_Context->Add(L"General_2",TNoParam(),TNoParam(),TNoParam());
Exeditlib_tlb::IEditContextPtr var_Context1 = Edit1->Context[TVariant("46")];
   var_Context1->Add(L"Property_1",TNoParam(),TNoParam(),TNoParam());
   var_Context1->Add(L"Property_2",TNoParam(),TNoParam(),TNoParam());
   var_Context1->Add(L"Property_3",TNoParam(),TNoParam(),TNoParam());
Exeditlib_tlb::IEditContextPtr var_Context2 = Edit1->Context[TVariant("58")];
   var_Context2->Add(L"Method_1",TNoParam(),TNoParam(),TNoParam());
   var_Context2->Add(L"Method_2",TNoParam(),TNoParam(),TNoParam());
   var_Context2->Add(L"Method_3",TNoParam(),TNoParam(),TNoParam());
```

C#

```
// Change event - Indicates that the control's text have changed.
private void exedit1_Change(object sender)
{
   exedit1.ShowContext(exedit1.ChangeOnKey);
}
//this.exedit1.Change += new
exontrol.EXEDITLib.exg2antt.ChangeEventHandler(this.exedit1_Change);

exedit1.Text = "";
exedit1.InsertText("Press .(dot), :(colon) or CTRL + SPACE to invoke the control's
context",null);
exontrol.EXEDITLib.Context var_Context = exedit1.get_Context(null);
   var_Context.Add("General_1",null,null,null);
   var_Context.Add("General_2",null,null,null);
exontrol.EXEDITLib.Context var_Context1 = exedit1.get_Context("46");
   var_Context1.Add("Property_1",null,null,null);
   var_Context1.Add("Property_2",null,null,null);
   var_Context1.Add("Property_3",null,null,null);
exontrol.EXEDITLib.Context var_Context2 = exedit1.get_Context("58");
   var_Context2.Add("Method_1",null,null,null);
   var_Context2.Add("Method_2",null,null,null);
   var_Context2.Add("Method_3",null,null,null);
```

JScript/JavaScript

```
<BODY onload="Init()">
<SCRIPT FOR="Edit1" EVENT="Change()" LANGUAGE="JScript">
   Edit1.ShowContext(Edit1.ChangeOnKey);
</SCRIPT>

<OBJECT CLASSID="clsid:39136531-DD0F-4281-B445-E36FC2CDDBC5" id="Edit1">
</OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
```

```
{
   Edit1.Text = "";
   Edit1.InsertText("Press .(dot), :(colon) or CTRL + SPACE to invoke the control's
context",null);
   var var_Context = Edit1.Context(null);
      var_Context.Add("General_1",null,null,null);
      var_Context.Add("General_2",null,null,null);
   var var_Context1 = Edit1.Context("46");
      var_Context1.Add("Property_1",null,null,null);
      var_Context1.Add("Property_2",null,null,null);
      var_Context1.Add("Property_3",null,null,null);
   var var_Context2 = Edit1.Context("58");
      var_Context2.Add("Method_1",null,null,null);
      var_Context2.Add("Method_2",null,null,null);
      var_Context2.Add("Method_3",null,null,null);
}
</SCRIPT>
</BODY>
```

VBScript

```
<BODY onload="Init()">
<SCRIPT LANGUAGE="VBScript">
Function Edit1_Change()
   With Edit1
      .ShowContext .ChangeOnKey
   End With
End Function
</SCRIPT>

<OBJECT CLASSID="clsid:39136531-DD0F-4281-B445-E36FC2CDDBC5" id="Edit1">
</OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
   With Edit1
```

```
        .Text = ""
        .InsertText "Press .(dot), :(colon) or CTRL + SPACE to invoke the control's context"
        With .Context()
            .Add "General_1"
            .Add "General_2"
        End With
        With .Context("46")
            .Add "Property_1"
            .Add "Property_2"
            .Add "Property_3"
        End With
        With .Context("58")
            .Add "Method_1"
            .Add "Method_2"
            .Add "Method_3"
        End With
    End With
End Function
</SCRIPT>
</BODY>
```

C# for /COM

```
// Change event - Indicates that the control's text have changed.
private void axEdit1_Change(object sender, EventArgs e)
{
    axEdit1.ShowContext(axEdit1.ChangeOnKey);
}
//this.axEdit1.Change += new EventHandler(this.axEdit1_Change);

axEdit1.Text = "";
axEdit1.InsertText("Press .(dot), :(colon) or CTRL + SPACE to invoke the control's
context",null);
EXEDITLib.Context var_Context = axEdit1.get_Context(null);
    var_Context.Add("General_1",null,null,null);
    var_Context.Add("General_2",null,null,null);
```

```
EXEDITLib.Context var_Context1 = axEdit1.get_Context("46");
   var_Context1.Add("Property_1",null,null,null);
   var_Context1.Add("Property_2",null,null,null);
   var_Context1.Add("Property_3",null,null,null);
EXEDITLib.Context var_Context2 = axEdit1.get_Context("58");
   var_Context2.Add("Method_1",null,null,null);
   var_Context2.Add("Method_2",null,null,null);
   var_Context2.Add("Method_3",null,null,null);
```

X++ (Dynamics Ax 2009)

```
// Change event - Indicates that the control's text have changed.
void onEvent_Change()
{
   ;
   exedit1.ShowContext(exedit1.ChangeOnKey());
}

public void init()
{
   COM com_Context,com_Context1,com_Context2;
   anytype var_Context,var_Context1,var_Context2;
   ;

   super();

   exedit1.Text("");
   exedit1.InsertText("Press .(dot), :(colon) or CTRL + SPACE to invoke the control's
context");
   var_Context = exedit1.Context(); com_Context = var_Context;
      com_Context.Add("General_1");
      com_Context.Add("General_2");
   var_Context1 = exedit1.Context("46"); com_Context1 = var_Context1;
      com_Context1.Add("Property_1");
      com_Context1.Add("Property_2");
      com_Context1.Add("Property_3");
```

```
    var_Context2 = exedit1.Context("58"); com_Context2 = var_Context2;
        com_Context2.Add("Method_1");
        com_Context2.Add("Method_2");
        com_Context2.Add("Method_3");
}
```

Delphi 8 (.NET only)

```
// Change event - Indicates that the control's text have changed.
procedure TWinForm1.AxEdit1_Change(sender: System.Object; e: System.EventArgs);
begin
    with AxEdit1 do
    begin
        ShowContext(TObject(ChangeOnKey));
    end
end;

with AxEdit1 do
begin
    Text := '';
    InsertText('Press .(dot), :(colon) or CTRL + SPACE to invoke the control''s
context',Nil);
    with get_Context(Nil) do
    begin
        Add('General_1',Nil,Nil,Nil);
        Add('General_2',Nil,Nil,Nil);
    end;
    with get_Context('46') do
    begin
        Add('Property_1',Nil,Nil,Nil);
        Add('Property_2',Nil,Nil,Nil);
        Add('Property_3',Nil,Nil,Nil);
    end;
    with get_Context('58') do
    begin
        Add('Method_1',Nil,Nil,Nil);
        Add('Method_2',Nil,Nil,Nil);
```

```
      Add('Method_3',Nil,Nil,Nil);
   end;
end
```

Delphi (standard)

```
// Change event - Indicates that the control's text have changed.
procedure TForm1.Edit1Change(ASender: TObject; );
begin
   with Edit1 do
   begin
      ShowContext(OleVariant(ChangeOnKey));
   end
end;

with Edit1 do
begin
   Text := '';
   InsertText('Press .(dot), :(colon) or CTRL + SPACE to invoke the control''s
context',Null);
   with Context[Null] do
   begin
      Add('General_1',Null,Null,Null);
      Add('General_2',Null,Null,Null);
   end;
   with Context['46'] do
   begin
      Add('Property_1',Null,Null,Null);
      Add('Property_2',Null,Null,Null);
      Add('Property_3',Null,Null,Null);
   end;
   with Context['58'] do
   begin
      Add('Method_1',Null,Null,Null);
      Add('Method_2',Null,Null,Null);
      Add('Method_3',Null,Null,Null);
   end;
```

```
end
```

VFP

```
*** Change event - Indicates that the control's text have changed. ***
LPARAMETERS nop
   with thisform.Edit1
      .ShowContext(.ChangeOnKey)
   endwith

with thisform.Edit1
   .Text = ""
   .InsertText("Press .(dot), :(colon) or CTRL + SPACE to invoke the control's context")
   with .Context()
      .Add("General_1")
      .Add("General_2")
   endwith
   with .Context("46")
      .Add("Property_1")
      .Add("Property_2")
      .Add("Property_3")
   endwith
   with .Context("58")
      .Add("Method_1")
      .Add("Method_2")
      .Add("Method_3")
   endwith
endwith
```

dBASE Plus

```
/*
with (this.EXEDITACTIVEXCONTROL1.nativeObject)
    Change = class::nativeObject_Change
endwith
*/
// Indicates that the control's text have changed.
function nativeObject_Change()
```

```
   oEdit = form.EXEDITACTIVEXCONTROL1.nativeObject
   oEdit.ShowContext(oEdit.ChangeOnKey)
return

local oEdit,var_Context,var_Context1,var_Context2

oEdit = form.EXEDITACTIVEXCONTROL1.nativeObject
oEdit.Text = ""
oEdit.InsertText("Press .(dot), :(colon) or CTRL + SPACE to invoke the control's
context")
var_Context = oEdit.Context()
   var_Context.Add("General_1")
   var_Context.Add("General_2")
var_Context1 = oEdit.Context("46")
   var_Context1.Add("Property_1")
   var_Context1.Add("Property_2")
   var_Context1.Add("Property_3")
var_Context2 = oEdit.Context("58")
   var_Context2.Add("Method_1")
   var_Context2.Add("Method_2")
   var_Context2.Add("Method_3")
```

XBasic (Alpha Five)

```
' Indicates that the control's text have changed.
function Change as v ()
   oEdit = topparent:CONTROL_ACTIVEX1.activex
   oEdit.ShowContext(oEdit.ChangeOnKey)
end function

Dim oEdit as P
Dim var_Context as P
Dim var_Context1 as P
Dim var_Context2 as P

oEdit = topparent:CONTROL_ACTIVEX1.activex
```

```
oEdit.Text = ""
oEdit.InsertText("Press .(dot), :(colon) or CTRL + SPACE to invoke the control's
context")
var_Context = oEdit.Context()
   var_Context.Add("General_1")
   var_Context.Add("General_2")
var_Context1 = oEdit.Context("46")
   var_Context1.Add("Property_1")
   var_Context1.Add("Property_2")
   var_Context1.Add("Property_3")
var_Context2 = oEdit.Context("58")
   var_Context2.Add("Method_1")
   var_Context2.Add("Method_2")
   var_Context2.Add("Method_3")
```

Visual Objects

```
METHOD OCX_Exontrol1Change() CLASS MainDialog
   // Change event - Indicates that the control's text have changed.
   oDCOCX_Exontrol1:ShowContext(oDCOCX_Exontrol1:ChangeOnKey)
RETURN NIL

local var_Context,var_Context1,var_Context2 as IEditContext

oDCOCX_Exontrol1:Text := ""
oDCOCX_Exontrol1:InsertText("Press .(dot), :(colon) or CTRL + SPACE to invoke the
control's context",nil)
var_Context := oDCOCX_Exontrol1:[Context,nil]
   var_Context:Add("General_1",nil,nil,nil)
   var_Context:Add("General_2",nil,nil,nil)
var_Context1 := oDCOCX_Exontrol1:[Context,"46"]
   var_Context1:Add("Property_1",nil,nil,nil)
   var_Context1:Add("Property_2",nil,nil,nil)
   var_Context1:Add("Property_3",nil,nil,nil)
var_Context2 := oDCOCX_Exontrol1:[Context,"58"]
   var_Context2:Add("Method_1",nil,nil,nil)
```

```
   var_Context2:Add("Method_2",nil,nil,nil)
   var_Context2:Add("Method_3",nil,nil,nil)
```

PowerBuilder

```
/*begin event Change() - Indicates that the control's text have changed.*/
/*
    oEdit = ole_1.Object
    oEdit.ShowContext(oEdit.ChangeOnKey)
*/
/*end event Change*/

OleObject oEdit,var_Context,var_Context1,var_Context2

oEdit = ole_1.Object
oEdit.Text = ""
oEdit.InsertText("Press .(dot), :(colon) or CTRL + SPACE to invoke the control's
context")
var_Context = oEdit.Context()
   var_Context.Add("General_1")
   var_Context.Add("General_2")
var_Context1 = oEdit.Context("46")
   var_Context1.Add("Property_1")
   var_Context1.Add("Property_2")
   var_Context1.Add("Property_3")
var_Context2 = oEdit.Context("58")
   var_Context2.Add("Method_1")
   var_Context2.Add("Method_2")
   var_Context2.Add("Method_3")
```

Visual DataFlex

```
// Indicates that the control's text have changed.
Procedure OnComChange
   Forward Send OnComChange
   Send ComShowContext (ComChangeOnKey(Self))
```

```
End_Procedure

Procedure OnCreate
   Forward Send OnCreate
   Set ComText to ""
   Send ComInsertText "Press .(dot), :(colon) or CTRL + SPACE to invoke the control's
context" Nothing
   Variant voContext
   Get ComContext Nothing to voContext
   Handle hoContext
   Get Create (RefClass(cComContext)) to hoContext
   Set pvComObject of hoContext to voContext
      Send ComAdd of hoContext "General_1" Nothing Nothing Nothing
      Send ComAdd of hoContext "General_2" Nothing Nothing Nothing
   Send Destroy to hoContext
   Variant voContext1
   Get ComContext "46" to voContext1
   Handle hoContext1
   Get Create (RefClass(cComContext)) to hoContext1
   Set pvComObject of hoContext1 to voContext1
      Send ComAdd of hoContext1 "Property_1" Nothing Nothing Nothing
      Send ComAdd of hoContext1 "Property_2" Nothing Nothing Nothing
      Send ComAdd of hoContext1 "Property_3" Nothing Nothing Nothing
   Send Destroy to hoContext1
   Variant voContext2
   Get ComContext "58" to voContext2
   Handle hoContext2
   Get Create (RefClass(cComContext)) to hoContext2
   Set pvComObject of hoContext2 to voContext2
      Send ComAdd of hoContext2 "Method_1" Nothing Nothing Nothing
      Send ComAdd of hoContext2 "Method_2" Nothing Nothing Nothing
      Send ComAdd of hoContext2 "Method_3" Nothing Nothing Nothing
   Send Destroy to hoContext2
End_Procedure
```

XBase++

```
PROCEDURE OnChange(oEdit)
   oEdit:ShowContext(oEdit:ChangeOnKey())
RETURN

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

PROCEDURE Main
   LOCAL oForm
   LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
   LOCAL oEdit
   LOCAL oContext,oContext1,oContext2

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

   oEdit := XbpActiveXControl():new( oForm:drawingArea )
   oEdit:CLSID  := "Exontrol.Edit.1" /*{39136531-DD0F-4281-B445-
E36FC2CDDBC5}*/
   oEdit:create(,, {10,60},{610,370} )

      oEdit:Change := {|| OnChange(oEdit)} /*Indicates that the control's text have
changed.*/

      oEdit:Text := ""
      oEdit:InsertText("Press .(dot), :(colon) or CTRL + SPACE to invoke the control's
context")
      oContext := oEdit:Context()
         oContext:Add("General_1")
         oContext:Add("General_2")
      oContext1 := oEdit:Context("46")
         oContext1:Add("Property_1")
         oContext1:Add("Property_2")
         oContext1:Add("Property_3")
      oContext2 := oEdit:Context("58")
```

```
        oContext2:Add("Method_1")
        oContext2:Add("Method_2")
        oContext2:Add("Method_3")

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

method Edit.ClearBackColorLine (Index as Long)

Clears the line's background color.

Type	Description
Index as Long	A long expression that indicates the index of line.

Use the ClearBackColorLine method to clear the line's background color, that was previously set by BackColorLine property. Use the BackColorLine property to change the line's background color. Use the BackColor property to specify the control's background color.

method Edit.ClearBookmarks ()

Clears all bookmarks.

Type	Description

Clears all bookmarks into the control. Use the Bookmark property to delete a specific bookmark. Use the BookmarksList property to bookmark multiple lines at a single moment. Use the BookmarkWidth property to hide the control's bookmarks header bar. Use the BookMarkBackColor and BookMarkBackColor2 properties to customize colors of the control's bookmark bar.

method Edit.ClearExpressions ()

Clears the expressions collection into a sensitive control.

Type	Description

Clears the entire expressions collection. The ClearExpressions method has effect if the control's EditType property is exSensitive . Use the DeleteExpression method to remove a specific expression. Use the Refresh method to reflect changes in the control's client area.

method Edit.ClearForeColorLine (Index as Long)

Clears the line's foreground color.

Type	Description
Index as Long	A long expression that indicates the index of line.

Use the ClearForeColorLine method to clear the line's foreground color, that was previously set by ForeColorLine property. Use the ForeColorLine property to change the line's foreground color. Use the ForeColor property to specify the control's foreground color.

method Edit.ClearKeywords ()

Clears the keywords collection into a sensitive control.

Type	Description

Clears the keywords collection. The ClearKeywords method has effect if the control's EditType property is exSensitive. Use the DeleteKeyword method to remove a particular keyword. Use the Refresh method to reflect changes in the control's visible area.

method Edit.ClearWilds ()

Clears the wild characters expressions collection into a sensitive control.

Type	Description

Clears the wild characters expression into a sensitive control. Use the DeleteWild method to delete a specific wild characters expression.

method Edit.ClientToCursor (X as OLE_XPOS_PIXELS, Y as OLE_YPOS_PIXELS, XCursor as Long, YCursor as Long)

Converts client coordinates to cursor coordinates.

Type	Description
X as OLE_XPOS_PIXELS	A single expression that specifies the x client coordinate. The x value is in pixels.
Y as OLE_YPOS_PIXELS	A single expression that specifies the y client coordinate. The y value is in pixels.
XCursor as Long	A long expression that indicates the x position of the cursor. The value is in cursor coordinate.
YCursor as Long	A long expression that indicates the y position of the cursor. The value is in cursor coordinate.

Use the ClientToCursor methdo to converts the client coordinates to cursor coordinates. The X and Y coordinates must be pixels coordinates. Make sure that these coordinates are not twips (By default, the VB uses the twips coordinates, C++ uses pixels). Use the TextLine property to get the line giving its index.

The following VB sample displays the line from the cursor:

```
Private Sub Edit1_MouseMove(Button As Integer, Shift As Integer, x As Single, y As Single)
  With Edit1
    Dim i As Long, c As Long
    .ClientToCursor x / Screen.TwipsPerPixelX, y / Screen.TwipsPerPixelY, c, i
    If (i > 0) Then
      Debug.Print .TextLine(i)
    End If
  End With
End Sub
```

The following C++ sample displays the line from the cursor:

```
void OnMouseMoveEdit1(short Button, short Shift, long X, long Y)
{
  long nLine = 0, nColumn = 0;
  m_edit.ClientToCursor( X, Y, &nColumn, &nLine );
  if ( nLine > 0 )
    OutputDebugString( m_edit.GetTextLine( nLine ) );
```

```
}
```

The following VB.NET sample displays the line from the cursor:

```
Private Sub AxEdit1_MouseMoveEvent(ByVal sender As Object, ByVal e As
AxEXEDITLib._IEditEvents_MouseMoveEvent) Handles AxEdit1.MouseMoveEvent
   With AxEdit1
      Dim i As Integer, c As Integer
      .ClientToCursor(e.x, e.y, c, i)
      If (i > 0) Then
         Debug.WriteLine(.get_TextLine(i))
      End If
   End With
End Sub
```

The following C# sample displays the line from the cursor:

```
private void axEdit1_MouseMoveEvent(object sender,
AxEXEDITLib._IEditEvents_MouseMoveEvent e)
{
   int iLine = 0, iColumn = 0;
   axEdit1.ClientToCursor(e.x, e.y, out iColumn, out iLine);
   if (iLine > 0)
      System.Diagnostics.Debug.WriteLine(axEdit1.get_TextLine(iLine));
}
```

The following VFP sample displays the line from the cursor:

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

with thisform.Edit1
   local i, c
   i = 0
   c = 0
   .ClientToCursor(x, y, @c, @i)
   If (i > 0) Then
      wait window nowait .TextLine(i)
   EndIf
```

```
endwith
```

property Edit.CloseBrackets as String

Specifies the list of closed brackets.

Type	Description
String	A string expression that defines the list of characters the represent a closed bracket.

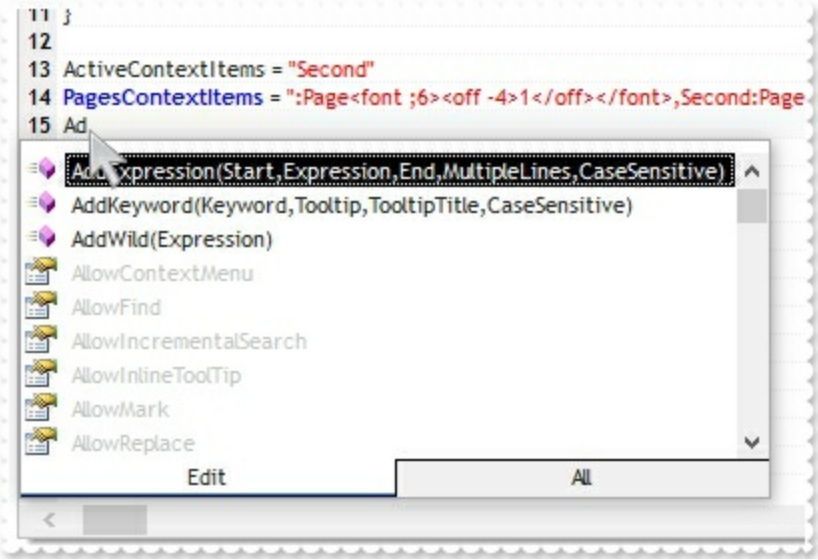
Use the OpenBrackets and CloseBrackets properties to define the list of available brackets. By default, the CloseBrackets property is "})]>". The control uses the OpenBrackets and CloseBrackets properties to get the brackets available when user presses the CTRL + {. When user presses the CTRL + { and the character at caret position is contained by the OpenBrackets property or the CloseBrackets property the control looks for the opposite bracket.

property Edit.CodeCompletion as CodeCompletionEnum

Specifies whether the code completion feature is enabled or disabled.

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

The control provides code completion support. By default, the user can invoke the control's context list by pressing CTRL+SPACE combination (the CodeCompletion property is True, by default). Use the Context collection to add your keywords to the control's context list. Use the Add method to add new items to the control's context list. The ActiveContextItems property defines the key of the current context. The ContextKey property specifies the key combination to display the control's context window.

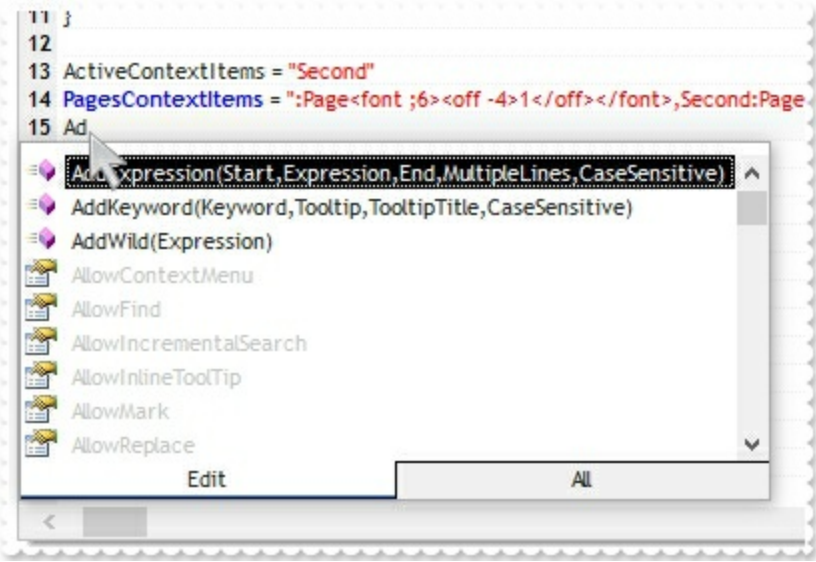


property Edit.Context ([Context as Variant]) as EditContext

Gets the context's collection.

Type	Description
Context as Variant	A string expression that defines the key of the context window to be defined / shown. Leave it empty, if single context is defined.
EditContext	A Context object that indicates the collection of keywords in the control's context list.

Use the Context collection to customize the list of keywords when the code completion is enabled. The control can display multiple pages on the control's sensitive context, if multiple key-context are defined. The ActiveContextItems property defines the key of the current context. Use the CodeCompletion property to disable the code completion support. By default, the user can display the control's context window by pressing the CTRL + SPACE key combination. Use the ContextKey property to define the key combination to open the control's context window. The ShowContext method shows programmatically the control's context, as the user would press the CTRL + SPACE key. The PagesContextItems property specifies the pages (key[:caption]) of the control's sensitive context, separated by comma character.



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. Use the AllowContextMenu property to display the control's context menu when user right clicks the control.

How can I add a sensitive context window?

VBA (MS Access, Excell...)

```
With Edit1
   .AddKeyword "<b>class</b>","this is a bit of text that's shown when the cursor hovers the <b>class</b> keyword.","exontrol.edit"
   .Refresh
   .Context().Add "class"
End With
```

VB6

```
With Edit1
   .AddKeyword "<b>class</b>","this is a bit of text that's shown when the cursor hovers the <b>class</b> keyword.","exontrol.edit"
   .Refresh
   .Context().Add "class"
End With
```

VB.NET

```
With Exedit1
   .AddKeyword("<b>class</b>","this is a bit of text that's shown when the cursor hovers the <b>class</b> keyword.","exontrol.edit")
   .Refresh()
   .get_Context().Add("class")
End With
```

VB.NET for /COM

```
With AxEdit1
   .AddKeyword("<b>class</b>","this is a bit of text that's shown when the cursor hovers the <b>class</b> keyword.","exontrol.edit")
   .Refresh()
   .get_Context().Add("class")
End With
```

C++

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

```
    #import <ExEdit.dll>
    using namespace EXEDITLib;
*/
EXEDITLib::IEditPtr spEdit1 = GetDlgItem(IDC_EDIT1)->GetControlUnknown();
spEdit1->AddKeyword(L"<b>class</b>","this is a bit of text that's shown when the cursor hovers the <b>class</b> keyword.","exontrol.edit",vtMissing);
spEdit1->Refresh();
spEdit1->GetContext(vtMissing)->Add(L"class",vtMissing,vtMissing,vtMissing);
```

C++ Builder

```
Edit1->AddKeyword(L"<b>class</b>",TVariant("this is a bit of text that's shown when the cursor hovers the <b>class</b> keyword."),TVariant("exontrol.edit"),TNoParam());
Edit1->Refresh();
Edit1->Context[TNoParam()]->Add(L"class",TNoParam(),TNoParam(),TNoParam());
```

C#

```
exedit1.AddKeyword("<b>class</b>","this is a bit of text that's shown when the cursor hovers the <b>class</b> keyword.","exontrol.edit",null);
exedit1.Refresh();
exedit1.get_Context(null).Add("class",null,null,null);
```

JScript/JavaScript

```
<BODY onload="Init()">
<OBJECT CLASSID="clsid:39136531-DD0F-4281-B445-E36FC2CDDBC5" id="Edit1">
</OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
   Edit1.AddKeyword("<b>class</b>","this is a bit of text that's shown when the
```

```
cursor hovers the <b>class</b> keyword.","exontrol.edit",null);
   Edit1.Refresh();
   Edit1.Context(null).Add("class",null,null,null);
}
</SCRIPT>
</BODY>
```

VBScript

```
<BODY onload="Init()">
<OBJECT CLASSID="clsid:39136531-DD0F-4281-B445-E36FC2CDDBC5" id="Edit1">
</OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
   With Edit1
      .AddKeyword "<b>class</b>","this is a bit of text that's shown when the cursor
hovers the <b>class</b> keyword.","exontrol.edit"
      .Refresh
      .Context().Add "class"
   End With
End Function
</SCRIPT>
</BODY>
```

C# for /COM

```
axEdit1.AddKeyword("<b>class</b>","this is a bit of text that's shown when the
cursor hovers the <b>class</b> keyword.","exontrol.edit",null);
axEdit1.Refresh();
axEdit1.get_Context(null).Add("class",null,null,null);
```

X++ (Dynamics Ax 2009)

```
public void init()
```

```
{
   COM com_Context;
   anytype var_Context;
   ;

   super();

   exedit1.AddKeyword("<b>class</b>","this is a bit of text that's shown when the cursor hovers the <b>class</b> keyword.","exontrol.edit");
   exedit1.Refresh();
   var_Context = COM::createFromObject(exedit1.Context()); com_Context = var_Context;
   com_Context.Add("class");
}
```

Delphi 8 (.NET only)

```
with AxEdit1 do
begin
   AddKeyword('<b>class</b>','this is a bit of text that''s shown when the cursor hovers the <b>class</b> keyword.','exontrol.edit',Nil);
   Refresh();
   get_Context(Nil).Add('class',Nil,Nil,Nil);
end
```

Delphi (standard)

```
with Edit1 do
begin
   AddKeyword('<b>class</b>','this is a bit of text that''s shown when the cursor hovers the <b>class</b> keyword.','exontrol.edit',Null);
   Refresh();
   Context[Null].Add('class',Null,Null,Null);
end
```

VFP

```
with thisform.Edit1
```

```
   .AddKeyword("<b>class</b>","this is a bit of text that's shown when the cursor hovers the <b>class</b> keyword.","exontrol.edit")
   .Refresh
   .Context().Add("class")
endwith
```

dBASE Plus

```
local oEdit

oEdit = form.EXEDITACTIVEXCONTROL1.nativeObject
oEdit.AddKeyword("<b>class</b>","this is a bit of text that's shown when the cursor hovers the <b>class</b> keyword.","exontrol.edit")
oEdit.Refresh()
oEdit.Context().Add("class")
```

XBasic (Alpha Five)

```
Dim oEdit as P

oEdit = topparent:CONTROL_ACTIVEX1.activex
oEdit.AddKeyword("<b>class</b>","this is a bit of text that's shown when the cursor hovers the <b>class</b> keyword.","exontrol.edit")
oEdit.Refresh()
oEdit.Context().Add("class")
```

Visual Objects

```

oDCOCX_Exontrol1:AddKeyword("<b>class</b>","this is a bit of text that's shown when the cursor hovers the <b>class</b> keyword.","exontrol.edit",nil)
oDCOCX_Exontrol1:Refresh()
oDCOCX_Exontrol1:[Context,nil]:Add("class",nil,nil,nil)
```

PowerBuilder

```
OleObject oEdit

oEdit = ole_1.Object
oEdit.AddKeyword("<b>class</b>","this is a bit of text that's shown when the cursor hovers the <b>class</b> keyword.","exontrol.edit")
oEdit.Refresh()
oEdit.Context().Add("class")
```

Visual DataFlex

```
Procedure OnCreate
   Forward Send OnCreate
   Send ComAddKeyword "<b>class</b>" "this is a bit of text that's shown when the cursor hovers the <b>class</b> keyword." "exontrol.edit" Nothing
   Send ComRefresh
   Variant voContext
   Get ComContext Nothing to voContext
   Handle hoContext
   Get Create (RefClass(cComContext)) to hoContext
   Set pvComObject of hoContext to voContext
      Send ComAdd of hoContext "class" Nothing Nothing Nothing
   Send Destroy to hoContext
End_Procedure
```

XBase++

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

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

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

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

   oEdit := XbpActiveXControl():new( oForm:drawingArea )
   oEdit:CLSID  := "Exontrol.Edit.1" /*{39136531-DD0F-4281-B445-
E36FC2CDDBC5}*/
   oEdit:create(,, {10,60},{610,370} )

      oEdit:AddKeyword("<b>class</b>","this is a bit of text that's shown when the
cursor hovers the <b>class</b> keyword.","exontrol.edit")
      oEdit:Refresh()
      oEdit:Context():Add("class")

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

property Edit.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. Use the Context property to access the control's context collection. Use the Add method to add new items to the context collection. The ShowContext method shows programmatically the control's context, as the user would press the CTRL + SPACE key. The ChangeOnKey property specifies the code of the last key that alters the control's text.

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 Edit1
   .ContextKey = vbKeySpace + 256 * 2
End With
```

or

```
With Edit1
   .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 Edit1
    .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 Edit1_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 Edit.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 Edit1
      .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_edit.SetAllowContextMenu( TRUE );
CString strCustom( "\r" );
strCustom += "Item 1";
strCustom += "\r\r";
```

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

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

```
With AxEdit1
   .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:

```
axEdit1.AllowContextMenu = true;
axEdit1.ContextMenuItems = "\rItem1\r\rItem2\rItem3";
```

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

```
with thisform.Edit1
   .AllowContextMenu = .t.
   .ContextMenuItems = chr(13) + "Item 1" + chr(13) + chr(13) + "Item 2" + chr(13) +
"Item 3"
endwith
```

property Edit.Count as Long

Counts the lines in the control.

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

The Count property gets the number of lines in the control. Use the TextLine property to access the line based on its index. Use the Text property to access the control's text. Use the InsertText method to insert lines to the control. Use the DeleteLine method to delete a specific line.

The following VB sample prints the line in the control:

```
With Edit1
   Dim i As Long
   For i = 1 To .Count
      Debug.Print .TextLine(i)
   Next
End With
```

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

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

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

```
With AxEdit1
   Dim i As Integer
   For i = 1 To .Count
      Debug.WriteLine(.get_TextLine(i))
   Next
End With
```

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

```
for (int i = 1; i <= axEdit1.Count; i++)
   System.Diagnostics.Debug.WriteLine(axEdit1.get_TextLine(i));
```

The following VFP sample prints the line in the control:

```
with thisform.Edit1.Object
   local i
   for i = 1 to .Count
      wait window nowait .TextLine(i)
   next
endwith
```

property Edit.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 bellow, 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: LaadCursor 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 Edit1
   .Cursor(exLineNumberArea) = "exCross"
End With
```

Here's the VB.NET alternative:

```
AxEdit1.Ctlset_Cursor(EXEDITLib.ClientAreaEnum.exLineNumberArea, "exCross")
```

The following VB sample loads a cursor from a file:

```
With Edit1
   .Cursor(exLineNumberArea) = "C:\WINNT\Cursors\metronom.ani"
End With
```

And here's the VB.NET alternative:

```
AxEdit1.Ctlset_Cursor(EXEDITLib.ClientAreaEnum.exLineNumberArea,
"C:\WINNT\Cursors\metronom.ani")
```

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

```
AxEdit1.Ctlset_Cursor(EXEDITLib.ClientAreaEnum.exLineNumberArea,
Cursors.PanEast.Handle)
```

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

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

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

```
axEdit1.Ctlset_Cursor(EXEDITLib.ClientAreaEnum.exEditArea,
"C:\\WINNT\\Cursors\\metronom.ani");
```

The following VFP sample loads the cursor from a file:

```
with thisform.Edit1.Object
   .Cursor(0) = "C:\WINNT\Cursors\metronom.ani"
endwith
```

method Edit.DeleteExpression (Expression as String)

Deletes an expression from expressions collection.

Type	Description
Expression as String	A string expression that defines the expression being deleted. The Expression parameter **must NOT** include HTML tags. For instance, if your expression is "/*" the name of the expression is "/*" not "/*"

Use the DeleteExpression method to remove a particular expression into a sensitive control. The DeleteExpression method has effect if the control's EditType method is exSensitive. For instance, if you defined an expression AddExpression("//"), the DeleteExpression("//") should be called, else the expression will not be removed. Call the Refresh method to reflect changes in the control. Use the ClearExpressions method to remove the entire collection of expressions. The DeleteExpression method deletes also the list of prefixes defined by the IgnorePrefixInExpression property.

method Edit.DeleteKeyword (Keyword as String)

Deletes a keyword from keywords collection.

Type	Description
Keyword as String	A string expression that defines the keyword being deleted. The Keyword parameter must not include HTML tags.

Use the DeleteKeyword method to delete a particular keyword from the keywords collection. The DeleteKeyword method has effect only if the control's EditType property is exSensitive. Use the ClearKeywords method to clear all keywords. Use the Refresh method to let control reflects changes in the visible area.

The following VB sample adds a keyword and removes it when then user presses a button:

```
Private Sub Form_Load()
   With Edit1
      .AddKeyword "<b>public</b>", "The public keyword's tooltip."
      .Refresh
   End With
End Sub
```

```
Private Sub Command1_Click()
   With Edit1
      .DeleteKeyword "public"
      .Refresh
   End With
End Sub
```

Important to note is that the "public" keyword doesn't include the HTML tags, when DeleteKeyword method is called.

