# ExMaskEdit

When you have several people entering data in your database, you can define how users must enter data in specific fields to help maintain consistency and to make your database easier to manage. For example, you can set an input mask for a form so that users can only enter telephone numbers in the Swedish format or addresses in the French format. You can set a specific format for the input mask, and select another format so that the same data is displayed differently.

Features of eXMaskEdit include:

- Ability to define and use one or more masks at runtime
- Insert or **Overtype** mode support
- **Restrict Input Data** until the user enters the appropriate value
- **HTML ToolTip**, Warning, Beep support, so a tooltip is shown once the user enters any invalid data
- Ability to highlight entities with a different color, while the entire mask is not completed
- Ability to define valid characters using the [] directive, or specify the margins of the number to be entered using the {} directives
- Left/Right alignment support

Here's a list of types you can mask:

- floating/decimal point numbers support, including grouping of digits
- license keys
- IP addresses
- urls
- e-mails
- phone numbers
- extension, zip code
- social security numbers
- decimal numbers
- hexa numbers
- binary numbers
- alpha and digit characters
- RGB, A-RGB colors
- date, time
- passwords
- and more
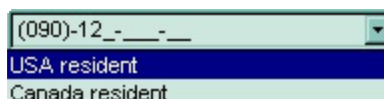
How to get support?

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 AllowEmptyValueEnum

The AllowEmptyValueEnum type supports the values that the AllowEmptyValue property supports. The AllowEmptyValueEnum type supports the following values.

Name	Value	Description
exNoEmptyValue	0	No empty value is accepted, so the control displays the control's mask when its content is empty.
exAllowEmptyValueShowNothing	1	While the control's content is empty, the control's mask is showing as soon as the user starts typing character inside.
exAllowEmptyValueShowMask	1	While the control's content is empty, the control's mask is showing as soon as the control gains the focus.

constants AppearanceEnum

The AppearanceEnum enumeration is used to specify the appearance of the control's border/frame. The [Appearance](#) property of the control specifies the control's border. The AppearanceEnum type supports the following values:

Name	Value	Description
None2	0	No border
Flat	1	Flat border
Sunken	2	Sunken border
Raised	3	Raised border
Etched	4	Etched border
Bump	5	Bump border

constants BackgroundPartEnum

The BackgroundPartEnum type specifies parts of the control to be visually changed. The [Background](#) property specifies the control's part background/foreground color or visual appearance. The BackgroundPartEnum type supports the following values:

Name	Value	Description
exToolTipAppearance	64	Specifies the visual appearance of the borders of the tooltips.
exToolTipBackColor	65	Specifies the tooltip's background color.
exToolTipForeColor	66	Specifies the tooltip's foreground color.

constants ClipModeLiteralsEnum

The ClipModeLiteralsEnum type indicates the way the control's text/value is gathered. The [TextIncludeLiterals](#), [TextIncludeLiteralsLoseFocus](#) or [Value](#) property may use the ClipModeLiteralsEnum type. The ClipModeLiteralsEnum type supports the following values:

Name	Value	Description
exClipModeLiteralsNone	0	Gets the full text.
exClipModeLiteralsInclude	1	Gets the value with no placeholders.
exClipModeLiteralsExclude	2	Gets the value with no placeholders and literals.
exClipModeLiteralsEscape	3	Gets the value of the optional, required and escaped entities. The quoted literals are not included.

constants InsertModeEnum

The InsertModeEnum type specifies the insertion mode the control supports. The [InsertMode](#) property specifies the current control's insertion mode. The InsertModeEnum type supports the following values:

Name	Value	Description
exEditInsertMode	0	Indicates the control's insert-type mode.
exEditOvertypeMode	1	Indicates the control's over-type mode.

constants PictureDisplayEnum

Only for internal use.

Name	Value	Description
UpperLeft	0	UpperLeft
UpperCenter	1	UpperCenter
UpperRight	2	UpperRight
MiddleLeft	16	MiddleLeft
MiddleCenter	17	MiddleCenter
MiddleRight	18	MiddleRight
LowerLeft	32	LowerLeft
LowerCenter	33	LowerCenter
LowerRight	34	LowerRight
Tile	48	Tile
Stretch	49	Stretch

constants SelectGotFocusEnum

The SelectGotFocusEnum type indicates how the control specifies the selection once the control gains the focus. The [SelectGotFocus](#) property indicates how the control specifies the selection once the control gets the focus. The SelectGotFocusEnum type supports the following values.

Name	Value	Description
exSelectNoGotFocus	0	No effect.
exSelectAllGotFocus	1	Selects all once the field receives the focus.
exSelectEditableGotFocus	2	Selects the first empty and editable entity of the field.
exMoveEditableGotFocus	3	Moves the cursor to the first empty and editable entity of the field.
exSelectRequiredEditableGotFocus	4	Selects the first empty, required and editable entity of the field.
exMoveRequiredEditableGotFocus	5	Moves the cursor to the first empty, required and editable entity of the field.

constants TypeEnum

Specifies whether the control subclasses a standard edit control or the Rich editor.

Name	Value	Description
exTypeEdit	0	Indicates the standard edit control.
exTypeRichEdit	1	Indicates the rich control. The control subclass the system's richedit class. You can use the ForeColorRich property to specify the foreground color for editable entities while the field is not validated/completed.

constants UIVisualThemeEnum

Only for internal use.

Name	Value	Description
exNoVisualTheme	0	exNoVisualTheme
exDefaultVisualTheme	16777215	exDefaultVisualTheme
exHeaderVisualTheme	1	exHeaderVisualTheme
exFilterBarVisualTheme	2	exFilterBarVisualTheme
exButtonsVisualTheme	4	exButtonsVisualTheme
exCalendarVisualTheme	8	exCalendarVisualTheme
exSliderVisualTheme	16	exSliderVisualTheme
exSpinVisualTheme	32	exSpinVisualTheme
exCheckBoxVisualTheme	64	exCheckBoxVisualTheme
exProgressVisualTheme	128	exProgressVisualTheme
exCalculatorVisualTheme	256	exCalculatorVisualTheme

constants ValidateAsEnum

The ValidateAsEnum type specifies the additional validation the control can make when user leaves the field. The ValidateAs property specifies the additional validation the control can make when user leaves the field. The ValidateAsEnum type supports the following values:

Name	Value	Description
exValidateAsNone	0	No effect.
exValidateAsDate	1	The field is validated as date.

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.
RenderType	Specifies the way colored EBN objects are displayed on the component.

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.
Skin as Variant	A string expression that indicates: • an Windows XP Theme part, it should start with "XP:". 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. In this case the format of the Skin parameter should be: "XP: Control/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 like listed at the end of the document. This option is available only on Windows XP that supports Themes API. • copy of another skin with different coordinates, if it begins with "CP:" . For instance, you may need to display a specified skin on a smaller rectangle. In this case, the string starts with "CP:", and contains the following "CP:n l t r b", where the n is the identifier being copied, the l, t, r, and b indicate the left, top, right and bottom coordinates being used to adjust the rectangle where the skin is displayed. For instance, the "**CP:1 4 0 -4 0**", indicates that the skin is displayed on a smaller rectangle like follows. Let's say that the control requests painting the {10, 10, 30, 20} area, a rectangle with the width of 20 pixels, and the height of 10 pixels, the skin will be displayed on the {14,10,26,20} as each coordinates in the "CP" syntax is added to the displayed rectangle, so the skin looks smaller. This way you can apply different effects to your objects in your control. • the path to the skin file (*.ebn). The Exontrol's exButton component installs a skin builder that should be used to create new skins

- the BASE64 encoded string that holds a skin file (*.ebn). Use the Exontrol's exImages tool to build BASE 64 encoded strings on the skin file (*.ebn) you have created. Loading the skin from a file (eventually uncompressed file) is always faster then loading from a BASE64 encoded string

A byte[] or safe arrays of VT_I1 or VT_UI1 expression that indicates the content of the EBN file. You can use this 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)).

| 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. Use the Background property to apply a skin to a part of the control. Use the Appearance property to change the visual appearance of the control's frame.

The following screen shot shows the control's with visual appearance changed:
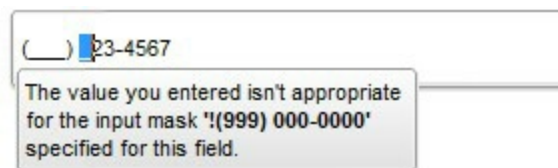
The following screen shot shows the control's with no visual appearance changed:
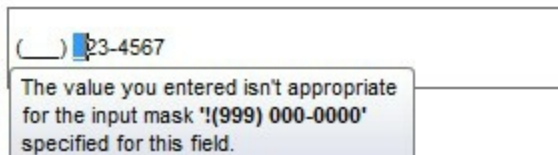
The identifier you choose for the skin is very important to be used in the background properties like explained bellow. Shortly, the color properties uses 4 bytes (DWORD, double WORD, and so on) to hold a RGB value. More than that, the first byte (most significant byte in the color) is used only to specify 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. For instance, the BackColor = BackColor Or &H2000000 indicates that we apply the skin with the index 2 using the old color, to the object that BackColor is applied.