The following C++ sample adds a keyword when the form is loaded, and removes it when the user presses a button:

```
COleVariant vtMissing; vtMissing.vt = VT_ERROR;
m_edit.AddKeyword("<b>public</b>", COleVariant( "The public keyword's tooltip." ),
vtMissing, vtMissing);
m_edit.Refresh();
```

```
void OnButton1()
{
   m_edit.DeleteKeyword( "public" );
   m_edit.Refresh();
}
```

The following VB.NET sample adds a keyword when the form is loaded, and removes it when the user presses a button:

```
With AxEdit1
   .AddKeyword("<b>public</b>", "The public keyword's tooltip.", Nothing, Nothing)
   .CtlRefresh()
End With
```

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click
   With AxEdit1
      .DeleteKeyword("public")
      .CtlRefresh()
   End With
End Sub
```

The following C# sample adds a keyword when the form is loaded, and removes it when the user presses a button:

```
axEdit1.AddKeyword("<b>public</b>", "The public keyword's tooltip.", null, null);
axEdit1.CtlRefresh();
```

```
private void button1_Click(object sender, EventArgs e)
{
   axEdit1.DeleteKeyword("public");
   axEdit1.CtlRefresh();
}
```

The following VFP sample adds a keyword when the form is loaded, and removes it when the user presses a button:

```
With thisform.Edit1.Object
   .AddKeyword("<b>public</b>", "The public keyword's tooltip.")
```

```
    .Refresh()
EndWith
```

```
with thisform.Edit1.Object
    .DeleteKeyword("public")
    .Refresh()
endwith
```

method Edit.DeleteLine (Index as Long)

Deletes the line.

Type	Description
Index as Long	A long expression that indicates the index of line being accessed. The index is 1 based. The index of the first line in the control's text is 1.

Use the DeleteLine method to remove programmatically a line based on its index. Use the Text property to clear all lines in the control's text. Use the InsertText method to insert programmatically new lines to the control. Use the CaretLine property to specify the index of the current line.

method Edit.DeleteWild (Expression as String)

Deletes an entry from the wild characters expressions collection.

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

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

```
With Edit1
    .AddWild ("<b><fgcolor=FF0000>*;</fgcolor></b>")
End With
```

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

```
With Edit1
    .DeleteWild "*;"
    .Refresh
End With
```

Use the ClearWilds method to clear the all wild characters expressions. The Refresh method should be called after DeleteWild method was called to reflect the latest changes.

property Edit.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. Use the GetSelection method to determine the coordinates of the selected text. Use the SetSelection method to select a text giving the end and start coordinates. 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.

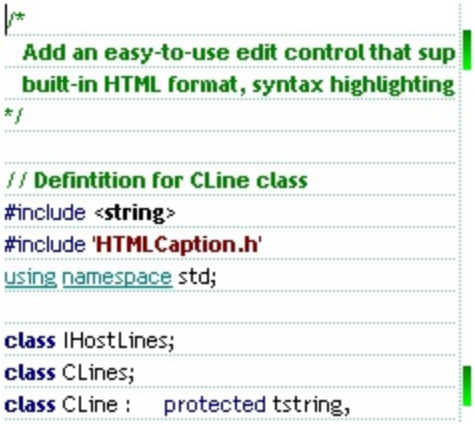
property Edit.DrawGridLines as Boolean

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

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

By default, the DrawGridLines property is False. Use the BookMarkBackColor2 property to specify the grid lines color. Use the BackColor property to specify the control's background color.

The following screen displays the control's grid lines:



property Edit.EditType as EditTypeEnum

Specifies the control's type.

Type	Description
EditTypeEnum	An EditTypeEnum expression that defines the control's edit type.

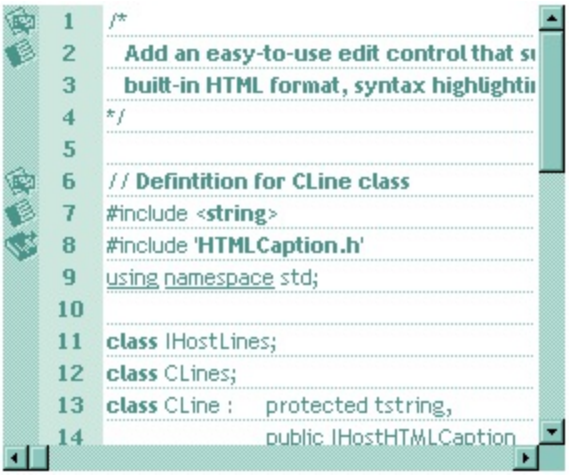
The EditType property defines the control's edit type. The control supports standard, sensitive or HTML type. If the EditType property is exSensitive use the AddKeyword, AddExpression, AddWild, DeleteKeyword, DeleteExpression, DeleteWild, ClearKeywords, ClearExpressions, ClearWilds methods to update the control's keywords and expressions collections. By default, the EditType property is exSensitive.

property Edit.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:

```
Edit1.Enabled = False
```

The following C++ sample disables the control:

```
m_edit.SetEnabled( FALSE );
```

The following VB.NET sample disables the control:

```
AxEdit1.Enabled = False
```

The following C# sample disables the control:

```
axEdit1.Enabled = false;
```

The following VFP sample disables the control:

```
with thisform.Edit1.Object
```

```
    .Enabled = .f.
endwith
```

method Edit.EnsureVisibleLine (Index as Long)

Ensures that specified line is visible.

Type	Description
Index as Long	A long expression that indicates the index of line being displayed.

Use the EnsureVisibleLine method to ensure that the line is visible in the control's client area. The control scrolls its content if necessary to make sure that the line is visible. Use the ClientToCursor method to get the line from the point. Use the CaretPos to ensure that specified character fits the control's client area.

property Edit.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.

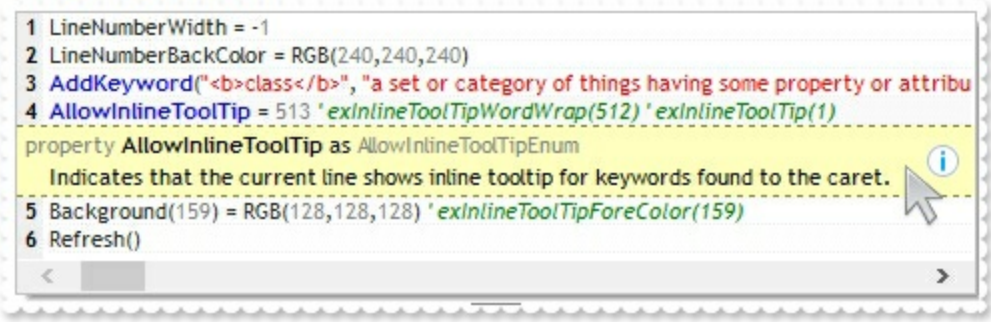
Calling the EventParam property outside of an event produces an OLE error, such as pointer invalid, as its scope was designed to be used only during events.

property Edit.ExcludeInlineToolTip as String

Specifies a list of keywords, separated by , (comma character), that specifies the keywords to be excluded from the inline tooltip support.

Type	Description
String	A String expression that defines a list of keywords, separated by , (comma character), that specifies the keywords to be excluded from the inline tooltip support.

By default, the ExcludeInlineToolTip property is empty. The ExcludeInlineToolTip property specifies a list of keywords, separated by , (comma character), that specifies the keywords to be excluded from the inline tooltip support. The Inline-ToolTip feature allows displaying inline tooltips of the keywords found at the current line as shown in the picture above. Use the AllowInlineToolTip property to enable or disable the control's inline-tooltip feature. For instance, you can use the Inline-ToolTip feature to display inside the control at current line, any information/html text you may need. The AddKeyword method adds a keyword to a sensitive control. The ToolTip/ToolTipTile parameters of the AddKeyword method defines the inline-tooltip for the specified keyword. The TempInlineToolTip property specifies a temporarily inline tooltip to be shown at the current line.



The Background(exInlineToolTipForeColor) / Background(exInlineToolTipBackColor) property define the foreground / background color / visual appearance of the inline-tooltip panel.

method Edit.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 Edit1.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 Edit.FileName as String

Specifies the name of the file being loaded or saved.

Type	Description
String	A string expression that indicates the name of the file being loaded or saved.

Use the FileName property to identify the name of the file being loaded or saved. Use the Load/LoadFromUnicode method to load files. Use the Save/SaveAsUnicode method to save control's content to a file.

The following VB sample extracts the name of the file from the path:

```
Private Function title(ByVal s As String) As String
   Dim i As Long
   i = InStrRev(s, "\")
   If i > 0 Then
      title = Right(s, Len(s) - i)
   Else
      title = s
   End If
End Function
```

The following C++ sample extracts the name of the file from the path:

```
CString title( LPCTSTR szFileName )
{
   CString strTitle( szFileName );
   int p = strTitle.ReverseFind('\\');
   if ( p >= 0 )
      strTitle = strTitle.operator LPCTSTR() + p + 1;
   return strTitle;
}
```

The following VB.NET sample extracts the name of the file from the path:

```
Private Function title(ByVal s As String) As String
   Dim i As Long
   i = InStrRev(s, "\")
```

```
    If i > 0 Then
        title = Microsoft.VisualBasic.Right(s, Len(s) - i)
    Else
        title = s
    End If
End Function
```

The following C# sample extracts the name of the file from the path:

```
private string title(string szFilename)
{
    string strTitle = szFilename;
    int p = strTitle.LastIndexOf('\\');
    if (p >= 0)
        strTitle = strTitle.Substring(p + 1);
    return strTitle;
}
```

The following VFP sample extracts the name of the file from the path:

```
local strTitle, p
strTitle = thisform.Edit1.FileName
p = rat('\',strTitle)
if ( p > 0 )
    strTitle = substr(strTitle,p + 1)
endif
wait window strTitle
```

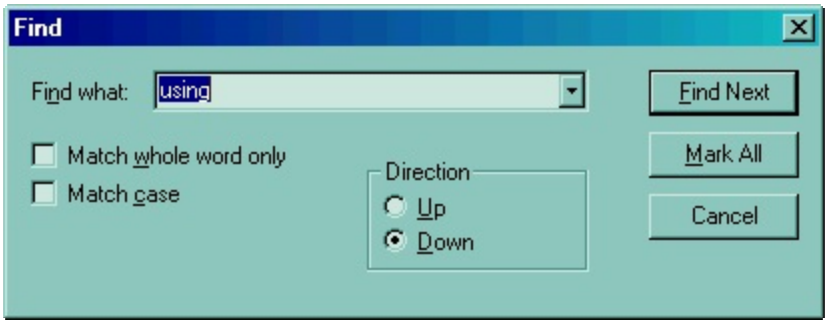
method Edit.Find (FindWhat as String, FindOption as FindOptionEnum)

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
#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 wMsg As Long, ByVal wParam As Long, lParam As Any) As Long
Private Const WM_COMMAND = &H111

Private Sub Command1_Click()
   Edit1.SetFocus
   SendMessage Edit1.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_edit.SetFocus();
   m_edit.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 wMsg 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 AxEdit1
      .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 wMsg, int wParam, int lParam);

private void button1_Click(object sender, EventArgs e)
{
   axEdit1.Focus();
   SendMessage(axEdit1.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.Edit1.Object
   SendMessage( .hWnd, 273, BITLSHIFT(57636, 16), 0)
endwith
```

property Edit.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 Edit.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 LineNumberFont property to specify the font for the control's line numbers bar. 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 LineHeight property specifies an expression that determines the height of the line within the editor.

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 Edit1
   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_edit.GetFont();
font.SetName( "Tahoma" );
m_edit.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 AxEdit1
   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);
axEdit1.Font = font;
axEdit1.CtlRefresh();
```

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

```
with thisform.Edit1.Object
   .Font.Name = "Tahoma"
   .Refresh()
endwith
```

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

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

property Edit.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 BackColorLine property to specify the line's background color. Use the ForeColorLine property to specify the line's foreground color. If the control is read-only (Locked property is True), the BackColorLockedLine property indicates the control's background color, if it is not zero. If the control is read-only (Locked property is True), the ForeColorLockedLine property indicates the control's foreground color, if it is not zero.

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

```
With Edit1
   .ForeColor = ColorConstants.vbBlack
End With
```

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

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

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

```
With AxEdit1
   .ForeColor = Color.Black
End With
```

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

```
axEdit1.ForeColor = Color.Black;
```

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

```
With thisform.Edit1.Object
   .ForeColor = RGB(0,0,0)
endwith
```

property Edit.ForeColorLine(Index as Long) as Color

Specifies the line's foreground color.

Type	Description
Index as Long	A long expression that indicates the index of line being colored.
Color	A color expression that indicates the line's background color.

Use the ForeColorLine property to change the line's foreground color. Use the ClearForeColorLine method to clear the line's foreground color. Use the BackColorLine property to specify the line's background color. Use the BackColor property to specify the control's background color. Use the ForeColor property to specify the control's foreground color.

property Edit.ForeColorLockedLine as Color

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

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

Use the ForeColorLockedLine property to specify the foreground color for locked lines. Use the LockedLine property to lock a specific line. Use the Locked property to make read-only the entire control. Use the BackColorLockedLine property to specify the background color for locked lines.

property Edit.FormatNumbers as String

Specifies the HTML format that's applied to numbers.

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

By default, the FormatNumbers property is "<fgcolor=0000FF> </fgcolor>". By default the numbers get colored in blue. 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. Use the AddKeyword method to add new keywords to the control. Use the AddExpression method to add new expressions to the control. Use the AddWild method to add expression that may contain wild character like "*" or "?".

For instance the following VB sample bolds the numbers in the control:

```
With Edit1
    .FormatNumbers = "<b> </b>"
End With
```

Important to note is that the " " contains an empty space character, else the format for numbers in the control will not be updated.

The following C++ sample bolds the numbers in the control:

```
m_edit.SetFormatNumbers( "<b> </b>" );
```

The following VB.NET sample bolds the numbers in the control:

```
With AxEdit1
  .FormatNumbers = "<b> </b>"
End With
```

The following C# sample bolds the numbers in the control:

```
axEdit1.FormatNumbers = "<b> </b>";
```

The following VFP sample bolds the numbers in the control:

```
With thisform.Edit1
   .FormatNumbers = "<b> </b>"
endwith
```

method Edit.GetSelection (SelStartLine as Variant, SelStartPos as Variant, SelEndLine as Variant, SelEndPos as Variant)

Retrieves the coordinates of the selection.

Type	Description
SelStartLine as Variant	A long expression that indicates the index of the line where the selection starts.
SelStartPos as Variant	A long expression that indicates the index of the character in the line where the selection starts.
SelEndLine as Variant	A long expression that indicates the index of the line where the selection ends.
SelEndPos as Variant	A long expression that indicates the index of the character in the line where the selection ends.

Use the GetSelection method to retrieve the coordinates of the selected text. Use the SelLenght property to return or set the number of characters selected. Use the SetSelection method to select a text giving start and end coordinates. The SelText property returns or sets the string containing the currently selected text; consists of a zero-length string ("") if no characters are selected. Use the CaretLine and CaretPos coordinates to determine the coordinates of the caret/cursor in the control. The control fires SelChange event when selection is changed or when the cursor is moved.

The following VB sample prints the coordinates of the selection when the user changes the selection:

```
Private Sub Edit1_SelChange()
   Dim sl, sp, el, ep
   With Edit1
      .GetSelection sl, sp, el, ep
   End With
   Debug.Print "The selection starts at (" & sl & "," & sp & ") and ends at (" & el & "," & ep
& ")"
End Sub
```

The following VFP sample prints the coordinates of the selection when the user changes the selection:

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

local SelStartLine, SelStartPos, SelEndLine, SelEndPos
```

```
SelStartLine = 0
SelStartPos = 0
SelEndLine = 0
SelEndPos = 0

With thisform.Edit1
	.GetSelection( @SelStartLine, @SelStartPos , @SelEndLine, @SelEndPos)
EndWith

wait window nowait Str(SelStartLine) + Str(SelStartPos) + Str(SelEndLine) + Str(SelEndPos)
```

The following C++ sample prints the coordinates of the selection when the user changes the selection:

```
static long V2I( VARIANT* pv, long nDefault = 0 )
{
   if ( pv )
   {
      if ( pv->vt == VT_ERROR )
         return nDefault;

      COleVariant vt;
      vt.ChangeType( VT_I4, pv );
      return V_I4( &vt );
   }
   return nDefault;
}

void OnSelChangeEdit1()
{
   COleVariant vtSelStartLine, vtSelStartPos, vtSelEndLine, vtSelEndPos;
   m_edit.GetSelection( &vtSelStartLine, &vtSelStartPos, &vtSelEndLine, &vtSelEndPos );

   TCHAR szOutput[1024];
   wsprintf( szOutput, "The selection starts at (%i,%i) and ends at (%i,%i)\n",
V2I(&vtSelStartLine), V2I( &vtSelStartPos ), V2I(&vtSelEndLine), V2I( &vtSelEndPos ) );
   OutputDebugString( szOutput );
```

```
}
```

The following VB.NET sample prints the coordinates of the selection:

```
Private Sub AxEdit1_SelChange(ByVal sender As Object, ByVal e As System.EventArgs)
Handles AxEdit1.SelChange
   With AxEdit1
      Dim sl As Object = 0, sp As Object = 0, el As Object = 0, ep As Object = 0
      .GetSelection(sl, sp, el, ep)
      Debug.WriteLine("Selection: " & sl.ToString() & " " & sp.ToString() & " " & el.ToString()
& " " & ep.ToString())
   End With
End Sub
```

The following C# sample prints the coordinates of the selection:

```
private void axEdit1_SelChange(object sender, EventArgs e)
{
   object sl = 0, sp = 0, el = 0, ep = 0;
   axEdit1.GetSelection(out sl, out sp, out el, out ep);
   System.Diagnostics.Debug.WriteLine("Selection: " + sl.ToString() + " " + sp.ToString() + "
" + el.ToString() + " " + ep.ToString());
}
```

The following VFP sample prints the coordinates of the selection :

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

with thisform.Edit1.Object
   local sl, sp, el, ep
   sl = 0
   sp = 0
   el = 0
   ep = 0
   .GetSelection( @sl, @sp, @el, @ep )
endwith
```

property Edit.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 SelForeColor and SelBackColor properties to define the colors used to paint the selection. Use the SelStart, SelLength and SelText properties to access the selection. Use the GetSelection method to determine the coordinates of the selected text. Use the SetSelection method to select a text giving the end and start coordinates. 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 Edit.HTMLText as String

Specifies the control's HTML text.

Type	Description
String	A string expression that specifies the control's text in MS HTML format.

Use the HTMLText property to retrieve the MS HTML text form the control's text. Use the Text property to specify the control's text. Use the Load/LoadFromUnicode method to load the control's text from a file. Use the Save/SaveAsUnicode method to save the control's text to a file. Use the InsertText method to insert programmatically new lines to the control. Use the DeleteLine method to delete programmatically a line based on its index. The Count property counts the lines in the control. Use the SelText property to insert text at cursor position. Use the CaretLine property to specify the index of the current line.

property Edit.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 Edit.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.

property Edit.IgnorePrefixInExpression(Expression as String) as String

Specifies a value that indicates the prefix that's ignored in the expression.

Type	Description
Expression as String	A string expression that indicates the expression being accessed. The Expression parameter must not include the HTML tags like they are passed to the AddExpression method. For instance, if we have the /* expression the name of the expression is /* not /*
String	A string expression that indicates the list of prefixes being ignored in the expression.

Use the IgnorePrefixInExpression property to define the prefixes being ignored by an expression. Use the AddExpression property to add new expression to your control.

For instance if we defined an expression for strings like follows:

```
With Edit1
    .AddExpression "<fgcolor=800000><b>""</b></fgcolor>", _
      "<b><fgcolor=FF0000> </fgcolor></b>", _
      "<fgcolor=800000><b>""</b></fgcolor>", _
      True
End With
```

In this case we have defined the expression ".

So, if the user types in the control the text:

'This is a string "this is a quote \" bla ... bla \" "'

the control displays the line like follows:

'This is a string **"this is a quote \"** bla ... bla **" ""**

, but we want to ignore \" expressions.

In this case we need to use the following code:

```
With Edit1
    .AddExpression "<fgcolor=800000><b>""</b></fgcolor>", _
      "<b><fgcolor=FF0000> </fgcolor></b>", _
```

```
        "<fgcolor=800000><b>""</b></fgcolor>", _
        True
    .IgnorePrefixInExpression("""") = "\"
End With
```

In this case, if the user types:

'This is a string "this is a quote \" bla ... bla \" "'

the control displays:

'This is a string **"this is a quote \" bla ... bla \""**'

, and this is what we need.

The IgnorePrefixInExpression property is cumulative, so you can add multiple prefixes to be ignored for an expression like in the following sample:

```
With Edit1
   .AddExpression "<fgcolor=800000><b>""</b></fgcolor>", _
      "<b><fgcolor=FF0000> </fgcolor></b>", _
      "<fgcolor=800000><b>""</b></fgcolor>", _
      True
   .IgnorePrefixInExpression("""") = "\"
   .IgnorePrefixInExpression("""") = """"
End With
```

Passing an empty string to the IgnorePrefixInExpression property clears the collection of the prefixes for the expression, so the .IgnorePrefixInExpression("""") = "" clears all prefixes previously defined for an expression.

The following C++ sample adds an expression to highlight the text between " (quote) characters:

```
COleVariant vtMissing; vtMissing.vt = VT_ERROR;
m_edit.AddExpression( "<fgcolor=800000><b>\"</b></fgcolor>", "<b>
<fgcolor=FF0000> </fgcolor></b>", "<fgcolor=800000><b>\"</b></fgcolor>",
vtMissing, vtMissing );
m_edit.SetIgnorePrefixInExpression( "\"", "\\" );
m_edit.SetIgnorePrefixInExpression( "\"", "\"" );
m_edit.Refresh();
```

The following VB.NET sample adds an expression to highlight the text between " (quote) characters:

```
With AxEdit1
   .AddExpression("<fgcolor=800000><b>""</b></fgcolor>", "<b><fgcolor=FF0000>
</fgcolor></b>", "<fgcolor=800000><b>""</b></fgcolor>")
   .set_IgnorePrefixInExpression("""", "\")
   .set_IgnorePrefixInExpression("""", """")
   .CtlRefresh()
End With
```

The following C# sample adds an expression to highlight the text between " (quote) characters:

```
axEdit1.AddExpression("<fgcolor=800000><b>\"</b></fgcolor>", "<b>
<fgcolor=FF0000> </fgcolor></b>", "<fgcolor=800000><b>\"</b></fgcolor>");
axEdit1.set_IgnorePrefixInExpression("\"", "\\");
axEdit1.set_IgnorePrefixInExpression("\"", "\"");
axEdit1.CtlRefresh();
```

The following VFP sample adds an expression to highlight the text between " (quote) characters:

```
With thisform.Edit1.Object
   .AddExpression("<fgcolor=800000><b>" + chr(34) + "</b></fgcolor>", "<b>
<fgcolor=FF0000> </fgcolor></b>", "<fgcolor=800000><b>"+chr(34)+"</b>
</fgcolor>")
   .IgnorePrefixInExpression(chr(34)) = chr(92)
   .IgnorePrefixInExpression(chr(34)) = chr(34)
   .Refresh
EndWith
```

method Edit.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
Handle as Variant	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 llVal 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(

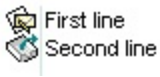
(LONGLONG)hImageList)), where hImageList is of HIMAGELIST type. The GetSafeHandle() method of the CImageList gets the HIMAGELIST handle (long, loads icon from HIMAGELIST type)

The control provides an image list window, that's displayed at design time. The ImageSize property defines the size (width/height) of the icons within the control's Images collection. 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 ReplaceIcon method to change the Images collection. The Images collection is 1 based. The control's context list may contain icons. Use the BookmarkImageLine property to assign an icon to the control's bookmarks header. Use the BookmarkImage property to display the control's bookmarks header.

The following VB sample loads two icons using BASE64 encoded strings:

```
With Edit1
   .BookmarkWidth = 18
   .Images
("gBJJgBggAAwAAgACEKAD/hz/EMNh8TIRNGwAjEZAEXjAojJAjIgjIBAEijUlk8plUrlktl0vmEx

   .Images
("gBJJgBggAAkGAAQhIAf8Nf4hhkOiRCJo2AEXjAAi0XFEYIEYhUXAIAEEZi8hk0plUrlktl0vmEx

   .InsertText ("First line")
   .InsertText ("Second line")
   .BookmarkImageLine(1) = 1
   .BookmarkImageLine(2) = 2
End With
```

If you run the sample you get:



The following sample shows how to replace the entire list of icons, using a Microsoft Image List control (ImageList1):

```
Edit1.Images ImageList1.hImageList
```

The following C++ sample loads two icons using BASE64 encoded strings:

```
m_edit.SetBookmarkWidth( 18 );
m_edit.Images(
COleVariant("gBJJgBggAAwAAgACEKAD/hz/EMNh8TIRNGwAjEZAEXjAojJAjIgjIBAEijUIk8plU
 );
m_edit.Images(
COleVariant("gBJJgBggAAkGAAQhIAf8Nf4hhkOiRCJo2AEXjAAi0XFEYIEYhUXAIAEEZi8hk0pl
 );
m_edit.InsertText("First line", vtMissing );
m_edit.InsertText("Second line", vtMissing );
m_edit.SetBookmarkImageLine( 1, 1 );
m_edit.SetBookmarkImageLine( 2, 2 );
```

The following VB.NET sample loads two icons using BASE64 encoded strings:

```
With AxEdit1
   .BookmarkWidth = 18

.Images("gBJJgBggAAwAAgACEKAD/hz/EMNh8TIRNGwAjEZAEXjAojJAjIgjIBAEijUIk8plUrlkt



.Images("gBJJgBggAAkGAAQhIAf8Nf4hhkOiRCJo2AEXjAAi0XFEYIEYhUXAIAEEZi8hk0plUrlk

   .InsertText("First line")
   .InsertText("Second line")
   .set_BookmarkImageLine(1, 1)
   .set_BookmarkImageLine(2, 2)
End With
```

The following C# sample loads two icons using BASE64 encoded strings:

```
axEdit1.BookmarkWidth = 18;
axEdit1.Images("gBJJgBggAAwAAgACEKAD/hz/EMNh8TIRNGwAjEZAEXjAojJAjIgjIBAEijUIk8

axEdit1.Images("gBJJgBggAAkGAAQhIAf8Nf4hhkOiRCJo2AEXjAAi0XFEYIEYhUXAIAEEZi8hk

axEdit1.InsertText("First line");
```

```
axEdit1.InsertText("Second line");
axEdit1.set_BookmarkImageLine(1, 1);
axEdit1.set_BookmarkImageLine(2, 2);
```

The following VFP sample loads two icons using BASE64 encoded strings:

```
With thisform.Edit1.Object
   local s
   .BookmarkWidth = 18
   s =
"gBJJgBggAAkGAAQhIAf8Nf4hhkOiRCJo2AEXjAAi0XFEYIEYhUXAIAEEZi8hk0plUrlktl0vmExr

   s = s +
"2PpWDutSxVhqGIqGVs+QzmdneYu0XxVr0GLvuZvOe1Wrmul1mv2GP0uzzex223n+00O43

   .Images (s)
   s =
"gBJJgBggAAkGAAQhIAf8Nf4hhkOiRCJo2AEXjAAi0XFEYIEYhUXAIAEEZi8hk0plUrlktl0vmExr

   s = s +
"tvwFCpmDsGToVxp+MzWbs2Ov+Vtk/t8XymUx2c1GppOOw1Ty1T0WismmtlM1W33FEz+

   s = s + "MgGlUOJMYAEwiIR+BylJ/j3Ey3JMgIA="
   .Images (s)
   .InsertText ("First line")
   .InsertText ("Second line")
   .BookmarkImageLine(1) = 1
   .BookmarkImageLine(2) = 2
EndWith
```

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:

Edit1
Images
Drag here an executable file,
a dynamic-link library, an
icon file or a regular file

property Edit.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 Edit.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 Edit.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 IndentSel property to indent the selection by code. Use the UseTabKey property to specify whether the control handles the TAB key.

method Edit.IndentSel (Forward as Boolean)

Indents the selected text.

Type	Description
Forward as Boolean	A boolean expression that indicates whether the selection is indented forward or backward.

Use the IndentSel method to indent the selected text by code. The action is similar to pressing the TAB or SHIFT + TAB key while the control has the focus, and you have selected multiple lines. Use the IndentOnTab property to disable indentation of the selection when user presses the TAB key.

The following VB sample indents the selection to the right when the user press the F1 key:

```
Private Sub Edit1_KeyDown(KeyCode As Integer, Shift As Integer)
   If KeyCode = KeyCodeConstants.vbKeyF1 Then
      Edit1.IndentSel True
   End If
End Sub
```

The following C++ sample indents the selection to the right when the user press the F1 key:

```
void OnKeyDownEdit1(short FAR* KeyCode, short Shift)
{
   if ( *KeyCode == VK_F11 )
      m_edit.IndentSel( TRUE );
}
```

The following VB.NET sample indents the selection to the right when the user press the F1 key:

```
Private Sub AxEdit1_KeyDownEvent(ByVal sender As Object, ByVal e As
AxEXEDITLib._IEditEvents_KeyDownEvent) Handles AxEdit1.KeyDownEvent
   If (Convert.ToInt32(e.keyCode) = Convert.ToInt32(Keys.F11)) Then
      AxEdit1.IndentSel(True)
   End If
End Sub
```

The following C# sample indents the selection to the right when the user press the F1 key:

```
private void axEdit1_KeyDownEvent(object sender,
AxEXEDITLib._IEditEvents_KeyDownEvent e)
{
   if (Convert.ToInt32(e.keyCode) == Convert.ToInt32(Keys.F11))
      axEdit1.IndentSel(true);
}
```

The following VFP sample indents the selection to the right when the user press the F1 key:

```
*** ActiveX Control Event ***
LPARAMETERS keycode, shift

if ( keycode = 122 )
   thisform.Edit1.IndentSel(.t.)
endif
```

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

Inserts text to control.

Type	Description
Text as String	A string expression being inserted.
Index as Variant	A long expression that defines the index of line where the text follows to be inserted.

Use the InsertText method to add programmatically new lines to the control. Use the DeleteLine method to delete a particular line. Use the Text property to clear the control's text. If the Text contains multiple cr lf sequences multiple lines are inserted. Checks the control's MultiLine property when multiple lines are accepted. Use the SelText property to insert text at cursor position.

The following VB sample inserts a line at the top:

```
Edit1.InsertText "top line" + vbCrLf, 1
```

The following C++ sample inserts a line at the top:

```
m_edit.InsertText( "top line\r\n", COleVariant( long(1) ) );
```

The following VB.NET sample inserts a line at the top:

```
AxEdit1.InsertText("top line" + vbCrLf, 1)
```

The following C# sample inserts a line at the top:

```
axEdit1.InsertText("top line\r\n", 1);
```

The following VFP sample inserts a line at the top:

```
with thisform.Edit1
   .InsertText("top line" + chr(13) + chr(1), 1)
endwith
```

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

The supported unary boolean operators are:

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

The supported binary boolean operators are:

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

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

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

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

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

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

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

operator is

variable := expression

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

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

=: variable

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

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

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

expression ? true_part : false_part

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

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

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

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

, where the 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 that iif (immediate if operator) alterative.

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

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

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

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

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

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

The supported conversion unary operators are:

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

  Here's few predefined types:

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

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

Other known operators for numbers are:

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

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

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

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

Other known operators for strings are:

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

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

- **upper** (unary operator) returns a string expression in uppercase letters. For instance, the *upper("mihai")* returns "MIHAI"
- **proper** (unary operator) returns from a character expression a string capitalized as appropriate for proper names. For instance, the *proper("mihai")* returns "Mihai"
- **ltrim** (unary operator) removes spaces on the left side of a string. For instance, the *ltrim(" mihai")* returns "mihai"
- **rtrim** (unary operator) removes spaces on the right side of a string. For instance, the *rtrim("mihai ")* returns "mihai"
- **trim** (unary operator) removes spaces on both sides of a string. For instance, the *trim(" mihai ")* returns "mihai"
- **reverse** (unary operator) reverses the order of the characters in the string a. For instance, the *reverse("Mihai")* returns "iahiM"
- **startwith** (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 Edit.LineNumberBackColor as Color

Specifies the background color of the line number bar.

Type	Description
Color	A color expression that specifies the background color of the line number bar.

Use the LineNumberBackColor and LineNumberForeColor properties to customize colors for the control's line numbers bar. Use the LineNumberWidth property to show the line numbers header bar. Use the BackColor property to specify the control's background color.

property Edit.LineNumberBorderColor as Color

Specifies the color for line number's border.

Type	Description
Color	A color expression that indicates the line numbers header bar's border

Use the LineNumberBackColor and LineNumberForeColor properties to customize colors for the control's line numbers bar. Use the BackColor property to specify the control's background color.

property Edit.LineNumberFont as IFontDisp

Retrieves or sets the font of the line numbers bar.

Type	Description
IFontDisp	A font object being used to paint the line numbers.

Use the LineNumberFont property to specify the font for the line numbers bar. Use the LineNumberWidth property to display the control's line numbers bar. Use the Font property to assign a font for the control's lines. Use the Refresh method to update the control's visible area.

The following VB sample assigns a different font for the control's line numbers bar:

```
With Edit1
   .LineNumberWidth = 32
   With .LineNumberFont
      .Name = "Tahoma"
      .Bold = True
   End With
   .Refresh
End With
```

The following C++ sample assigns a different font for the control's line numbers bar:

```
m_edit.SetLineNumberWidth( 32 );
COleFont font = m_edit.GetLineNumberFont();
font.SetName( "Tahoma" );
font.SetBold( TRUE );
m_edit.Refresh();
```

The following VB.NET sample assigns a different font for the control's line numbers bar:

```
With AxEdit1
   .LineNumberWidth = 32
   .LineNumberFont = New System.Drawing.Font("Tahoma", 10, FontStyle.Bold)
   .CtlRefresh()
End With
```

The following C# sample assigns a different font for the control's line numbers bar:

```
axEdit1.LineNumberWidth = 32;
axEdit1.LineNumberFont = new System.Drawing.Font("Tahoma", 10, FontStyle.Bold);
axEdit1.CtlRefresh();
```

The following VFP sample assigns a different font for the control's line numbers bar:

```
With thisform.Edit1.Object
   .LineNumberWidth = 32
   With .LineNumberFont
      .Name = "Tahoma"
      .Bold = .t.
   EndWith
   .Refresh
EndWith
```

property Edit.LineNumberForeColor as Color

Specifies the foreground color of the line number bar.

Type	Description
Color	A color expression that specifies the foreground color of the line number bar.

Use the LineNumberBackColor and LineNumberForeColor properties to customize colors for the control's line number bar. Use the LineNumberWidth property to show the line numbers header bar. Use the ForeColor property to specify the control's foreground color.

property Edit.LineNumberWidth as Long

Specifies the width of the border where the that contains the line numbers.

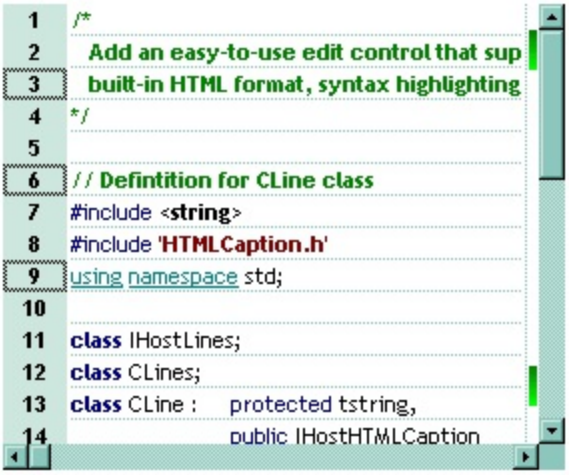
Type	Description
Long	A long expression that defines the width in pixels for the control's line number border.

Use the LineNumberWidth property to show the control's line number width. Use the BookmarkWidth property to show the control's bookmark border. By default, the LineNumberWidth property is 0. If the control's LineNumberWidth is zero the control's line number border is hidden. If the LineNumberWidth property is 0 the control displays no line numbers border. If the LineNumberWidth property is less than 0 the control sets the width of the line numbers border automatically. Use the LineNumberBackColor and LineNumberForeColor properties to customize colors for the control's line numbers bar. Use the LineNumberFont property to specify the font for the control's line numbers bar.

The following VB sample displays the line bookmarks and line numbers on the same column, and the column is automatically sized:

```
With Edit1
    .LineNumberWidth = -1
    .BookmarkWidth = -1
End With
```

The following screen show shows the control's line numbers header bar:



The following C++ sample displays the line bookmarks and line numbers on the same column, and the column is automatically sized:

```
m_edit.SetLineNumberWidth( -1 );
m_edit.SetBookmarkWidth( -1 );
```

The following VB.NET sample displays the line bookmarks and line numbers on the same column, and the column is automatically sized:

```
With AxEdit1
    .LineNumberWidth = -1
    .BookmarkWidth = -1
End With
```

The following C# sample displays the line bookmarks and line numbers on the same column, and the column is automatically sized:

```
axEdit1.LineNumberWidth = -1;
axEdit1.BookmarkWidth = -1;
```

The following VFP sample displays the line bookmarks and line numbers on the same column, and the column is automatically sized:

```
With thisform.Edit1
  .LineNumberWidth = -1
  .BookmarkWidth = -1
EndWith
```

method Edit.Load (FileName as String)

Loads a file.

Type	Description
FileName as String	A string expression that specifies the file being loaded. If the FileName parameter is empty, the control uses the FileName property to specify the file being loaded.

Use the Load method to load the control's text from a file (ANSI file). Use the LoadFromUnicode method to load UNICODE text. If the file doesn't exist no errors occur. Use the FileName property to get the name of the file being loaded or saved. Use the Text property to access the control's text. Use the Save/SaveAsUnicode method to save the control's text to a file. The Load method changes the FileName property. If the AllowMark property is True, the control could load the text slowly, because each line requires to be parsed in order to find the expression to be highlighted on the control's vertical scroll bar, so we would suggest to use this option only before you check the time to load the file using this option.

method Edit.LoadFromUnicode (FileName as String)

Loads from an UNICODE file.

Type	Description
FileName as String	A string expression that specifies the file being loaded. If the FileName parameter is empty, the control uses the FileName property to specify the file being loaded.

Use the LoadFromUnicode method to load UNICODE text. Use the Load method to load the control's text from a file (ANSI file). If the file doesn't exist no errors occur. Use the FileName property to get the name of the file being loaded or saved. Use the Text property to access the control's text. Use the Save/SaveAsUnicode method to save the control's text to a file. The Load method changes the FileName property. If the AllowMark property is True, the control could load the text slowly, because each line requires to be parsed in order to find the expression to be highlighted on the control's vertical scroll bar, so we would suggest to use this option only before you check the time to load the file using this option.

property Edit.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 LockedLine property to lock or unlock a specific line. Use the ShowCaret property to hide the control's caret. If the Locked property is True, the UseTabKey property is automatically False. If the control is read-only (Locked property is True), the BackColorLockedLine property indicates the control's background color, if it is not zero. If the control is read-only (Locked property is True), the ForeColorLockedLine property indicates the control's foreground color, if it is not zero. Use the ApplyColors property to hide the colors inside the control.

property Edit.LockedLine(Index as Long) as Boolean

Specifies whether a line is locked or unlocked.

Type	Description
Index as Long	A long expression that indicates the index of line being locked or unlocked. 1 means the first line, in the text, 2, indicates the second line in the text and so on.
Boolean	A boolean expression that indicates whether the line is locked or unlocked.

Use the LockedLine property to lock or unlock a line. A locked line can't be edited. Use the Locked property to lock the entire control. A locked control is read-only. Use the BackColorLockedLine or ForeColorLockedLine property to specify the background and foreground colors for locked lines. Use the ForeColor property to specify the foreground color for a specific line. Use the BackColorLine property to specify the background color for a locked line.

property Edit.Margin as Long

Defines the distance between text and inner border (border, line-numbers or bookmarks).

Type	Description
Long	A long expression that specifies the distance between text and inner border (border, line-numbers or bookmarks)

Defines the distance between text and inner border (border, line-numbers or bookmarks).

property Edit.MarkColor(Expression as String) as Color

Specifies a color being used to mark the position of the expression, on the control's vertical scroll bar.

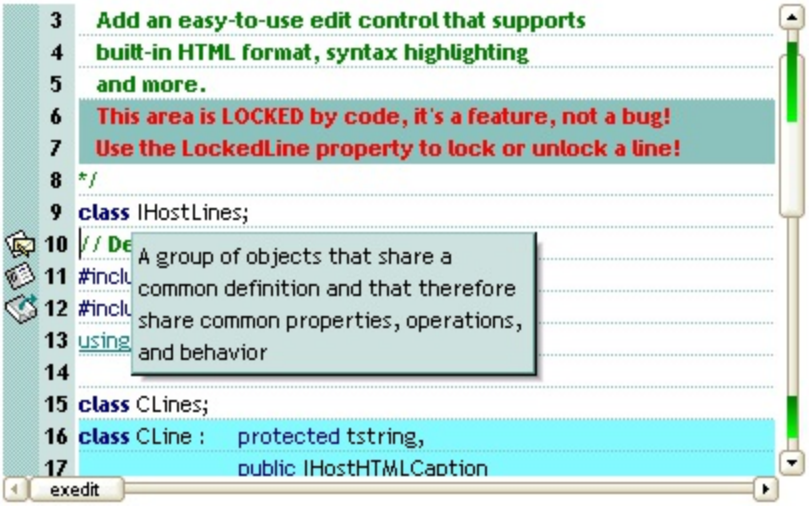
Type	Description
Expression as String	A string expression that indicates the expression being accessed. The Expression parameter must not include the HTML tags like it is passed to the AddExpression or AddKeyword method.
Color	A color expression that defines the marking color on the control's vertical scroll bar.

Use the MarkColor property to specify a marking color on the control's vertical scroll bar for an expression/keyword. The MarkColor property has effect only if the AllowMark property is True. You can add multiple expressions to be marked on the control's vertical scroll bar. Use the AddKeyword to add new keywords to the control's context. Use the AddExpression to add new expressions to the control. Use the Expression = "" to clear all the marking colors. Use the MarkContinueBlocks property to mark multiple lines block.

The following sample defines the // TODO expression, and marks on the control's vertical scroll bar each position of the // TODO expression in the control's text:

```
With Edit1
   .AllowMark = True
   .AddExpression "<fgcolor=000080>// TODO</fgcolor>", "<b><fgcolor=000080>
</fgcolor></b>", ""
   .MarkColor("// TODO") = RGB(0, 0, &H80)
   .Refresh
End With
```

As you can see from the following screen shot, there are lines (on the control's vertical scroll bar) that marks the position for each // TODO expression in the text.



The following C++ sample marks the position of the // TODO expressions on the control's vertical scroll bar:

```
COleVariant vtMissing; vtMissing.vt = VT_ERROR;
m_edit.SetAllowMark( TRUE );
m_edit.AddExpression("<fgcolor=000080>// TODO</fgcolor>", "<fgcolor=000080>
</fgcolor>", "", vtMissing, vtMissing );
m_edit.SetMarkColor("// TODO", RGB(0, 0, 0x80) );
m_edit.Refresh();
```

The following VB.NET sample marks the position of the // TODO expressions on the control's vertical scroll bar:

```
With AxEdit1
   .AllowMark = True
   .LineNumberWidth = -1
   .AddExpression("<fgcolor=000080>// TODO</fgcolor>", "<b><fgcolor=000080>
</fgcolor></b>", "")
   .set_MarkColor("// TODO", ToUInt32(Color.FromArgb(0, 0, &H80)))
   .CtlRefresh()
End With
```

where the ToUInt32 function converts a Color expression to an OLE_COLOR expressions:

```
Shared Function ToUInt32(ByVal c As Color) As UInt32
   Dim i As Long
   i = c.R
   i = i + 256 * c.G
   i = i + 256 * 256 * c.B
```

```
    ToUInt32 = Convert.ToUInt32(i)
End Function
```

The following C# sample marks the position of the // TODO expressions on the control's vertical scroll bar:

```
axEdit1.AllowMark = true;
axEdit1.AddExpression("<fgcolor=000080>// TODO</fgcolor>", "<b>
<fgcolor=000080> </fgcolor></b>", "");
axEdit1.set_MarkColor("// TODO", ToUInt32(Color.FromArgb(0, 0, 0x80)));
axEdit1.CtlRefresh();
```

where the ToUInt32 function converts a Color expression to an OLE_COLOR expressions:

```
private UInt32 ToUInt32(Color c)
{
    long i;
    i = c.R;
    i = i + 256 * c.G;
    i = i + 256 * 256 * c.B;
    return Convert.ToUInt32(i);
}
```

The following VFP sample marks the position of the // TODO expressions on the control's vertical scroll bar:

```
With thisform.Edit1.Object
    .AllowMark = .t.
    .AddExpression("<fgcolor=000080>// TODO</fgcolor>", "<b><fgcolor=000080>
</fgcolor></b>", "")
    .MarkColor("// TODO") = RGB(0, 0, 128)
    .Refresh
EndWith
```

property Edit.MarkContinueBlocks as Boolean

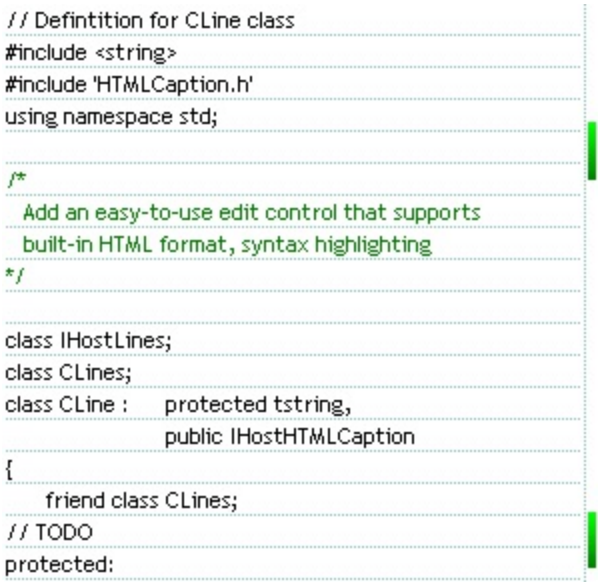
Marks continuously the blocks.

Type	Description
Boolean	A boolean expression that indicates whether the blocks are continuously marked on the control's vertical scroll bar

Use the MarkContinueBlocks property to mark the whole block on the control's vertical scroll bar. By default, the property MarkContinueBlocks property is False. The MarkContinueBlocks property has effect only if the AllowMark property is True, and the control contains expressions added using the MarkColor property. Use the AddExpression method to add new expressions to the control. Use the Refresh method to refresh the control's visible area, when adding expressions on the fly.

The following VB sample adds blocks (the block starts with /* and ends with /*) on the control's vertical scroll bar:

```
With Edit1
   .AllowMark = True
   .MarkContinueBlocks = True
   .AddExpression "<fgcolor=008000>/*</fgcolor>", "<fgcolor=008000> </fgcolor>", "
<fgcolor=008000>*/</fgcolor>", True
   .MarkColor("/*") = RGB(0, 255, 0)
   .MarkColor("*/") = RGB(0, 128, 0)
   .Refresh
End With
```



The following C++ sample adds blocks (the block starts with /* and ends with /*) on the

control's vertical scroll bar:

```
COleVariant vtMissing; vtMissing.vt = VT_ERROR;
m_edit.SetAllowMark( TRUE );
m_edit.SetMarkContinueBlocks( TRUE );
m_edit.AddExpression("<fgcolor=008000>/*</fgcolor>", "<fgcolor=008000>
</fgcolor>", "<fgcolor=008000>*/</fgcolor>", COleVariant( VARIANT_TRUE ), vtMissing
);
m_edit.SetMarkColor("/*", RGB(0, 255, 0) );
m_edit.SetMarkColor("*/", RGB(0, 128, 0) );
m_edit.Refresh();
```

The following VB.NET sample adds blocks (the block starts with /* and ends with /*) on the control's vertical scroll bar:

```
With AxEdit1
   .AllowMark = True
   .MarkContinueBlocks = True
   .AddExpression("<fgcolor=008000>/*</fgcolor>", "<fgcolor=008000> </fgcolor>", "
<fgcolor=008000>*/</fgcolor>", True, Nothing)
   .set_MarkColor("/*", ToUInt32(Color.FromArgb(0, 255, 0)))
   .set_MarkColor("*/", ToUInt32(Color.FromArgb(0, 128, 0)))
   .CtlRefresh()
End With
```

where the ToUInt32 function converts the Color expression to OLE_COLOR expression

```
Shared Function ToUInt32(ByVal c As Color) As UInt32
   Dim i As Long
   i = c.R
   i = i + 256 * c.G
   i = i + 256 * 256 * c.B
   ToUInt32 = Convert.ToUInt32(i)
End Function
```

The following C# sample adds blocks (the block starts with /* and ends with /*) on the control's vertical scroll bar:

```
axEdit1.AllowMark = true;
```

```
axEdit1.MarkContinueBlocks = true;
axEdit1.AddExpression("<fgcolor=008000>/*</fgcolor>", "<fgcolor=008000>
</fgcolor>", "<fgcolor=008000>*/</fgcolor>", true, null);
axEdit1.set_MarkColor("/*", ToUInt32(Color.FromArgb(0, 255, 0)));
axEdit1.set_MarkColor("*/", ToUInt32(Color.FromArgb(0, 128, 0)));
axEdit1.CtlRefresh();
```

where the ToUInt32 function converts the Color expression to OLE_COLOR expression

```
private UInt32 ToUInt32(Color c)
{
   long i;
   i = c.R;
   i = i + 256 * c.G;
   i = i + 256 * 256 * c.B;
   return Convert.ToUInt32(i);
}
```

The following VFP sample adds blocks (the block starts with /* and ends with /*) on the control's vertical scroll bar:

```
With thisform.Edit1.Object
   .AllowMark = .t.
   .MarkContinueBlocks = .t.
   .AddExpression("<fgcolor=008000>/*</fgcolor>", "<fgcolor=008000> </fgcolor>", "
<fgcolor=008000>*/</fgcolor>", .t.)
   .MarkColor("/*") = RGB(0, 255, 0)
   .MarkColor("*/") = RGB(0, 128, 0)
   .Refresh
EndWith
```

property Edit.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. Use the TextLine property to access a particular item when MultiLine property is True.

method Edit.NextBookmark ()

Moves the cursor to the next bookmark.

Type	Description

Use the NextBookmark method to move the cursor to the next bookmark. Use the PrevBookmark method to move the cursor to the prior bookmark. Use the Bookmark property to bookmark a line by code. Use the BookmarkList property to get the list of lines that have bookmarks. Use the CaretLine property to specify the current line.

For instance, the following VB sample moves the cursor to next bookmark when the user presses the F8 key:

```
Private Sub Edit1_KeyDown(KeyCode As Integer, Shift As Integer)
   If vbKeyF8 = KeyCode Then
      Edit1.NextBookmark
   End If
End Sub
```

The following C++ sample moves the cursor to the next bookmark when the user presses the F8 key:

```
void OnKeyDownEdit1(short FAR* KeyCode, short Shift)
{
   if ( *KeyCode == VK_F8 )
      m_edit.NextBookmark();
}
```

The following VB.NET sample moves the cursor to the next bookmark when the user presses the F8 key:

```
Private Sub AxEdit1_KeyDownEvent(ByVal sender As Object, ByVal e As
AxEXEDITLib._IEditEvents_KeyDownEvent) Handles AxEdit1.KeyDownEvent
   If (Convert.ToInt32(e.keyCode) = Convert.ToInt32(Keys.F8)) Then
      AxEdit1.NextBookmark()
   End If
End Sub
```

The following C# sample moves the cursor to the next bookmark when the user presses the F8 key:

```
private void axEdit1_KeyDownEvent(object sender,
AxEXEDITLib._IEditEvents_KeyDownEvent e)
{
   if (Convert.ToInt32(e.keyCode) == Convert.ToInt32(Keys.F8))
      axEdit1.NextBookmark();
}
```

The following VFP sample moves the cursor to the next bookmark when the user presses the F8 key:

```
*** ActiveX Control Event ***
LPARAMETERS keycode, shift

if ( keycode = 119 ) && F8
   thisform.Edit1.NextBookmark()
endif
```

method Edit.OLEDrag ()

Causes a component to initiate an OLE drag/drop operation.

Type	Description

Only for internal use.

property Edit.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 ExEdit 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 Edit1
   .OLEDropMode = exOLEDropManual
End With
```

```
Private Sub Edit1_OLEStartDrag(ByVal Data As EXEDITLibCtl.IExDataObject, AllowedEffects
As Long)
   Dim s As String
   s = Edit1.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_edit.SetOLEDropMode( 1 );
```

```
void OnOLEStartDragEdit1(LPDISPATCH Data, long FAR* AllowedEffects)
{
   CString s( m_edit.GetSelText() );
   if ( s.GetLength() > 0 )
   {
      *AllowedEffects = 1; /*exOLEDropEffectCopy*/
      if ( EXEDITLib::IExDataObjectPtr spData( Data ) )
```

```
        spData->SetData( COleVariant( s ), COleVariant( long(EXEDITLib::exCFText) ) );

    }
}
```

The C++ sample uses the #import <exedit.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 AxEdit1
    .OLEDropMode = EXEDITLib.exOLEDropModeEnum.exOLEDropManual
End With
```

```
Private Sub AxEdit1_OLEStartDrag(ByVal sender As Object, ByVal e As
AxEXEDITLib._IEditEvents_OLEStartDragEvent) Handles AxEdit1.OLEStartDrag
    Dim s As String = AxEdit1.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:

```
axEdit1.OLEDropMode = EXEDITLib.exOLEDropModeEnum.exOLEDropManual;
```

```
private void axEdit1_OLEStartDrag(object sender,
AxEXEDITLib._IEditEvents_OLEStartDragEvent e)
{
    string s = axEdit1.SelText;
    if (s.Length > 0)
    {
        e.allowedEffects = 1;
        e.data.SetData(s, EXEDITLib.exClipboardFormatEnum.exCFText );
    }
}
```

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

```
with thisform.Edit1
   .OLEDropMode = 1
endwith
```

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

with thisform.Edit1
   local s
   s = .SelText
   if ( len(s) > 0 ) then
      allowedeffects = 1
      data.SetData( s, 1 )
   endif
endwith
```

property Edit.OpenBrackets as String

Specifies the list of opened brackets.

Type	Description
String	A string expression that defines the list of characters the represent an opened bracket.

Use the OpenBrackets and CloseBrackets properties to define the list of available brackets. By default, the OpenBrackets property is "{([<". The control uses the OpenBrackets and CloseBrackets properties to get the brackets available when user presses the CTRL + {. When user presses the CTRL + { and the character at caret position is contained by the OpenBrackets property or by the CloseBrackets the control looks for the opposite bracket.

property Edit.Overtype as Boolean

Specifies whether the control is running in overtype mode.

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

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

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

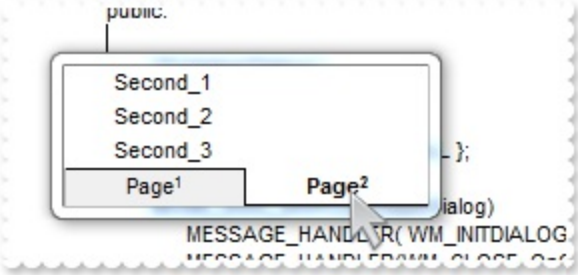
```
Private Sub Edit1_KeyDown(KeyCode As Integer, Shift As Integer)
   If (KeyCode = vbKeyInsert) Then
      KeyCode = 0
   End If
End Sub
```

property Edit.PagesContextItems as String

Specifies the pages (key[:caption]) of the control's sensitive context, separated by comma character.

Type	Description
String	A String expression that specifies the pages (key[:caption]) of the control's sensitive context, separated by comma character. For instance, ":Page<off -4>1</off>,Second:Page<off -4>2</off> " defines two pages one for the context "" with the name Page 1, and the second for context "Second" with the name Page 2.

By default, the PagesContextItems property is empty. The PagesContextItems property specifies the pages (key[:caption]) of the control's sensitive context, separated by comma character. The ActiveContextItems property defines the key of the current context. Use the Context collection to customize the list of keywords when the code completion is enabled. The control can display multiple pages on the control's sensitive context, if multiple key-context are defined. Use the CodeCompletion property to disable the code completion support. By default, the user can display the control's context window by pressing the CTRL + SPACE key combination. Use the ContextKey property to define the key combination to open the control's context window. The ShowContext method shows programmatically the control's context, as the user would press the CTRL + SPACE key.



The caption of the pair (key[:caption]), supports built-HTML as listed:

- ** ... ** displays the text in **bold**
- ** ... ** 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
- **<img>number[:width]</img>** 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.
- **<img>key[:width]</img>** 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 $_{\text{subscript}}$ 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** FFFFFF;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

How can I display more pages on the control's sensitive context?

VBA (MS Access, Excell...)

```
With Edit1
   With .Context()
      .Add "First_1"
      .Add "First_2"
   End With
   With .Context("Second")
      .Add "Second_1"
      .Add "Second_2"
      .Add "Second_3"
   End With
   .ActiveContextItems = "Second"
   .PagesContextItems = ":Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>"
End With
```

VB6

```
With Edit1
   With .Context()
      .Add "First_1"
      .Add "First_2"
   End With
   With .Context("Second")
      .Add "Second_1"
      .Add "Second_2"
      .Add "Second_3"
   End With
   .ActiveContextItems = "Second"
   .PagesContextItems = ":Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>"
End With
```

VB.NET

```
With Exedit1
```

```
    With .get_Context()
        .Add("First_1")
        .Add("First_2")
    End With
    With .get_Context("Second")
        .Add("Second_1")
        .Add("Second_2")
        .Add("Second_3")
    End With
    .ActiveContextItems = "Second"
    .PagesContextItems = ":Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>"
End With
```

VB.NET for /COM

```
With AxEdit1
    With .get_Context()
        .Add("First_1")
        .Add("First_2")
    End With
    With .get_Context("Second")
        .Add("Second_1")
        .Add("Second_2")
        .Add("Second_3")
    End With
    .ActiveContextItems = "Second"
    .PagesContextItems = ":Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>"
End With
```

C++

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

    #import <ExEdit.dll>
```

```
    using namespace EXEDITLib;
*/
EXEDITLib::IEditPtr spEdit1 = GetDlgItem(IDC_EDIT1)->GetControlUnknown();
EXEDITLib::IEditContextPtr var_Context = spEdit1->GetContext(vtMissing);
   var_Context->Add(L"First_1",vtMissing,vtMissing,vtMissing);
   var_Context->Add(L"First_2",vtMissing,vtMissing,vtMissing);
EXEDITLib::IEditContextPtr var_Context1 = spEdit1->GetContext("Second");
   var_Context1->Add(L"Second_1",vtMissing,vtMissing,vtMissing);
   var_Context1->Add(L"Second_2",vtMissing,vtMissing,vtMissing);
   var_Context1->Add(L"Second_3",vtMissing,vtMissing,vtMissing);
spEdit1->PutActiveContextItems(L"Second");
spEdit1->PutPagesContextItems(L":Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>");
```

C++ Builder

```
Exeditlib_tlb::IEditContextPtr var_Context = Edit1->Context[TNoParam()];
   var_Context->Add(L"First_1",TNoParam(),TNoParam(),TNoParam());
   var_Context->Add(L"First_2",TNoParam(),TNoParam(),TNoParam());
Exeditlib_tlb::IEditContextPtr var_Context1 = Edit1->Context[TVariant("Second")];
   var_Context1->Add(L"Second_1",TNoParam(),TNoParam(),TNoParam());
   var_Context1->Add(L"Second_2",TNoParam(),TNoParam(),TNoParam());
   var_Context1->Add(L"Second_3",TNoParam(),TNoParam(),TNoParam());
Edit1->ActiveContextItems = L"Second";
Edit1->PagesContextItems = L":Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>";
```

C#

```
exontrol.EXEDITLib.Context var_Context = exedit1.get_Context(null);
   var_Context.Add("First_1",null,null,null);
   var_Context.Add("First_2",null,null,null);
exontrol.EXEDITLib.Context var_Context1 = exedit1.get_Context("Second");
   var_Context1.Add("Second_1",null,null,null);
   var_Context1.Add("Second_2",null,null,null);
   var_Context1.Add("Second_3",null,null,null);
```

```
exedit1.ActiveContextItems = "Second";
exedit1.PagesContextItems = ":Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>";
```

JScript/JavaScript

```
<BODY onload="Init()">
<OBJECT CLASSID="clsid:39136531-DD0F-4281-B445-E36FC2CDDBC5" id="Edit1">
</OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
   var var_Context = Edit1.Context(null);
      var_Context.Add("First_1",null,null,null);
      var_Context.Add("First_2",null,null,null);
   var var_Context1 = Edit1.Context("Second");
      var_Context1.Add("Second_1",null,null,null);
      var_Context1.Add("Second_2",null,null,null);
      var_Context1.Add("Second_3",null,null,null);
   Edit1.ActiveContextItems = "Second";
   Edit1.PagesContextItems = ":Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>";
}
</SCRIPT>
</BODY>
```

VBScript

```
<BODY onload="Init()">
<OBJECT CLASSID="clsid:39136531-DD0F-4281-B445-E36FC2CDDBC5" id="Edit1">
</OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
   With Edit1
```

```
        With .Context()
            .Add "First_1"
            .Add "First_2"
        End With
        With .Context("Second")
            .Add "Second_1"
            .Add "Second_2"
            .Add "Second_3"
        End With
        .ActiveContextItems = "Second"
        .PagesContextItems = ":Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>"
    End With
End Function
</SCRIPT>
</BODY>
```

C# for /COM

```
EXEDITLib.Context var_Context = axEdit1.get_Context(null);
    var_Context.Add("First_1",null,null,null);
    var_Context.Add("First_2",null,null,null);
EXEDITLib.Context var_Context1 = axEdit1.get_Context("Second");
    var_Context1.Add("Second_1",null,null,null);
    var_Context1.Add("Second_2",null,null,null);
    var_Context1.Add("Second_3",null,null,null);
axEdit1.ActiveContextItems = "Second";
axEdit1.PagesContextItems = ":Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>";
```

X++ (Dynamics Ax 2009)

```
public void init()
{
    COM com_Context,com_Context1;
    anytype var_Context,var_Context1;
```

```
    ;

    super();

    var_Context = exedit1.Context(); com_Context = var_Context;
        com_Context.Add("First_1");
        com_Context.Add("First_2");
    var_Context1 = exedit1.Context("Second"); com_Context1 = var_Context1;
        com_Context1.Add("Second_1");
        com_Context1.Add("Second_2");
        com_Context1.Add("Second_3");
    exedit1.ActiveContextItems("Second");
    exedit1.PagesContextItems(":Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>");
}
```

Delphi 8 (.NET only)

```
with AxEdit1 do
begin
    with get_Context(Nil) do
    begin
        Add('First_1',Nil,Nil,Nil);
        Add('First_2',Nil,Nil,Nil);
    end;
    with get_Context('Second') do
    begin
        Add('Second_1',Nil,Nil,Nil);
        Add('Second_2',Nil,Nil,Nil);
        Add('Second_3',Nil,Nil,Nil);
    end;
    ActiveContextItems := 'Second';
    PagesContextItems := ':Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>';
end
```

Delphi (standard)

```
with Edit1 do
begin
   with Context[Null] do
   begin
      Add('First_1',Null,Null,Null);
      Add('First_2',Null,Null,Null);
   end;
   with Context['Second'] do
   begin
      Add('Second_1',Null,Null,Null);
      Add('Second_2',Null,Null,Null);
      Add('Second_3',Null,Null,Null);
   end;
   ActiveContextItems := 'Second';
   PagesContextItems := ':Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>';
end
```

VFP

```
with thisform.Edit1
   with .Context()
      .Add("First_1")
      .Add("First_2")
   endwith
   with .Context("Second")
      .Add("Second_1")
      .Add("Second_2")
      .Add("Second_3")
   endwith
   .ActiveContextItems = "Second"
   .PagesContextItems = ":Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>"
endwith
```

dBASE Plus

```
local oEdit,var_Context,var_Context1
```

```
oEdit = form.EXEDITACTIVEXCONTROL1.nativeObject
var_Context = oEdit.Context()
   var_Context.Add("First_1")
   var_Context.Add("First_2")
var_Context1 = oEdit.Context("Second")
   var_Context1.Add("Second_1")
   var_Context1.Add("Second_2")
   var_Context1.Add("Second_3")
oEdit.ActiveContextItems = "Second"
oEdit.PagesContextItems = ":Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>"
```

XBasic (Alpha Five)

```
Dim oEdit as P
Dim var_Context as P
Dim var_Context1 as P

oEdit = topparent:CONTROL_ACTIVEX1.activex
var_Context = oEdit.Context()
   var_Context.Add("First_1")
   var_Context.Add("First_2")
var_Context1 = oEdit.Context("Second")
   var_Context1.Add("Second_1")
   var_Context1.Add("Second_2")
   var_Context1.Add("Second_3")
oEdit.ActiveContextItems = "Second"
oEdit.PagesContextItems = ":Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>"
```

Visual Objects

```
local var_Context,var_Context1 as IEditContext

var_Context := oDCOCX_Exontrol1:[Context,nil]
```

```
   var_Context:Add("First_1",nil,nil,nil)
   var_Context:Add("First_2",nil,nil,nil)
var_Context1 := oDCOCX_Exontrol1:[Context,"Second"]
   var_Context1:Add("Second_1",nil,nil,nil)
   var_Context1:Add("Second_2",nil,nil,nil)
   var_Context1:Add("Second_3",nil,nil,nil)
oDCOCX_Exontrol1:ActiveContextItems := "Second"
oDCOCX_Exontrol1:PagesContextItems := ":Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>"
```

PowerBuilder

```
OleObject oEdit,var_Context,var_Context1

oEdit = ole_1.Object
var_Context = oEdit.Context()
   var_Context.Add("First_1")
   var_Context.Add("First_2")
var_Context1 = oEdit.Context("Second")
   var_Context1.Add("Second_1")
   var_Context1.Add("Second_2")
   var_Context1.Add("Second_3")
oEdit.ActiveContextItems = "Second"
oEdit.PagesContextItems = ":Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>"
```

Visual DataFlex

```
Procedure OnCreate
   Forward Send OnCreate
   Variant voContext
   Get ComContext Nothing to voContext
   Handle hoContext
   Get Create (RefClass(cComContext)) to hoContext
   Set pvComObject of hoContext to voContext
      Send ComAdd of hoContext "First_1" Nothing Nothing Nothing
```

```
      Send ComAdd of hoContext "First_2" Nothing Nothing Nothing
   Send Destroy to hoContext
   Variant voContext1
   Get ComContext "Second" to voContext1
   Handle hoContext1
   Get Create (RefClass(cComContext)) to hoContext1
   Set pvComObject of hoContext1 to voContext1
      Send ComAdd of hoContext1 "Second_1" Nothing Nothing Nothing
      Send ComAdd of hoContext1 "Second_2" Nothing Nothing Nothing
      Send ComAdd of hoContext1 "Second_3" Nothing Nothing Nothing
   Send Destroy to hoContext1
   Set ComActiveContextItems to "Second"
   Set ComPagesContextItems to ":Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>"
End_Procedure
```

XBase++

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

PROCEDURE Main
   LOCAL oForm
   LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
   LOCAL oEdit
   LOCAL oContext,oContext1

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

   oEdit := XbpActiveXControl():new( oForm:drawingArea )
   oEdit:CLSID  := "Exontrol.Edit.1" /*{39136531-DD0F-4281-B445-
E36FC2CDDBC5}*/
   oEdit:create(,, {10,60},{610,370} )
```

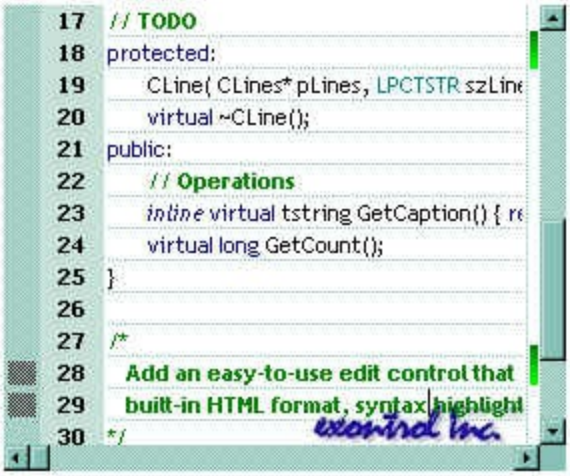
```
      oContext := oEdit:Context()
         oContext:Add("First_1")
         oContext:Add("First_2")
      oContext1 := oEdit:Context("Second")
         oContext1:Add("Second_1")
         oContext1:Add("Second_2")
         oContext1:Add("Second_3")
      oEdit:ActiveContextItems := "Second"
      oEdit:PagesContextItems := ":Page<font ;6><off -4>1</off>
</font>,Second:Page<font ;6><off -4>2</off></font>"

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

property Edit.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. Use the BackColorAlternate property to specify the background color for alternate lines. Use the BackColorLine property to specify the line's background color. Use the ForeColorLine property to specify the line's foreground color.

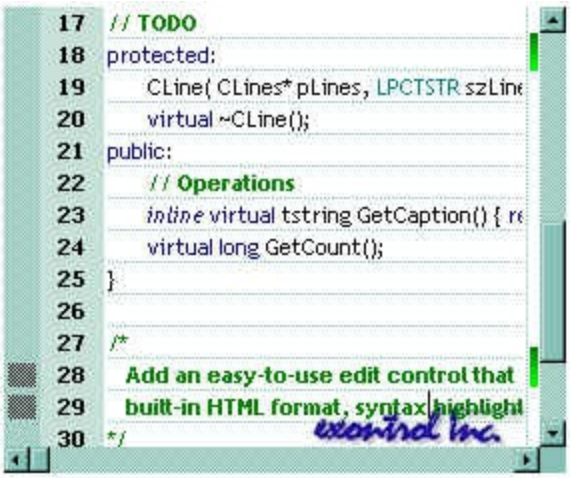


property Edit.PictureDisplay as PictureDisplayEnum

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

Type	Description
PictureDisplayEnum	A PictureDisplayEnum expression that defines how the control's picture is arranged on control's background.

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



method Edit.PrevBookmark ()

Moves the cursor to the prior bookmark.

Type	Description

Use the PrevBookmark method to move the cursor to the prior bookmark. Use the NextBookmark method to move the cursor to the next bookmark. Use the Bookmark property to bookmark a line by code. Use the BookmarkList property to get the list of lines that have bookmarks. Use the CaretLine property to specify the current line.

For instance, the following VB sample moves the cursor to next bookmark when the user presses the F7 key:

```
Private Sub Edit1_KeyDown(KeyCode As Integer, Shift As Integer)
   If vbKeyF7 = KeyCode Then
      Edit1.PrevBookmark
   End If
End Sub
```

The following C++ sample moves the cursor to the next bookmark when the user presses the F7 key:

```
void OnKeyDownEdit1(short FAR* KeyCode, short Shift)
{
   if ( *KeyCode == VK_F7 )
      m_edit.PrevBookmark();
}
```

The following VB.NET sample moves the cursor to the next bookmark when the user presses the F7 key:

```
Private Sub AxEdit1_KeyDownEvent(ByVal sender As Object, ByVal e As
AxEXEDITLib._IEditEvents_KeyDownEvent) Handles AxEdit1.KeyDownEvent
   If (Convert.ToInt32(e.keyCode) = Convert.ToInt32(Keys.F7)) Then
      AxEdit1.PrevBookmark()
   End If
End Sub
```

The following C# sample moves the cursor to the next bookmark when the user presses the F7 key:

```
private void axEdit1_KeyDownEvent(object sender,
AxEXEDITLib._IEditEvents_KeyDownEvent e)
{
   if (Convert.ToInt32(e.keyCode) == Convert.ToInt32(Keys.F7))
      axEdit1.PrevBookmark();
}
```

The following VFP sample moves the cursor to the next bookmark when the user presses the F7 key:

```
*** ActiveX Control Event ***
LPARAMETERS keycode, shift

if ( keycode = 118 ) && F7
   thisform.Edit1.PrevBookmark()
endif
```

property Edit.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 Edit.Redo ()

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

Type	Description

Use the Redo method to redo the last edit-control operation. Use the CanRedo property to specify whether an redo operation is available. Use the Undo method to undo the last edit-control operation. Use the AllowContextMenu property to specify whether the control displays a context menu when user right clicks the control.

method Edit.Refresh ()

Refreshes the control.

Type	Description

The Refresh method repaints the control. If the control's EditType property is exSensitive or exHTML the latest changes to keywords or expressions collection will be updated too.

The following VB sample updates the control:

```
Edit1.Refresh
```

The following C++ sample updates the control:

```
m_edit.Refresh();
```

The following VB.NET sample updates the control:

```
AxEdit1.CtlRefresh()
```

The following C# sample updates the control:

```
axEdit1.CtlRefresh();
```

The following VFP sample updates the control:

```
With thisform.Edit1
   .Object.Refresh
EndWith
```

method Edit.ReplaceIcon ([Icon as Variant], [Index as Variant])

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

Type	Description
Icon as Variant	A Variant expression that specifies the icon to add or insert, as one of the following options: • 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) 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. By default, if the Icon parameter is not specified or is missing, a value of 0 is used.
Index as Variant	A long expression that defines the index of the icon to insert or remove, as follows: • 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 By default, if the Index parameter is not specified or is missing, a value of -1 is used.

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

Use the ReplaceIcon property to add, remove or replace an icon in the control's images

collection. Also, the ReplaceIcon property can clear the images collection. Use the Images method to attach an image list to the control. Use the BookmarkImageLine property to assign an icon to the control's bookmarks header.

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

i = Edit1.ReplaceIcon(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 = Edit1.ReplaceIcon(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:

Edit1.ReplaceIcon 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:

Edit1.ReplaceIcon 0, -1

method Edit.Save (FileName as String)

Saves the control's text to a file.

Type	Description
FileName as String	A string expression that specifies the file being saved. If the FileName parameter is empty, the control uses the FileName property to specify the file being loaded.