On **Windows XP**, the following table shows how the common controls are broken into parts and states:

Control/ClassName	Part	States
BUTTON	BP_CHECKBOX = 3	CBS_UNCHECKED 1 CBS_UNCHECKE CBS_UNCHECKED = 3 CBS_UNCHECKED = 4 CBS_CHECKEI 5 CBS_CHECKEDH CBS_CHECKEDPR CBS_CHECKEDDIS CBS_MIXEDNORM CBS_MIXEDHOT = CBS_MIXEDPRESS CBS_MIXEDDISAB
	BP_GROUPBOX = 4	GBS_NORMAL = 1 GBS_DISABLED =
	BP_PUSHBUTTON = 1	PBS_NORMAL = 1 = 2 PBS_PRESSED PBS_DISABLED = PBS_DEFAULTED =
	BP_RADIOBUTTON = 2	RBS_UNCHECKED 1 RBS_UNCHECKE RBS_UNCHECKED = 3 RBS_UNCHECKED = 4 RBS_CHECKEI

		5 RBS_CHECKEDH RBS_CHECKEDPR RBS_CHECKEDDIS
	BP_USERBUTTON = 5	
CLOCK	CLP_TIME = 1	CLS_NORMAL = 1
COMBOBOX	CP_DROPDOWNBUTTON = 1	CBXS_NORMAL = CBXS_HOT = 2 CBXS_PRESSED = CBXS_DISABLED =
EDIT	EP_CARET = 2	
	EP_EDITTEXT = 1	ETS_NORMAL = 1 2 ETS_SELECTED ETS_DISABLED = ETS_FOCUSED = ! ETS_READONLY = ETS_ASSIST = 7
EXPLORERBAR	EBP_HEADERBACKGROUND = 1	
	EBP_HEADERCLOSE = 2	EBHC_NORMAL = EBHC_HOT = 2 EBHC_PRESSED =
	EBP_HEADERPIN = 3	EBHP_NORMAL = EBHP_HOT = 2 EBHP_PRESSED = EBHP_SELECTEDM 4 EBHP_SELECTEI EBHP_SELECTEDP 6
	EBP_IEBARMENU = 4	EBM_NORMAL = 1 = 2 EBM_PRESSEI
	EBP_NORMALGROUPBACKGROUND = 5	
	EBP_NORMALGROUPCOLLAPSE = 6	EBNGC_NORMAL = EBNGC_HOT = 2 EBNGC_PRESSED
	EBP_NORMALGROUPEXPAND = 7	EBNGE_NORMAL = EBNGE_HOT = 2 EBNGE_PRESSED
	EBP_NORMALGROUPHEAD = 8	
	EBP_SPECIALGROUPBACKGROUND = 9	
	EBP_SPECIALGROUPCOLLAPSE = 10	EBSGC_NORMAL = EBSGC_HOT = 2

	EBP_SPECIALGROUPEXPAND = 11	EBSGC_PRESSED EBSGE_NORMAL = EBSGE_HOT = 2 EBSGE_PRESSED
	EBP_SPECIALGROUPHEAD = 12	
HEADER	HP_HEADERITEM = 1	HIS_NORMAL = 1 H 2 HIS_PRESSED =
	HP_HEADERITEMLEFT = 2	HILS_NORMAL = 1 = 2 HILS_PRESSEI
	HP_HEADERITEMRIGHT = 3	HIRS_NORMAL = 1 = 2 HIRS_PRESSE
	HP_HEADERSORTARROW = 4	HSAS_SORTEDUP HSAS_SORTEDDC
LISTVIEW	LVP_EMPTYTEXT = 5 LVP_LISTDETAIL = 3 LVP_LISTGROUP = 2	
	LVP_LISTITEM = 1	LIS_NORMAL = 1 L 2 LIS_SELECTED = LIS_DISABLED = 4 LIS_SELECTEDNO 5
	LVP_LISTSORTEDDETAIL = 4	
MENU	MP_MENUBARDROPDOWN = 4	MS_NORMAL = 1 MS_SELECTED = : MS_DEMOTED = 3
	MP_MENUBARITEM = 3	MS_NORMAL = 1 MS_SELECTED = : MS_DEMOTED = 3
	MP_CHEVRON = 5	MS_NORMAL = 1 MS_SELECTED = : MS_DEMOTED = 3
	MP_MENUDROPDOWN = 2	MS_NORMAL = 1 MS_SELECTED = : MS_DEMOTED = 3
	MP_MENUITEM = 1	MS_NORMAL = 1 MS_SELECTED = : MS_DEMOTED = 3
	MP_SEPARATOR = 6	MS_NORMAL = 1 MS_SELECTED = :

Category	Part	State
MENUBAND	MDP_NEWAPPBUTTON = 1	MS_DEMOTED = 3 MDS_NORMAL = 1 = 2 MDS_PRESSE... MDS_DISABLED = MDS_CHECKED = MDS_HOTCHECKE...
	MDP_SEPERATOR = 2	
PAGE	PGRP_DOWN = 2	DNS_NORMAL = 1 = 2 DNS_PRESSED DNS_DISABLED =
	PGRP_DOWNHORZ = 4	DNHZS_NORMAL = DNHZS_HOT = 2 DNHZS_PRESSED DNHZS_DISABLED
	PGRP_UP = 1	UPS_NORMAL = 1 = 2 UPS_PRESSED UPS_DISABLED =
	PGRP_UPHORZ = 3	UPHZS_NORMAL = UPHZS_HOT = 2 UPHZS_PRESSED UPHZS_DISABLED
PROGRESS	PP_BAR = 1	
	PP_BARVERT = 2	
	PP_CHUNK = 3	
	PP_CHUNKVERT = 4	
REBAR	RP_BAND = 3	
	RP_CHEVRON = 4	CHEVS_NORMAL = CHEVS_HOT = 2 CHEVS_PRESSED
	RP_CHEVRONVERT = 5	
	RP_GRIPPER = 1	
	RP_GRIPPERVERT = 2	
		ABS_DOWNDISAB... ABS_DOWNHOT, ABS_DOWNNORM... ABS_DOWNPRESS... ABS_UPDISABLED ABS_UPHOT, ABS_UPNORMAL, ABS_UPPRESSED,

SCROLLBAR	SBP_ARROWBTN = 1	ABS_LEFTDISABLI ABS_LEFTHOT, ABS_LEFTNORMA ABS_LEFTPRESSE ABS_RIGHTDISAB ABS_RIGHTHOT, ABS_RIGHTNORM ABS_RIGHTPRESS
	SBP_GRIPPERHORZ = 8 SBP_GRIPPERVERT = 9	
	SBP_LOWERTRACKHORZ = 4	SCRBS_NORMAL = SCRBS_HOT = 2 SCRBS_PRESSED SCRBS_DISABLED
	SBP_LOWERTRACKVERT = 6	SCRBS_NORMAL = SCRBS_HOT = 2 SCRBS_PRESSED SCRBS_DISABLED
	SBP_THUMBBTNHORZ = 2	SCRBS_NORMAL = SCRBS_HOT = 2 SCRBS_PRESSED SCRBS_DISABLED
	SBP_THUMBBTNVERT = 3	SCRBS_NORMAL = SCRBS_HOT = 2 SCRBS_PRESSED SCRBS_DISABLED
	SBP_UPPERTRACKHORZ = 5	SCRBS_NORMAL = SCRBS_HOT = 2 SCRBS_PRESSED SCRBS_DISABLED
	SBP_UPPERTRACKVERT = 7	SCRBS_NORMAL = SCRBS_HOT = 2 SCRBS_PRESSED SCRBS_DISABLED
	SBP_SIZEBOX = 10	SZB_RIGHTALIGN SZB_LEFTALIGN =
SPIN	SPNP_DOWN = 2	DNS_NORMAL = 1 = 2 DNS_PRESSE[DNS_DISABLED = DNHZS_NORMAL = DNHZS_HOT = 2

		DNHZS_PRESSED DNHZS_DISABLED
	SPNP_DOWNHORZ = 4	
		UPS_NORMAL = 1 = 2 UPS_PRESSED UPS_DISABLED =
	SPNP_UP = 1	
		UPHZS_NORMAL = UPHZS_HOT = 2 UPHZS_PRESSED UPHZS_DISABLED
	SPNP_UPHORZ = 3	
STARTPANEL	SPP_LOGOFF = 8	
		SPLS_NORMAL = · SPLS_HOT = 2 SPLS_PRESSED =
	SPP_LOGOFFBUTTONS = 9	
	SPP_MOREPROGRAMS = 2	
		SPS_NORMAL = 1 = 2 SPS_PRESSED
	SPP_MOREPROGRAMSARROW = 3	
	SPP_PLACESLIST = 6	
	SPP_PLACESLISTSEPARATOR = 7	
	SPP_PREVIEW = 11	
	SPP_PROGLIST = 4	
	SPP_PROGLISTSEPARATOR = 5	
	SPP_USERPANE = 1	
	SPP_USERPICTURE = 10	
STATUS	SP_GRIPPER = 3	
	SP_PANE = 1	
	SP_GRIPPERPANE = 2	
TAB	TABP_BODY = 10	
	TABP_PANE = 9	
		TIS_NORMAL = 1 T 2 TIS_SELECTED = TIS_DISABLED = 4 TIS_FOCUSED = 5
	TABP_TABITEM = 1	
		TIBES_NORMAL = TIBES_HOT = 2
		TIBES_SELECTED TIBES_DISABLED TIBES_FOCUSED : TILES_NORMAL =
	TABP_TABITEMBOTHEDGE = 4	

TABP_TABITEMLEFTEDGE = 2

TABP_TABITEMRIGHTEDGE = 3

TABP_TOPTABITEM = 5

TABP_TOPTABITEMBOTHEDGE = 8

TABP_TOPTABITEMLEFTEDGE = 6

TABP_TOPTABITEMRIGHTEDGE = 7

TASKBAND
TDP_GROUPCOUNT = 1
TDP_FLASHBUTTON = 2
TDP_FLASHBUTTONGROUPMENU = 3

TASKBAR
TBP_BACKGROUNDBOTTOM = 1
TBP_BACKGROUNDLEFT = 4
TBP_BACKGROUNDRIGHT = 2
TBP_BACKGROUNDTOP = 3
TBP_SIZINGBARBOTTOM = 5
TBP_SIZINGBARBOTTOMLEFT = 8
TBP_SIZINGBARRIGHT = 6
TBP_SIZINGBARTOP = 7

TILES_HOT = 2
TILES_SELECTED
TILES_DISABLED
TILES_FOCUSED

TIRES_NORMAL =
TIRES_HOT = 2
TIRES_SELECTED
TIRES_DISABLED
TIRES_FOCUSED

TTIS_NORMAL = 1
= 2 TTIS_SELECTE
TTIS_DISABLED =
TTIS_FOCUSED =

TTIBES_NORMAL
TTIBES_HOT = 2
TTIBES_SELECTE
TTIBES_DISABLED
TTIBES_FOCUSED

TTILES_NORMAL
TTILES_HOT = 2
TTILES_SELECTE
TTILES_DISABLED
TTILES_FOCUSED

TTIRES_NORMAL
TTIRES_HOT = 2
TTIRES_SELECTE
TTIRES_DISABLED
TTIRES_FOCUSED

TS_NORMAL = 1 T

TOOLBAR	TP_BUTTON = 1	TS_PRESSED = 3 TS_DISABLED = 4 TS_CHECKED = 5 TS_HOTCHECKED
	TP_DROPDOWNBUTTON = 2	TS_NORMAL = 1 T TS_PRESSED = 3 TS_DISABLED = 4 TS_CHECKED = 5 TS_HOTCHECKED
	TP_SPLITBUTTON = 3	TS_NORMAL = 1 T TS_PRESSED = 3 TS_DISABLED = 4 TS_CHECKED = 5 TS_HOTCHECKED
	TP_SPLITBUTTONDROPDOWN = 4	TS_NORMAL = 1 T TS_PRESSED = 3 TS_DISABLED = 4 TS_CHECKED = 5 TS_HOTCHECKED
	TP_SEPARATOR = 5	TS_NORMAL = 1 T TS_PRESSED = 3 TS_DISABLED = 4 TS_CHECKED = 5 TS_HOTCHECKED
	TP_SEPARATORVERT = 6	TS_NORMAL = 1 T TS_PRESSED = 3 TS_DISABLED = 4 TS_CHECKED = 5 TS_HOTCHECKED
TOOLTIP	TTP_BALLOON = 3	TTBS_NORMAL = TTBS_LINK = 2
	TTP_BALLOONTITLE = 4	TTBS_NORMAL = TTBS_LINK = 2
	TTP_CLOSE = 5	TTCS_NORMAL = TTCS_HOT = 2 TTCS_PRESSED =
	TTP_STANDARD = 1	TTSS_NORMAL = TTSS_LINK = 2
	TTP_STANDARDTITLE = 2	TTSS_NORMAL = TTSS_LINK = 2
		TUS_NORMAL = 1

TRACKBAR	TKP_THUMB = 3	2 TUS_PRESSED =
		TUS_FOCUSED = 4
		TUS_DISABLED =
		TUBS_NORMAL =
		TUBS_HOT = 2
	TKP_THUMBBOTTOM = 4	TUBS_PRESSED =
		TUBS_FOCUSED =
		TUBS_DISABLED =
		TUVLS_NORMAL =
		TUVLS_HOT = 2
	TKP_THUMBLEFT = 7	TUVLS_PRESSED
		TUVLS_FOCUSED
		TUVLS_DISABLED
		TUVRS_NORMAL =
		TUVRS_HOT = 2
	TKP_THUMBRIGHT = 8	TUVRS_PRESSED
		TUVRS_FOCUSED
		TUVRS_DISABLED
		TUTS_NORMAL =
		TUTS_HOT = 2
	TKP_THUMBTOP = 5	TUTS_PRESSED =
		TUTS_FOCUSED =
		TUTS_DISABLED =
		TUVS_NORMAL =
		TUVS_HOT = 2
	TKP_THUMBVERT = 6	TUVS_PRESSED =
		TUVS_FOCUSED =
		TUVS_DISABLED =
	TKP_TICS = 9	TSS_NORMAL = 1
	TKP_TICSVERT = 10	TSVS_NORMAL =
	TKP_TRACK = 1	TRS_NORMAL = 1
	TKP_TRACKVERT = 2	TRVS_NORMAL =
TRAYNOTIFY	TNP_ANIMBACKGROUND = 2	
	TNP_BACKGROUND = 1	
TREEVIEW	TVP_BRANCH = 3	
		GLPS_CLOSED =
	TVP_GLYPH = 2	GLPS_OPENED =
		TREIS_NORMAL =
		TREIS_HOT = 2
		TREIS_SELECTED

	TVP_TREEITEM = 1	TREIS_DISABLED TREIS_SELECTED = 5
WINDOW	WP_CAPTION = 1	CS_ACTIVE = 1 CS = 2 CS_DISABLED
	WP_CAPTIONSIZINGTEMPLATE = 30	
	WP_CLOSEBUTTON = 18	CBS_NORMAL = 1 = 2 CBS_PUSHED CBS_DISABLED =
	WP_DIALOG = 29	
	WP_FRAMEBOTTOM = 9	FS_ACTIVE = 1 FS = 2
	WP_FRAMEBOTTOMSIZINGTEMPLATE = 36	
	WP_FRAMELEFT = 7	FS_ACTIVE = 1 FS = 2
	WP_FRAMELEFTSIZINGTEMPLATE = 32	
	WP_FRAMERIGHT = 8	FS_ACTIVE = 1 FS = 2
	WP_FRAMERIGHTSIZINGTEMPLATE = 34	
	WP_HELPBUTTON = 23	HBS_NORMAL = 1 = 2 HBS_PUSHED HBS_DISABLED =
	WP_HORZSCROLL = 25	HSS_NORMAL = 1 = 2 HSS_PUSHED HSS_DISABLED =
	WP_HORZTHUMB = 26	HTS_NORMAL = 1 2 HTS_PUSHED = HTS_DISABLED =
	WP_MAX_BUTTON	MAXBS_NORMAL MAXBS_HOT = 2 MAXBS_PUSHED = MAXBS_DISABLED
	WP_MAXCAPTION = 5	MXCS_ACTIVE = 1 MXCS_INACTIVE = MXCS_DISABLED
	WP_MDICLOSEBUTTON = 20	CBS_NORMAL = 1 = 2 CBS_PUSHED CBS_DISABLED =
		HBS_NORMAL = 1