Use the Save method to save the control's text to a file. Use the FileName property to get the name of the file being loaded or saved. Use the Text property to access the control's text. The Save method updates the FileName property. Use the Load method to load text from a file. The Save method saves the file as ANSI, while the SaveAsUnicode saves the control's text as UNICODE.

method Edit.SaveAsUnicode (FileName as String)

Saves the control's text to file as UNICODE.

Type	Description
FileName as String	A string expression that specifies the file being saved. If the FileName parameter is empty, the control uses the FileName property to specify the file being loaded.

Use the Save method to save the control's text to a file. Use the FileName property to get the name of the file being loaded or saved. Use the Text property to access the control's text. The Save/SaveAsUnicode method updates the FileName property. Use the Load method to load ANSI text from a file. The Save method saves the file as ANSI, while the SaveAsUnicode saves the control's text as UNICODE.

property Edit.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. Use the Count property to count the number of lines in the control. Use the EnsureVisibleLine method to ensure that the specified line fits the control's client area. If required, the control is scrolled until the line is visible.

property Edit.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 Edit.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 Edit.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 ScrolPartCaption 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 Edit.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 Edit.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 forth 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 forth 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.

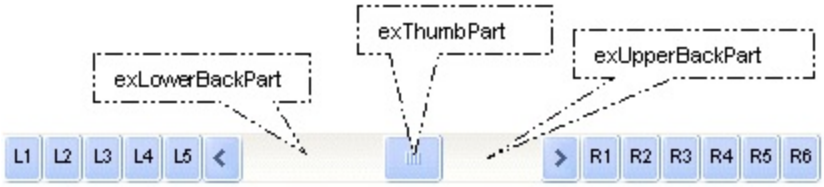
Any other literal between commas is ignored. If duplicate literals are found, the second is ignored, and so on. For instance, "t,l,r" indicates that the left/top and right/bottom buttons are displayed right/bottom after the thumb area.

property Edit.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 displated
String	A String expression that specifies the caption being displayed on the part of the scroll bar.

Use the ScrolPartCaption 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 Edit1
    .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 AxEdit1
   .set_ScrollPartVisible(EXEDITLib.ScrollBarEnum.exVScroll,
EXEDITLib.ScrollPartEnum.exLeftB1Part Or EXEDITLib.ScrollPartEnum.exRightB1Part, True)
   .set_ScrollPartCaption(EXEDITLib.ScrollBarEnum.exVScroll,
EXEDITLib.ScrollPartEnum.exLeftB1Part, "1")
   .set_ScrollPartCaption(EXEDITLib.ScrollBarEnum.exVScroll,
EXEDITLib.ScrollPartEnum.exRightB1Part, "2")
End With
```

The following C# sample adds up and down additional buttons to the control's vertical scroll bar :

```
axEdit1.set_ScrollPartVisible(EXEDITLib.ScrollBarEnum.exVScroll,
EXEDITLib.ScrollPartEnum.exLeftB1Part | EXEDITLib.ScrollPartEnum.exRightB1Part, true);
axEdit1.set_ScrollPartCaption(EXEDITLib.ScrollBarEnum.exVScroll,
EXEDITLib.ScrollPartEnum.exLeftB1Part , "1");
axEdit1.set_ScrollPartCaption(EXEDITLib.ScrollBarEnum.exVScroll,
EXEDITLib.ScrollPartEnum.exRightB1Part, "2");
```

The following C++ sample adds up and down additional buttons to the control's vertical scroll bar :

```
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") );
```

The following VFP sample adds up and down additional buttons to the control's vertical scroll bar :

```
With thisform.Edit1
    .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 Edit.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.

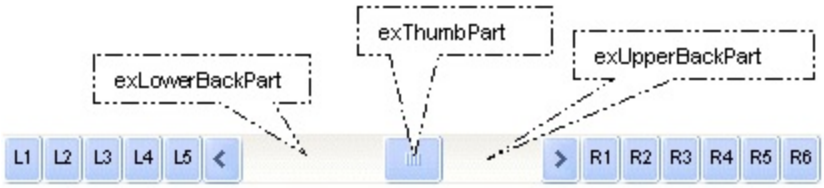
The ScrollPartCaptionAlignment property specifies the alignment of the caption in the part of the scroll bar. By default, the caption is centered. Use the ScrolPartCaption property to specify the caption being displayed on specified part of the scroll bar. Use the ScrollPartVisible property to add or remove buttons/parts in the control's scrollbar.

property Edit.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 ScrolPartCaption 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.

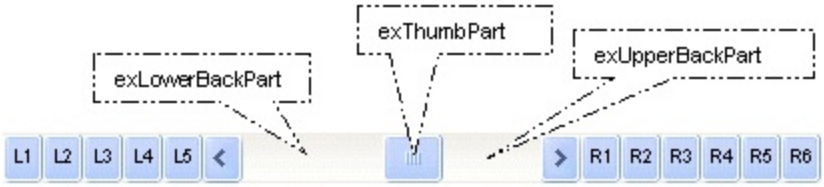


property Edit.ScrollPartVisible(ScrollBar as ScrollBarEnum, Part as ScrollPartEnum) as Boolean

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 ScrolPartCaption 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 Edit1
	.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 AxEdit1
   .set_ScrollPartVisible(EXEDITLib.ScrollBarEnum.exVScroll,
EXEDITLib.ScrollPartEnum.exLeftB1Part Or EXEDITLib.ScrollPartEnum.exRightB1Part, True)
   .set_ScrollPartCaption(EXEDITLib.ScrollBarEnum.exVScroll,
EXEDITLib.ScrollPartEnum.exLeftB1Part, "1")
   .set_ScrollPartCaption(EXEDITLib.ScrollBarEnum.exVScroll,
EXEDITLib.ScrollPartEnum.exRightB1Part, "2")
End With
```

The following C# sample adds up and down additional buttons to the control's vertical scroll bar :

```
axEdit1.set_ScrollPartVisible(EXEDITLib.ScrollBarEnum.exVScroll,
EXEDITLib.ScrollPartEnum.exLeftB1Part | EXEDITLib.ScrollPartEnum.exRightB1Part, true);
axEdit1.set_ScrollPartCaption(EXEDITLib.ScrollBarEnum.exVScroll,
EXEDITLib.ScrollPartEnum.exLeftB1Part , "1");
axEdit1.set_ScrollPartCaption(EXEDITLib.ScrollBarEnum.exVScroll,
EXEDITLib.ScrollPartEnum.exRightB1Part, "2");
```

The following C++ sample adds up and down additional buttons to the control's vertical scroll bar :

```
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") );
```

The following VFP sample adds up and down additional buttons to the control's vertical scroll bar :

```
With thisform.Edit1
```

```
    .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 Edit.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(exVSThumb) or Background(exHSThumb) 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 Edit.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 Edit.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 Edit.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. Use the GetSelection method to determine the coordinates of the selected text. Use the SetSelection method to select a text giving the end and start coordinates. The control fires the SelChange event when user changes the selection.

property Edit.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.

method Edit.SelectLine (Index as Long)

Selects a line by code.

Type	Description
Index as Long	A long expression that indicates the index of line being selected. The index is 1 based.

Use the SelectLine method to select a line by code. Use the SelText property to retrieve the selected text. Use the GetSelection method to determine the coordinates of the selected text. Use the SetSelection method to select a text giving the end and start coordinates. The SelStart property returns or sets the starting point of text selected; indicates the position of the insertion point if no text is selected.

property Edit.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. Use the GetSelection method to determine the coordinates of the selected text. Use the SetSelection method to select a text giving the end and start coordinates. The control fires the SelChange event when user changes the selection.

property Edit.SelForeColorHide 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 SelForeColorHide 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 Edit.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. Use the GetSelection method to determine the coordinates of the selected text. Use the SetSelection method to select a text giving the end and start coordinates. Use the SelectLine method to select a line by code. 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 Edit.SelStart as Long

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

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

Returns the starting point of a text selection made by the user in a text-entry area of a control, or indicates the position of the insertion point if no text is selected. Also, specifies the starting point of a text selection in a text-entry area of a control. Not available at design time; read-write at run time. Use the SelLenght property to get the selection's length. Use the SelectLine method to select a line by code. Use the GetSelection method to determine the coordinates of the selected text. Use the SetSelection method to select a text giving the end and start coordinates.

property Edit.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. Use the CaretPos and CaretLine properties to get the control's caret position. Use the SelectLine method to select a line by code. Use the GetSelection method to determine the coordinates of the selected text. 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 Edit1_SelChange()
   If Not Edit1.SelText = "" Then
      Debug.Print Edit1.SelText
   End If
End Sub
```

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

```
void OnSelChangeEdit1()
{
   OutputDebugString( m_edit.GetSelText() );
}
```

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

```
Private Sub AxEdit1_SelChange(ByVal sender As Object, ByVal e As System.EventArgs)
Handles AxEdit1.SelChange
   With AxEdit1
      Debug.WriteLine(.SelText)
   End With
End Sub
```

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

```
private void axEdit1_SelChange(object sender, EventArgs e)
{
    System.Diagnostics.Debug.WriteLine(axEdit1.SelText);
}
```

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

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

with thisform.Edit1
    wait window nowait .SelText
endwith
```

method Edit.SetSelection ([SelStartLine as Variant], [SelStartPos as Variant], [SelEndLine as Variant], [SelEndPos as Variant])

Selects the text giving the start and end coordinates.

Type	Description
SelStartLine as Variant	A long expression that indicates the index of the line where the selection starts. If the SelStartLine parameter is negative, the selection starts from the cursor position.
SelStartPos as Variant	A long expression that indicates the index of the character in the line where the selection starts. If the SelStartPos parameter is negative, the selection starts from the cursor position.
SelEndLine as Variant	A long expression that indicates the index of the line where the selection ends. If the SelEndLine parameter is negative the control selects the text to the end.
SelEndPos as Variant	A long expression that indicates the index of the character in the line where the selection ends. If the SelEndPos parameter is negative the control selects the text to the end.

Use the SetSelection method selects a text giving the start and end position. Use the CaretPos and CaretLine properties to determine the caret/cursor's position. Use the GetSelection method to retrieve the coordinates of the selection. The SelLenght property returns or sets the number of characters selected. Use the SelectLine property to select a single line. The control fires the SelChange event when the user changes the selection or the cursor's is moved. The HideSelection property specifies whether the selection in the control is hidden when the control loses the focus.

The following VB sample selects the entire text in the control:

```
With Edit1
   .SetSelection 0, 0, -1, -1
End With
```

The following VB sample selects the text from the caret position to the end:

```
With Edit1
   .SetSelection -1, -1, -1, -1
End With
```

The following C++ sample selects the entire text in the control:

```
m_edit.SetSelection( COleVariant( (long)0 ), COleVariant( (long)0 ), COleVariant( (long)-1 ),
COleVariant( (long)-1 ) );
```

The following VB.NET sample selects the entire text in the control:

```
With AxEdit1
   .SetSelection(0, 0, -1, -1)
End With
```

The following C# sample selects the entire text in the control:

```
axEdit1.SetSelection(0, 0, -1, -1);
```

The following VFP sample selects the entire text in the control:

```
With thisform.Edit1
   .SetSelection(0, 0, -1, -1)
EndWith
```

property Edit.Show as String

Specifies the order and visibility of wild, keyword, expression and format-numbers within the control.

Type	Description
String	A string expression that defines the order and visibility of wild, keyword, expression and format-numbers within the control.

By default, the Show property is "expression,key,wild,number". The Show property defines the order and visibility of wild, keyword, expression and format-numbers within the control.

property Edit.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.

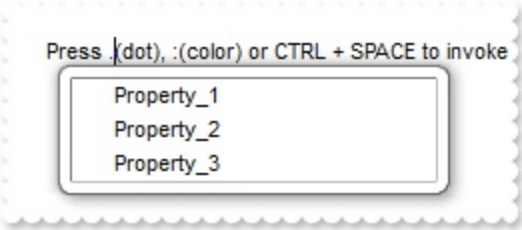
Use the ShowCaret property to hide the control's caret. By default, the ShowCaret property is True. The control hides the caret when the control is disabled. Use the Enabled property to disable the control. Use the Locked property to lock the control.

method Edit.ShowContext ([Context as Variant])

Shows the control's context, as the user would press the CTRL + SPACE key.

Type	Description
Context as Variant	A string expression that defines the current context.

The ShowContext method shows programmatically the control's context, as the user would press the CTRL + SPACE key. The ActiveContextItems property defines the key of the current context. Use the Context collection to customize the list of keywords when the code completion is enabled. The control can display multiple pages on the control's sensitive context, if multiple key-context are defined. Use the CodeCompletion property to disable the code completion support. By default, the user can display the control's context window by pressing the CTRL + SPACE key combination. Use the ContextKey property to define the key combination to open the control's context window. The ChangeOnKey property specifies the code of the last key that alters the control's text.



How can I provide different sensitive context?

VBA (MS Access, Excell...)

```
' Change event - Indicates that the control's text have changed.
Private Sub Edit1_Change()
   With Edit1
      .ShowContext .ChangeOnKey
   End With
End Sub

With Edit1
   .Text = ""
   .InsertText "Press .(dot), :(colon) or CTRL + SPACE to invoke the control's context"
   With .Context()
      .Add "General_1"
      .Add "General_2"
   End With
```

```
    With .Context("46")
        .Add "Property_1"
        .Add "Property_2"
        .Add "Property_3"
    End With
    With .Context("58")
        .Add "Method_1"
        .Add "Method_2"
        .Add "Method_3"
    End With
End With
```

VB6

```
' Change event - Indicates that the control's text have changed.
Private Sub Edit1_Change()
    With Edit1
        .ShowContext .ChangeOnKey
    End With
End Sub

With Edit1
    .Text = ""
    .InsertText "Press .(dot), :(colon) or CTRL + SPACE to invoke the control's context"
    With .Context()
        .Add "General_1"
        .Add "General_2"
    End With
    With .Context("46")
        .Add "Property_1"
        .Add "Property_2"
        .Add "Property_3"
    End With
    With .Context("58")
        .Add "Method_1"
        .Add "Method_2"
        .Add "Method_3"
```

```
    End With
End With
```

VB.NET

```
' Change event - Indicates that the control's text have changed.
Private Sub Exedit1_Change(ByVal sender As System.Object) Handles Exedit1.Change
    With Exedit1
        .ShowContext(.ChangeOnKey)
    End With
End Sub

With Exedit1
    .Text = ""
    .InsertText("Press .(dot), :(colon) or CTRL + SPACE to invoke the control's context")
    With .get_Context()
        .Add("General_1")
        .Add("General_2")
    End With
    With .get_Context("46")
        .Add("Property_1")
        .Add("Property_2")
        .Add("Property_3")
    End With
    With .get_Context("58")
        .Add("Method_1")
        .Add("Method_2")
        .Add("Method_3")
    End With
End With
```

VB.NET for /COM

```
' Change event - Indicates that the control's text have changed.
Private Sub AxEdit1_Change(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles AxEdit1.Change
    With AxEdit1
        .ShowContext(.ChangeOnKey)
```

```
    End With
End Sub

With AxEdit1
    .Text = ""
    .InsertText("Press .(dot), :(colon) or CTRL + SPACE to invoke the control's context")
    With .get_Context()
        .Add("General_1")
        .Add("General_2")
    End With
    With .get_Context("46")
        .Add("Property_1")
        .Add("Property_2")
        .Add("Property_3")
    End With
    With .get_Context("58")
        .Add("Method_1")
        .Add("Method_2")
        .Add("Method_3")
    End With
End With
```

C++

```
// Change event - Indicates that the control's text have changed.
void OnChangeEdit1()
{
    /*
        Copy and paste the following directives to your header file as
        it defines the namespace 'EXEDITLib' for the library: 'ExEdit 1.0 Control Library'
        #import <ExEdit.dll>
        using namespace EXEDITLib;
     */
    EXEDITLib::IEditPtr spEdit1 = GetDlgItem(IDC_EDIT1)->GetControlUnknown();
    spEdit1->ShowContext(spEdit1->GetChangeOnKey());
}
```

```
EXEDITLib::IEditPtr spEdit1 = GetDlgItem(IDC_EDIT1)->GetControlUnknown();
spEdit1->PutText(L"");
spEdit1->InsertText(L"Press .(dot), :(colon) or CTRL + SPACE to invoke the control's
context",vtMissing);
EXEDITLib::IEditContextPtr var_Context = spEdit1->GetContext(vtMissing);
   var_Context->Add(L"General_1",vtMissing,vtMissing,vtMissing);
   var_Context->Add(L"General_2",vtMissing,vtMissing,vtMissing);
EXEDITLib::IEditContextPtr var_Context1 = spEdit1->GetContext("46");
   var_Context1->Add(L"Property_1",vtMissing,vtMissing,vtMissing);
   var_Context1->Add(L"Property_2",vtMissing,vtMissing,vtMissing);
   var_Context1->Add(L"Property_3",vtMissing,vtMissing,vtMissing);
EXEDITLib::IEditContextPtr var_Context2 = spEdit1->GetContext("58");
   var_Context2->Add(L"Method_1",vtMissing,vtMissing,vtMissing);
   var_Context2->Add(L"Method_2",vtMissing,vtMissing,vtMissing);
   var_Context2->Add(L"Method_3",vtMissing,vtMissing,vtMissing);
```

C++ Builder

```
// Change event - Indicates that the control's text have changed.
void __fastcall TForm1::Edit1Change(TObject *Sender)
{
   Edit1->ShowContext(TVariant(Edit1->ChangeOnKey));
}

Edit1->Text = L"";
Edit1->InsertText(L"Press .(dot), :(colon) or CTRL + SPACE to invoke the control's
context",TNoParam());
Exeditlib_tlb::IEditContextPtr var_Context = Edit1->Context[TNoParam()];
   var_Context->Add(L"General_1",TNoParam(),TNoParam(),TNoParam());
   var_Context->Add(L"General_2",TNoParam(),TNoParam(),TNoParam());
Exeditlib_tlb::IEditContextPtr var_Context1 = Edit1->Context[TVariant("46")];
   var_Context1->Add(L"Property_1",TNoParam(),TNoParam(),TNoParam());
   var_Context1->Add(L"Property_2",TNoParam(),TNoParam(),TNoParam());
   var_Context1->Add(L"Property_3",TNoParam(),TNoParam(),TNoParam());
Exeditlib_tlb::IEditContextPtr var_Context2 = Edit1->Context[TVariant("58")];
   var_Context2->Add(L"Method_1",TNoParam(),TNoParam(),TNoParam());
```

```
var_Context2->Add(L"Method_2",TNoParam(),TNoParam(),TNoParam());
var_Context2->Add(L"Method_3",TNoParam(),TNoParam(),TNoParam());
```

C#

```
// Change event - Indicates that the control's text have changed.
private void exedit1_Change(object sender)
{
   exedit1.ShowContext(exedit1.ChangeOnKey);
}
//this.exedit1.Change += new
exontrol.EXEDITLib.exg2antt.ChangeEventHandler(this.exedit1_Change);

exedit1.Text = "";
exedit1.InsertText("Press .(dot), :(colon) or CTRL + SPACE to invoke the control's
context",null);
exontrol.EXEDITLib.Context var_Context = exedit1.get_Context(null);
   var_Context.Add("General_1",null,null,null);
   var_Context.Add("General_2",null,null,null);
exontrol.EXEDITLib.Context var_Context1 = exedit1.get_Context("46");
   var_Context1.Add("Property_1",null,null,null);
   var_Context1.Add("Property_2",null,null,null);
   var_Context1.Add("Property_3",null,null,null);
exontrol.EXEDITLib.Context var_Context2 = exedit1.get_Context("58");
   var_Context2.Add("Method_1",null,null,null);
   var_Context2.Add("Method_2",null,null,null);
   var_Context2.Add("Method_3",null,null,null);
```

JScript/JavaScript

```
<BODY onload="Init()">
<SCRIPT FOR="Edit1" EVENT="Change()" LANGUAGE="JScript">
   Edit1.ShowContext(Edit1.ChangeOnKey);
</SCRIPT>

<OBJECT CLASSID="clsid:39136531-DD0F-4281-B445-E36FC2CDDBC5" id="Edit1">
```

```
</OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
   Edit1.Text = "";
   Edit1.InsertText("Press .(dot), :(colon) or CTRL + SPACE to invoke the control's
context",null);
   var var_Context = Edit1.Context(null);
      var_Context.Add("General_1",null,null,null);
      var_Context.Add("General_2",null,null,null);
   var var_Context1 = Edit1.Context("46");
      var_Context1.Add("Property_1",null,null,null);
      var_Context1.Add("Property_2",null,null,null);
      var_Context1.Add("Property_3",null,null,null);
   var var_Context2 = Edit1.Context("58");
      var_Context2.Add("Method_1",null,null,null);
      var_Context2.Add("Method_2",null,null,null);
      var_Context2.Add("Method_3",null,null,null);
}
</SCRIPT>
</BODY>
```

VBScript

```
<BODY onload="Init()">
<SCRIPT LANGUAGE="VBScript">
Function Edit1_Change()
   With Edit1
      .ShowContext .ChangeOnKey
   End With
End Function
</SCRIPT>

<OBJECT CLASSID="clsid:39136531-DD0F-4281-B445-E36FC2CDDBC5" id="Edit1">
</OBJECT>
```

```
<SCRIPT LANGUAGE="VBScript">
Function Init()
   With Edit1
      .Text = ""
      .InsertText "Press .(dot), :(colon) or CTRL + SPACE to invoke the control's context"
      With .Context()
         .Add "General_1"
         .Add "General_2"
      End With
      With .Context("46")
         .Add "Property_1"
         .Add "Property_2"
         .Add "Property_3"
      End With
      With .Context("58")
         .Add "Method_1"
         .Add "Method_2"
         .Add "Method_3"
      End With
   End With
End Function
</SCRIPT>
</BODY>
```

C# for /COM

```
// Change event - Indicates that the control's text have changed.
private void axEdit1_Change(object sender, EventArgs e)
{
   axEdit1.ShowContext(axEdit1.ChangeOnKey);
}
//this.axEdit1.Change += new EventHandler(this.axEdit1_Change);

axEdit1.Text = "";
axEdit1.InsertText("Press .(dot), :(colon) or CTRL + SPACE to invoke the control's
```

```
context",null);
EXEDITLib.Context var_Context = axEdit1.get_Context(null);
   var_Context.Add("General_1",null,null,null);
   var_Context.Add("General_2",null,null,null);
EXEDITLib.Context var_Context1 = axEdit1.get_Context("46");
   var_Context1.Add("Property_1",null,null,null);
   var_Context1.Add("Property_2",null,null,null);
   var_Context1.Add("Property_3",null,null,null);
EXEDITLib.Context var_Context2 = axEdit1.get_Context("58");
   var_Context2.Add("Method_1",null,null,null);
   var_Context2.Add("Method_2",null,null,null);
   var_Context2.Add("Method_3",null,null,null);
```

X++ (Dynamics Ax 2009)

```
// Change event - Indicates that the control's text have changed.
void onEvent_Change()
{
   ;
   exedit1.ShowContext(exedit1.ChangeOnKey());
}

public void init()
{
   COM com_Context,com_Context1,com_Context2;
   anytype var_Context,var_Context1,var_Context2;
   ;

   super();

   exedit1.Text("");
   exedit1.InsertText("Press .(dot), :(colon) or CTRL + SPACE to invoke the control's
context");
   var_Context = exedit1.Context(); com_Context = var_Context;
      com_Context.Add("General_1");
      com_Context.Add("General_2");
```

```
    var_Context1 = exedit1.Context("46"); com_Context1 = var_Context1;
        com_Context1.Add("Property_1");
        com_Context1.Add("Property_2");
        com_Context1.Add("Property_3");
    var_Context2 = exedit1.Context("58"); com_Context2 = var_Context2;
        com_Context2.Add("Method_1");
        com_Context2.Add("Method_2");
        com_Context2.Add("Method_3");
}
```

Delphi 8 (.NET only)

```
// Change event - Indicates that the control's text have changed.
procedure TWinForm1.AxEdit1_Change(sender: System.Object; e: System.EventArgs);
begin
    with AxEdit1 do
    begin
        ShowContext(TObject(ChangeOnKey));
    end
end;

with AxEdit1 do
begin
    Text := '';
    InsertText('Press .(dot), :(colon) or CTRL + SPACE to invoke the control''s
context',Nil);
    with get_Context(Nil) do
    begin
        Add('General_1',Nil,Nil,Nil);
        Add('General_2',Nil,Nil,Nil);
    end;
    with get_Context('46') do
    begin
        Add('Property_1',Nil,Nil,Nil);
        Add('Property_2',Nil,Nil,Nil);
        Add('Property_3',Nil,Nil,Nil);
    end;
```

```
  with get_Context('58') do
  begin
    Add('Method_1',Nil,Nil,Nil);
    Add('Method_2',Nil,Nil,Nil);
    Add('Method_3',Nil,Nil,Nil);
  end;
end
```

Delphi (standard)

```
// Change event - Indicates that the control's text have changed.
procedure TForm1.Edit1Change(ASender: TObject; );
begin
  with Edit1 do
  begin
    ShowContext(OleVariant(ChangeOnKey));
  end
end;

with Edit1 do
begin
  Text := '';
  InsertText('Press .(dot), :(colon) or CTRL + SPACE to invoke the control''s
context',Null);
  with Context[Null] do
  begin
    Add('General_1',Null,Null,Null);
    Add('General_2',Null,Null,Null);
  end;
  with Context['46'] do
  begin
    Add('Property_1',Null,Null,Null);
    Add('Property_2',Null,Null,Null);
    Add('Property_3',Null,Null,Null);
  end;
  with Context['58'] do
  begin
```

```
        Add('Method_1',Null,Null,Null);
        Add('Method_2',Null,Null,Null);
        Add('Method_3',Null,Null,Null);
    end;
end
```

VFP

```
*** Change event - Indicates that the control's text have changed. ***
LPARAMETERS nop
    with thisform.Edit1
        .ShowContext(.ChangeOnKey)
    endwith

with thisform.Edit1
    .Text = ""
    .InsertText("Press .(dot), :(colon) or CTRL + SPACE to invoke the control's context")
    with .Context()
        .Add("General_1")
        .Add("General_2")
    endwith
    with .Context("46")
        .Add("Property_1")
        .Add("Property_2")
        .Add("Property_3")
    endwith
    with .Context("58")
        .Add("Method_1")
        .Add("Method_2")
        .Add("Method_3")
    endwith
endwith
```

dBASE Plus

```
/*
with (this.EXEDITACTIVEXCONTROL1.nativeObject)
    Change = class::nativeObject_Change
```

```
endwith
*/
// Indicates that the control's text have changed.
function nativeObject_Change()
   oEdit = form.EXEDITACTIVEXCONTROL1.nativeObject
   oEdit.ShowContext(oEdit.ChangeOnKey)
return

local oEdit,var_Context,var_Context1,var_Context2

oEdit = form.EXEDITACTIVEXCONTROL1.nativeObject
oEdit.Text = ""
oEdit.InsertText("Press .(dot), :(colon) or CTRL + SPACE to invoke the control's
context")
var_Context = oEdit.Context()
   var_Context.Add("General_1")
   var_Context.Add("General_2")
var_Context1 = oEdit.Context("46")
   var_Context1.Add("Property_1")
   var_Context1.Add("Property_2")
   var_Context1.Add("Property_3")
var_Context2 = oEdit.Context("58")
   var_Context2.Add("Method_1")
   var_Context2.Add("Method_2")
   var_Context2.Add("Method_3")
```

XBasic (Alpha Five)

```
' Indicates that the control's text have changed.
function Change as v ()
   oEdit = topparent:CONTROL_ACTIVEX1.activex
   oEdit.ShowContext(oEdit.ChangeOnKey)
end function

Dim oEdit as P
Dim var_Context as P
```

```
Dim var_Context1 as P
Dim var_Context2 as P

oEdit = topparent:CONTROL_ACTIVEX1.activex
oEdit.Text = ""
oEdit.InsertText("Press .(dot), :(colon) or CTRL + SPACE to invoke the control's
context")
var_Context = oEdit.Context()
   var_Context.Add("General_1")
   var_Context.Add("General_2")
var_Context1 = oEdit.Context("46")
   var_Context1.Add("Property_1")
   var_Context1.Add("Property_2")
   var_Context1.Add("Property_3")
var_Context2 = oEdit.Context("58")
   var_Context2.Add("Method_1")
   var_Context2.Add("Method_2")
   var_Context2.Add("Method_3")
```

Visual Objects

```
METHOD OCX_Exontrol1Change() CLASS MainDialog
   // Change event - Indicates that the control's text have changed.
   oDCOCX_Exontrol1:ShowContext(oDCOCX_Exontrol1:ChangeOnKey)
RETURN NIL

local var_Context,var_Context1,var_Context2 as IEditContext

oDCOCX_Exontrol1:Text := ""
oDCOCX_Exontrol1:InsertText("Press .(dot), :(colon) or CTRL + SPACE to invoke the
control's context",nil)
var_Context := oDCOCX_Exontrol1:[Context,nil]
   var_Context:Add("General_1",nil,nil,nil)
   var_Context:Add("General_2",nil,nil,nil)
var_Context1 := oDCOCX_Exontrol1:[Context,"46"]
   var_Context1:Add("Property_1",nil,nil,nil)
```

```
   var_Context1:Add("Property_2",nil,nil,nil)
   var_Context1:Add("Property_3",nil,nil,nil)
var_Context2 := oDCOCX_Exontrol1:[Context,"58"]
   var_Context2:Add("Method_1",nil,nil,nil)
   var_Context2:Add("Method_2",nil,nil,nil)
   var_Context2:Add("Method_3",nil,nil,nil)
```

PowerBuilder

```
/*begin event Change() - Indicates that the control's text have changed.*/
/*
    oEdit = ole_1.Object
    oEdit.ShowContext(oEdit.ChangeOnKey)
*/
/*end event Change*/

OleObject oEdit,var_Context,var_Context1,var_Context2

oEdit = ole_1.Object
oEdit.Text = ""
oEdit.InsertText("Press .(dot), :(colon) or CTRL + SPACE to invoke the control's
context")
var_Context = oEdit.Context()
   var_Context.Add("General_1")
   var_Context.Add("General_2")
var_Context1 = oEdit.Context("46")
   var_Context1.Add("Property_1")
   var_Context1.Add("Property_2")
   var_Context1.Add("Property_3")
var_Context2 = oEdit.Context("58")
   var_Context2.Add("Method_1")
   var_Context2.Add("Method_2")
   var_Context2.Add("Method_3")
```

Visual DataFlex

```
// Indicates that the control's text have changed.
Procedure OnComChange
   Forward Send OnComChange
   Send ComShowContext (ComChangeOnKey(Self))
End_Procedure

Procedure OnCreate
   Forward Send OnCreate
   Set ComText to ""
   Send ComInsertText "Press .(dot), :(colon) or CTRL + SPACE to invoke the control's
context" Nothing
   Variant voContext
   Get ComContext Nothing to voContext
   Handle hoContext
   Get Create (RefClass(cComContext)) to hoContext
   Set pvComObject of hoContext to voContext
      Send ComAdd of hoContext "General_1" Nothing Nothing Nothing
      Send ComAdd of hoContext "General_2" Nothing Nothing Nothing
   Send Destroy to hoContext
   Variant voContext1
   Get ComContext "46" to voContext1
   Handle hoContext1
   Get Create (RefClass(cComContext)) to hoContext1
   Set pvComObject of hoContext1 to voContext1
      Send ComAdd of hoContext1 "Property_1" Nothing Nothing Nothing
      Send ComAdd of hoContext1 "Property_2" Nothing Nothing Nothing
      Send ComAdd of hoContext1 "Property_3" Nothing Nothing Nothing
   Send Destroy to hoContext1
   Variant voContext2
   Get ComContext "58" to voContext2
   Handle hoContext2
   Get Create (RefClass(cComContext)) to hoContext2
   Set pvComObject of hoContext2 to voContext2
      Send ComAdd of hoContext2 "Method_1" Nothing Nothing Nothing
      Send ComAdd of hoContext2 "Method_2" Nothing Nothing Nothing
      Send ComAdd of hoContext2 "Method_3" Nothing Nothing Nothing
   Send Destroy to hoContext2
```

```
End_Procedure
```

XBase++

```
PROCEDURE OnChange(oEdit)
   oEdit:ShowContext(oEdit:ChangeOnKey())
RETURN

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

PROCEDURE Main
   LOCAL oForm
   LOCAL nEvent := 0, mp1 := NIL, mp2 := NIL, oXbp := NIL
   LOCAL oEdit
   LOCAL oContext,oContext1,oContext2

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

   oEdit := XbpActiveXControl():new( oForm:drawingArea )
   oEdit:CLSID  := "Exontrol.Edit.1" /*{39136531-DD0F-4281-B445-
E36FC2CDDBC5}*/
   oEdit:create(,, {10,60},{610,370} )

      oEdit:Change := {|| OnChange(oEdit)} /*Indicates that the control's text have
changed.*/

      oEdit:Text := ""
      oEdit:InsertText("Press .(dot), :(colon) or CTRL + SPACE to invoke the control's
context")
      oContext := oEdit:Context()
         oContext:Add("General_1")
         oContext:Add("General_2")
      oContext1 := oEdit:Context("46")
```

```
        oContext1:Add("Property_1")
        oContext1:Add("Property_2")
        oContext1:Add("Property_3")
      oContext2 := oEdit:Context("58")
        oContext2:Add("Method_1")
        oContext2:Add("Method_2")
        oContext2:Add("Method_3")

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

property Edit.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. Use the BookmarkImageLine property to assign an icon to the control's bookmarks header.

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 Edit.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	The ToolTip parameter can be any of the following: • 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	The Title parameter can be any of the following: • 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	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. The Alignment parameter can be one of the following: • 0 - exTopLeft • 1 - exTopRight • 2 - exBottomLeft • 3 - exBottomRight • 0x10 - exCenter • 0x11 - exCenterLeft • 0x12 - exCenterRight • 0x13 - exCenterTop • 0x14 - exCenterBottom By default, the tooltip is aligned relative to the top-left corner (0 - exTopLeft).

X as Variant	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)
Y 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)

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**
- ** ... ** 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
- **<img>number[:width]</img>** 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.
- **<img>key[:width]</img>** 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 $_{\text{subscript}}$ 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** FFFFFF;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:

- **<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:

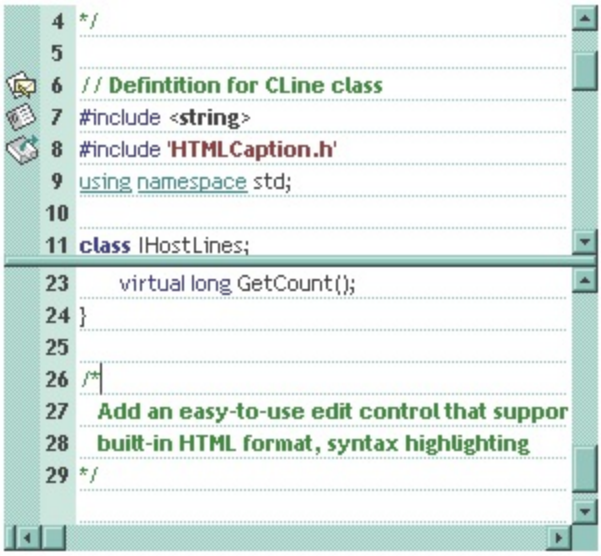
or "*<**sha** 404040;5;0><fgcolor=FFFFFF>outline anti-aliasing</fgcolor></**sha**>*" gets:

property Edit.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 edit control. Use the SplitPaneWidth property to split horizontally the edit 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 Edit.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 edit control. Use the SplitPaneHeight property to split vertically the edit control. Use the AllowSplitter property to show or hide certain splitters. The SplitterChange event notifies your application when user splits the control.

property Edit.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.

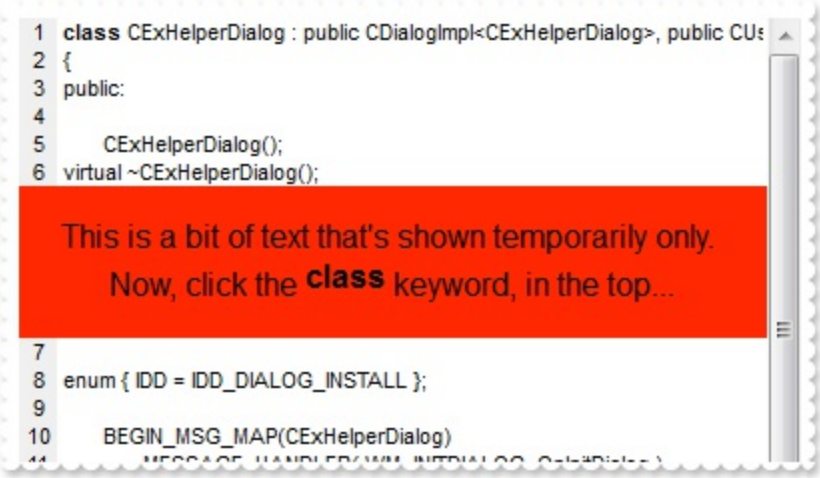
Use the TabLength property to defines the size for each tab character. By default, the TabLength property is 4.

property Edit.TempInlineToolTip as String

Specifies a temporarily inline tooltip to be shown on the current line.

Type	Description
String	A String expression that defines the temporarily inline HTML text to be shown on the current line.

By default, the TempInlineToolTip property is empty. The TempInlineToolTip property can be used, even if the AllowInlineToolTip property is exDisableInlineToolTip (by default). For instance, you can use the Inline-ToolTip feature to display inside the control at current line, any information/html text you may need. Use the AllowInlineToolTip property to enable or disable the control's inline-tooltip feature. The TempInlineToolTip is hidden as soon as the selection is changed (the caret is moved). The Background(exTempInlineToolTipForeColor) / Background(exTempInlineToolTipBackColor) property define the foreground / background color / visual appearance of the temporarily inline-tooltip panel. If these are not defined, the Background(exInlineToolTipForeColor) / Background(exInlineToolTipBackColor) is used instead.



The TempInlineToolTip, supports built-HTML as listed:

- ** ... ** displays the text in **bold**
- ** ... ** 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
- **<img>number[:width]</img>** 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.
- **<img>key[:width]</img>** 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 $_{\text{subscript}}$ 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** FFFFFF;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

Is it possible to display the inline tooltip with a different appearance than temporarily inline tooltip?

VBA (MS Access, Excell...)