WP_MDIHELPBUTTON = 24	= 2 HBS_PUSHED HBS_DISABLED =
WP_MDIMINBUTTON = 16	MINBS_NORMAL = MINBS_HOT = 2 MINBS_PUSHED = MINBS_DISABLED
WP_MDIRESTOREBUTTON = 22	RBS_NORMAL = 1 = 2 RBS_PUSHED RBS_DISABLED =
WP_MDISYSBUTTON = 14	SBS_NORMAL = 1 = 2 SBS_PUSHED SBS_DISABLED =
WP_MINBUTTON = 15	MINBS_NORMAL = MINBS_HOT = 2 MINBS_PUSHED = MINBS_DISABLED
WP_MINCAPTION = 3	MNCS_ACTIVE = 1 MNCS_INACTIVE = MNCS_DISABLED
WP_RESTOREBUTTON = 21	RBS_NORMAL = 1 = 2 RBS_PUSHED RBS_DISABLED =
WP_SMALLCAPTION = 2	CS_ACTIVE = 1 CS = 2 CS_DISABLED
WP_SMALLCAPTIONSIZINGTEMPLATE = 31	
WP_SMALLCLOSEBUTTON = 19	CBS_NORMAL = 1 = 2 CBS_PUSHED CBS_DISABLED =
WP_SMALLFRAMEBOTTOM = 12	FS_ACTIVE = 1 FS = 2
WP_SMALLFRAMEBOTTOMSIZINGTEMPLATE = 37	
WP_SMALLFRAMELEFT = 10	FS_ACTIVE = 1 FS = 2
WP_SMALLFRAMELEFTSIZINGTEMPLATE = 33	
WP_SMALLFRAMERIGHT = 11	FS_ACTIVE = 1 FS = 2
WP_SMALLFRAMERIGHTSIZINGTEMPLATE = 35	

WP_SMALLHELPBUTTON	HBS_NORMAL = 1 = 2 HBS_PUSHED HBS_DISABLED =
WP_SMALLMAXBUTTON	MAXBS_NORMAL MAXBS_HOT = 2 MAXBS_PUSHED = MAXBS_DISABLED
WP_SMALLMAXCAPTION = 6	MXCS_ACTIVE = 1 MXCS_INACTIVE = MXCS_DISABLED
WP_SMALLMINCAPTION = 4	MNCS_ACTIVE = 1 MNCS_INACTIVE = MNCS_DISABLED
WP_SMALLRESTOREBUTTON	RBS_NORMAL = 1 = 2 RBS_PUSHED RBS_DISABLED =
WP_SMALLSYSBUTTON	SBS_NORMAL = 1 = 2 SBS_PUSHED SBS_DISABLED =
WP_SYSBUTTON = 13	SBS_NORMAL = 1 = 2 SBS_PUSHED SBS_DISABLED =
WP_VERTSCROLL = 27	VSS_NORMAL = 1 = 2 VSS_PUSHED VSS_DISABLED =
WP_VERTTHUMB = 28	VTS_NORMAL = 1 2 VTS_PUSHED = VTS_DISABLED =

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. Use the Background property to apply a skin to a part of the control. Use the Appearance property to change the visual appearance of the control's frame.

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. Use the [Background](#) property to apply a skin to a part of the control. Use the [Appearance](#) property to change the visual appearance of the control's frame.

property Appearance.RenderType as Long

Specifies the way colored EBN objects are displayed on the component.

Type	Description
Long	A long expression that indicates how the EBN objects are shown in the control, like explained bellow.

By default, the RenderType property is 0, which indicates an A-color scheme. The RenderType property can be used to change the colors for the entire control, for parts of the controls that uses EBN objects. The RenderType property is not applied to the currently XP-theme if using.

The RenderType property is applied to all parts that displays an EBN object. The properties of color type may support the EBN object if the property's description includes "*A color expression that indicates the cell's background color. 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.*" In other words, a property that supports EBN objects should be of format 0xIDRRGGBB, where the ID is the identifier of the EBN to be applied, while the BBGGRR is the (Red,Green,Blue, RGB-Color) color to be applied on the selected EBN. For instance, the 0x1000000 indicates displaying the EBN as it is, with no color applied, while the 0x1FF0000, applies the Blue color (RGB(0x0,0x0,0xFF), RGB(0,0,255) on the EBN with the identifier 1. You can use the [EBNColor](#) tool to visualize applying EBN colors.

Click here ▣ to watch a movie on how you can change the colors to be applied on EBN objects.

For instance, the following sample changes the control's appearance, by using an EBN object:

```
With Control
    .VisualAppearance.Add 1,"c:\exontrol\images\normal.ebn"
    .Appearance = &H1000000
End With
```

In the following screen shot the following objects displays the current EBN with a different color:

- "A" in Red (RGB(255,0,0)), for instance the bar's property exBarColor is 0x10000FF
- "B" in Green (RGB(0,255,0)), for instance the bar's property exBarColor is 0x100FF00

- "C" in Blue (RGB(0,0,255), for instance the bar's property exBarColor is 0x1FF0000
- "Default", no color is specified, for instance the bar's property exBarColor is 0x1000000

The RenderType property could be one of the following:

- **-3, *no color is applied*.** For instance, the BackColorHeader = &H1FF0000 is displayed as would be .BackColorHeader = &H1000000, so the 0xFF0000 color (Blue color) is ignored. You can use this option to allow the control displays the EBN colors or not.
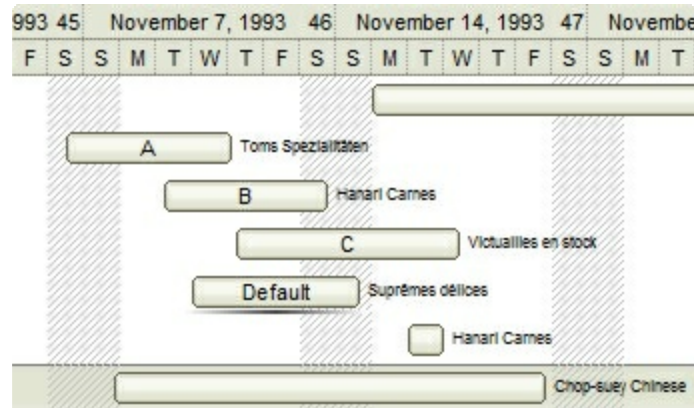

- **-2, *OR-color scheme*.** The color to be applied on the part of the control is a OR bit combination between the original EBN color and the specified color. For instance, the BackColorHeader = &H1FF0000, applies the OR bit for the entire Blue channel, or in other words, it applies a less Blue to the part of the control. This option should be used with solid colors (RGB(255,0,0), RGB(0,255,0), RGB(0,0,255), RGB(255,255,0), RGB(255,0,255), RGB(0,255,255),RGB(127,0,0),RGB(0,127,0), ...)
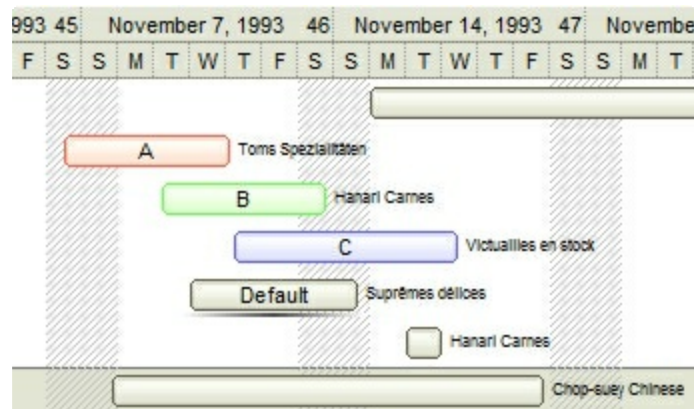
- **-1, *AND-color scheme*,** The color to be applied on the part of the control is an AND bit combination between the original EBN color and the specified color. For instance, the BackColorHeader = &H1FF0000, applies the AND bit for the entire Blue channel, or in other words, it applies a more Blue to the part of the control. This option should be used with solid colors (RGB(255,0,0), RGB(0,255,0), RGB(0,0,255), RGB(255,255,0), RGB(255,0,255), RGB(0,255,255),RGB(127,0,0),RGB(0,127,0), ...)
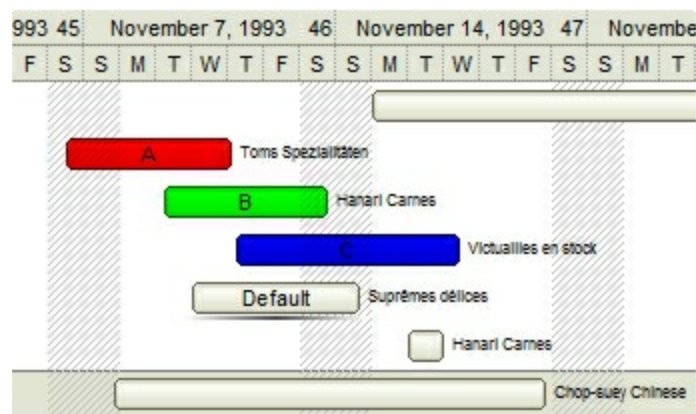
- **0, default**, the specified color is applied to the EBN. For instance, the BackColorHeader = &H1FF0000, applies a Blue color to the object. This option could be used to specify any color for the part of the components, that support EBN objects, not only solid colors.
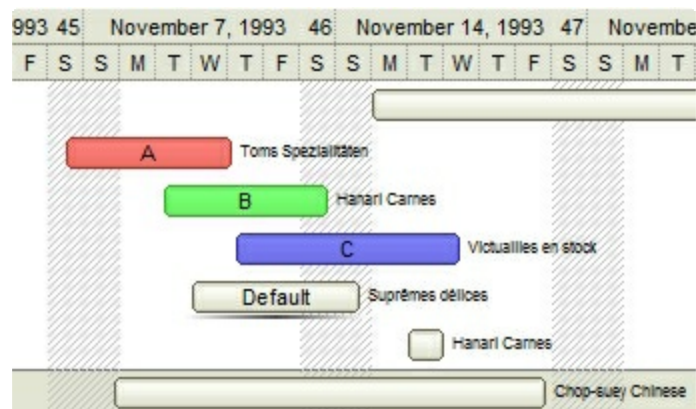