```
With Edit1
   .LineNumberWidth = -1
   .LineNumberBackColor = RGB(240,240,240)
   .AddKeyword "<b>class</b>","<r>a set or category of things having some property or attribute in common and differentiated from others by kind, type, or qua" & _
"lity."
   .AllowInlineToolTip = 513 ' AllowInlineToolTipEnum.exInlineToolTipWordWrap Or AllowInlineToolTipEnum.exInlineToolTip
   .Background(159) = RGB(128,128,128)
   .Background(158) = RGB(240,240,240)
   .CaretLine = 6
   .Background(160) = RGB(255,40,0)
   .Background(161) = RGB(0,0,1)
   .TempInlineToolTip = "<br><c><font ;12>This is a bit of text that's shown temporarily only. <br><c>Now, click the <off -4><b>class</b></off> keyword," & _
" in the top...<br>"
   .Refresh
End With
```

VB6

```
With Edit1
   .LineNumberWidth = -1
   .LineNumberBackColor = RGB(240,240,240)
   .AddKeyword "<b>class</b>","<r>a set or category of things having some property or attribute in common and differentiated from others by kind, type, or qua" & _
"lity."
   .AllowInlineToolTip = AllowInlineToolTipEnum.exInlineToolTipWordWrap Or AllowInlineToolTipEnum.exInlineToolTip
   .Background(exInlineToolTipForeColor) = RGB(128,128,128)
```

```
    .Background(exInlineToolTipBackColor) = RGB(240,240,240)
    .CaretLine = 6
    .Background(exTempInlineToolTipBackColor) = RGB(255,40,0)
    .Background(exTempInlineToolTipForeColor) = RGB(0,0,1)
    .TempInlineToolTip = "<br><c><font ;12>This is a bit of text that's shown
temporarily only. <br><c>Now, click the <off -4><b>class</b></off> keyword," &
_
" in the top...<br>"
    .Refresh
End With
```

VB.NET

```
With Exedit1
    .LineNumberWidth = -1
    .LineNumberBackColor = Color.FromArgb(240,240,240)
    .AddKeyword("<b>class</b>","<r>a set or category of things having some
property or attribute in common and differentiated from others by kind, type, or
qua" & _
"lity.")
    .AllowInlineToolTip =
exontrol.EXEDITLib.AllowInlineToolTipEnum.exInlineToolTipWordWrap Or
exontrol.EXEDITLib.AllowInlineToolTipEnum.exInlineToolTip

.set_Background(exontrol.EXEDITLib.BackgroundPartEnum.exInlineToolTipForeColor,C



.set_Background(exontrol.EXEDITLib.BackgroundPartEnum.exInlineToolTipBackColor,C

    .CaretLine = 6

.set_Background(exontrol.EXEDITLib.BackgroundPartEnum.exTempInlineToolTipBackC



.set_Background(exontrol.EXEDITLib.BackgroundPartEnum.exTempInlineToolTipForeC

    .TempInlineToolTip = "<br><c><font ;12>This is a bit of text that's shown
```

```
temporarily only. <br><c>Now, click the <off -4><b>class</b></off> keyword," &
_
" in the top...<br>"
    .Refresh()
End With
```

VB.NET for /COM

```
With AxEdit1
    .LineNumberWidth = -1
    .LineNumberBackColor = RGB(240,240,240)
    .AddKeyword("<b>class</b>","<r>a set or category of things having some
property or attribute in common and differentiated from others by kind, type, or
qua" & _
"lity.")
    .AllowInlineToolTip = EXEDITLib.AllowInlineToolTipEnum.exInlineToolTipWordWrap
Or EXEDITLib.AllowInlineToolTipEnum.exInlineToolTip

.set_Background(EXEDITLib.BackgroundPartEnum.exInlineToolTipForeColor,8421504)

.set_Background(EXEDITLib.BackgroundPartEnum.exInlineToolTipBackColor,15790320)

    .CaretLine = 6

.set_Background(EXEDITLib.BackgroundPartEnum.exTempInlineToolTipBackColor,1049


.set_Background(EXEDITLib.BackgroundPartEnum.exTempInlineToolTipForeColor,6553

    .TempInlineToolTip = "<br><c><font ;12>This is a bit of text that's shown
temporarily only. <br><c>Now, click the <off -4><b>class</b></off> keyword," &
_
" in the top...<br>"
    .Refresh()
End With
```

C++

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

    #import <ExEdit.dll>
    using namespace EXEDITLib;
*/
EXEDITLib::IEditPtr spEdit1 = GetDlgItem(IDC_EDIT1)->GetControlUnknown();
spEdit1->PutLineNumberWidth(-1);
spEdit1->PutLineNumberBackColor(RGB(240,240,240));
spEdit1->AddKeyword(L"<b>class</b>",_bstr_t("<r>a set or category of things
having some property or attribute in common and differentiated from others by
kind, type, or qua") +
"lity.",vtMissing,vtMissing);
spEdit1-
>PutAllowInlineToolTip(EXEDITLib::AllowInlineToolTipEnum(EXEDITLib::exInlineToolTipV
 | EXEDITLib::exInlineToolTip));
spEdit1->PutBackground(EXEDITLib::exInlineToolTipForeColor,RGB(128,128,128));
spEdit1->PutBackground(EXEDITLib::exInlineToolTipBackColor,RGB(240,240,240));
spEdit1->PutCaretLine(6);
spEdit1->PutBackground(EXEDITLib::exTempInlineToolTipBackColor,RGB(255,40,0));
spEdit1->PutBackground(EXEDITLib::exTempInlineToolTipForeColor,RGB(0,0,1));
spEdit1->PutTempInlineToolTip(_bstr_t("<br><c><font ;12>This is a bit of text that's
shown temporarily only. <br><c>Now, click the <off -4><b>class</b></off>
keyword,") +
" in the top...<br>");
spEdit1->Refresh();
```

C++ Builder

```
Edit1->LineNumberWidth = -1;
Edit1->LineNumberBackColor = RGB(240,240,240);
Edit1->AddKeyword(L"<b>class</b>",TVariant(String("<r>a set or category of
things having some property or attribute in common and differentiated from others
by kind, type, or qua") +
"lity."),TNoParam(),TNoParam());
```

```
Edit1->AllowInlineToolTip =
Exeditlib_tlb::AllowInlineToolTipEnum::exInlineToolTipWordWrap |
Exeditlib_tlb::AllowInlineToolTipEnum::exInlineToolTip;
Edit1->Background[Exeditlib_tlb::BackgroundPartEnum::exInlineToolTipForeColor] =
RGB(128,128,128);
Edit1->Background[Exeditlib_tlb::BackgroundPartEnum::exInlineToolTipBackColor] =
RGB(240,240,240);
Edit1->CaretLine = 6;
Edit1-
>Background[Exeditlib_tlb::BackgroundPartEnum::exTempInlineToolTipBackColor] =
RGB(255,40,0);
Edit1-
>Background[Exeditlib_tlb::BackgroundPartEnum::exTempInlineToolTipForeColor] =
RGB(0,0,1);
Edit1->TempInlineToolTip = TVariant(String("<br><c><font ;12>This is a bit of text
that's shown temporarily only. <br><c>Now, click the <off -4><b>class</b></off>
keyword,") +
" in the top...<br>");
Edit1->Refresh();
```

C#

```
exedit1.LineNumberWidth = -1;
exedit1.LineNumberBackColor = Color.FromArgb(240,240,240);
exedit1.AddKeyword("<b>class</b>","<r>a set or category of things having some
property or attribute in common and differentiated from others by kind, type, or
qua" +
"lity.",null,null);
exedit1.AllowInlineToolTip =
exontrol.EXEDITLib.AllowInlineToolTipEnum.exInlineToolTipWordWrap |
exontrol.EXEDITLib.AllowInlineToolTipEnum.exInlineToolTip;
exedit1.set_Background(exontrol.EXEDITLib.BackgroundPartEnum.exInlineToolTipFore

exedit1.set_Background(exontrol.EXEDITLib.BackgroundPartEnum.exInlineToolTipBack

exedit1.CaretLine = 6;
```

```
exedit1.set_Background(exontrol.EXEDITLib.BackgroundPartEnum.exTempInlineToolTip

exedit1.set_Background(exontrol.EXEDITLib.BackgroundPartEnum.exTempInlineToolTip

exedit1.TempInlineToolTip = "<br><c><font ;12>This is a bit of text that's shown
temporarily only. <br><c>Now, click the <off -4><b>class</b></off> keyword," +
" in the top...<br>";
exedit1.Refresh();
```

JScript/JavaScript

```
<BODY onload="Init()">
<OBJECT CLASSID="clsid:39136531-DD0F-4281-B445-E36FC2CDDBC5" id="Edit1">
</OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
   Edit1.LineNumberWidth = -1;
   Edit1.LineNumberBackColor = 15790320;
   Edit1.AddKeyword("<b>class</b>","<r>a set or category of things having some
property or attribute in common and differentiated from others by kind, type, or
qua" +
   "lity.",null,null);
   Edit1.AllowInlineToolTip = 513;
   Edit1.Background(159) = 8421504;
   Edit1.Background(158) = 15790320;
   Edit1.CaretLine = 6;
   Edit1.Background(160) = 10495;
   Edit1.Background(161) = 65536;
   Edit1.TempInlineToolTip = "<br><c><font ;12>This is a bit of text that's shown
temporarily only. <br><c>Now, click the <off -4><b>class</b></off> keyword," +
   " in the top...<br>";
   Edit1.Refresh();
}
</SCRIPT>
```

```
</BODY>
```

VBScript

```
<BODY onload="Init()">
<OBJECT CLASSID="clsid:39136531-DD0F-4281-B445-E36FC2CDDBC5" id="Edit1">
</OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
   With Edit1
      .LineNumberWidth = -1
      .LineNumberBackColor = RGB(240,240,240)
      .AddKeyword "<b>class</b>","<r>a set or category of things having some
property or attribute in common and differentiated from others by kind, type, or
qua" & _
   "lity."
      .AllowInlineToolTip = 513 ' AllowInlineToolTipEnum.exInlineToolTipWordWrap Or
AllowInlineToolTipEnum.exInlineToolTip
      .Background(159) = RGB(128,128,128)
      .Background(158) = RGB(240,240,240)
      .CaretLine = 6
      .Background(160) = RGB(255,40,0)
      .Background(161) = RGB(0,0,1)
      .TempInlineToolTip = "<br><c><font ;12>This is a bit of text that's shown
temporarily only. <br><c>Now, click the <off -4><b>class</b></off> keyword," &
_
   " in the top...<br>"
      .Refresh
   End With
End Function
</SCRIPT>
</BODY>
```

C# for /COM

```
axEdit1.LineNumberWidth = -1;
axEdit1.LineNumberBackColor = Color.FromArgb(240,240,240);
axEdit1.AddKeyword("<b>class</b>","<r>a set or category of things having some
property or attribute in common and differentiated from others by kind, type, or
qua" +
"lity.",null,null);
axEdit1.AllowInlineToolTip =
EXEDITLib.AllowInlineToolTipEnum.exInlineToolTipWordWrap |
EXEDITLib.AllowInlineToolTipEnum.exInlineToolTip;
axEdit1.set_Background(EXEDITLib.BackgroundPartEnum.exInlineToolTipForeColor,
(uint)ColorTranslator.ToWin32(Color.FromArgb(128,128,128)));
axEdit1.set_Background(EXEDITLib.BackgroundPartEnum.exInlineToolTipBackColor,
(uint)ColorTranslator.ToWin32(Color.FromArgb(240,240,240)));
axEdit1.CaretLine = 6;
axEdit1.set_Background(EXEDITLib.BackgroundPartEnum.exTempInlineToolTipBackCol
(uint)ColorTranslator.ToWin32(Color.FromArgb(255,40,0)));
axEdit1.set_Background(EXEDITLib.BackgroundPartEnum.exTempInlineToolTipForeCol
(uint)ColorTranslator.ToWin32(Color.FromArgb(0,0,1)));
axEdit1.TempInlineToolTip = "<br><c><font ;12>This is a bit of text that's shown
temporarily only. <br><c>Now, click the <off -4><b>class</b></off> keyword," +
" in the top...<br>";
axEdit1.Refresh();
```

X++ (Dynamics Ax 2009)

```
public void init()
{
   str var_s,var_s1;
   ;

   super();

   exedit1.LineNumberWidth(-1);
   exedit1.LineNumberBackColor(WinApi::RGB2int(240,240,240));
   var_s = "<r>a set or category of things having some property or attribute in
common and differentiated from others by kind, type, or qual";
```

```
   var_s = var_s + "ity.";
   exedit1.AddKeyword("<b>class</b>",COMVariant::createFromStr(var_s));
   exedit1.AllowInlineToolTip(513/*exInlineToolTipWordWrap | exInlineToolTip*/);

exedit1.Background(159/*exInlineToolTipForeColor*/,WinApi::RGB2int(128,128,128));

exedit1.Background(158/*exInlineToolTipBackColor*/,WinApi::RGB2int(240,240,240));

   exedit1.CaretLine(6);

exedit1.Background(160/*exTempInlineToolTipBackColor*/,WinApi::RGB2int(255,40,0))

   exedit1.Background(161/*exTempInlineToolTipForeColor*/,WinApi::RGB2int(0,0,1));
   var_s1 = "<br><c><font ;12>This is a bit of text that's shown temporarily only.
<br><c>Now, click the <off -4><b>class</b></off> keyword, ";
   var_s1 = var_s1 + "in the top...<br>";
   exedit1.TempInlineToolTip(var_s1);
   exedit1.Refresh();
}
```

Delphi 8 (.NET only)

```
with AxEdit1 do
begin
   LineNumberWidth := -1;
   LineNumberBackColor := Color.FromArgb(240,240,240);
   AddKeyword('<b>class</b>','<r>a set or category of things having some
property or attribute in common and differentiated from others by kind, type, or
qual' +
   'ity.',Nil,Nil);
   AllowInlineToolTip :=
Integer(EXEDITLib.AllowInlineToolTipEnum.exInlineToolTipWordWrap) Or
Integer(EXEDITLib.AllowInlineToolTipEnum.exInlineToolTip);

set_Background(EXEDITLib.BackgroundPartEnum.exInlineToolTipForeColor,$808080);

set_Background(EXEDITLib.BackgroundPartEnum.exInlineToolTipBackColor,$f0f0f0);
```

```
   CaretLine := 6;

set_Background(EXEDITLib.BackgroundPartEnum.exTempInlineToolTipBackColor,$28ff

set_Background(EXEDITLib.BackgroundPartEnum.exTempInlineToolTipForeColor,$1000

   TempInlineToolTip := '<br><c><font ;12>This is a bit of text that''s shown
temporarily only. <br><c>Now, click the <off -4><b>class</b></off> keyword, ' +
   'in the top...<br>';
   Refresh();
end
```

Delphi (standard)

```
with Edit1 do
begin
   LineNumberWidth := -1;
   LineNumberBackColor := RGB(240,240,240);
   AddKeyword('<b>class</b>','<r>a set or category of things having some
property or attribute in common and differentiated from others by kind, type, or
qual' +
   'ity.',Null,Null);
   AllowInlineToolTip := Integer(EXEDITLib_TLB.exInlineToolTipWordWrap) Or
Integer(EXEDITLib_TLB.exInlineToolTip);
   Background[EXEDITLib_TLB.exInlineToolTipForeColor] := $808080;
   Background[EXEDITLib_TLB.exInlineToolTipBackColor] := $f0f0f0;
   CaretLine := 6;
   Background[EXEDITLib_TLB.exTempInlineToolTipBackColor] := $28ff;
   Background[EXEDITLib_TLB.exTempInlineToolTipForeColor] := $10000;
   TempInlineToolTip := '<br><c><font ;12>This is a bit of text that''s shown
temporarily only. <br><c>Now, click the <off -4><b>class</b></off> keyword, ' +
   'in the top...<br>';
   Refresh();
end
```

VFP

```
with thisform.Edit1
   .LineNumberWidth = -1
   .LineNumberBackColor = RGB(240,240,240)
   var_s = "<r>a set or category of things having some property or attribute in common and differentiated from others by kind, type, or qual"
   var_s = var_s + "ity."
   .AddKeyword("<b>class</b>",var_s)
   .AllowInlineToolTip = 513 && AllowInlineToolTipEnum.exInlineToolTipWordWrap Or AllowInlineToolTipEnum.exInlineToolTip
   .Object.Background(159) = RGB(128,128,128)
   .Object.Background(158) = RGB(240,240,240)
   .CaretLine = 6
   .Object.Background(160) = RGB(255,40,0)
   .Object.Background(161) = RGB(0,0,1)
   var_s1 = "<br><c><font ;12>This is a bit of text that's shown temporarily only. <br><c>Now, click the <off -4><b>class</b></off> keyword, "
   var_s1 = var_s1 + "in the top...<br>"
   .TempInlineToolTip = var_s1
   .Refresh
endwith
```

dBASE Plus

```
local oEdit

oEdit = form.EXEDITACTIVEXCONTROL1.nativeObject
oEdit.LineNumberWidth = -1
oEdit.LineNumberBackColor = 0xf0f0f0
oEdit.AddKeyword("<b>class</b>","<r>a set or category of things having some property or attribute in common and differentiated from others by kind, type, or quality.")
oEdit.AllowInlineToolTip = 513 /*exInlineToolTipWordWrap | exInlineToolTip*/
oEdit.Template = [Background(159) = 8421504] // oEdit.Background(159) = 0x808080
oEdit.Template = [Background(158) = 15790320] // oEdit.Background(158) = 0xf0f0f0
oEdit.CaretLine = 6
oEdit.Template = [Background(160) = 10495] // oEdit.Background(160) = 0x28ff
```

```
oEdit.Template = [Background(161) = 65536] // oEdit.Background(161) = 0x10000
oEdit.TempInlineToolTip = "<br><c><font ;12>This is a bit of text that's shown temporarily only. <br><c>Now, click the <off -4><b>class</b></off> keyword, in the top...<br>"
oEdit.Refresh()
```

XBasic (Alpha Five)

```
Dim oEdit as P

oEdit = topparent:CONTROL_ACTIVEX1.activex
oEdit.LineNumberWidth = -1
oEdit.LineNumberBackColor = 15790320
oEdit.AddKeyword("<b>class</b>","<r>a set or category of things having some property or attribute in common and differentiated from others by kind, type, or quality.")
oEdit.AllowInlineToolTip = 513 'exInlineToolTipWordWrap + exInlineToolTip
oEdit.Template = "Background(159) = 8421504" // oEdit.Background(159) = 8421504
oEdit.Template = "Background(158) = 15790320" // oEdit.Background(158) = 15790320
oEdit.CaretLine = 6
oEdit.Template = "Background(160) = 10495" // oEdit.Background(160) = 10495
oEdit.Template = "Background(161) = 65536" // oEdit.Background(161) = 65536
oEdit.TempInlineToolTip = "<br><c><font ;12>This is a bit of text that's shown temporarily only. <br><c>Now, click the <off -4><b>class</b></off> keyword, in the top...<br>"
oEdit.Refresh()
```

Visual Objects

```
oDCOCX_Exontrol1:LineNumberWidth := -1
oDCOCX_Exontrol1:LineNumberBackColor := RGB(240,240,240)
oDCOCX_Exontrol1:AddKeyword("<b>class</b>","<r>a set or category of things having some property or attribute in common and differentiated from others by kind, type, or quality.",nil,nil)
```

```
oDCOCX_Exontrol1:AllowInlineToolTip := exInlineToolTipWordWrap | exInlineToolTip
oDCOCX_Exontrol1:[Background,exInlineToolTipForeColor] := RGB(128,128,128)
oDCOCX_Exontrol1:[Background,exInlineToolTipBackColor] := RGB(240,240,240)
oDCOCX_Exontrol1:CaretLine := 6
oDCOCX_Exontrol1:[Background,exTempInlineToolTipBackColor] := RGB(255,40,0)
oDCOCX_Exontrol1:[Background,exTempInlineToolTipForeColor] := RGB(0,0,1)
oDCOCX_Exontrol1:TempInlineToolTip := "<br><c><font ;12>This is a bit of text
that's shown temporarily only. <br><c>Now, click the <off -4><b>class</b></off>
keyword, in the top...<br>"
oDCOCX_Exontrol1:Refresh()
```

PowerBuilder

```
OleObject oEdit

oEdit = ole_1.Object
oEdit.LineNumberWidth = -1
oEdit.LineNumberBackColor = RGB(240,240,240)
oEdit.AddKeyword("<b>class</b>","<r>a set or category of things having some
property or attribute in common and differentiated from others by kind, type, or
quality.")
oEdit.AllowInlineToolTip = 513 /*exInlineToolTipWordWrap | exInlineToolTip*/
oEdit.Background(159,RGB(128,128,128))
oEdit.Background(158,RGB(240,240,240))
oEdit.CaretLine = 6
oEdit.Background(160,RGB(255,40,0))
oEdit.Background(161,RGB(0,0,1))
oEdit.TempInlineToolTip = "<br><c><font ;12>This is a bit of text that's shown
temporarily only. <br><c>Now, click the <off -4><b>class</b></off> keyword, in
the top...<br>"
oEdit.Refresh()
```

Visual DataFlex

```
Procedure OnCreate
   Forward Send OnCreate
```

```
   Set ComLineNumberWidth to -1
   Set ComLineNumberBackColor to (RGB(240,240,240))
   Send ComAddKeyword "<b>class</b>" "<r>a set or category of things having some property or attribute in common and differentiated from others by kind, type, or quality." Nothing Nothing
   Set ComAllowInlineToolTip to (OLEexInlineToolTipWordWrap + OLEexInlineToolTip)
   Set ComBackground OLEexInlineToolTipForeColor to (RGB(128,128,128))
   Set ComBackground OLEexInlineToolTipBackColor to (RGB(240,240,240))
   Set ComCaretLine to 6
   Set ComBackground OLEexTempInlineToolTipBackColor to (RGB(255,40,0))
   Set ComBackground OLEexTempInlineToolTipForeColor to (RGB(0,0,1))
   Set ComTempInlineToolTip to "<br><c><font ;12>This is a bit of text that's shown temporarily only. <br><c>Now, click the <off -4><b>class</b></off> keyword, in the top...<br>"
   Send ComRefresh
End_Procedure
```

XBase++

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

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

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

   oEdit := XbpActiveXControl():new( oForm:drawingArea )
   oEdit:CLSID  := "Exontrol.Edit.1" /*{39136531-DD0F-4281-B445-E36FC2CDDBC5}*/
   oEdit:create(,, {10,60},{610,370} )
```

```
      oEdit:LineNumberWidth := -1
      oEdit:SetProperty("LineNumberBackColor",AutomationTranslateColor(
GraMakeRGBColor  ( { 240,240,240 } )  , .F. ))
      oEdit:AddKeyword("<b>class</b>","<r>a set or category of things having
some property or attribute in common and differentiated from others by kind, type,
or quality.")
      oEdit:AllowInlineToolTip := 513/*exInlineToolTipWordWrap+exInlineToolTip*/

oEdit:SetProperty("Background",159/*exInlineToolTipForeColor*/,AutomationTranslate(
 GraMakeRGBColor  ( { 128,128,128 } )  , .F. ))

oEdit:SetProperty("Background",158/*exInlineToolTipBackColor*/,AutomationTranslate
 GraMakeRGBColor  ( { 240,240,240 } )  , .F. ))
      oEdit:CaretLine := 6

oEdit:SetProperty("Background",160/*exTempInlineToolTipBackColor*/,AutomationTran
 GraMakeRGBColor  ( { 255,40,0 } )  , .F. ))

oEdit:SetProperty("Background",161/*exTempInlineToolTipForeColor*/,AutomationTran
 GraMakeRGBColor  ( { 0,0,1 } )  , .F. ))
      oEdit:TempInlineToolTip := "<br><c><font ;12>This is a bit of text that's shown
temporarily only. <br><c>Now, click the <off -4><b>class</b></off> keyword, in
the top...<br>"
      oEdit:Refresh()

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

property Edit.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.*

Here's a simple template code:

```
'Defines the items in the control's context list
Context
{
   Add("<fgcolor=000080>keyword</fgcolor>")
}

' Defines the keywords
AddKeyword("<fgcolor=008080>keyword</fgcolor>", "description of the tooltip", "title
of the tooltip")

' Defines the expressions
AddExpression("<fgcolor=008000>start</fgcolor>", "<b><fgcolor=008000>
</fgcolor></b>", "<fgcolor=008000>end</fgcolor>")
```

property Edit.TemplateDef as Variant

Defines inside variables for the next Template/ExecuteTemplate call.

Type	Description
Variant	A string expression that indicates the template script.

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

method Edit.TemplatePut (NewVal as Variant)

Defines inside variables for the next Template/ExecuteTemplate call.

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

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

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

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

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

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

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

The x-script may uses constant expressions as follow:

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

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

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

property Edit.Text as String

Specifies the control's text.

Type	Description
String	A string expression that defines the control's text.

Use the Text property to access the control's text. Use the HTMLText property to retrieve the MS HTML text form the control's text. Use the Load/LoadFromUnicode method to load the control's text from a file. Use the Save/SaveAsUnicode method to save the control's text to a file. Use the InsertText method to insert programmatically new lines to the control. Use the DeleteLine method to delete programmatically a line based on its index. The Count property counts the lines in the control. Use the SelText property to insert text at cursor position. Use the CaretLine property to specify the index of the current line.

property Edit.TextLine(Index as Long) as String

Specifies the line based on its index.

Type	Description
Index as Long	A long expression that defines the index of line being accessed. The Index is 1 based.
String	A string expression that defines the line's text.

Use the TextLine property to access a particular line in the control's text. Use the Text property to access the control's text when MultiLine property is False. Use the DeleteLine method to delete a particular line. Passing an empty string to the TextLine property doesn't remove the line. It just erases the line's content. The Count property counts the number of lines in the control.

The following VB sample prints the line in the control:

```
With Edit1
   Dim i As Long
   For i = 1 To .Count
      Debug.Print .TextLine(i)
   Next
End With
```

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

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

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

```
With AxEdit1
   Dim i As Integer
   For i = 1 To .Count
      Debug.WriteLine(.get_TextLine(i))
   Next
End With
```

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

```
for (int i = 1; i <= axEdit1.Count; i++)
```

```
System.Diagnostics.Debug.WriteLine(axEdit1.get_TextLine(i));
```

The following VFP sample prints the line in the control:

```
with thisform.Edit1.Object
   local i
   for i = 1 to .Count
      wait window nowait .TextLine(i)
   next
endwith
```

property Edit.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 AddKeyword method to specify the tooltip description for a keyword. 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 Edit.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 AddKeyword method to specify the tooltip description for a keyword. 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 Edit.ToolTipMargin as String

Defines the size of the control's tooltip margins.

Type	Description
String	A string expression that defines the horizontal and vertical margins (separated by comma) of the control's tooltip as one of the following formats: • "value", where value is a positive number, that specifies the horizontal and vertical margins, such as "4" equivalent of "4,4" • "value,", where value is a positive number, that specifies the horizontal margin, such as "4," equivalent of "4,0" • ",value", where value is a positive number, that specifies the vertical margin, such as ",4" equivalent of "0,4" • "horizontal,vertical", where horizontal and vertical are positive numbers, that specifies the horizontal and vertical margins, such as "4,4"

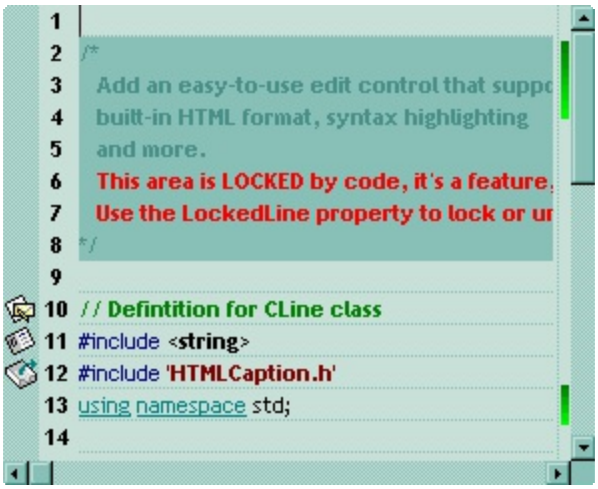
By default, the size of the tooltip margin is "4" (horizontal and vertical). For instance, ToolTipMargin = "8" changes the horizontal and vertical margins are set to 8 pixels. ToolTipMargin = "8,4" changes the horizontal margin to 8 pixels and the vertical margin to 4 pixels. The ToolTipWidth property specifies a value that indicates the width of the tooltip window, in pixels. Use the ShowToolTip method to display a custom tooltip. Use the ToolTipFont property to assign a font for the control's tooltip. 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.

property Edit.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 AddKeyword method to add new keywords to 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. The AllowInlineToolTip property indicates that the current line shows inline tooltip for keywords found to the caret.



property Edit.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.

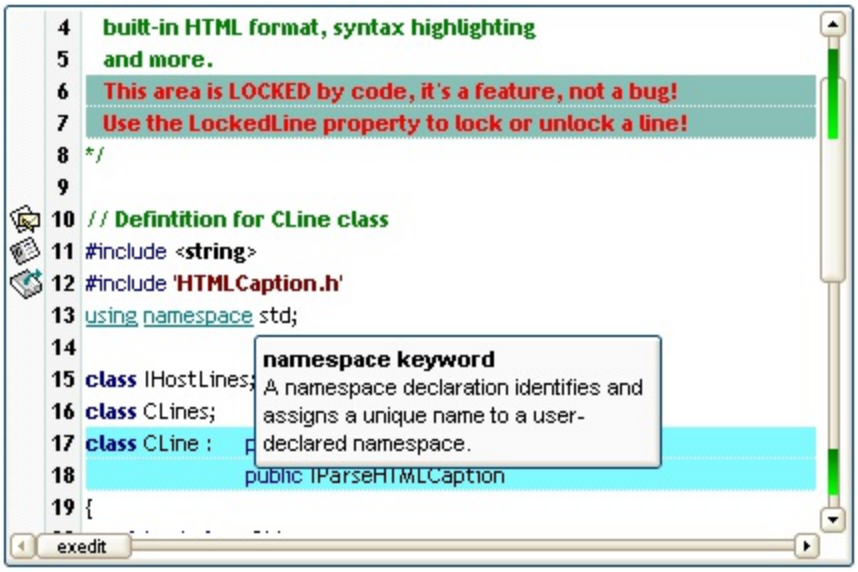
By default, the ToolTipPopDelay property is 5000 ms. If the ToolTipPopDelay property is 0, the control displays no tooltips for any keyword. Use the ToolTipDelay and ToolTipPopDelay properties to define the time before showing a tooltip and the period of the time that tooltip remains visible if the mouse pointer is stationary within a control. Use the AddKeyword method to specify the tooltip description for a keyword. 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 Edit.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 AddKeyword method to assign a tooltip to a keyword. 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 Edit.Undo ()

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

Type	Description

Use the Undo method to undo the last edit-control operation. Use the CanUndo property to specify whether an undo operation is available. Use the Redo method to redo the last edit-control operation. Use the AllowContextMenu property to specify whether the control displays a context menu when user right clicks the control.

property Edit.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 Edit.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 Edit.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 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 Background property to change the visual appearance for different parts of the control.

property Edit.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.

EditContext object

Use the Context property to access the control's Context collection. Use the ContextKey property to define the key combination for the control's context window. The OnContext event occurs when user invokes the control's context window.

The Context object supports the following methods and properties:

Name	Description
Add	Adds a new item to the context.
Clear	Removes all members of context collection.
Count	Returns the number of objects in a collection.
Options	Specifies an option for the sensitive context list.
Remove	Removes a specific member from the context collection.
Sort	Sorts the items.

method EditContext.Add (HTMLItem as String, [Item as Variant], [Image as Variant], [SortOrder as Variant])

Adds a new item to the control's context list.

Type	Description
HTMLItem as String	A string expression that may include HTML built- in tags that is displayed in the context popup list.
Item as Variant	A string expression that specifies the text being inserted in the control's text when the user selects the item in the context popup menu. If missing, the HTMLItem with no HTML tags is being used.
Image as Variant	A long expression that indicates the index of image in the item.
SortOrder as Variant	A long expression that indicates the order of the item if the list gets sorted.

Use the Add method to add new items to the control's context collection. The control provides a collection called Context that helps you to add new items to context drop down portion. You can invoke the drop down context window, by pressing CTRL+SPACE. If you want to provide icons for items in the drop down portion, you have to use the Images method. Use the Sort method to get the items sorted. You can use the OnContext event to add new items to the Context collection when user invokes the control's context window.

The following VB sample adds few items to control's drop down portion:

```
With Edit1.Context("")
   .Add "<fgcolor=FF0000>class</fgcolor>", , 1
   .Add "<fgcolor=0000FF><b>public</fgcolor>", , 2
   .Add "<fgcolor=0000FF><b>protected</fgcolor>"
   .Add "<fgcolor=0000FF><b>private</fgcolor>", , 3
End With
```

The following C++ sample adds few items to the control's drop down portion:

```
#include "Context.h"
COleVariant vtMissing; vtMissing.vt = VT_ERROR;
CContext context = m_edit.GetContext( vtMissing );
context.Add("<fgcolor=FF0000>class</fgcolor>", vtMissing , COleVariant( long(1) ),
vtMissing );
```

```
context.Add("<fgcolor=0000FF><b>public</fgcolor>", vtMissing , COleVariant( long(2) ), vtMissing );
context.Add("<fgcolor=0000FF><b>protected</fgcolor>", vtMissing , vtMissing, vtMissing );
context.Add("<fgcolor=0000FF><b>private</fgcolor>", vtMissing , COleVariant( long(3) ), vtMissing );
```

The following VB.NET sample adds few items to the control's drop down portion:

```
With AxEdit1.get_Context("")
   .Add("<fgcolor=FF0000>class</fgcolor>", , 1)
   .Add("<fgcolor=0000FF><b>public</fgcolor>", , 2)
   .Add("<fgcolor=0000FF><b>protected</fgcolor>")
   .Add("<fgcolor=0000FF><b>private</fgcolor>", , 3)
End With
```

The following C# sample adds few items to the control's drop down portion:

```
EXEDITLib.Context context = axEdit1.get_Context("");
context.Add("<fgcolor=FF0000>class</fgcolor>", null , 1, null );
context.Add("<fgcolor=0000FF><b>public</fgcolor>", null , 2, null);
context.Add("<fgcolor=0000FF><b>protected</fgcolor>", null, null, null);
context.Add("<fgcolor=0000FF><b>private</fgcolor>", null, 3, null);
```

The following VFP sample adds few items to the control's drop down portion:

```
With thisform.Edit1.Context("")
   .Add("<fgcolor=FF0000>class</fgcolor>", , 1)
   .Add("<fgcolor=0000FF><b>public</fgcolor>", , 2)
   .Add("<fgcolor=0000FF><b>protected</fgcolor>")
   .Add("<fgcolor=0000FF><b>private</fgcolor>", , 3)
EndWith
```

method EditContext.Clear ()

Removes all members of context collection.

Type	Description

Use the Clear method to clear the control's context list. The context popup list is not displayed if no items is in the Context collection. Use the Remove method to remove a specific item.

property EditContext.Count as Long

Returns the number of objects in a collection.

Type	Description
Long	A long expression that indicates the number of elements in a collection.

Use the Count property to count the number of elements in the control's context list.

property EditContext.Options(Option as ContextOptionEnum) as Variant

Specifies an option for the sensitive context list.

Type	Description
Option as ContextOptionEnum	A ContextOptionEnum expression that indicates the option being changed.
Variant	A variant value that indicates the value for the specified option.

Use the Options property to specify options for the control's context list. Use the Add method to add new items to the control's context list. Use the Remove method to remove an item from the control's context list. Use the Sort method to sort the items in the control's context list.