- **0xAABBGGRR**, where the AA a value between 0 to 255, which indicates the transparency, and RR, GG, BB the red, green and blue values. This option applies the same color to all parts that displays EBN objects, whit ignoring any specified color in the color property. For instance, the RenderType on 0x4000FFFF, indicates a 25% Yellow on EBN objects. The 0x40, or 64 in decimal, is a 25 % from in a 256 interal, and the 0x00FFFF, indicates the Yellow (RGB(255,255,0)). The same could be if the RenderType is 0x40000000 + vbYellow, or &H40000000 + RGB(255, 255, 0), and so, the RenderType could be the 0xAA000000 + Color, where the Color is the RGB format of the color.
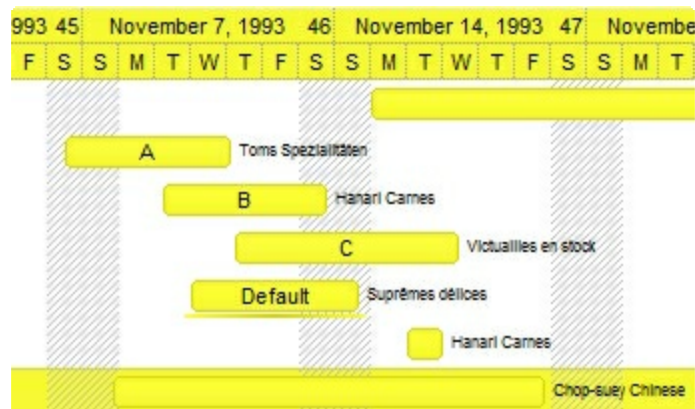
*The following picture shows the control with the RenderType property on 0x**4000FFFF** (25% Yellow, 0x40 or 64 in decimal is 25% from 256):*
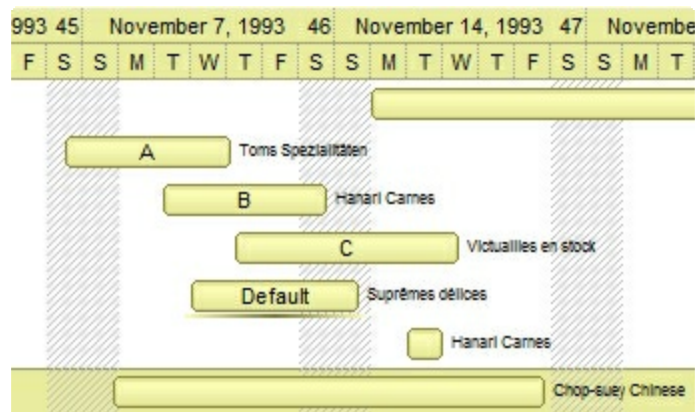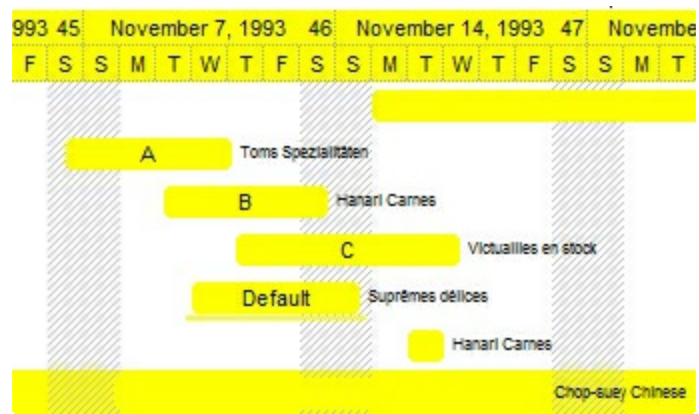
*The following picture shows the control with the RenderType property on 0x**80**00FFFF (50% Yellow, 0x80 or 128 in decimal is 50% from 256):*
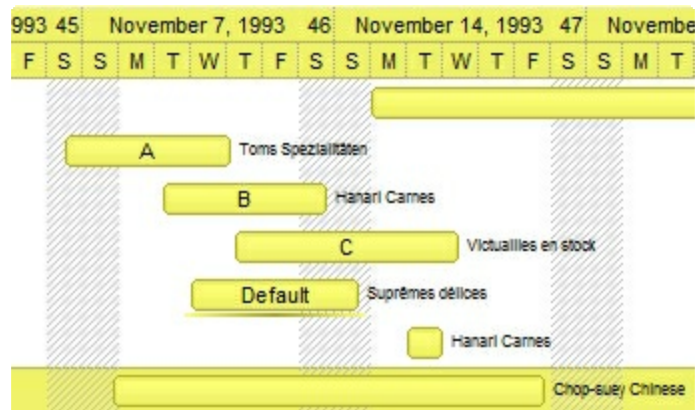
*The following picture shows the control with the RenderType property on 0x**C0**00FFFF (75% Yellow, 0xC0 or 192 in decimal is 75% from 256):*

*The following picture shows the control with the RenderType property on 0x**FF**00FFFF (100% Yellow, 0xFF or 255 in decimal is 100% from 255):*

F	S	S	M	T	W	T	F	S	S	M	T	W	T	F	S	S	M	T

A Toms Spezialitäten

B Hanari Carnes

C Victualilles en stock

Default Suprêmes délices

Hanari Carnes

Chop-suey Chinese

MaskEdit 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: {43F80262-F652-11D3-AD39-00C0DFC59237}. The object's program identifier is: "Exontrol.MaskEdit". The /COM object module is: "MaskEdit.dll"

The ExMaskEdit ActiveX control handles the format of text entered by the user. What does ExMaskEdit provide, and other mask edit control doesn't? The ExMaskEdit control provides a nice and useful feature, that we called multi-mask (haven't seen it before) , that means that the control is able to handle more than a mask at runtime. Sometime, your mask doesn't fit all the requirements, or the masking rules are limited, so you are unable to use a standard mask edit control. That's the reason why we decided to get released the ExMaskEdit ActiveX control. The ExMaskEdit control can subclass a standard edit control or a rich control edit. When the control subclasses a rich edit control, the mask is painted using an user color. Here's a list of types of data that you might want to mask: license keys, IP addresses, urls, e-mails, phone numbers, decimal numbers, hexa numbers, binary numbers, alpha and digit characters, date, time, and so on.

Name	Description
ActiveMask	Specifies the index of active mask.
AllowBeep	Specifies whether the control plays a beep once the user enters any invalid character.
AllowContextMenu	Specifies whether the control displays the content menu when user right clicks the control.
AllowEmptyValue	Specifies whether the field supports empty values.
AllowToggleInsertMode	Specifies whether the control is toggling the InsertMode when the user presses the Insert key.
Appearance	Retrieves or sets the control's appearance.
AttachTemplate	Attaches a script to the current object, including the events, from a string, file, a safe array of bytes.
BackColor	Specifies the control's background color.
BackColorReadOnly	Indicates the control's background when the control is read only.
Background	Returns or sets a value that indicates the background color for parts in the control.

[CursorPos](#)	Retrieves or sets a value that indicates the cursor position.
[Enabled](#)	Enables or disables the control.
[EventParam](#)	Retrieves or sets a value that indicates the current's event parameter.
[ExecuteTemplate](#)	Executes a template and returns the result.
[Font](#)	Retrieves or sets the control's font.
[ForeColor](#)	Specifies the control's foreground color.
[ForeColorReadOnly](#)	Indicates the control's foreground when the control is read only.
[ForeColorRich](#)	Retrieves or sets the color to show the editable entity, while the Type is RichEdit and the value is not completed.
[HTMLPicture](#)	Adds or replaces a picture in HTML captions.
[hWnd](#)	Retrieves the control's window handle.
[Images](#)	Sets at runtime the control's image list. The Handle should be a handle to an Images List Control.
[ImageSize](#)	Retrieves or sets the size of icons the control displays..
[InsertMode](#)	Specifies the control's inserting mode that could be insert-type or over-type.
[Invalid](#)	Indicates the html-tooltip message to be shown when the user enters an invalid value.
[Mask](#)	Retrieves or sets a string expression that indicates the control's mask.
[MaskChar](#)	Retrieves or sets a value that determines masking character.
[MaskFloat](#)	Specifies whether the Mask property masks a floating point number.
[Masks](#)	Retrieves or sets a value that determines all possible masks that user can use at runtime.
[MultiLine](#)	Retrieves or sets a value that determines whether the control can accept multiple lines of text.
[Password](#)	Displays all characters as an asterisk (*)
[PasswordChar](#)	Retrieves or sets a value that determines password character.
[ReadOnly](#)	Retrieves or sets a value that determines whether the control is read only.

[Refresh](#)	Refreshes the control's field.
[ReplaceIcon](#)	Adds a new icon, replaces an icon or clears the control's image list.
[Right](#)	Right aligns text in a single-line or multiline edit control.
[Select](#)	Selects the text between Start and End
[SelectGotFocus](#)	Indicates whether the entire text is selected once the field receives the focus.
[SelEnd](#)	Returns or sets the ending point of text selected.
[SelStart](#)	Returns or sets the starting point of text selected.
[ShowImageList](#)	Specifies whether the control's image list window is visible or hidden.
[ShowToolTip](#)	Shows the specified tooltip at given position.
[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](#)	Retrieves or sets the text displayed in the control.
[TextIncludeLiterals](#)	Determines the way the Text property returns or set the value of the field
[TextIncludeLiteralsLoseFocus](#)	Determines how the field shows its content once it loses the focus.
[Type](#)	Retrieves or sets the base class for the mask field.
[Valid](#)	Retrieves a value indicating whether the contrl contains a valid value.
[ValidateAs](#)	Indicates the additional validation is performed, once the user leaves the field.
[Value](#)	Retrieves the control's value with or without literals.
[Version](#)	Retrieves the control's version.
[VisibleMasks](#)	Retrieves or sets a value that indicates the number of visible items in the control masks list.
[VisualAppearance](#)	Retrieves the control's appearance.
[Warning](#)	Indicates the html-tooltip message to be shown when the user enters an invalid character.

property MaskEdit.ActiveMask as Long

Specifies the index of active mask.

Type	Description
Long	A long expression that indicates the index of active mask.

The number of masks used is determined by the [Masks](#) property. If the Masks property is not empty, the control loads multiple masks. Else, if the [Mask](#) property is used, the control uses only a single mask.

The following sample changes the active mask after loading the masks list to the control:

```
Private Sub Form_Load()
    With MaskEdit1
        .Masks = "USA resident;(090)-###-###-###;Canada resident;(091)-###-###-###"
        .ActiveMask = 1
    End With
End Sub
```

property MaskEdit.AllowBeep as Boolean

Specifies whether the control plays a beep once the user enters any invalid character.

Type	Description
Boolean	A Boolean expression that specifies whether the control beeps once the user enters any invalid character.

By default, the AllowBeep property is False. Use the AllowBeep property to let control beeps once the user enters any invalid character. Use the [Warning](#) property to specify a HTML tooltip to be shown at the cursor position when user enters an invalid character. Use the [Invalid](#) property to specify a HTML tooltip to keep the control focused while the user enters an inappropriate value for the field.

property MaskEdit.AllowContextMenu as Boolean

Specifies whether the control displays the content menu when user right clicks the control.

Type	Description
Boolean	A Boolean expression

By default, the AllowContextMenu property is True. Use the AllowContextMenu property on False, to prevent showing the control's context menu when the user right clicks the control. The AllowContextMenu property indicates that the field provides no context menu when user right clicks the field. For instance, Mask on ";;;password,nocontext" displays a password field, where the user can not invoke the default context menu, usually when a right click occurs.

property MaskEdit.AllowEmptyValue as AllowEmptyValueEnum

Specifies whether the field supports empty values.

Type	Description
[AllowEmptyValueEnum](#)	A AllowEmptyValueEnum expression that specifies whether the field supports empty values.

By default, AllowEmptyValue property is exNoEmptyValue. The AllowEmptyValue property indicates whether the field supports empty values. This option can be used with invalid flag, which indicates that the user can leave the field if it is empty. If empty flag is present, the field displays nothing if no entity is completed (empty). Once the user starts typing characters the current mask is displayed. For instance, having the mask "!(999) 000 0000;;;empty,select=4,overtype,invalid=invalid phone number,beep", it specifies an empty or valid phone to be entered.

property MaskEdit.AllowToggleInsertMode as Boolean

Specifies whether the control is toggling the InsertMode when the user presses the Insert key.

Type	Description
Boolean	A Boolean expression that specifies whether the control toggles the insertion mode when the user presses the Insert key.

By default, the AllowToggleInsertMode property is True. The AllowToggleInsertMode property specifies whether the control toggles the insertion mode when the user presses the Insert key. The [InsertMode](#) property specifies the control's insertion mode. Use the InsertMode property on exEditOvertypeMode to allow user to edit the control's field in overtype mode.

property MaskEdit.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 list/hierarchy/chart, scrollbars are always shown in the control's client area. The skin may contain transparent objects, and so you can define round corners. The normal.ebn file contains such of objects. Use the eXButton's Skin builder to view or change this file***

By default, the Appearance property is Flat. 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 control's tooltip.

The following screen shot shows the control's with visual appearance changed:
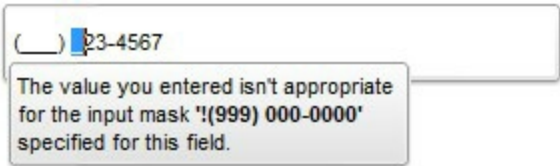
The following screen shot shows the control's with no visual appearance changed:
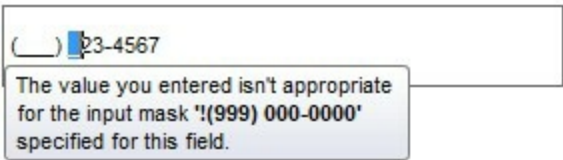
method MaskEdit.AttachTemplate (Template as Variant)

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 MaskEdit1_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 MaskEdit.BackColor as Color

Retrieves or sets the control's background color.

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

Use the BackColor property to change the control's background color. Use the ForeColor property to change the control's foreground color. The BackColorReadOnly property specifies the control's background color when the field is locked (ReadOnly property is True). The ForeColorReadOnly property specifies the control's foreground color when the field is locked (ReadOnly property is True).

property MaskEdit.BackColorReadOnly as Color

Indicates the control's background when the control is read only.

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

The BackColorReadOnly property specifies the control's background color when the field is locked (ReadOnly property is True). The ForeColorReadOnly property specifies the control's foreground color when the field is locked (ReadOnly property is True). Use the ForeColor property to change the control's foreground color. Use the BackColor property to change the control's background color. Use the ForeColorRich property to change the color used to paint the mask, while the control's Type is RichEdit, and the Value is not yet validated. Use the Valid property to check whenever the control's mask value is valid or invalid.

property MaskEdit.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. Use the Invalid/Warning/ShowToolTip to display a tooltip.