- The Context.Options(**exContextCaseSensitive**) as Boolean specifies whether the selection of items into a sensitive context list is case sensitive. By default, the Context.Options(exContextCaseSensitive) is False. For instance, if Context.Options(exContextCaseSensitive) is True, the selection of items from a context list is case sensitive.
- The Context.Options(**exContextAllowChars**) as String specifies the list of characters that are allowed to a context list. By default, the Context.Options(exContextAllowChars) property is "_ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz". For instance, if user presses any of characters that are not included to Context.Options(exContextAllowChars) property the context list is closed.

method EditContext.Remove (Index as Variant)

Removes a specific member from the context collection.

Type	Description
Index as Variant	A long expression that indicates the index of item being deleted.

Use the Remove method to remove an item for control's context list. Use the Clear method to clear the control's context list.

method EditContext.Sort (Ascending as Boolean)

Sorts the items.

Type	Description
Ascending as Boolean	A boolean value that indicates whether the items get sorted ascendant or descendent.

Use the Add method of the Context object to specifies the SortOrder number. All items that are added using the Add method without specifying the SortOrder parameter, have 0 for the SortOrder by default. The items are sorted by SortOrder number. The Sort method sorts alphabetically the list of items that have the same SortOrder parameter.

The following VB sample sorts the list of items as follows:

```
With Edit1.Context
    .Add "<b>a</b>", , , 9
    .Add "<b>b</b>", , , 8
    .Add "<b>c</b>", , , 7

    .Sort True
End With
```

and the list of items is 'c', 'b', 'a'.

The following VB sample sorts the list as follows:

```
With Edit1.Context
    .Add "<b>a</b>"
    .Add "<b>b</b>"
    .Add "<b>c</b>"

    .Sort True
End With
```

and the list of items is 'a', 'b', 'c'.

```
With Edit1.Context
    .Add "<b>a</b>"
    .Add "<b>b</b>", , , 9999
    .Add "<b>c</b>"
```

```
    .Sort True
End With
```

and the list of items is 'a', 'c', 'b'.

The following C++ sample sorts the list of items as follows:

```
#include "Context.h"
COleVariant vtMissing; vtMissing.vt = VT_ERROR;
CContext context = m_edit.GetContext( vtMissing );
context.Add("<b>a</b>", vtMissing , vtMissing, COleVariant( long(9) ) );
context.Add("<b>b</b>", vtMissing , vtMissing, COleVariant( long(8) ) );
context.Add("<b>c</b>", vtMissing , vtMissing, COleVariant( long(7) ) );
context.Sort( TRUE );
```

and the list of items is 'c', 'b', 'a'.

The following VB.NET sample sorts the list of items as follows:

```
With AxEdit1.get_Context("")
   .Add("<b>a</b>", , , 9)
   .Add("<b>b</b>", , , 8)
   .Add("<b>c</b>", , , 7)
   .Sort(True)
End With
```

and the list of items is 'c', 'b', 'a'.

The following C# sample sorts the list of items as follows:

```
EXEDITLib.Context context = axEdit1.get_Context("");
context.Add("<b>a</b>", null , null, 9 );
context.Add("<b>b</b>", null, null, 8);
context.Add("<b>c</b>", null, null, 7);
context.Sort(true);
```

and the list of items is 'c', 'b', 'a'.

The following VFP sample sorts the list of items as follows:

```
With thisform.Edit1.Context()
    .Add("<b>a</b>", , , 9)
    .Add("<b>b</b>", , , 8)
    .Add("<b>c</b>", , , 7)
    .Sort(.t.)
EndWith
```

and the list of items is 'c', 'b', 'a'.

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 ExDataObjectFiles 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 ExDataObject'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.GetFormat (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 ExDataObject'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 Edit1_OLEDragDrop(ByVal Data As EXEDITLibCtl.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
      Edit1.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 OnOLEDragDropEdit1(LPDISPATCH Data, long FAR* Effect, short Button, short Shift,
long X, long Y)
{
  if ( EXEDITLib::IExDataObjectPtr spData( Data ) )
    if ( spData->GetFormat( EXEDITLib::exCFFiles ) )
```

```
    {
        EXEDITLib::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 <exedit.dll> statement. The #import <exedit.dll> includes the EXEDITLib 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 AxEdit1_OLEDragDrop(ByVal sender As Object, ByVal e As
AxEXEDITLib._IEditEvents_OLEDragDropEvent) Handles AxEdit1.OLEDragDrop
    If (e.data.GetFormat(EXEDITLib.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
            AxEdit1.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 axEdit1_OLEDragDrop(object sender,
AxEXEDITLib._IEditEvents_OLEDragDropEvent e)
{
    if (e.data.GetFormat(Convert.ToInt16(EXEDITLib.exClipboardFormatEnum.exCFFiles)))
    {
```

```
        EXEDITLib.IExDataObjectFiles files = e.data.Files;
        if (files != null)
        {
            string s = "";
            for (int i = 0; i < files.Count; i++)
                s += files[i] + "\r\n";
            axEdit1.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.Edit1.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 ExDataObject 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 Edit1_OLEStartDrag(ByVal Data As EXEDITLibCtl.IExDataObject, AllowedEffects
As Long)
   Dim s As String
   s = Edit1.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 OnOLEStartDragEdit1(LPDISPATCH Data, long FAR* AllowedEffects)
{
   CString s( m_edit.GetSelText() );
   if ( s.GetLength() > 0 )
   {
      *AllowedEffects = 1; /*exOLEDropEffectCopy*/
      if ( EXEDITLib::IExDataObjectPtr spData( Data ) )
         spData->SetData( COleVariant( s ), COleVariant( long(EXEDITLib::exCFText) ) );

   }
}
```

The C++ sample uses the #import <exedit.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 AxEdit1_OLEStartDrag(ByVal sender As Object, ByVal e As
AxEXEDITLib._IEditEvents_OLEStartDragEvent) Handles AxEdit1.OLEStartDrag
   Dim s As String = AxEdit1.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 axEdit1_OLEStartDrag(object sender,
AxEXEDITLib._IEditEvents_OLEStartDragEvent e)
{
   string s = axEdit1.SelText;
   if (s.Length > 0)
   {
      e.allowedEffects = 1;
      e.data.SetData(s, EXEDITLib.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.Edit1
   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 Edit1_OLEDragDrop(ByVal Data As EXEDITLibCtl.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
         Edit1.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 OnOLEDragDropEdit1(LPDISPATCH Data, long FAR* Effect, short Button, short Shift,
long X, long Y)
{
   if ( EXEDITLib::IExDataObjectPtr spData( Data ) )
      if ( spData->GetFormat( EXEDITLib::exCFFiles ) )
      {
```

```
        EXEDITLib::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 <exedit.dll> statement. The #import <exedit.dll> includes the EXEDITLib 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 AxEdit1_OLEDragDrop(ByVal sender As Object, ByVal e As
AxEXEDITLib._IEditEvents_OLEDragDropEvent) Handles AxEdit1.OLEDragDrop
    If (e.data.GetFormat(EXEDITLib.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
            AxEdit1.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 axEdit1_OLEDragDrop(object sender,
AxEXEDITLib._IEditEvents_OLEDragDropEvent e)
{
    if (e.data.GetFormat(Convert.ToInt16(EXEDITLib.exClipboardFormatEnum.exCFFiles)))
    {
        EXEDITLib.IExDataObjectFiles files = e.data.Files;
```

```
        if (files != null)
        {
            string s = "";
            for (int i = 0; i < files.Count; i++)
                s += files[i] + "\r\n";
            axEdit1.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.Edit1.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 Edit1_OLEDragDrop(ByVal Data As EXEDITLibCtl.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
         Edit1.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 OnOLEDragDropEdit1(LPDISPATCH Data, long FAR* Effect, short Button, short Shift,
long X, long Y)
{
   if ( EXEDITLib::IExDataObjectPtr spData( Data ) )
      if ( spData->GetFormat( EXEDITLib::exCFFiles ) )
      {
         EXEDITLib::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 <exedit.dll> statement. The #import <exedit.dll> includes the EXEDITLib 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 AxEdit1_OLEDragDrop(ByVal sender As Object, ByVal e As
AxEXEDITLib._IEditEvents_OLEDragDropEvent) Handles AxEdit1.OLEDragDrop
    If (e.data.GetFormat(EXEDITLib.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
            AxEdit1.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 axEdit1_OLEDragDrop(object sender,
AxEXEDITLib._IEditEvents_OLEDragDropEvent e)
{
    if (e.data.GetFormat(Convert.ToInt16(EXEDITLib.exClipboardFormatEnum.exCFFiles)))
    {
        EXEDITLib.IExDataObjectFiles files = e.data.Files;
        if (files != null)
```

```
        {
            string s = "";
            for (int i = 0; i < files.Count; i++)
                s += files[i] + "\r\n";
            axEdit1.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.Edit1.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.

ExEdit events

The ExEdit component supports the following events:

Name	Description
BookmarkChange	Occurs when the user adds or removes bookmarks.
Change	Indicates that the control's text have changed.
Click	Occurs when the user presses and then releases the left mouse button over the list control.
DblClick	Occurs when the user double clicks the left mouse button over an object.
Event	Notifies the application once the control fires an event.
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.
QueryContext	Queries for the context at the specified location, to provide different tooltips for the same keyword on QueryContextToolTip event.
QueryContextToolTip	Asks for the tooltip/title of the keyword on the context retrieved by the QueryContext event.
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 BookmarkChange (Index as Long)

Occurs when the user adds or removes bookmarks.

Type	Description
Index as Long	A long expression that indicates the line being changed.

The BokmarkChange event notifies your application that user changes the line's bookmark. The BookmarkWidth property specifies whether the bookmarks header is visible or hidden. The Bookmark property specifies whether a line has a bookmark. The BookmarksList property specifies the lines with bookmarks. Use the TextLine property to retrieve a line by its index. Use the NextBookmark method to move the cursor to the next bookmark. Use the PrevBookmark method to move the cursor to the prior bookmark.

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

C#

```
private void BookmarkChange(object sender,int Index)
{
}
```

VB

```
Private Sub BookmarkChange(ByVal sender As System.Object,ByVal Index As
Integer) Handles BookmarkChange
End Sub
```

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

C#

```
private void BookmarkChange(object sender,
AxEXEDITLib._IEditEvents_BookmarkChangeEvent e)
{
}
```

C++

```
void OnBookmarkChange(long Index)
{
}
```

C++ Builder

```
void __fastcall BookmarkChange(TObject *Sender,long Index)
{
}
```

Delphi

```
procedure BookmarkChange(ASender: TObject; Index : Integer);
```

```
begin
end;
```

Delphi 8 (.NET only)

```
procedure BookmarkChange(sender: System.Object; e:
AxEXEDITLib._IEditEvents_BookmarkChangeEvent);
begin
end;
```

Powe…

```
begin event BookmarkChange(long Index)
end event BookmarkChange
```

VB.NET

```
Private Sub BookmarkChange(ByVal sender As System.Object, ByVal e As
AxEXEDITLib._IEditEvents_BookmarkChangeEvent) Handles BookmarkChange
End Sub
```

VB6

```
Private Sub BookmarkChange(ByVal Index As Long)
End Sub
```

VBA

```
Private Sub BookmarkChange(ByVal Index As Long)
End Sub
```

VFP

```
LPARAMETERS Index
```

Xbas…

```
PROCEDURE OnBookmarkChange(oEdit,Index)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComBookmarkChange Integer llIndex
   Forward Send OnComBookmarkChange llIndex
End_Procedure
```

Visual Objects

```
METHOD OCX_BookmarkChange(Index) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_BookmarkChange(int _Index)
{
}
```

XBasic

```
function BookmarkChange as v (Index as N)
end function
```

dBASE

```
function nativeObject_BookmarkChange(Index)
return
```

The following VB sample displays the lines with bookmarks:

```
Private Sub Edit1_BookmarkChange(ByVal Index As Long)
   With Edit1
      Dim i As Long, b As Variant
      b = .BookmarksList
      For i = LBound(b) To UBound(b)
         Debug.Print .TextLine(b(i))
      Next
   End With
End Sub
```

The following C++ sample displays the lines with bookmarks:

```
void OnBookmarkChangeEdit1(long Index)
{
   COleVariant vtBookmarks = m_edit.GetBookmarksList();
   SAFEARRAY* pBookmarks = V_ARRAY( &vtBookmarks );
   long nDim = 0;
```

```
    for ( long i = pBookmarks->rgsabound[nDim].lLbound; i < (long)pBookmarks->rgsabound[nDim].cElements; i++ )
    {
        long nLine = 0;
        SafeArrayGetElement( pBookmarks, &i, &nLine );
        OutputDebugString( m_edit.GetTextLine( nLine ) );
    }
}
```

The following VB.NET sample displays the lines with bookmarks:

```
Private Sub AxEdit1_BookmarkChange(ByVal sender As Object, ByVal e As AxEXEDITLib._IEditEvents_BookmarkChangeEvent) Handles AxEdit1.BookmarkChange
    With AxEdit1
        Dim i As Integer, b As Integer() = .BookmarksList
        For Each i In b
            Debug.WriteLine(.get_TextLine(i))
        Next
    End With
End Sub
```

or

```
Private Sub AxEdit1_BookmarkChange(ByVal sender As Object, ByVal e As AxEXEDITLib._IEditEvents_BookmarkChangeEvent) Handles AxEdit1.BookmarkChange
    With AxEdit1
        Dim i As Long, b As Object
        b = .BookmarksList
        For i = LBound(b) To UBound(b)
            Debug.WriteLine(.get_TextLine(b(i)))
        Next
    End With
End Sub
```

The following C# sample displays the lines with bookmarks:

```
private void axEdit1_BookmarkChange(object sender, AxEXEDITLib._IEditEvents_BookmarkChangeEvent e)
{
```

```
    int[] b = axEdit1.BookmarksList as int[];
    foreach (int i in b)
        System.Diagnostics.Debug.WriteLine(axEdit1.get_TextLine(i));
}
```

The following VFP sample displays the lines with bookmarks:

```
*** ActiveX Control Event ***
LPARAMETERS index

with thisform.Edit1
    local b, i
    b = .BookmarksList
    for i = 1 to alen(b)
        wait window .TextLine(b[i])
    next
endwith
```

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 in the control. Use the Text property to access the control's text. Use the InsertText and DeleteLine methods to update the control's text. Use the AddKeyword method to add keywords to the control's content. Use the AddExpression method to add new expressions to the control's content. Use the CaretLine property to determine the position of the caret. Use the TextLine property to access a line giving its index. The ChangeOnKey property specifies the code of the last key that alters the control's text.

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(oEdit)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComChange
   Forward Send OnComChange
```

```
End_Procedure
```

Visual Objects

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

X++

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

XBasic

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

dBASE

```
function nativeObject_Change()
return
```

The following VB sample displays the line where the change occurs:

```
Private Sub Edit1_Change()
   With Edit1
      Debug.Print .TextLine(.CaretLine)
   End With
End Sub
```

The following C++ sample displays the line where the change occurs:

```
void OnChangeEdit1()
{
   if ( ::IsWindow( m_edit.m_hWnd ) )
      OutputDebugString( m_edit.GetTextLine( m_edit.GetCaretLine() ) );
}
```

The following VB.NET sample displays the line where the change occurs:

```
Private Sub AxEdit1_Change(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles AxEdit1.Change
   With AxEdit1
      Debug.WriteLine(.get_TextLine(.CaretLine))
   End With
```

```
End Sub
```

The following C# sample displays the line where the change occurs:

```
private void axEdit1_Change(object sender, EventArgs e)
{
    System.Diagnostics.Debug.WriteLine(axEdit1.get_TextLine(axEdit1.CaretLine));
}
```

The following VFP sample displays the line where the change occurs:

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

with thisform.Edit1
    wait window nowait .TextLine(.CaretLine)
endwith
```

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. Use the ClientToCursor property and MouseDown event to get the line from 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(oEdit)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComClick
   Forward Send OnComClick
```

```
End_Procedure
```

Visual Objects

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

X++

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

XBasic

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

dBASE

```
function nativeObject_Click()
return
```

event DblClick (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 DblClick 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. Use the ClientToCursor method to convert client coordinates to cursor coordinates. Use the TextLine property to retrieve the line giving its index.

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

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

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

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

C#
```
private void DblClick(object sender, AxEXEDITLib._IEditEvents_DblClickEvent e)
{
}
```

C++
```
void OnDblClick(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:
AxEXEDITLib._IEditEvents_DblClickEvent);
begin
end;
```

Powe…

```
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
AxEXEDITLib._IEditEvents_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
```

Xbas…

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

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComDblClick Short llShift OLE_XPOS_PIXELS llX OLE_YPOS_PIXELS llY
   Forward Send OnComDblClick llShift llX llY
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.Edit.1::OLE_XPOS_PIXELS,Y as OLE::Exontrol.Edit.1::OLE_YPOS_PIXELS)
end function
```

dBASE

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

The following VB sample displays the line double clicked:

```
Private Sub Edit1_DblClick(Shift As Integer, X As Single, Y As Single)
   With Edit1
      Dim i As Long, c As Long
      .ClientToCursor X / Screen.TwipsPerPixelX, Y / Screen.TwipsPerPixelY, c, i
      If (i > 0) Then
         Debug.Print .TextLine(i)
      End If
   End With
End Sub
```

The following C++ sample displays the line that user double clicks:

```
void OnDblClickEdit1(short Shift, long X, long Y)
{
   long nLine = 0, nColumn = 0;
   m_edit.ClientToCursor( X, Y, &nColumn, &nLine );
   if ( nLine > 0 )
      OutputDebugString( m_edit.GetTextLine( nLine ) );
}
```

The following VB.NET sample displays the line that user double clicks:

```
Private Sub AxEdit1_DblClick(ByVal sender As Object, ByVal e As
AxEXEDITLib._IEditEvents_DblClickEvent) Handles AxEdit1.DblClick
   With AxEdit1
      Dim i As Integer, c As Integer
      .ClientToCursor(e.x, e.y, c, i)
      If (i > 0) Then
         Debug.WriteLine(.get_TextLine(i))
      End If
   End With
End Sub
```

The following C# sample displays the line that user double clicks:

```
private void axEdit1_DblClick(object sender, AxEXEDITLib._IEditEvents_DblClickEvent e)
{
   int iLine = 0, iColumn = 0;
   axEdit1.ClientToCursor(e.x, e.y, out iColumn, out iLine);
   if ( iLine > 0 )
      System.Diagnostics.Debug.WriteLine(axEdit1.get_TextLine(iLine));
}
```

The following VFP sample displays the line that user double clicks:

```
*** ActiveX Control Event ***
LPARAMETERS shift, x, y

with thisform.Edit1
   local i, c
```

```
    i = 0
    c = 0
    .ClientToCursor(x, y, @c, @i)
    If (i > 0) Then
        wait window nowait .TextLine(i)
    EndIf
endwith
```

event Event (EventID as Long)

Notifies the application once the control fires an event.

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

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

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

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

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

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

```
// Notifies the application once the control fires an event.
void onEvent_Event(int _EventID)
{
   print exgantt1.EventParam(-2).toString();
}
```

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

```
MouseDown/-605( 1 , 0 , 102 , 644 )
SelChange/2
MouseMove/-606( 1 , 0 , 105 , 646 )
```

MouseUp/-607(1 , 0 , 128 , 682)
Click/-600

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

C#
```
private void Event(object sender,int   EventID)
{
}
```

VB
```
Private Sub Event(ByVal sender As System.Object,ByVal EventID As Integer)
Handles Event
End Sub
```

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

C#
```
private void Event(object sender, AxEXEDITLib._IEditEvents_EventEvent e)
{
}
```

C++
```
void OnEvent(long   EventID)
{
}
```

C++ Builder
```
void __fastcall Event(TObject *Sender,long   EventID)
{
}
```

Delphi
```
procedure Event(ASender: TObject; EventID : Integer);
begin
end;
```

Delphi 8 (.NET only)
```
procedure Event(sender: System.Object; e: AxEXEDITLib._IEditEvents_EventEvent);
begin
end;
```

Powe…
```
begin event Event(long  EventID)

end event Event
```

VB.NET

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

VB6

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

VBA

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

VFP

```
LPARAMETERS EventID
```

Xbas…

```
PROCEDURE OnEvent(oEdit,EventID)

RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComEvent Integer   llEventID
   Forward Send OnComEvent llEventID
End_Procedure
```

Visual Objects

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

X++

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

XBasic

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

dBASE

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

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 us: 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,
AxEXEDITLib._IEditEvents_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:
AxEXEDITLib._IEditEvents_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
AxEXEDITLib._IEditEvents_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(oEdit,Item,Position)
RETURN
```

Syntax for ExecuteContextMenu event, **/COM** 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 llItem Integer llPosition
    Forward Send OnComExecuteContextMenu llItem llPosition
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 Edit1
      .AllowContextMenu = True
      .ContextMenuItems = vbCr & "Item 1" & vbCr & vbCr & "Item 2" & vbCr & "Item 3"
   End With
End Sub
```

```
Private Sub Edit1_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 OnExecuteContextMenuEdit1(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 AxEdit1
   .AllowContextMenu = True
   .ContextMenuItems = vbCr & "Item 1" & vbCr & vbCr & "Item 2" & vbCr & "Item 3"
End With
```

```
Private Sub AxEdit1_ExecuteContextMenu(ByVal sender As Object, ByVal e As
AxEXEDITLib._IEditEvents_ExecuteContextMenuEvent) Handles
AxEdit1.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:

```
axEdit1.AllowContextMenu = true;
axEdit1.ContextMenuItems = "\rItem1\r\rItem2\rItem3";
```

```
private void axEdit1_ExecuteContextMenu(object sender,
AxEXEDITLib._IEditEvents_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.Edit1
   .AllowContextMenu = .t.
   .ContextMenuItems = 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, AxEXEDITLib._IEditEvents_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:
AxEXEDITLib._IEditEvents_KeyDownEvent);
begin
end;
```

Powe…

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

VB.NET

```
Private Sub KeyDownEvent(ByVal sender As System.Object, ByVal e As
AxEXEDITLib._IEditEvents_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(oEdit,KeyCode,Shift)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComKeyDown Short llKeyCode Short llShift
   Forward Send OnComKeyDown llKeyCode llShift
End_Procedure
```

Visual Objects

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

X++

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

XBasic

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

dBASE

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

event KeyPress (ByRef KeyAscii as Integer)

Occurs when the user presses and releases an ANSI key.

Type	Description
KeyAscii as Integer	(By Reference) An integer that returns a standard numeric ANSI keycode

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

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

C#

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

VB

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

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

C#

```
private void KeyPressEvent(object sender, AxEXEDITLib._IEditEvents_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:
AxEXEDITLib._IEditEvents_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
AxEXEDITLib._IEditEvents_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(oEdit,KeyAscii)
RETURN
```

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

Java…

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

VBSc…

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

```
</SCRIPT>
```

Visual Data…

```
Procedure OnComKeyPress Short llKeyAscii
    Forward Send OnComKeyPress llKeyAscii
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, AxEXEDITLib._IEditEvents_KeyUpEvent e)
{
}
```

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

C++ Builder
```
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:
AxEXEDITLib._IEditEvents_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
AxEXEDITLib._IEditEvents_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(oEdit,KeyCode,Shift)
RETURN
```

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

Java…

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

VBSc…

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

```
</SCRIPT>
```

Visual Data…

```
Procedure OnComKeyUp Short llKeyCode Short llShift
    Forward Send OnComKeyUp llKeyCode llShift
End_Procedure
```

Visual Objects

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

X++

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

XBasic

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

dBASE

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

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

Occurs when the user presses a mouse button.

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

Use a MouseDown or MouseUp event procedure to specify actions that will occur when a mouse button is pressed or released. Unlike the Click and DblClick 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. Use the ClientToCursor method to convert client coordinates to cursor coordinates. Use the TextLine property to retrieve the line giving its index.

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,
AxEXEDITLib._IEditEvents_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:
AxEXEDITLib._IEditEvents_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
AxEXEDITLib._IEditEvents_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(oEdit,Button,Shift,X,Y)
```

```
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComMouseDown Short llButton Short llShift OLE_XPOS_PIXELS llX
OLE_YPOS_PIXELS llY
   Forward Send OnComMouseDown llButton llShift llX llY
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.Edit.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.Edit.1::OLE_YPOS_PIXELS)
end function
```

dBASE

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

The following VB sample displays the line from the cursor:

```
Private Sub Edit1_MouseDown(Button As Integer, Shift As Integer, x As Single, y As Single)
   With Edit1
      Dim i As Long, c As Long
```

```
        .ClientToCursor x / Screen.TwipsPerPixelX, y / Screen.TwipsPerPixelY, c, i
        If (i > 0) Then
            Debug.Print .TextLine(i)
        End If
    End With
End Sub
```

The following C++ sample displays the line from the cursor:

```
void OnMouseDownEdit1(short Button, short Shift, long X, long Y)
{
    long nLine = 0, nColumn = 0;
    m_edit.ClientToCursor( X, Y, &nColumn, &nLine );
    if ( nLine > 0 )
        OutputDebugString( m_edit.GetTextLine( nLine ) );
}
```

The following VB.NET sample displays the line from the cursor:

```
Private Sub AxEdit1_MouseDownEvent(ByVal sender As Object, ByVal e As
AxEXEDITLib._IEditEvents_MouseDownEvent) Handles AxEdit1.MouseDownEvent
    With AxEdit1
        Dim i As Integer, c As Integer
        .ClientToCursor(e.x, e.y, c, i)
        If (i > 0) Then
            Debug.WriteLine(.get_TextLine(i))
        End If
    End With
End Sub
```

The following C# sample displays the line from the cursor:

```
private void axEdit1_MouseDownEvent(object sender,
AxEXEDITLib._IEditEvents_MouseDownEvent e)
{
    int iLine = 0, iColumn = 0;
    axEdit1.ClientToCursor(e.x, e.y, out iColumn, out iLine);
    if (iLine > 0)
        System.Diagnostics.Debug.WriteLine(axEdit1.get_TextLine(iLine));
```

```
}
```

The following VFP sample displays the line from the cursor:

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

with thisform.Edit1
   local i, c
   i = 0
   c = 0
   .ClientToCursor(x, y, @c, @i)
   If (i > 0) Then
      wait window nowait .TextLine(i)
   EndIf
endwith
```

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

Occurs when the user moves the mouse.

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

The MouseMove event is generated continually as the mouse pointer moves across objects. Unless another object has captured the mouse, an object recognizes a MouseMove event whenever the mouse position is within its borders. Use the ClientToCursor method to convert client coordinates to cursor coordinates. Use the TextLine property to retrieve the line giving its index.

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

C#

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

VB

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

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

C#

```
private void MouseMoveEvent(object sender,
AxEXEDITLib._IEditEvents_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:
AxEXEDITLib._IEditEvents_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
AxEXEDITLib._IEditEvents_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(oEdit,Button,Shift,X,Y)
RETURN
```

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

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

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

Visual Data...
```
Procedure OnComMouseMove Short llButton Short llShift OLE_XPOS_PIXELS llX
OLE_YPOS_PIXELS llY
   Forward Send OnComMouseMove llButton llShift llX llY
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.Edit.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.Edit.1::OLE_YPOS_PIXELS)
end function
```

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

The following VB sample displays the line from the cursor:

```
Private Sub Edit1_MouseMove(Button As Integer, Shift As Integer, x As Single, y As Single)
   With Edit1
      Dim i As Long, c As Long
      .ClientToCursor x / Screen.TwipsPerPixelX, y / Screen.TwipsPerPixelY, c, i
      If (i > 0) Then
```

```
        Debug.Print .TextLine(i)
    End If
  End With
End Sub
```

The following C++ sample displays the line from the cursor:

```
void OnMouseMoveEdit1(short Button, short Shift, long X, long Y)
{
  long nLine = 0, nColumn = 0;
  m_edit.ClientToCursor( X, Y, &nColumn, &nLine );
  if ( nLine > 0 )
    OutputDebugString( m_edit.GetTextLine( nLine ) );
}
```

The following VB.NET sample displays the line from the cursor:

```
Private Sub AxEdit1_MouseMoveEvent(ByVal sender As Object, ByVal e As
AxEXEDITLib._IEditEvents_MouseMoveEvent) Handles AxEdit1.MouseMoveEvent
  With AxEdit1
    Dim i As Integer, c As Integer
    .ClientToCursor(e.x, e.y, c, i)
    If (i > 0) Then
      Debug.WriteLine(.get_TextLine(i))
    End If
  End With
End Sub
```

The following C# sample displays the line from the cursor:

```
private void axEdit1_MouseMoveEvent(object sender,
AxEXEDITLib._IEditEvents_MouseMoveEvent e)
{
  int iLine = 0, iColumn = 0;
  axEdit1.ClientToCursor(e.x, e.y, out iColumn, out iLine);
  if (iLine > 0)
    System.Diagnostics.Debug.WriteLine(axEdit1.get_TextLine(iLine));
}
```

The following VFP sample displays the line from the cursor:

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

with thisform.Edit1
   local i, c
   i = 0
   c = 0
   .ClientToCursor(x, y, @c, @i)
   If (i > 0) Then
      wait window nowait .TextLine(i)
   EndIf
endwith
```

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 DblClick 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. Use the ClientToCursor method to convert client coordinates to cursor coordinates. Use the TextLine property to get the line giving its index.

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,
AxEXEDITLib._IEditEvents_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:
AxEXEDITLib._IEditEvents_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
AxEXEDITLib._IEditEvents_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(oEdit,Button,Shift,X,Y)
```

```
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComMouseUp Short llButton Short llShift OLE_XPOS_PIXELS llX
OLE_YPOS_PIXELS llY
   Forward Send OnComMouseUp llButton llShift llX llY
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.Edit.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.Edit.1::OLE_YPOS_PIXELS)
end function
```

dBASE

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

The following VB sample displays the line from the cursor:

```
Private Sub Edit1_MouseUp(Button As Integer, Shift As Integer, x As Single, y As Single)
   With Edit1
      Dim i As Long, c As Long
```

```
        .ClientToCursor x / Screen.TwipsPerPixelX, y / Screen.TwipsPerPixelY, c, i
        If (i > 0) Then
            Debug.Print .TextLine(i)
        End If
    End With
End Sub
```

The following C++ sample displays the line from the cursor:

```
void OnMouseUpEdit1(short Button, short Shift, long X, long Y)
{
    long nLine = 0, nColumn = 0;
    m_edit.ClientToCursor( X, Y, &nColumn, &nLine );
    if ( nLine > 0 )
        OutputDebugString( m_edit.GetTextLine( nLine ) );
}
```

The following VB.NET sample displays the line from the cursor:

```
Private Sub AxEdit1_MouseUpEvent(ByVal sender As Object, ByVal e As
AxEXEDITLib._IEditEvents_MouseUpEvent) Handles AxEdit1.MouseUpEvent
    With AxEdit1
        Dim i As Integer, c As Integer
        .ClientToCursor(e.x, e.y, c, i)
        If (i > 0) Then
            Debug.WriteLine(.get_TextLine(i))
        End If
    End With
End Sub
```

The following C# sample displays the line from the cursor:

```
private void axEdit1_MouseUpEvent(object sender,
AxEXEDITLib._IEditEvents_MouseUpEvent e)
{
    int iLine = 0, iColumn = 0;
    axEdit1.ClientToCursor(e.x, e.y, out iColumn, out iLine);
    if (iLine > 0)
        System.Diagnostics.Debug.WriteLine(axEdit1.get_TextLine(iLine));
```

```
}
```

The following VFP sample displays the line from the cursor:

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

with thisform.Edit1
   local i, c
   i = 0
   c = 0
   .ClientToCursor(x, y, @c, @i)
   If (i > 0) Then
      wait window nowait .TextLine(i)
   EndIf
endwith
```

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,
AxEXEDITLib._IEditEvents_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:
AxEXEDITLib._IEditEvents_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
AxEXEDITLib._IEditEvents_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(oEdit,Effect)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComOLECompleteDrag Integer llEffect
   Forward Send OnComOLECompleteDrag llEffect
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 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, AxEXEDITLib._IEditEvents_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,Exeditlib_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:
AxEXEDITLib._IEditEvents_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
AxEXEDITLib._IEditEvents_OLEDragDropEvent) Handles OLEDragDrop
End Sub
```

VB6

```
Private Sub OLEDragDrop(ByVal Data As EXEDITLibCtl.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(oEdit,Data,Effect,Button,Shift,X,Y)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComOLEDragDrop Variant llData Integer llEffect Short llButton
Short llShift OLE_XPOS_PIXELS llX OLE_YPOS_PIXELS llY
    Forward Send OnComOLEDragDrop llData llEffect llButton llShift llX llY
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.Edit.1::IExDataObject,Effect as
N,Button as N,Shift as N,X as OLE::Exontrol.Edit.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.Edit.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 Edit1_OLEDragDrop(ByVal Data As EXEDITLibCtl.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
      Edit1.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 OnOLEDragDropEdit1(LPDISPATCH Data, long FAR* Effect, short Button, short Shift,
long X, long Y)
{
    if ( EXEDITLib::IExDataObjectPtr spData( Data ) )
        if ( spData->GetFormat( EXEDITLib::exCFFiles ) )
        {
            EXEDITLib::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 <exedit.dll> statement. The #import <exedit.dll> includes the EXEDITLib 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 AxEdit1_OLEDragDrop(ByVal sender As Object, ByVal e As
AxEXEDITLib._IEditEvents_OLEDragDropEvent) Handles AxEdit1.OLEDragDrop
    If (e.data.GetFormat(EXEDITLib.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
            AxEdit1.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 axEdit1_OLEDragDrop(object sender,
AxEXEDITLib._IEditEvents_OLEDragDropEvent e)
{
  if (e.data.GetFormat(Convert.ToInt16(EXEDITLib.exClipboardFormatEnum.exCFFiles)))
  {
    EXEDITLib.IExDataObjectFiles files = e.data.Files;
    if (files != null)
    {
      string s = "";
      for (int i = 0; i < files.Count; i++)
        s += files[i] + "\r\n";
      axEdit1.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.Edit1.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 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	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 bellow.

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,
AxEXEDITLib._IEditEvents_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,Exeditlib_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:
AxEXEDITLib._IEditEvents_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
AxEXEDITLib._IEditEvents_OLEDragOverEvent) Handles OLEDragOver
End Sub
```

VB6

```
Private Sub OLEDragOver(ByVal Data As EXEDITLibCtl.IExDataObject,Effect As
Long,ByVal Button As Integer,ByVal Shift As Integer,ByVal X As Single,ByVal Y As
Single,ByVal State As Integer)
```

```
End Sub
```

VBA

```
Private Sub OLEDragOver(ByVal Data As Object,Effect As Long,ByVal Button As Integer,ByVal Shift As Integer,ByVal X As Long,ByVal Y As Long,ByVal State As Integer)
End Sub
```

VFP

```
LPARAMETERS Data,Effect,Button,Shift,X,Y,State
```

Xbas...

```
PROCEDURE OnOLEDragOver(oEdit,Data,Effect,Button,Shift,X,Y,State)
RETURN
```

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

Java...

```
<SCRIPT EVENT="OLEDragOver(Data,Effect,Button,Shift,X,Y,State)"
LANGUAGE="JScript">
</SCRIPT>
```

VBSc...

```
<SCRIPT LANGUAGE="VBScript">
Function OLEDragOver(Data,Effect,Button,Shift,X,Y,State)
End Function
</SCRIPT>
```

Visual Data...

```
Procedure OnComOLEDragOver Variant llData Integer llEffect Short llButton Short llShift OLE_XPOS_PIXELS llX OLE_YPOS_PIXELS llY Short llState
   Forward Send OnComOLEDragOver llData llEffect llButton llShift llX llY llState
End_Procedure
```

Visual Objects

```
METHOD OCX_OLEDragOver(Data,Effect,Button,Shift,X,Y,State) CLASS MainDialog
RETURN NIL
```

X++

```
// OLEDragOver event is not supported. Use the DragEnter,DragLeave,DragOver, DragDrop ... events.
```

XBasic

```
function OLEDragOver as v (Data as OLE::Exontrol.Edit.1::IExDataObject,Effect as N,Button as N,Shift as N,X as OLE::Exontrol.Edit.1::OLE_XPOS_PIXELS,Y as OLE::Exontrol.Edit.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 bellow.
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,
AxEXEDITLib._IEditEvents_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:
AxEXEDITLib._IEditEvents_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
AxEXEDITLib._IEditEvents_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(oEdit,Effect,DefaultCursors)
RETURN
```

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

Java…

```
<SCRIPT EVENT="OLEGiveFeedback(Effect,DefaultCursors)"
LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function OLEGiveFeedback(Effect,DefaultCursors)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComOLEGiveFeedback Integer llEffect Boolean llDefaultCursors
    Forward Send OnComOLEGiveFeedback llEffect llDefaultCursors
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 ExDataObject object in which to place the requested data. The component calls the SetData 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 ExDataObject 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,
AxEXEDITLib._IEditEvents_OLESetDataEvent e)
{
}
```

C++
```
void OnOLESetData(LPDISPATCH Data,short Format)
{
}
```

C++ Builder
```
void __fastcall OLESetData(TObject *Sender,Exeditlib_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:
AxEXEDITLib._IEditEvents_OLESetDataEvent);
begin
end;
```

Powe…

```
begin event OLESetData(oleobject Data,integer Format)
end event OLESetData
```

VB.NET

```
Private Sub OLESetData(ByVal sender As System.Object, ByVal e As
AxEXEDITLib._IEditEvents_OLESetDataEvent) Handles OLESetData
End Sub
```

VB6

```
Private Sub OLESetData(ByVal Data As EXEDITLibCtl.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(oEdit,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 llData Short llFormat
    Forward Send OnComOLESetData llData llFormat
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.Edit.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.

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, **/COM** version, on:

C#

```
private void OLEStartDrag(object sender, AxEXEDITLib._IEditEvents_OLEStartDragEvent e)
{
}
```

C++

```
void OnOLEStartDrag(LPDISPATCH Data,long FAR* AllowedEffects)
{
}
```

C++ Builder

```
void __fastcall OLEStartDrag(TObject *Sender,Exeditlib_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:
AxEXEDITLib._IEditEvents_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
AxEXEDITLib._IEditEvents_OLEStartDragEvent) Handles OLEStartDrag
End Sub
```

VB6

```
Private Sub OLEStartDrag(ByVal Data As
EXEDITLibCtl.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(oEdit,Data,AllowedEffects)
RETURN
```

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

Java...

```
<SCRIPT EVENT="OLEStartDrag(Data,AllowedEffects)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc...

```
<SCRIPT LANGUAGE="VBScript">
Function OLEStartDrag(Data,AllowedEffects)
End Function
</SCRIPT>
```

Visual Data...

```
Procedure OnComOLEStartDrag Variant llData Integer llAllowedEffects
   Forward Send OnComOLEStartDrag llData llAllowedEffects
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.Edit.1::IExDataObject,AllowedEffects as N)
end function
```

dBASE

```
function nativeObject_OLEStartDrag(Data,AllowedEffects)
return
```

To start accepting drag and drop sources the exGrid control should have the OLEDropMode to exOLEDropManual or exOLEDropAutomatic.

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 Edit1_OLEStartDrag(ByVal Data As EXEDITLibCtl.IExDataObject, AllowedEffects
As Long)
   Dim s As String
   s = Edit1.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 OnOLEStartDragEdit1(LPDISPATCH Data, long FAR* AllowedEffects)
{
   CString s( m_edit.GetSelText() );
   if ( s.GetLength() > 0 )
   {
      *AllowedEffects = 1; /*exOLEDropEffectCopy*/
      if ( EXEDITLib::IExDataObjectPtr spData( Data ) )
         spData->SetData( COleVariant( s ), COleVariant( long(EXEDITLib::exCFText) ) );

   }
}
```

The C++ sample uses the #import <exedit.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 AxEdit1_OLEStartDrag(ByVal sender As Object, ByVal e As
AxEXEDITLib._IEditEvents_OLEStartDragEvent) Handles AxEdit1.OLEStartDrag
   Dim s As String = AxEdit1.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 axEdit1_OLEStartDrag(object sender,
AxEXEDITLib._IEditEvents_OLEStartDragEvent e)
{
   string s = axEdit1.SelText;
   if (s.Length > 0)
   {
      e.allowedEffects = 1;
      e.data.SetData(s, EXEDITLib.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.Edit1
   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. For instance, during the OnContext event you can add a collection of the items, based on the context you have, using the Add method of the Context object. Use the CaretPos and CaretLine properties to determine the position of the cursor inside the text. Use the TextLine property to retrieve line at the cursor.

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, AxEXEDITLib._IEditEvents_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:
AxEXEDITLib._IEditEvents_OnContextEvent);
begin
end;
```

Powe…

```
begin event OnContext(long Start,string Context)
end event OnContext
```

VB.NET

```
Private Sub OnContext(ByVal sender As System.Object, ByVal e As
AxEXEDITLib._IEditEvents_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(oEdit,Start,Context)
RETURN
```

Syntax for OnContext event, **/COM** 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 llStart String llContext
    Forward Send OnComOnContext llStart llContext
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
```

The following VB sample displays the current line when the OnContext event occurs:

```
Private Sub Edit1_OnContext(ByVal Start As Long, ByVal Context As String)
    With Edit1
        Debug.Print "Current line: " & .TextLine(.CaretLine)
    End With
End Sub
```

The following C++ sample displays the current line when the OnContext event occurs:

```
void OnOnContextEdit1(long Start, LPCTSTR Context)
{
    OutputDebugString( m_edit.GetTextLine( m_edit.GetCaretLine() ) );
}
```

The following VB.NET sample displays the current line when the OnContext event occurs:

```
Private Sub AxEdit1_OnContext(ByVal sender As Object, ByVal e As
AxEXEDITLib._IEditEvents_OnContextEvent) Handles AxEdit1.OnContext
   With AxEdit1
      Debug.WriteLine(.get_TextLine(.CaretLine))
   End With
End Sub
```

The following C# sample displays the current line when the OnContext event occurs:

```
private void axEdit1_OnContext(object sender, AxEXEDITLib._IEditEvents_OnContextEvent
e)
{
   System.Diagnostics.Debug.WriteLine(axEdit1.get_TextLine(axEdit1.CaretLine));
}
```

The following VFP sample displays the current line when the OnContext event occurs:

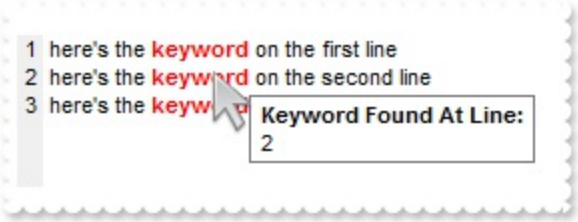
```
*** ActiveX Control Event ***
LPARAMETERS start, context

with thisform.Edit1
   wait window nowait .TextLine(.CaretLine)
endwith
```

event QueryContext (XCursor as Long, YCursor as Long, ByRef QContext as Variant)

Queries for the context at the specified location, to provide different tooltips for the same keyword on QueryContextToolTip event.

Type	Description
XCursor as Long	A Long expression that specifies the position of the caret inside the current line.
YCursor as Long	A Long expression that specifies the position current line.
QContext as Variant	(By Reference) A String expression that defines the context to be passed to the QContext parameter of the QueryContextToolTip event.

The QueryContext event queries for the context at the specified location, to provide different tooltips for the same keyword on QueryContextToolTip event. For instance, the QueryContext event occurs when the control is about to find / display the tooltip of the keyword from the cursor. Use the QueryContextToolTip event to define a new tooltip to be shown for specified context.



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

C#
```
private void QueryContext(object sender,int   XCursor,int   YCursor,ref object QContext)
{
}
```

VB
```
Private Sub QueryContext(ByVal sender As System.Object,ByVal XCursor As Integer,ByVal YCursor As Integer,ByRef QContext As Object) Handles QueryContext
End Sub
```

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

C#
```
private void QueryContext(object sender, AxEXEDITLib._IEditEvents_QueryContextEvent e)
```

```
{
}
```

C++

```
void OnQueryContext(long   XCursor,long   YCursor,VARIANT FAR*   QContext)
{
}
```

C++ Builder

```
void __fastcall QueryContext(TObject *Sender,long   XCursor,long   YCursor,Variant
*   QContext)
{
}
```

Delphi

```
procedure QueryContext(ASender: TObject; XCursor : Integer;YCursor : Integer;var
QContext : OleVariant);
begin
end;
```

Delphi 8 (.NET only)

```
procedure QueryContext(sender: System.Object; e:
AxEXEDITLib._IEditEvents_QueryContextEvent);
begin
end;
```

Powe…

```
begin event QueryContext(long  XCursor,long  YCursor,any  QContext)

end event QueryContext
```

VB.NET

```
Private Sub QueryContext(ByVal sender As System.Object, ByVal e As
AxEXEDITLib._IEditEvents_QueryContextEvent) Handles QueryContext
End Sub
```

VB6

```
Private Sub QueryContext(ByVal XCursor As Long,ByVal YCursor As Long,QContext
As Variant)
End Sub
```

VBA

```
Private Sub QueryContext(ByVal XCursor As Long,ByVal YCursor As Long,QContext
As Variant)
End Sub
```

VFP

```
LPARAMETERS XCursor,YCursor,QContext
```

Xbas…

```
PROCEDURE OnQueryContext(oEdit,XCursor,YCursor,QContext)

RETURN
```

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

Java…

```
<SCRIPT EVENT="QueryContext(XCursor,YCursor,QContext)"
LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function QueryContext(XCursor,YCursor,QContext)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComQueryContext Integer   llXCursor Integer   llYCursor Variant
llQContext
   Forward Send OnComQueryContext llXCursor llYCursor llQContext
End_Procedure
```

Visual Objects

```
METHOD OCX_QueryContext(XCursor,YCursor,QContext) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_QueryContext(int   _XCursor,int   _YCursor,COMVariant /*variant*/
_QContext)
{
}
```

XBasic

```
function QueryContext as v (XCursor  as  N,YCursor  as  N,QContext  as  A)
end function
```

dBASE

```
function nativeObject_QueryContext(XCursor,YCursor,QContext)
return
```

How can I provide different tooltip for the same keyword?

VBA (MS Access, Excell...)

```
' QueryContext event - Queries for the context at the specified location, to provide different tooltips for the same keyword on QueryContextToolTip event.
Private Sub Edit1_QueryContext(ByVal XCursor As Long,ByVal YCursor As Long,QContext As Variant)
   With Edit1
      QContext = YCursor
   End With
End Sub

' QueryContextToolTip event - Asks for the tooltip/title of the keyword on the context retrieved by the QueryContext event.
Private Sub Edit1_QueryContextToolTip(ByVal QContext As String,ByVal Keyword As String,QToolTip As Variant,QToolTipTitle As Variant)
   With Edit1
      QToolTip = QContext
      QToolTipTitle = "Keyword Found At Line:"
   End With
End Sub

With Edit1
   .LineNumberWidth = -1
   .LineNumberBackColor = RGB(240,240,240)
   .AddKeyword "<fgcolor=FF0000><b>keyword</b></fgcolor>"
   .Text = ""
   .InsertText "here's the keyword on the first line"
   .InsertText "\r\nhere's the keyword on the second line"
   .InsertText "\r\nhere's the keyword on the third line"
End With
```

VB6

```
' QueryContext event - Queries for the context at the specified location, to provide different tooltips for the same keyword on QueryContextToolTip event.
```

```
Private Sub Edit1_QueryContext(ByVal XCursor As Long,ByVal YCursor As
Long,QContext As Variant)
   With Edit1
      QContext = YCursor
   End With
End Sub

' QueryContextToolTip event - Asks for the tooltip/title of the keyword on the context
retrieved by the QueryContext event.
Private Sub Edit1_QueryContextToolTip(ByVal QContext As String,ByVal Keyword As
String,QToolTip As Variant,QToolTipTitle As Variant)
   With Edit1
      QToolTip = QContext
      QToolTipTitle = "Keyword Found At Line:"
   End With
End Sub

With Edit1
   .LineNumberWidth = -1
   .LineNumberBackColor = RGB(240,240,240)
   .AddKeyword "<fgcolor=FF0000><b>keyword</b></fgcolor>"
   .Text = ""
   .InsertText "here's the keyword on the first line"
   .InsertText "\r\nhere's the keyword on the second line"
   .InsertText "\r\nhere's the keyword on the third line"
End With
```

VB.NET

```
' QueryContext event - Queries for the context at the specified location, to provide
different tooltips for the same keyword on QueryContextToolTip event.
Private Sub Exedit1_QueryContext(ByVal sender As System.Object,ByVal XCursor As
Integer,ByVal YCursor As Integer,ByRef QContext As Object) Handles
Exedit1.QueryContext
   With Exedit1
      QContext = YCursor
   End With
```

```
End Sub

' QueryContextToolTip event - Asks for the tooltip/title of the keyword on the context retrieved by the QueryContext event.
Private Sub Exedit1_QueryContextToolTip(ByVal sender As System.Object,ByVal QContext As String,ByVal Keyword As String,ByRef QToolTip As Object,ByRef QToolTipTitle As Object) Handles Exedit1.QueryContextToolTip
   With Exedit1
      QToolTip = QContext
      QToolTipTitle = "Keyword Found At Line:"
   End With
End Sub

With Exedit1
   .LineNumberWidth = -1
   .LineNumberBackColor = Color.FromArgb(240,240,240)
   .AddKeyword("<fgcolor=FF0000><b>keyword</b></fgcolor>")
   .Text = ""
   .InsertText("here's the keyword on the first line")
   .InsertText("\r\nhere's the keyword on the second line")
   .InsertText("\r\nhere's the keyword on the third line")
End With
```

VB.NET for /COM

```
' QueryContext event - Queries for the context at the specified location, to provide different tooltips for the same keyword on QueryContextToolTip event.
Private Sub AxEdit1_QueryContext(ByVal sender As System.Object, ByVal e As AxEXEDITLib._IEditEvents_QueryContextEvent) Handles AxEdit1.QueryContext
   With AxEdit1
      e.qContext = e.yCursor
   End With
End Sub

' QueryContextToolTip event - Asks for the tooltip/title of the keyword on the context retrieved by the QueryContext event.
Private Sub AxEdit1_QueryContextToolTip(ByVal sender As System.Object, ByVal e As
```

```
AxEXEDITLib._IEditEvents_QueryContextToolTipEvent) Handles
AxEdit1.QueryContextToolTip
    With AxEdit1
        e.qToolTip = e.qContext
        e.qToolTipTitle = "Keyword Found At Line:"
    End With
End Sub

With AxEdit1
    .LineNumberWidth = -1
    .LineNumberBackColor = RGB(240,240,240)
    .AddKeyword("<fgcolor=FF0000><b>keyword</b></fgcolor>")
    .Text = ""
    .InsertText("here's the keyword on the first line")
    .InsertText("\r\nhere's the keyword on the second line")
    .InsertText("\r\nhere's the keyword on the third line")
End With
```

C++

```
// QueryContext event - Queries for the context at the specified location, to provide
different tooltips for the same keyword on QueryContextToolTip event.
void OnQueryContextEdit1(long   XCursor,long   YCursor,VARIANT FAR*   QContext)
{
    /*
        Copy and paste the following directives to your header file as
        it defines the namespace 'EXEDITLib' for the library: 'ExEdit 1.0 Control Library'
        #import <ExEdit.dll>
        using namespace EXEDITLib;
     */
    EXEDITLib::IEditPtr spEdit1 = GetDlgItem(IDC_EDIT1)->GetControlUnknown();
    QContext = YCursor;
}

// QueryContextToolTip event - Asks for the tooltip/title of the keyword on the context
retrieved by the QueryContext event.
void OnQueryContextToolTipEdit1(LPCTSTR   QContext,LPCTSTR   Keyword,VARIANT
```

```
FAR*   QToolTip,VARIANT FAR*   QToolTipTitle)
{
   EXEDITLib::IEditPtr spEdit1 = GetDlgItem(IDC_EDIT1)->GetControlUnknown();
   QToolTip = QContext;
   QToolTipTitle = "Keyword Found At Line:";
}

EXEDITLib::IEditPtr spEdit1 = GetDlgItem(IDC_EDIT1)->GetControlUnknown();
spEdit1->PutLineNumberWidth(-1);
spEdit1->PutLineNumberBackColor(RGB(240,240,240));
spEdit1->AddKeyword(L"<fgcolor=FF0000><b>keyword</b>
</fgcolor>",vtMissing,vtMissing,vtMissing);
spEdit1->PutText(L"");
spEdit1->InsertText(L"here's the keyword on the first line",vtMissing);
spEdit1->InsertText(L"\\r\\nhere's the keyword on the second line",vtMissing);
spEdit1->InsertText(L"\\r\\nhere's the keyword on the third line",vtMissing);
```

C++ Builder

```
// QueryContext event - Queries for the context at the specified location, to provide
different tooltips for the same keyword on QueryContextToolTip event.
void __fastcall TForm1::Edit1QueryContext(TObject *Sender,long   XCursor,long
YCursor,Variant *   QContext)
{
   QContext = YCursor;
}

// QueryContextToolTip event - Asks for the tooltip/title of the keyword on the context
retrieved by the QueryContext event.
void __fastcall TForm1::Edit1QueryContextToolTip(TObject *Sender,BSTR
QContext,BSTR   Keyword,Variant *   QToolTip,Variant *   QToolTipTitle)
{
   QToolTip = QContext;
   QToolTipTitle = "Keyword Found At Line:";
}
```

```
Edit1->LineNumberWidth = -1;
Edit1->LineNumberBackColor = RGB(240,240,240);
Edit1->AddKeyword(L"<fgcolor=FF0000><b>keyword</b>
</fgcolor>",TNoParam(),TNoParam(),TNoParam());
Edit1->Text = L"";
Edit1->InsertText(L"here's the keyword on the first line",TNoParam());
Edit1->InsertText(L"\\r\\nhere's the keyword on the second line",TNoParam());
Edit1->InsertText(L"\\r\\nhere's the keyword on the third line",TNoParam());
```

C#

```
// QueryContext event - Queries for the context at the specified location, to provide
different tooltips for the same keyword on QueryContextToolTip event.
private void exedit1_QueryContext(object sender,int   XCursor,int   YCursor,ref object
QContext)
{
   QContext = YCursor;
}
//this.exedit1.QueryContext += new
exontrol.EXEDITLib.exg2antt.QueryContextEventHandler(this.exedit1_QueryContext);

// QueryContextToolTip event - Asks for the tooltip/title of the keyword on the context
retrieved by the QueryContext event.
private void exedit1_QueryContextToolTip(object sender,string   QContext,string
Keyword,ref object   QToolTip,ref object   QToolTipTitle)
{
   QToolTip = QContext;
   QToolTipTitle = "Keyword Found At Line:";
}
//this.exedit1.QueryContextToolTip += new
exontrol.EXEDITLib.exg2antt.QueryContextToolTipEventHandler(this.exedit1_QueryConte


exedit1.LineNumberWidth = -1;
exedit1.LineNumberBackColor = Color.FromArgb(240,240,240);
exedit1.AddKeyword("<fgcolor=FF0000><b>keyword</b>
```

```
</fgcolor>",null,null,null);
exedit1.Text = "";
exedit1.InsertText("here's the keyword on the first line",null);
exedit1.InsertText("\\r\\nhere's the keyword on the second line",null);
exedit1.InsertText("\\r\\nhere's the keyword on the third line",null);
```

JScript/JavaScript

```
<BODY onload="Init()">
<SCRIPT FOR="Edit1" EVENT="QueryContext(XCursor,YCursor,QContext)"
LANGUAGE="JScript">
   QContext = YCursor;
</SCRIPT>

<SCRIPT FOR="Edit1"
EVENT="QueryContextToolTip(QContext,Keyword,QToolTip,QToolTipTitle)"
LANGUAGE="JScript">
   QToolTip = QContext;
   QToolTipTitle = "Keyword Found At Line:";
</SCRIPT>

<OBJECT CLASSID="clsid:39136531-DD0F-4281-B445-E36FC2CDDBC5" id="Edit1">
</OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
   Edit1.LineNumberWidth = -1;
   Edit1.LineNumberBackColor = 15790320;
   Edit1.AddKeyword("<fgcolor=FF0000><b>keyword</b>
</fgcolor>",null,null,null);
   Edit1.Text = "";
   Edit1.InsertText("here's the keyword on the first line",null);
   Edit1.InsertText("\\r\\nhere's the keyword on the second line",null);
   Edit1.InsertText("\\r\\nhere's the keyword on the third line",null);
}
```

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

VBScript

```
<BODY onload="Init()">
<SCRIPT LANGUAGE="VBScript">
Function Edit1_QueryContext(XCursor,YCursor,QContext)
   With Edit1
      QContext = YCursor
   End With
End Function
</SCRIPT>

<SCRIPT LANGUAGE="VBScript">
Function Edit1_QueryContextToolTip(QContext,Keyword,QToolTip,QToolTipTitle)
   With Edit1
      QToolTip = QContext
      QToolTipTitle = "Keyword Found At Line:"
   End With
End Function
</SCRIPT>

<OBJECT CLASSID="clsid:39136531-DD0F-4281-B445-E36FC2CDDBC5" id="Edit1">
</OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
   With Edit1
      .LineNumberWidth = -1
      .LineNumberBackColor = RGB(240,240,240)
      .AddKeyword "<fgcolor=FF0000><b>keyword</b></fgcolor>"
      .Text = ""
      .InsertText "here's the keyword on the first line"
      .InsertText "\r\nhere's the keyword on the second line"
      .InsertText "\r\nhere's the keyword on the third line"
```

```
   End With
End Function
</SCRIPT>
</BODY>
```

C# for /COM

```
// QueryContext event - Queries for the context at the specified location, to provide
different tooltips for the same keyword on QueryContextToolTip event.
private void axEdit1_QueryContext(object sender,
AxEXEDITLib._IEditEvents_QueryContextEvent e)
{
   e.qContext = e.yCursor;
}
//this.axEdit1.QueryContext += new
AxEXEDITLib._IEditEvents_QueryContextEventHandler(this.axEdit1_QueryContext);

// QueryContextToolTip event - Asks for the tooltip/title of the keyword on the context
retrieved by the QueryContext event.
private void axEdit1_QueryContextToolTip(object sender,
AxEXEDITLib._IEditEvents_QueryContextToolTipEvent e)
{
   e.qToolTip = e.qContext;
   e.qToolTipTitle = "Keyword Found At Line:";
}
//this.axEdit1.QueryContextToolTip += new
AxEXEDITLib._IEditEvents_QueryContextToolTipEventHandler(this.axEdit1_QueryContext


axEdit1.LineNumberWidth = -1;
axEdit1.LineNumberBackColor = Color.FromArgb(240,240,240);
axEdit1.AddKeyword("<fgcolor=FF0000><b>keyword</b>
</fgcolor>",null,null,null);
axEdit1.Text = "";
axEdit1.InsertText("here's the keyword on the first line",null);
axEdit1.InsertText("\\r\\nhere's the keyword on the second line",null);
```

```
axEdit1.InsertText("\\r\\nhere's the keyword on the third line",null);
```

X++ (Dynamics Ax 2009)

```
// QueryContext event - Queries for the context at the specified location, to provide
different tooltips for the same keyword on QueryContextToolTip event.
void onEvent_QueryContext(int   _XCursor,int   _YCursor,COMVariant /*variant*/
_QContext)
{
    ;
    _QContext = _YCursor;
}

// QueryContextToolTip event - Asks for the tooltip/title of the keyword on the context
retrieved by the QueryContext event.
void onEvent_QueryContextToolTip(str   _QContext,str   _Keyword,COMVariant
/*variant*/   _QToolTip,COMVariant /*variant*/   _QToolTipTitle)
{
    ;
    _QToolTip = _QContext;
    _QToolTipTitle = "Keyword Found At Line:";
}

public void init()
{
    ;

    super();

    exedit1.LineNumberWidth(-1);
    exedit1.LineNumberBackColor(WinApi::RGB2int(240,240,240));
    exedit1.AddKeyword("<fgcolor=FF0000><b>keyword</b></fgcolor>");
    exedit1.Text("");
    exedit1.InsertText("here's the keyword on the first line");
    exedit1.InsertText("\\r\\nhere's the keyword on the second line");
    exedit1.InsertText("\\r\\nhere's the keyword on the third line");
```

```
}
```

Delphi 8 (.NET only)

```
// QueryContext event - Queries for the context at the specified location, to provide
different tooltips for the same keyword on QueryContextToolTip event.
procedure TWinForm1.AxEdit1_QueryContext(sender: System.Object; e:
AxEXEDITLib._IEditEvents_QueryContextEvent);
begin
  with AxEdit1 do
  begin
    e.qContext := e.yCursor;
  end
end;

// QueryContextToolTip event - Asks for the tooltip/title of the keyword on the context
retrieved by the QueryContext event.
procedure TWinForm1.AxEdit1_QueryContextToolTip(sender: System.Object; e:
AxEXEDITLib._IEditEvents_QueryContextToolTipEvent);
begin
  with AxEdit1 do
  begin
    e.qToolTip := e.qContext;
    e.qToolTipTitle := 'Keyword Found At Line:';
  end
end;

with AxEdit1 do
begin
  LineNumberWidth := -1;
  LineNumberBackColor := Color.FromArgb(240,240,240);
  AddKeyword('<fgcolor=FF0000><b>keyword</b></fgcolor>',Nil,Nil,Nil);
  Text := '';
  InsertText('here''s the keyword on the first line',Nil);
  InsertText('\r\nhere''s the keyword on the second line',Nil);
  InsertText('\r\nhere''s the keyword on the third line',Nil);
end
```

Delphi (standard)

```
// QueryContext event - Queries for the context at the specified location, to provide
different tooltips for the same keyword on QueryContextToolTip event.
procedure TForm1.Edit1QueryContext(ASender: TObject; XCursor : Integer;YCursor :
Integer;var QContext : OleVariant);
begin
   with Edit1 do
   begin
      QContext := YCursor;
   end
end;

// QueryContextToolTip event - Asks for the tooltip/title of the keyword on the context
retrieved by the QueryContext event.
procedure TForm1.Edit1QueryContextToolTip(ASender: TObject; QContext :
WideString;Keyword : WideString;var QToolTip : OleVariant;var QToolTipTitle :
OleVariant);
begin
   with Edit1 do
   begin
      QToolTip := QContext;
      QToolTipTitle := 'Keyword Found At Line:';
   end
end;

with Edit1 do
begin
   LineNumberWidth := -1;
   LineNumberBackColor := RGB(240,240,240);
   AddKeyword('<fgcolor=FF0000><b>keyword</b></fgcolor>',Null,Null,Null);
   Text := '';
   InsertText('here''s the keyword on the first line',Null);
   InsertText('\r\nhere''s the keyword on the second line',Null);
   InsertText('\r\nhere''s the keyword on the third line',Null);
end
```

VFP

```
*** QueryContext event - Queries for the context at the specified location, to provide
different tooltips for the same keyword on QueryContextToolTip event. ***
LPARAMETERS XCursor,YCursor,QContext
   with thisform.Edit1
      QContext = YCursor
   endwith

*** QueryContextToolTip event - Asks for the tooltip/title of the keyword on the
context retrieved by the QueryContext event. ***
LPARAMETERS QContext,Keyword,QToolTip,QToolTipTitle
   with thisform.Edit1
      QToolTip = QContext
      QToolTipTitle = "Keyword Found At Line:"
   endwith

with thisform.Edit1
   .LineNumberWidth = -1
   .LineNumberBackColor = RGB(240,240,240)
   .AddKeyword("<fgcolor=FF0000><b>keyword</b></fgcolor>")
   .Text = ""
   .InsertText("here's the keyword on the first line")
   .InsertText("\r\nhere's the keyword on the second line")
   .InsertText("\r\nhere's the keyword on the third line")
endwith
```

dBASE Plus

```
/*
with (this.EXEDITACTIVEXCONTROL1.nativeObject)
    QueryContext = class::nativeObject_QueryContext
endwith
*/
// Queries for the context at the specified location, to provide different tooltips for the
same keyword on QueryContextToolTip event.
function nativeObject_QueryContext(XCursor,YCursor,QContext)
   oEdit = form.EXEDITACTIVEXCONTROL1.nativeObject
```

```
   QContext = YCursor
return

/*
with (this.EXEDITACTIVEXCONTROL1.nativeObject)
    QueryContextToolTip = class::nativeObject_QueryContextToolTip
endwith
*/
// Asks for the tooltip/title of the keyword on the context retrieved by the
QueryContext event.
function
nativeObject_QueryContextToolTip(QContext,Keyword,QToolTip,QToolTipTitle)
   oEdit = form.EXEDITACTIVEXCONTROL1.nativeObject
   QToolTip = QContext
   QToolTipTitle = "Keyword Found At Line:"
return

local oEdit

oEdit = form.EXEDITACTIVEXCONTROL1.nativeObject
oEdit.LineNumberWidth = -1
oEdit.LineNumberBackColor = 0xf0f0f0
oEdit.AddKeyword("<fgcolor=FF0000><b>keyword</b></fgcolor>")
oEdit.Text = ""
oEdit.InsertText("here's the keyword on the first line")
oEdit.InsertText("\r\nhere's the keyword on the second line")
oEdit.InsertText("\r\nhere's the keyword on the third line")
```

XBasic (Alpha Five)

```
' Queries for the context at the specified location, to provide different tooltips for the
same keyword on QueryContextToolTip event.
function QueryContext as v (XCursor  as  N,YCursor  as  N,QContext  as  A)
   oEdit = topparent:CONTROL_ACTIVEX1.activex
   QContext = YCursor
end function
```

```
' Asks for the tooltip/title of the keyword on the context retrieved by the
QueryContext event.
function QueryContextToolTip as v (QContext as C,Keyword as C,QToolTip as
A,QToolTipTitle as A)
   oEdit = topparent:CONTROL_ACTIVEX1.activex
   QToolTip = QContext
   QToolTipTitle = "Keyword Found At Line:"
end function

Dim oEdit as P

oEdit = topparent:CONTROL_ACTIVEX1.activex
oEdit.LineNumberWidth = -1
oEdit.LineNumberBackColor = 15790320
oEdit.AddKeyword("<fgcolor=FF0000><b>keyword</b></fgcolor>")
oEdit.Text = ""
oEdit.InsertText("here's the keyword on the first line")
oEdit.InsertText("\r\nhere's the keyword on the second line")
oEdit.InsertText("\r\nhere's the keyword on the third line")
```

Visual Objects

```
METHOD OCX_Exontrol1QueryContext(XCursor,YCursor,QContext) CLASS MainDialog
   // QueryContext event - Queries for the context at the specified location, to provide
different tooltips for the same keyword on QueryContextToolTip event.
   QContext := YCursor
RETURN NIL

METHOD
OCX_Exontrol1QueryContextToolTip(QContext,Keyword,QToolTip,QToolTipTitle)
CLASS MainDialog
   // QueryContextToolTip event - Asks for the tooltip/title of the keyword on the
context retrieved by the QueryContext event.
   QToolTip := QContext
   QToolTipTitle := "Keyword Found At Line:"
```

```
RETURN NIL


oDCOCX_Exontrol1:LineNumberWidth := -1
oDCOCX_Exontrol1:LineNumberBackColor := RGB(240,240,240)
oDCOCX_Exontrol1:AddKeyword("<fgcolor=FF0000><b>keyword</b>
</fgcolor>",nil,nil,nil)
oDCOCX_Exontrol1:Text := ""
oDCOCX_Exontrol1:InsertText("here's the keyword on the first line",nil)
oDCOCX_Exontrol1:InsertText("\r\nhere's the keyword on the second line",nil)
oDCOCX_Exontrol1:InsertText("\r\nhere's the keyword on the third line",nil)
```

PowerBuilder

```
/*begin event QueryContext(long  XCursor,long  YCursor,any  QContext) - Queries for
the context at the specified location, to provide different tooltips for the same
keyword on QueryContextToolTip event.*/
/*
    oEdit = ole_1.Object
    QContext = YCursor
*/
/*end event QueryContext*/

/*begin event QueryContextToolTip(string  QContext,string  Keyword,any
QToolTip,any  QToolTipTitle) - Asks for the tooltip/title of the keyword on the context
retrieved by the QueryContext event.*/
/*
    oEdit = ole_1.Object
    QToolTip = QContext
    QToolTipTitle = "Keyword Found At Line:"
*/
/*end event QueryContextToolTip*/

OleObject oEdit

oEdit = ole_1.Object
```

```
oEdit.LineNumberWidth = -1
oEdit.LineNumberBackColor = RGB(240,240,240)
oEdit.AddKeyword("<fgcolor=FF0000><b>keyword</b></fgcolor>")
oEdit.Text = ""
oEdit.InsertText("here's the keyword on the first line")
oEdit.InsertText("\r\nhere's the keyword on the second line")
oEdit.InsertText("\r\nhere's the keyword on the third line")
```

Visual DataFlex

```
// Queries for the context at the specified location, to provide different tooltips for the
same keyword on QueryContextToolTip event.
Procedure OnComQueryContext Integer   llXCursor Integer   llYCursor Variant
llQContext
   Forward Send OnComQueryContext llXCursor llYCursor llQContext
   Move llYCursor to QContext
End_Procedure

// Asks for the tooltip/title of the keyword on the context retrieved by the
QueryContext event.
Procedure OnComQueryContextToolTip String   llQContext String   llKeyword Variant
llQToolTip Variant   llQToolTipTitle
   Forward Send OnComQueryContextToolTip llQContext llKeyword llQToolTip
llQToolTipTitle
   Move llQContext to QToolTip
   Move "Keyword Found At Line:" to QToolTipTitle
End_Procedure

Procedure OnCreate
   Forward Send OnCreate
   Set ComLineNumberWidth to -1
   Set ComLineNumberBackColor to (RGB(240,240,240))
   Send ComAddKeyword "<fgcolor=FF0000><b>keyword</b></fgcolor>"
Nothing Nothing Nothing
   Set ComText to ""
   Send ComInsertText "here's the keyword on the first line" Nothing
```

```
   Send ComInsertText "\r\nhere's the keyword on the second line" Nothing
   Send ComInsertText "\r\nhere's the keyword on the third line" Nothing
End_Procedure
```

XBase++

```
PROCEDURE OnQueryContext(oEdit,XCursor,YCursor,QContext)
   QContext := YCursor
RETURN

PROCEDURE
OnQueryContextToolTip(oEdit,QContext,Keyword,QToolTip,QToolTipTitle)
   QToolTip := QContext
   QToolTipTitle := "Keyword Found At Line:"
RETURN

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

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

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

   oEdit := XbpActiveXControl():new( oForm:drawingArea )
   oEdit:CLSID  := "Exontrol.Edit.1" /*{39136531-DD0F-4281-B445-
E36FC2CDDBC5}*/
   oEdit:create(,, {10,60},{610,370} )

      oEdit:QueryContext := {|XCursor,YCursor,QContext|
OnQueryContext(oEdit,XCursor,YCursor,QContext)} /*Queries for the context at the
specified location, to provide different tooltips for the same keyword on
```

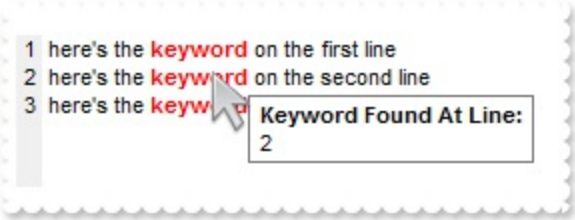
```
QueryContextToolTip event.*/
      oEdit:QueryContextToolTip := {|QContext,Keyword,QToolTip,QToolTipTitle|
OnQueryContextToolTip(oEdit,QContext,Keyword,QToolTip,QToolTipTitle)} /*Asks for
the tooltip/title of the keyword on the context retrieved by the QueryContext event.*/

      oEdit:LineNumberWidth := -1
      oEdit:SetProperty("LineNumberBackColor",AutomationTranslateColor(
GraMakeRGBColor  ( { 240,240,240 } )  , .F. ))
      oEdit:AddKeyword("<fgcolor=FF0000><b>keyword</b></fgcolor>")
      oEdit:Text := ""
      oEdit:InsertText("here's the keyword on the first line")
      oEdit:InsertText("\r\nhere's the keyword on the second line")
      oEdit:InsertText("\r\nhere's the keyword on the third line")

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

event QueryContextToolTip (QContext as String, Keyword as String, ByRef QToolTip as Variant, ByRef QToolTipTitle as Variant)

Asks for the tooltip/title of the keyword on the context retrieved by the QueryContext event.

Type	Description
QContext as String	A String expression defined by the QContext parameter of the QueryContext event
Keyword as String	A String expression that defines the keyword at the current position.
QToolTip as Variant	(By Reference) A String expression that defines the HTML-tooltip to be displayed. The QToolTip parameter supports built-in HTML tags as shown bellow.
QToolTipTitle as Variant	(By Reference) A String expression that defines the title of the tooltip to be displayed.

Use the QueryContextToolTip event to define a new tooltip to be shown for specified context. The QueryContext event queries for the context at the specified location, to provide different tooltips for the same keyword on QueryContextToolTip event. For instance, the QueryContext event occurs when the control is about to find / display the tooltip of the keyword from the cursor.