The following screen shot shows the control's tooltip with a new visual apperance:

The following sample shows how you can change the tooltip's visual appearance:

VBA (MS Access, Excell...)

```
With MaskEdit1
    .VisualAppearance.Add 1,"c:\exontrol\images\normal.ebn"
    .Background(64) = &H1000000
    .Warning = "Invalid character"
    .Mask = "`Time: `00:00:00"
End With
```

VB6

```
With MaskEdit1
    .VisualAppearance.Add 1,"c:\exontrol\images\normal.ebn"
    .Background(exToolTipAppearance) = &H1000000
    .Warning = "Invalid character"
    .Mask = "`Time: `00:00:00"
End With
```

VB.NET

```
With Exmaskedit1
    .VisualAppearance.Add(1,"c:\exontrol\images\normal.ebn")

.set_Background32(exontrol.EXMASKEDITLib.BackgroundPartEnum.exToolTipAppearan

    .Warning = "Invalid character"
    .Mask = "`Time: `00:00:00"
End With
```

VB.NET for /COM

```
With AxMaskEdit1
    .VisualAppearance.Add(1,"c:\exontrol\images\normal.ebn")

.set_Background(EXMASKEDITLib.BackgroundPartEnum.exToolTipAppearance,167772

    .Warning = "Invalid character"
    .Mask = "`Time: `00:00:00"
End With
```

C++

```
/*
    Copy and paste the following directives to your header file as
    it defines the namespace 'EXMASKEDITLib' for the library: 'ExMaskEdit 7.1 Control
Library'

    #import <MaskEdit.dll>
    using namespace EXMASKEDITLib;
```

```
*/
EXMASKEDITLib::IMaskEditPtr spMaskEdit1 = GetDlgItem(IDC_MASKEDIT1)-
>GetControlUnknown();
spMaskEdit1->GetVisualAppearance()-
>Add(1,"c:\\exontrol\\images\\normal.ebn");
spMaskEdit1->PutBackground(EXMASKEDITLib::exToolTipAppearance,0x1000000);
spMaskEdit1->PutWarning(L"Invalid character");
spMaskEdit1->PutMask(L"`Time: `00:00:00");
```

C++ Builder

```
MaskEdit1->VisualAppearance-
>Add(1,TVariant("c:\\exontrol\\images\\normal.ebn"));
MaskEdit1-
>Background[Exmaskeditlib_tlb::BackgroundPartEnum::exToolTipAppearance] =
0x1000000;
MaskEdit1->Warning = L"Invalid character";
MaskEdit1->Mask = L"`Time: `00:00:00";
```

C#

```
exmaskedit1.VisualAppearance.Add(1,"c:\\exontrol\\images\\normal.ebn");
exmaskedit1.set_Background32(exontrol.EXMASKEDITLib.BackgroundPartEnum.exTool

exmaskedit1.Warning = "Invalid character";
exmaskedit1.Mask = "`Time: `00:00:00";
```

JavaScript

```
<OBJECT classid="clsid:43F80262-F652-11D3-AD39-00C0DFC59237"
id="MaskEdit1"></OBJECT>

<SCRIPT LANGUAGE="JScript">
    MaskEdit1.VisualAppearance.Add(1,"c:\\exontrol\\images\\normal.ebn");
    MaskEdit1.Background(64) = 16777216;
```

```
    MaskEdit1.Warning = "Invalid character";
    MaskEdit1.Mask = "`Time: `00:00:00";
</SCRIPT>
```

C# for /COM

```
axMaskEdit1.VisualAppearance.Add(1,"c:\\exontrol\\images\\normal.ebn");
axMaskEdit1.set_Background(EXMASKEDITLib.BackgroundPartEnum.exToolTipAppear

axMaskEdit1.Warning = "Invalid character";
axMaskEdit1.Mask = "`Time: `00:00:00";
```

X++ (Dynamics Ax 2009)

```
public void init()
{
    ;

    super();

    exmaskedit1.VisualAppearance().Add(1,"c:\\exontrol\\images\\normal.ebn");
    exmaskedit1.Background(64/*exToolTipAppearance*/,0x1000000);
    exmaskedit1.Warning("Invalid character");
    exmaskedit1.Mask("`Time: `00:00:00");
}
```

Delphi 8 (.NET only)

```
with AxMaskEdit1 do
begin
    VisualAppearance.Add(1,'c:\exontrol\images\normal.ebn');

    set_Background(EXMASKEDITLib.BackgroundPartEnum.exToolTipAppearance,$100000

    Warning := 'Invalid character';
    Mask := '`Time: `00:00:00';
end
```

Delphi (standard)

```
with MaskEdit1 do
begin
   VisualAppearance.Add(1,'c:\exontrol\images\normal.ebn');
   Background[EXMASKEDITLib_TLB.exToolTipAppearance] := $1000000;
   Warning := 'Invalid character';
   Mask := '`Time: `00:00:00';
end
```

VFP

```
with thisform.MaskEdit1
   .VisualAppearance.Add(1,"c:\exontrol\images\normal.ebn")
   .Object.Background(64) = 0x1000000
   .Warning = "Invalid character"
   .Mask = "`Time: `00:00:00"
endwith
```

dBASE Plus

```
local oMaskEdit

oMaskEdit = form.Activex1.nativeObject
oMaskEdit.VisualAppearance.Add(1,"c:\exontrol\images\normal.ebn")
oMaskEdit.Template = [Background(64) = 0x1000000] //
oMaskEdit.Background(64) = 0x1000000
oMaskEdit.Warning = "Invalid character"
oMaskEdit.Mask = "`Time: `00:00:00"
```

XBasic (Alpha Five)

```
Dim oMaskEdit as P

oMaskEdit = topparent:CONTROL_ACTIVEX1.activex
oMaskEdit.VisualAppearance.Add(1,"c:\exontrol\images\normal.ebn")
oMaskEdit.Template = "Background(64) = 16777216" ' oMaskEdit.Background(64)
= 16777216
```

```
oMaskEdit.Warning = "Invalid character"
oMaskEdit.Mask = "`Time: `00:00:00"
```

Visual Objects

```
oDCOCX_Exontrol1:VisualAppearance:Add(1,"c:\exontrol\images\normal.ebn")
oDCOCX_Exontrol1:[Background,exToolTipAppearance] := 0x1000000
oDCOCX_Exontrol1:Warning := "Invalid character"
oDCOCX_Exontrol1:Mask := "`Time: `00:00:00"
```

PowerBuilder

```
OleObject oMaskEdit

oMaskEdit = ole_1.Object
oMaskEdit.VisualAppearance.Add(1,"c:\exontrol\images\normal.ebn")
oMaskEdit.Background(64,16777216 /*0x1000000*/)
oMaskEdit.Warning = "Invalid character"
oMaskEdit.Mask = "`Time: `00:00:00"
```

Visual DataFlex

```
Procedure OnCreate
    Forward Send OnCreate
    Variant voAppearance
    Get ComVisualAppearance to voAppearance
    Handle hoAppearance
    Get Create (RefClass(cComAppearance)) to hoAppearance
    Set pvComObject of hoAppearance to voAppearance
        Get ComAdd of hoAppearance 1 "c:\exontrol\images\normal.ebn" to Nothing
    Send Destroy to hoAppearance
    Set ComBackground OLEexToolTipAppearance to |CI$1000000
    Set ComWarning to "Invalid character"
    Set ComMask to "`Time: `00:00:00"
```

End_Procedure

property MaskEdit.CursorPos as Long

Retrieves or sets a value that indicates the cursor position.

Type	Description
Long	A long expression that indicates the cursor position.

The CursorPos property determines the cursor position. The CursorPos property starts from 0.

The following sample displays the cursor position using a Timer:

```
Private Sub Timer1_Timer()
    Label1 = MaskEdit1.CursorPos
End Sub
```

property MaskEdit.Enabled as Boolean

Retrieves or sets a value indicating whether the control is enabled or disabled.

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

Use the Enable property to disable the control. Use the ReadOnly property to make your control read-only.

property MaskEdit.EventParam(Parameter as Long) as Variant

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

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

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

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

In most languages you will type something like:

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

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

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

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

reference.

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

method MaskEdit.ExecuteTemplate (Template as String)

Executes a template and returns the result.

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

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

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

```
Debug.Print MaskEdit1.ExecuteTemplate("Items.ItemBar(FirstVisibleItem(),`,1)")
```

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

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

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

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

"\n\r" (newline) characters.

An instruction can be one of the following:

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

The Template supports the following general functions:

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

property MaskEdit.Font as IFontDisp

Retrieves or sets the control's font.

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

The Font property determines the control's font.

property MaskEdit.ForeColor as Color

Retrieves or sets the control's foreground color.

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

Use the ForeColor property to change the control's foreground color. Use the BackColor property to change the control's background color. Use the ForeColorRich property to change the color used to paint the mask, while the control's Type is RichEdit, and the Value is not yet validated. Use the Valid property to check whenever the control's mask value is valid or invalid. The BackColorReadOnly property specifies the control's background color when the field is locked (ReadOnly property is True). The ForeColorReadOnly property specifies the control's foreground color when the field is locked (ReadOnly property is True).

property MaskEdit.ForeColorReadOnly as Color

Indicates the control's foreground when the control is read only.