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

C#
```
private void QueryContextToolTip(object sender,string   QContext,string
Keyword,ref object   QToolTip,ref object   QToolTipTitle)
{
}
```

VB
```
Private Sub QueryContextToolTip(ByVal sender As System.Object,ByVal QContext
As String,ByVal Keyword As String,ByRef QToolTip As Object,ByRef QToolTipTitle As
Object) Handles QueryContextToolTip
End Sub
```

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

C#

```
private void QueryContextToolTip(object sender,
AxEXEDITLib._IEditEvents_QueryContextToolTipEvent e)
{
}
```

C++

```
void OnQueryContextToolTip(LPCTSTR   QContext,LPCTSTR   Keyword,VARIANT
FAR*   QToolTip,VARIANT FAR*   QToolTipTitle)
{
}
```

C++ Builder

```
void __fastcall QueryContextToolTip(TObject *Sender,BSTR   QContext,BSTR
Keyword,Variant *   QToolTip,Variant *   QToolTipTitle)
{
}
```

Delphi

```
procedure QueryContextToolTip(ASender: TObject; QContext :
WideString;Keyword : WideString;var QToolTip : OleVariant;var QToolTipTitle :
OleVariant);
begin
end;
```

Delphi 8 (.NET only)

```
procedure QueryContextToolTip(sender: System.Object; e:
AxEXEDITLib._IEditEvents_QueryContextToolTipEvent);
begin
end;
```

Powe…

```
begin event QueryContextToolTip(string  QContext,string  Keyword,any
QToolTip,any  QToolTipTitle)

end event QueryContextToolTip
```

VB.NET

```
Private Sub QueryContextToolTip(ByVal sender As System.Object, ByVal e As
AxEXEDITLib._IEditEvents_QueryContextToolTipEvent) Handles
QueryContextToolTip
End Sub
```

VB6

```
Private Sub QueryContextToolTip(ByVal QContext As String,ByVal Keyword As
String,QToolTip As Variant,QToolTipTitle As Variant)
```

```
End Sub
```

VBA

```
Private Sub QueryContextToolTip(ByVal QContext As String,ByVal Keyword As
String,QToolTip As Variant,QToolTipTitle As Variant)
End Sub
```

VFP

```
LPARAMETERS QContext,Keyword,QToolTip,QToolTipTitle
```

Xbas…

```
PROCEDURE
OnQueryContextToolTip(oEdit,QContext,Keyword,QToolTip,QToolTipTitle)

RETURN
```

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

Java…

```
<SCRIPT
EVENT="QueryContextToolTip(QContext,Keyword,QToolTip,QToolTipTitle)"
LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function QueryContextToolTip(QContext,Keyword,QToolTip,QToolTipTitle)
End Function
</SCRIPT>
```

Visual Data…

```
Procedure OnComQueryContextToolTip String   llQContext String   llKeyword
Variant   llQToolTip Variant   llQToolTipTitle
   Forward Send OnComQueryContextToolTip llQContext llKeyword llQToolTip
llQToolTipTitle
End_Procedure
```

Visual Objects

```
METHOD OCX_QueryContextToolTip(QContext,Keyword,QToolTip,QToolTipTitle)
CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_QueryContextToolTip(str   _QContext,str   _Keyword,COMVariant
/*variant*/   _QToolTip,COMVariant /*variant*/   _QToolTipTitle)
```

```
{
}
```

XBasic

```
function QueryContextToolTip as v (QContext as C,Keyword as C,QToolTip as A,QToolTipTitle as A)
end function
```

dBASE

```
function nativeObject_QueryContextToolTip(QContext,Keyword,QToolTip,QToolTipTitle)
return
```

The QToolTip supports built-HTML as listed:

- ** ... ** displays the text in **bold**
- ** ... ** 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
- **<img>number[:width]</img>** 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.
- **<img>key[:width]</img>** 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 $_{\text{subscript}}$ 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** FFFFFF;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

How can I provide different tooltip for the same keyword?

VBA (MS Access, Excell...)

```
' QueryContext event - Queries for the context at the specified location, to provide different tooltips for the same keyword on QueryContextToolTip event.
Private Sub Edit1_QueryContext(ByVal XCursor As Long,ByVal YCursor As Long,QContext As Variant)
   With Edit1
      QContext = YCursor
   End With
End Sub

' QueryContextToolTip event - Asks for the tooltip/title of the keyword on the context retrieved by the QueryContext event.
Private Sub Edit1_QueryContextToolTip(ByVal QContext As String,ByVal Keyword As String,QToolTip As Variant,QToolTipTitle As Variant)
```

```
    With Edit1
        QToolTip = QContext
        QToolTipTitle = "Keyword Found At Line:"
    End With
End Sub

With Edit1
    .LineNumberWidth = -1
    .LineNumberBackColor = RGB(240,240,240)
    .AddKeyword "<fgcolor=FF0000><b>keyword</b></fgcolor>"
    .Text = ""
    .InsertText "here's the keyword on the first line"
    .InsertText "\r\nhere's the keyword on the second line"
    .InsertText "\r\nhere's the keyword on the third line"
End With
```

VB6

```
' QueryContext event - Queries for the context at the specified location, to provide different tooltips for the same keyword on QueryContextToolTip event.
Private Sub Edit1_QueryContext(ByVal XCursor As Long,ByVal YCursor As Long,QContext As Variant)
    With Edit1
        QContext = YCursor
    End With
End Sub

' QueryContextToolTip event - Asks for the tooltip/title of the keyword on the context retrieved by the QueryContext event.
Private Sub Edit1_QueryContextToolTip(ByVal QContext As String,ByVal Keyword As String,QToolTip As Variant,QToolTipTitle As Variant)
    With Edit1
        QToolTip = QContext
        QToolTipTitle = "Keyword Found At Line:"
    End With
End Sub
```

```
With Edit1
   .LineNumberWidth = -1
   .LineNumberBackColor = RGB(240,240,240)
   .AddKeyword "<fgcolor=FF0000><b>keyword</b></fgcolor>"
   .Text = ""
   .InsertText "here's the keyword on the first line"
   .InsertText "\r\nhere's the keyword on the second line"
   .InsertText "\r\nhere's the keyword on the third line"
End With
```

VB.NET

```
' QueryContext event - Queries for the context at the specified location, to provide different tooltips for the same keyword on QueryContextToolTip event.
Private Sub Exedit1_QueryContext(ByVal sender As System.Object,ByVal XCursor As Integer,ByVal YCursor As Integer,ByRef QContext As Object) Handles Exedit1.QueryContext
   With Exedit1
      QContext = YCursor
   End With
End Sub

' QueryContextToolTip event - Asks for the tooltip/title of the keyword on the context retrieved by the QueryContext event.
Private Sub Exedit1_QueryContextToolTip(ByVal sender As System.Object,ByVal QContext As String,ByVal Keyword As String,ByRef QToolTip As Object,ByRef QToolTipTitle As Object) Handles Exedit1.QueryContextToolTip
   With Exedit1
      QToolTip = QContext
      QToolTipTitle = "Keyword Found At Line:"
   End With
End Sub

With Exedit1
   .LineNumberWidth = -1
   .LineNumberBackColor = Color.FromArgb(240,240,240)
   .AddKeyword("<fgcolor=FF0000><b>keyword</b></fgcolor>")
```

```
    .Text = ""
    .InsertText("here's the keyword on the first line")
    .InsertText("\r\nhere's the keyword on the second line")
    .InsertText("\r\nhere's the keyword on the third line")
End With
```

VB.NET for /COM

```
' QueryContext event - Queries for the context at the specified location, to provide different tooltips for the same keyword on QueryContextToolTip event.
Private Sub AxEdit1_QueryContext(ByVal sender As System.Object, ByVal e As AxEXEDITLib._IEditEvents_QueryContextEvent) Handles AxEdit1.QueryContext
    With AxEdit1
        e.qContext = e.yCursor
    End With
End Sub

' QueryContextToolTip event - Asks for the tooltip/title of the keyword on the context retrieved by the QueryContext event.
Private Sub AxEdit1_QueryContextToolTip(ByVal sender As System.Object, ByVal e As AxEXEDITLib._IEditEvents_QueryContextToolTipEvent) Handles AxEdit1.QueryContextToolTip
    With AxEdit1
        e.qToolTip = e.qContext
        e.qToolTipTitle = "Keyword Found At Line:"
    End With
End Sub

With AxEdit1
    .LineNumberWidth = -1
    .LineNumberBackColor = RGB(240,240,240)
    .AddKeyword("<fgcolor=FF0000><b>keyword</b></fgcolor>")
    .Text = ""
    .InsertText("here's the keyword on the first line")
    .InsertText("\r\nhere's the keyword on the second line")
    .InsertText("\r\nhere's the keyword on the third line")
End With
```

C++

```
// QueryContext event - Queries for the context at the specified location, to provide
different tooltips for the same keyword on QueryContextToolTip event.
void OnQueryContextEdit1(long   XCursor,long   YCursor,VARIANT FAR*   QContext)
{
   /*
       Copy and paste the following directives to your header file as
       it defines the namespace 'EXEDITLib' for the library: 'ExEdit 1.0 Control Library'
       #import <ExEdit.dll>
       using namespace EXEDITLib;
    */
   EXEDITLib::IEditPtr spEdit1 = GetDlgItem(IDC_EDIT1)->GetControlUnknown();
   QContext = YCursor;
}

// QueryContextToolTip event - Asks for the tooltip/title of the keyword on the context
retrieved by the QueryContext event.
void OnQueryContextToolTipEdit1(LPCTSTR   QContext,LPCTSTR   Keyword,VARIANT
FAR*   QToolTip,VARIANT FAR*   QToolTipTitle)
{
   EXEDITLib::IEditPtr spEdit1 = GetDlgItem(IDC_EDIT1)->GetControlUnknown();
   QToolTip = QContext;
   QToolTipTitle = "Keyword Found At Line:";
}

EXEDITLib::IEditPtr spEdit1 = GetDlgItem(IDC_EDIT1)->GetControlUnknown();
spEdit1->PutLineNumberWidth(-1);
spEdit1->PutLineNumberBackColor(RGB(240,240,240));
spEdit1->AddKeyword(L"<fgcolor=FF0000><b>keyword</b>
</fgcolor>",vtMissing,vtMissing,vtMissing);
spEdit1->PutText(L"");
spEdit1->InsertText(L"here's the keyword on the first line",vtMissing);
spEdit1->InsertText(L"\\r\\nhere's the keyword on the second line",vtMissing);
spEdit1->InsertText(L"\\r\\nhere's the keyword on the third line",vtMissing);
```

C++ Builder

```
// QueryContext event - Queries for the context at the specified location, to provide different tooltips for the same keyword on QueryContextToolTip event.
void __fastcall TForm1::Edit1QueryContext(TObject *Sender,long   XCursor,long YCursor,Variant *   QContext)
{
   QContext = YCursor;
}

// QueryContextToolTip event - Asks for the tooltip/title of the keyword on the context retrieved by the QueryContext event.
void __fastcall TForm1::Edit1QueryContextToolTip(TObject *Sender,BSTR QContext,BSTR   Keyword,Variant *   QToolTip,Variant *   QToolTipTitle)
{
   QToolTip = QContext;
   QToolTipTitle = "Keyword Found At Line:";
}

Edit1->LineNumberWidth = -1;
Edit1->LineNumberBackColor = RGB(240,240,240);
Edit1->AddKeyword(L"<fgcolor=FF0000><b>keyword</b></fgcolor>",TNoParam(),TNoParam(),TNoParam());
Edit1->Text = L"";
Edit1->InsertText(L"here's the keyword on the first line",TNoParam());
Edit1->InsertText(L"\\r\\nhere's the keyword on the second line",TNoParam());
Edit1->InsertText(L"\\r\\nhere's the keyword on the third line",TNoParam());
```

C#

```
// QueryContext event - Queries for the context at the specified location, to provide different tooltips for the same keyword on QueryContextToolTip event.
private void exedit1_QueryContext(object sender,int   XCursor,int   YCursor,ref object QContext)
{
   QContext = YCursor;
}
```

```
//this.exedit1.QueryContext += new
exontrol.EXEDITLib.exg2antt.QueryContextEventHandler(this.exedit1_QueryContext);

// QueryContextToolTip event - Asks for the tooltip/title of the keyword on the context
retrieved by the QueryContext event.
private void exedit1_QueryContextToolTip(object sender,string   QContext,string
Keyword,ref object   QToolTip,ref object   QToolTipTitle)
{
   QToolTip = QContext;
   QToolTipTitle = "Keyword Found At Line:";
}
//this.exedit1.QueryContextToolTip += new
exontrol.EXEDITLib.exg2antt.QueryContextToolTipEventHandler(this.exedit1_QueryConte


exedit1.LineNumberWidth = -1;
exedit1.LineNumberBackColor = Color.FromArgb(240,240,240);
exedit1.AddKeyword("<fgcolor=FF0000><b>keyword</b>
</fgcolor>",null,null,null);
exedit1.Text = "";
exedit1.InsertText("here's the keyword on the first line",null);
exedit1.InsertText("\\r\\nhere's the keyword on the second line",null);
exedit1.InsertText("\\r\\nhere's the keyword on the third line",null);
```

JScript/JavaScript

```
<BODY onload="Init()">
<SCRIPT FOR="Edit1" EVENT="QueryContext(XCursor,YCursor,QContext)"
LANGUAGE="JScript">
   QContext = YCursor;
</SCRIPT>

<SCRIPT FOR="Edit1"
EVENT="QueryContextToolTip(QContext,Keyword,QToolTip,QToolTipTitle)"
LANGUAGE="JScript">
   QToolTip = QContext;
```

```
   QToolTipTitle = "Keyword Found At Line:";
</SCRIPT>

<OBJECT CLASSID="clsid:39136531-DD0F-4281-B445-E36FC2CDDBC5" id="Edit1">
</OBJECT>

<SCRIPT LANGUAGE="JScript">
function Init()
{
   Edit1.LineNumberWidth = -1;
   Edit1.LineNumberBackColor = 15790320;
   Edit1.AddKeyword("<fgcolor=FF0000><b>keyword</b>
</fgcolor>",null,null,null);
   Edit1.Text = "";
   Edit1.InsertText("here's the keyword on the first line",null);
   Edit1.InsertText("\\r\\nhere's the keyword on the second line",null);
   Edit1.InsertText("\\r\\nhere's the keyword on the third line",null);
}
</SCRIPT>
</BODY>
```

VBScript

```
<BODY onload="Init()">
<SCRIPT LANGUAGE="VBScript">
Function Edit1_QueryContext(XCursor,YCursor,QContext)
   With Edit1
      QContext = YCursor
   End With
End Function
</SCRIPT>

<SCRIPT LANGUAGE="VBScript">
Function Edit1_QueryContextToolTip(QContext,Keyword,QToolTip,QToolTipTitle)
   With Edit1
      QToolTip = QContext
```

```
        QToolTipTitle = "Keyword Found At Line:"
    End With
End Function
</SCRIPT>

<OBJECT CLASSID="clsid:39136531-DD0F-4281-B445-E36FC2CDDBC5" id="Edit1">
</OBJECT>

<SCRIPT LANGUAGE="VBScript">
Function Init()
    With Edit1
        .LineNumberWidth = -1
        .LineNumberBackColor = RGB(240,240,240)
        .AddKeyword "<fgcolor=FF0000><b>keyword</b></fgcolor>"
        .Text = ""
        .InsertText "here's the keyword on the first line"
        .InsertText "\r\nhere's the keyword on the second line"
        .InsertText "\r\nhere's the keyword on the third line"
    End With
End Function
</SCRIPT>
</BODY>
```

C# for /COM

```
// QueryContext event - Queries for the context at the specified location, to provide
different tooltips for the same keyword on QueryContextToolTip event.
private void axEdit1_QueryContext(object sender,
AxEXEDITLib._IEditEvents_QueryContextEvent e)
{
    e.qContext = e.yCursor;
}
//this.axEdit1.QueryContext += new
AxEXEDITLib._IEditEvents_QueryContextEventHandler(this.axEdit1_QueryContext);

// QueryContextToolTip event - Asks for the tooltip/title of the keyword on the context
```

```
retrieved by the QueryContext event.
private void axEdit1_QueryContextToolTip(object sender,
AxEXEDITLib._IEditEvents_QueryContextToolTipEvent e)
{
    e.qToolTip = e.qContext;
    e.qToolTipTitle = "Keyword Found At Line:";
}
//this.axEdit1.QueryContextToolTip += new
AxEXEDITLib._IEditEvents_QueryContextToolTipEventHandler(this.axEdit1_QueryContext


axEdit1.LineNumberWidth = -1;
axEdit1.LineNumberBackColor = Color.FromArgb(240,240,240);
axEdit1.AddKeyword("<fgcolor=FF0000><b>keyword</b>
</fgcolor>",null,null,null);
axEdit1.Text = "";
axEdit1.InsertText("here's the keyword on the first line",null);
axEdit1.InsertText("\\r\\nhere's the keyword on the second line",null);
axEdit1.InsertText("\\r\\nhere's the keyword on the third line",null);
```

X++ (Dynamics Ax 2009)

```
// QueryContext event - Queries for the context at the specified location, to provide
different tooltips for the same keyword on QueryContextToolTip event.
void onEvent_QueryContext(int   _XCursor,int   _YCursor,COMVariant /*variant*/
_QContext)
{
    ;
    _QContext = _YCursor;
}

// QueryContextToolTip event - Asks for the tooltip/title of the keyword on the context
retrieved by the QueryContext event.
void onEvent_QueryContextToolTip(str   _QContext,str   _Keyword,COMVariant
/*variant*/   _QToolTip,COMVariant /*variant*/   _QToolTipTitle)
{
```

```
    ;
    _QToolTip = _QContext;
    _QToolTipTitle = "Keyword Found At Line:";
}

public void init()
{
    ;

    super();

    exedit1.LineNumberWidth(-1);
    exedit1.LineNumberBackColor(WinApi::RGB2int(240,240,240));
    exedit1.AddKeyword("<fgcolor=FF0000><b>keyword</b></fgcolor>");
    exedit1.Text("");
    exedit1.InsertText("here's the keyword on the first line");
    exedit1.InsertText("\\r\\nhere's the keyword on the second line");
    exedit1.InsertText("\\r\\nhere's the keyword on the third line");
}
```

Delphi 8 (.NET only)

```
// QueryContext event - Queries for the context at the specified location, to provide
different tooltips for the same keyword on QueryContextToolTip event.
procedure TWinForm1.AxEdit1_QueryContext(sender: System.Object; e:
AxEXEDITLib._IEditEvents_QueryContextEvent);
begin
    with AxEdit1 do
    begin
        e.qContext := e.yCursor;
    end
end;

// QueryContextToolTip event - Asks for the tooltip/title of the keyword on the context
retrieved by the QueryContext event.
procedure TWinForm1.AxEdit1_QueryContextToolTip(sender: System.Object; e:
AxEXEDITLib._IEditEvents_QueryContextToolTipEvent);
```

```
begin
   with AxEdit1 do
   begin
      e.qToolTip := e.qContext;
      e.qToolTipTitle := 'Keyword Found At Line:';
   end
end;

with AxEdit1 do
begin
   LineNumberWidth := -1;
   LineNumberBackColor := Color.FromArgb(240,240,240);
   AddKeyword('<fgcolor=FF0000><b>keyword</b></fgcolor>',Nil,Nil,Nil);
   Text := '';
   InsertText('here''s the keyword on the first line',Nil);
   InsertText('\r\nhere''s the keyword on the second line',Nil);
   InsertText('\r\nhere''s the keyword on the third line',Nil);
end
```

Delphi (standard)

```
// QueryContext event - Queries for the context at the specified location, to provide
different tooltips for the same keyword on QueryContextToolTip event.
procedure TForm1.Edit1QueryContext(ASender: TObject; XCursor : Integer;YCursor :
Integer;var QContext : OleVariant);
begin
   with Edit1 do
   begin
      QContext := YCursor;
   end
end;

// QueryContextToolTip event - Asks for the tooltip/title of the keyword on the context
retrieved by the QueryContext event.
procedure TForm1.Edit1QueryContextToolTip(ASender: TObject; QContext :
WideString;Keyword : WideString;var QToolTip : OleVariant;var QToolTipTitle :
OleVariant);
```

```
begin
   with Edit1 do
   begin
      QToolTip := QContext;
      QToolTipTitle := 'Keyword Found At Line:';
   end
end;

with Edit1 do
begin
   LineNumberWidth := -1;
   LineNumberBackColor := RGB(240,240,240);
   AddKeyword('<fgcolor=FF0000><b>keyword</b></fgcolor>',Null,Null,Null);
   Text := '';
   InsertText('here''s the keyword on the first line',Null);
   InsertText('\r\nhere''s the keyword on the second line',Null);
   InsertText('\r\nhere''s the keyword on the third line',Null);
end
```

VFP

```
*** QueryContext event - Queries for the context at the specified location, to provide
different tooltips for the same keyword on QueryContextToolTip event. ***
LPARAMETERS XCursor,YCursor,QContext
   with thisform.Edit1
      QContext = YCursor
   endwith

*** QueryContextToolTip event - Asks for the tooltip/title of the keyword on the
context retrieved by the QueryContext event. ***
LPARAMETERS QContext,Keyword,QToolTip,QToolTipTitle
   with thisform.Edit1
      QToolTip = QContext
      QToolTipTitle = "Keyword Found At Line:"
   endwith

with thisform.Edit1
```

```
    .LineNumberWidth = -1
    .LineNumberBackColor = RGB(240,240,240)
    .AddKeyword("<fgcolor=FF0000><b>keyword</b></fgcolor>")
    .Text = ""
    .InsertText("here's the keyword on the first line")
    .InsertText("\r\nhere's the keyword on the second line")
    .InsertText("\r\nhere's the keyword on the third line")
endwith
```

dBASE Plus

```
/*
with (this.EXEDITACTIVEXCONTROL1.nativeObject)
    QueryContext = class::nativeObject_QueryContext
endwith
*/
// Queries for the context at the specified location, to provide different tooltips for the
same keyword on QueryContextToolTip event.
function nativeObject_QueryContext(XCursor,YCursor,QContext)
   oEdit = form.EXEDITACTIVEXCONTROL1.nativeObject
   QContext = YCursor
return

/*
with (this.EXEDITACTIVEXCONTROL1.nativeObject)
    QueryContextToolTip = class::nativeObject_QueryContextToolTip
endwith
*/
// Asks for the tooltip/title of the keyword on the context retrieved by the
QueryContext event.
function
nativeObject_QueryContextToolTip(QContext,Keyword,QToolTip,QToolTipTitle)
   oEdit = form.EXEDITACTIVEXCONTROL1.nativeObject
   QToolTip = QContext
   QToolTipTitle = "Keyword Found At Line:"
return
```

```
local oEdit

oEdit = form.EXEDITACTIVEXCONTROL1.nativeObject
oEdit.LineNumberWidth = -1
oEdit.LineNumberBackColor = 0xf0f0f0
oEdit.AddKeyword("<fgcolor=FF0000><b>keyword</b></fgcolor>")
oEdit.Text = ""
oEdit.InsertText("here's the keyword on the first line")
oEdit.InsertText("\r\nhere's the keyword on the second line")
oEdit.InsertText("\r\nhere's the keyword on the third line")
```

XBasic (Alpha Five)

```
' Queries for the context at the specified location, to provide different tooltips for the same keyword on QueryContextToolTip event.
function QueryContext as v (XCursor as N,YCursor as N,QContext as A)
   oEdit = topparent:CONTROL_ACTIVEX1.activex
   QContext = YCursor
end function

' Asks for the tooltip/title of the keyword on the context retrieved by the QueryContext event.
function QueryContextToolTip as v (QContext as C,Keyword as C,QToolTip as A,QToolTipTitle as A)
   oEdit = topparent:CONTROL_ACTIVEX1.activex
   QToolTip = QContext
   QToolTipTitle = "Keyword Found At Line:"
end function

Dim oEdit as P

oEdit = topparent:CONTROL_ACTIVEX1.activex
oEdit.LineNumberWidth = -1
oEdit.LineNumberBackColor = 15790320
oEdit.AddKeyword("<fgcolor=FF0000><b>keyword</b></fgcolor>")
oEdit.Text = ""
```

```
oEdit.InsertText("here's the keyword on the first line")
oEdit.InsertText("\r\nhere's the keyword on the second line")
oEdit.InsertText("\r\nhere's the keyword on the third line")
```

Visual Objects

```
METHOD OCX_Exontrol1QueryContext(XCursor,YCursor,QContext) CLASS MainDialog
   // QueryContext event - Queries for the context at the specified location, to provide
different tooltips for the same keyword on QueryContextToolTip event.
   QContext := YCursor
RETURN NIL

METHOD
OCX_Exontrol1QueryContextToolTip(QContext,Keyword,QToolTip,QToolTipTitle)
CLASS MainDialog
   // QueryContextToolTip event - Asks for the tooltip/title of the keyword on the
context retrieved by the QueryContext event.
   QToolTip := QContext
   QToolTipTitle := "Keyword Found At Line:"
RETURN NIL



oDCOCX_Exontrol1:LineNumberWidth := -1
oDCOCX_Exontrol1:LineNumberBackColor := RGB(240,240,240)
oDCOCX_Exontrol1:AddKeyword("<fgcolor=FF0000><b>keyword</b>
</fgcolor>",nil,nil,nil)
oDCOCX_Exontrol1:Text := ""
oDCOCX_Exontrol1:InsertText("here's the keyword on the first line",nil)
oDCOCX_Exontrol1:InsertText("\r\nhere's the keyword on the second line",nil)
oDCOCX_Exontrol1:InsertText("\r\nhere's the keyword on the third line",nil)
```

PowerBuilder

```
/*begin event QueryContext(long  XCursor,long  YCursor,any  QContext) - Queries for
the context at the specified location, to provide different tooltips for the same
keyword on QueryContextToolTip event.*/
```

```
/*
    oEdit = ole_1.Object
    QContext = YCursor
*/
/*end event QueryContext*/

/*begin event QueryContextToolTip(string  QContext,string  Keyword,any
QToolTip,any  QToolTipTitle) - Asks for the tooltip/title of the keyword on the context
retrieved by the QueryContext event.*/
/*
    oEdit = ole_1.Object
    QToolTip = QContext
    QToolTipTitle = "Keyword Found At Line:"
*/
/*end event QueryContextToolTip*/

OleObject oEdit

oEdit = ole_1.Object
oEdit.LineNumberWidth = -1
oEdit.LineNumberBackColor = RGB(240,240,240)
oEdit.AddKeyword("<fgcolor=FF0000><b>keyword</b></fgcolor>")
oEdit.Text = ""
oEdit.InsertText("here's the keyword on the first line")
oEdit.InsertText("\r\nhere's the keyword on the second line")
oEdit.InsertText("\r\nhere's the keyword on the third line")
```

Visual DataFlex

```
// Queries for the context at the specified location, to provide different tooltips for the
same keyword on QueryContextToolTip event.
Procedure OnComQueryContext Integer  llXCursor Integer  llYCursor Variant
llQContext
   Forward Send OnComQueryContext llXCursor llYCursor llQContext
   Move llYCursor to QContext
End_Procedure
```

```
// Asks for the tooltip/title of the keyword on the context retrieved by the
QueryContext event.
Procedure OnComQueryContextToolTip String   llQContext String   llKeyword Variant
llQToolTip Variant   llQToolTipTitle
   Forward Send OnComQueryContextToolTip llQContext llKeyword llQToolTip
llQToolTipTitle
   Move llQContext to QToolTip
   Move "Keyword Found At Line:" to QToolTipTitle
End_Procedure

Procedure OnCreate
   Forward Send OnCreate
   Set ComLineNumberWidth to -1
   Set ComLineNumberBackColor to (RGB(240,240,240))
   Send ComAddKeyword "<fgcolor=FF0000><b>keyword</b></fgcolor>"
Nothing Nothing Nothing
   Set ComText to ""
   Send ComInsertText "here's the keyword on the first line" Nothing
   Send ComInsertText "\r\nhere's the keyword on the second line" Nothing
   Send ComInsertText "\r\nhere's the keyword on the third line" Nothing
End_Procedure
```

XBase++

```
PROCEDURE OnQueryContext(oEdit,XCursor,YCursor,QContext)
   QContext := YCursor
RETURN

PROCEDURE
OnQueryContextToolTip(oEdit,QContext,Keyword,QToolTip,QToolTipTitle)
   QToolTip := QContext
   QToolTipTitle := "Keyword Found At Line:"
RETURN

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

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

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

   oEdit := XbpActiveXControl():new( oForm:drawingArea )
   oEdit:CLSID  := "Exontrol.Edit.1" /*{39136531-DD0F-4281-B445-
E36FC2CDDBC5}*/
   oEdit:create(,, {10,60},{610,370} )

      oEdit:QueryContext := {|XCursor,YCursor,QContext|
OnQueryContext(oEdit,XCursor,YCursor,QContext)} /*Queries for the context at the
specified location, to provide different tooltips for the same keyword on
QueryContextToolTip event.*/
      oEdit:QueryContextToolTip := {|QContext,Keyword,QToolTip,QToolTipTitle|
OnQueryContextToolTip(oEdit,QContext,Keyword,QToolTip,QToolTipTitle)} /*Asks for
the tooltip/title of the keyword on the context retrieved by the QueryContext event.*/

      oEdit:LineNumberWidth := -1
      oEdit:SetProperty("LineNumberBackColor",AutomationTranslateColor(
GraMakeRGBColor  ( { 240,240,240 } )  , .F. ))
      oEdit:AddKeyword("<fgcolor=FF0000><b>keyword</b></fgcolor>")
      oEdit:Text := ""
      oEdit:InsertText("here's the keyword on the first line")
      oEdit:InsertText("\r\nhere's the keyword on the second line")
      oEdit:InsertText("\r\nhere's the keyword on the third line")

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

```
   ENDDO
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 ScrolPartCaption 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.EXEDITLib.ScrollBarEnum
ScrollBar,exontrol.EXEDITLib.ScrollPartEnum ScrollPart)
{
}
```

VB

```
Private Sub ScrollButtonClick(ByVal sender As System.Object,ByVal ScrollBar As
exontrol.EXEDITLib.ScrollBarEnum,ByVal ScrollPart As
exontrol.EXEDITLib.ScrollPartEnum) Handles ScrollButtonClick
End Sub
```

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

C#

```
private void ScrollButtonClick(object sender,
AxEXEDITLib._IEditEvents_ScrollButtonClickEvent e)
{
}
```

C++

```
void OnScrollButtonClick(long ScrollBar,long ScrollPart)
{
```

```
}
```

C++ Builder

```
void __fastcall ScrollButtonClick(TObject *Sender,Exeditlib_tlb::ScrollBarEnum
ScrollBar,Exeditlib_tlb::ScrollPartEnum ScrollPart)
{
}
```

Delphi

```
procedure ScrollButtonClick(ASender: TObject; ScrollBar :
ScrollBarEnum;ScrollPart : ScrollPartEnum);
begin
end;
```

Delphi 8 (.NET only)

```
procedure ScrollButtonClick(sender: System.Object; e:
AxEXEDITLib._IEditEvents_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
AxEXEDITLib._IEditEvents_ScrollButtonClickEvent) Handles ScrollButtonClick
End Sub
```

VB6

```
Private Sub ScrollButtonClick(ByVal ScrollBar As EXEDITLibCtl.ScrollBarEnum,ByVal
ScrollPart As EXEDITLibCtl.ScrollPartEnum)
End Sub
```

VBA

```
Private Sub ScrollButtonClick(ByVal ScrollBar As Long,ByVal ScrollPart As Long)
End Sub
```

VFP

```
LPARAMETERS ScrollBar,ScrollPart
```

Xbas…

```
PROCEDURE OnScrollButtonClick(oEdit,ScrollBar,ScrollPart)
RETURN
```

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

Java...
```
<SCRIPT EVENT="ScrollButtonClick(ScrollBar,ScrollPart)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc...
```
<SCRIPT LANGUAGE="VBScript">
Function ScrollButtonClick(ScrollBar,ScrollPart)
End Function
</SCRIPT>
```

Visual Data...
```
Procedure OnComScrollButtonClick OLEScrollBarEnum llScrollBar
OLEScrollPartEnum llScrollPart
   Forward Send OnComScrollButtonClick llScrollBar llScrollPart
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.Edit.1::ScrollBarEnum,ScrollPart as
OLE::Exontrol.Edit.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 Edit1
    .ScrollPartVisible(exVScroll, exLeftB1Part Or exRightB1Part) = True
    .ScrollPartCaption(exVScroll, exLeftB1Part) = "1"
    .ScrollPartCaption(exVScroll, exRightB1Part) = "2"
End With
```

```
Private Sub Edit1_ScrollButtonClick(ByVal ScrollPart As EXEDITLibCtl.ScrollPartEnum)
   MsgBox (ScrollPart)
End Sub
```

The following VB.NET sample displays the identifier of the scroll's button being clicked:

```
With AxEdit1
   .set_ScrollPartVisible(EXEDITLib.ScrollBarEnum.exVScroll,
EXEDITLib.ScrollPartEnum.exLeftB1Part Or EXEDITLib.ScrollPartEnum.exRightB1Part, True)
   .set_ScrollPartCaption(EXEDITLib.ScrollBarEnum.exVScroll,
EXEDITLib.ScrollPartEnum.exLeftB1Part, "1")
   .set_ScrollPartCaption(EXEDITLib.ScrollBarEnum.exVScroll,
EXEDITLib.ScrollPartEnum.exRightB1Part, "2")
End With
```

```
Private Sub AxEdit1_ScrollButtonClick(ByVal sender As System.Object, ByVal e As
AxEXEDITLib._IEditEvents_ScrollButtonClickEvent) Handles AxEdit1.ScrollButtonClick
      MessageBox.Show( e.scrollPart.ToString())
End Sub
```

The following C# sample displays the identifier of the scroll's button being clicked:

```
axEdit1.set_ScrollPartVisible(EXEDITLib.ScrollBarEnum.exVScroll,
EXEDITLib.ScrollPartEnum.exLeftB1Part | EXEDITLib.ScrollPartEnum.exRightB1Part, true);
axEdit1.set_ScrollPartCaption(EXEDITLib.ScrollBarEnum.exVScroll,
EXEDITLib.ScrollPartEnum.exLeftB1Part , "1");
axEdit1.set_ScrollPartCaption(EXEDITLib.ScrollBarEnum.exVScroll,
EXEDITLib.ScrollPartEnum.exRightB1Part, "2");
```

```
private void axEdit1_ScrollButtonClick(object sender,
AxEXEDITLib._IEditEvents_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 OnScrollButtonClickEdit1(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.Edit1
      .ScrollPartVisible(0, bitor(32768,32)) = .t.
      .ScrollPartCaption(0,32768) = "1"
      .ScrollPartCaption(0, 32) = "2"
EndWit
```

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. Use the GetSelection method to determine the coordinates of the selected text. The SelStart and SelLenght properties determine the position of the selected text. Use the SelForeColor and SelBackColor properties to specify the colors for the selected text.

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;
```

Powe…

```
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(oEdit)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComSelChange
   Forward Send OnComSelChange
```

```
End_Procedure
```

Visual Objects

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

X++

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

XBasic

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

dBASE

```
function nativeObject_SelChange()
return
```

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

```
Private Sub Edit1_SelChange()
   If Not Edit1.SelText = "" Then
      Debug.Print Edit1.SelText
   End If
End Sub
```

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

```
void OnSelChangeEdit1()
{
   OutputDebugString( m_edit.GetSelText() );
}
```

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

```
Private Sub AxEdit1_SelChange(ByVal sender As Object, ByVal e As System.EventArgs)
Handles AxEdit1.SelChange
   With AxEdit1
      Debug.WriteLine(.SelText)
   End With
End Sub
```

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

```
private void axEdit1_SelChange(object sender, EventArgs e)
{
    System.Diagnostics.Debug.WriteLine(axEdit1.SelText);
}
```

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

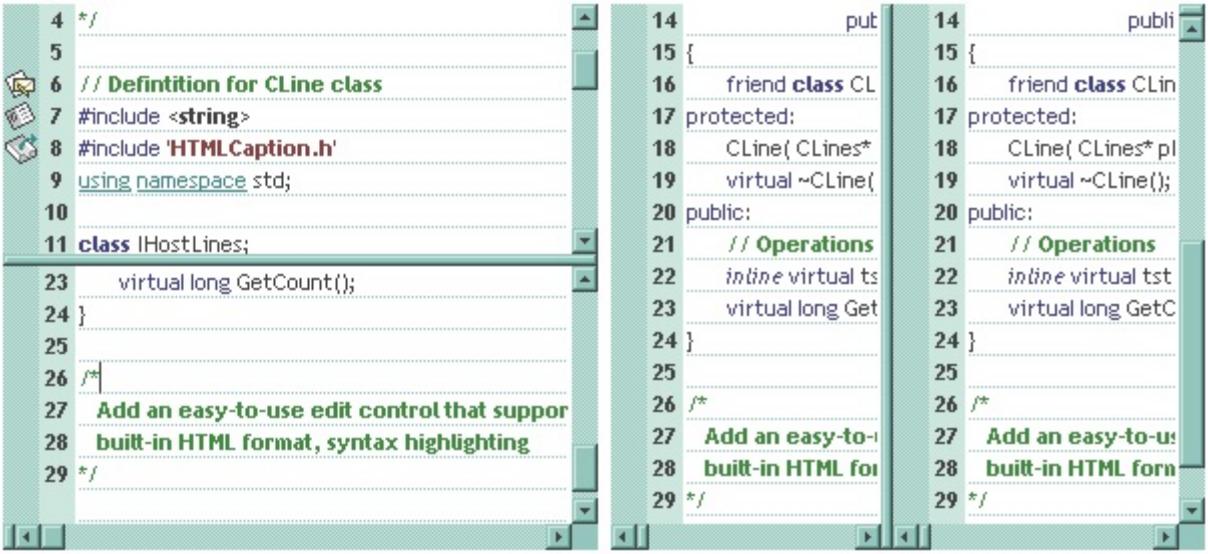
```
*** ActiveX Control Event ***

with thisform.Edit1
    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(oEdit)
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
```