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

The ForeColorReadOnly property specifies the control's foreground color when the field is locked (ReadOnly property is True). The BackColorReadOnly property specifies the control's background color when the field is locked (ReadOnly property is True). Use the ForeColor property to change the control's foreground color. Use the BackColor property to change the control's background color. Use the ForeColorRich property to change the color used to paint the mask, while the control's Type is RichEdit, and the Value is not yet validated. Use the Valid property to check whenever the control's mask value is valid or invalid.

property MaskEdit.ForeColorRich as Color

Retrieves or sets a color that is used to paint the mask value, while the Type is RichEdit and the value is not valid.

Type	Description
Color	A color expression that indicates the color used by control to paint the mask value, while it is not valid.

The ForeColorRich property has effect only if the control's Type is RichEdit Use the ForeColor property to change the control's foreground color. You can use the ForeColorRich property to specify the foreground color for editable entities while the field is not validated/completed. The BackColorReadOnly property specifies the control's background color when the field is locked (ReadOnly property is True). The ForeColorReadOnly property specifies the control's foreground color when the field is locked (ReadOnly property is True).

property MaskEdit.HTMLPicture(Key as String) as Variant

Adds or replaces a picture in HTML captions.

Type	Description
Key as String	A String expression that indicates the key of the picture being added or replaced. If the Key property is Empty string, the entire collection of pictures is cleared.
Variant	The HTMLPicture specifies the picture being associated to a key. It can be one of the followings: • a string expression that indicates the path to the picture file, being loaded. • a string expression that indicates the base64 encoded string that holds a picture object, Use the eximages tool to save your picture as base64 encoded format. • A Picture object that indicates the picture being added or replaced. (A Picture object implements IPicture interface), If empty, the picture being associated to a key is removed. If the key already exists the new picture is replaced. If the key is not empty, and it doesn't not exist a new picture is added

The HTMLPicture property handles a collection of custom size picture being displayed in the HTML captions, using the <img> tags. By default, the HTMLPicture collection is empty. Use the HTMLPicture property to add new pictures to be used in HTML captions. For instance, the HTMLPicture("pic1") = "c:\winnt\zapotec.bmp", loads the zapotec picture and associates the pic1 key to it. Any "<img>pic1</img>" sequence in HTML captions, displays the pic1 picture. On return, the HTMLPicture property retrieves a Picture object (this implements the IPictureDisp interface). Use the Images property to assign icons to the control. Use the Warning property to specify a HTML tooltip to be shown at the cursor position when user enters an invalid character. Use the Invalid property to specify a HTML tooltip to keep the control focused while the user enters an inappropriate value for the field.

The following sample shows how to display icons in the control's invalid tooltip:

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

<COLUMN1>.Invalid = "A <img>pic1</img>"
```

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

property MaskEdit.hWnd as Long

Retrieves the control's window handle.

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

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

method MaskEdit.Images (Handle as Variant)

Sets the control's image list at runtime.

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 user can add images at design time, by drag and drop files to combo's image holder. The ImageSize property defines the size (width/height) of the icons within the control's Images collection. Use the ReplaceIcon method to add, remove or clear icons in the control's images collection. Use the <img> HTML tag to specify the index of the icon to be displayed. Use the Warning property to specify a HTML tooltip to be shown at the cursor position when user enters an invalid character. Use the Invalid property to specify a HTML tooltip to keep the control focused while the user enters an inappropriate value for the field.

property MaskEdit.ImageSize as Long

Retrieves or sets the size of icons the control displays..

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 MaskEdit.InsertMode as InsertModeEnum

Specifies the control's inserting mode that could be insert-type or over-type.

Type	Description
InsertModeEnum	An InsertModeEnum expression that specifies the insertion mode.

By default, the InsertMode property is exEditInsertMode. The InsertMode property specifies the control's insertion mode. Use the InsertMode property on exEditOvertypeMode to allow user to edit the control's field in overtype mode. The AllowToggleInsertMode property specifies whether the control toggles the insertion mode when the user presses the Insert key.

property MaskEdit.Invalid as String

Indicates the html-tooltip message to be shown when the user enters an invalid value.

Type	Description
String	A String expression that defines the built-in HTML tooltip to be shown at cursor position when user enters any invalid character.

By default, the Invalid property is "". Use the Invalid property to specify a HTML tooltip to keep the control focused while the user enters an inappropriate value for the field. The Invalid property has no effect if the property is empty or the control is not masked. Use the Warning property to specify a HTML tooltip to be shown at the cursor position when user enters an invalid character. Use the AllowBeep property to let control beeps once the user enters any invalid character. The Invalid property indicates the html message to be displayed when the user enters an inappropriate value for the field. If the value is missing or empty, the option has no effect, so no validation is performed. If the value is a not-empty value, the validation is performed. If the value is single space, no message is displayed and the field is keep opened while the value is inappropriate. For instance, "!(999) 000 0000;;;invalid=The value you entered isn't appropriate for the input mask '<%mask%>' specified for this field." displays the "The value you entered isn't appropriate for the input mask '...' specified for this field." tooltip once the user leaves the field and it is not-valid (for instance, the field includes entities required and uncompleted). The <%mask%> keyword in value, substitute the current mask of the field, while the <%value%> keyword substitutes the current value (including the literals). If this option should display/use the semicolon (;) character is should be \; (escape). This option can be combined with AllowEmptyValue or ValidateAs property.

The following screen shot shows the Warning once the user enters an invalid character:

The control uses built-in HTML tags to display the caption using HTML format. The control supports the following HTML tags:

- ** ... ** displays the text in **bold**
- **<i> ... </i>** displays the text in *italics*
- **<u> ... </u>** underlines the text
- **<s> ... </s>** ~~Strike~~-through text
- **<a id;options> ... ** displays an anchor element that can be clicked. An anchor is a piece of text or some other object (for example an image) which marks the beginning

and/or the end of a hypertext link. The <a> element is used to mark that piece of text (or inline image), and to give its hypertextual relationship to other documents. The control fires the *AnchorClick(AnchorID, Options)* event when the user clicks the anchor element. The *FormatAnchor* property customizes the visual effect for anchor elements.

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

- **<fgcolor rrggbb> ... </fgcolor>** or <fgcolor=rrggbb> ... </fgcolor> displays text with a specified foreground color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.

- **<bgcolor rrggbb> ... </bgcolor>** or <bgcolor=rrggbb> ... </bgcolor> displays text with a specified <mark>background</mark> 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 <u>[Add](Add)</u> 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 ₛᵤᵦₛcᵣᵢₚₜ 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

property MaskEdit.Mask as String

Retrieves or sets a string expression that indicates the control's mask.

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

Indicates the control's mask. Use the Value property to get the value entered, and use the Valid property to check if the field is completed and valid. Use the Masks property to provide multiple masks to the control. Use the MaskFloat property to mask floating point numbers without displaying the masking character. The MaskChange event is fired when the Mask property is changed. The Mask property is composed by a combination of regular characters, literal escape characters, and masking characters. The Mask property can contain also alternative characters, or range rules. A literal escape character is preceded by a \ character, and it is used to display a character that is used in masking rules. The control fires the MaskChange event once the control's Mask property is changed.

Starting from the version **7.1** the Mask property has been radically changed as explained bellow. If using any previously version, you can find the specifications at the bottom of the page.

For instance, the following input-mask (ext-phone)

"*!(999) 000 0000;1;;select=1,empty,overtype,warning=invalid character,invalid=The value you entered isn't appropriate for the input mask '<%mask%>' specified for this field.*"

indicates the following:

- The pattern should contain 3 optional digits *999*, and 7 required digits *000 0000*, aligned to the right, *!*.
- The second part of the input mask indicates *1*, which means that all literals are included when the user leaves the field.
- The entire field is selected when it receives the focus, *select=1*
- The field supports *empty* value, so the user can leave the field with no content
- The field enters in *overtype* mode, and insert-type mode is not allowed when user pressed the Insert key
- If the user enters any invalid character, a *warning* tooltip with the message "*invalid character*" is displayed.
- If the user tries to leave the field, while the field is not validated (all 7 required digits completed), the *invalid* tooltip is shown with the message "*The value you entered isn't appropriate for the input mask '<%mask%>' specified for this field.*" The *<%mask%>* is replaced with the first part of the input mask *!(999) 000 0000*

The four parts of an input mask, or the Mask property supports up to four parts, separated by a semicolon (;). For instance, "`Time: `00:00:00;;0;overtype,warning=<fgcolor FF0000>invalid character,beep", indicates the pattern "00:00" with the prefix Time:, the masking character being the 0, instead _, the field enters in over-type mode, insert-type mode is not allowed, and the field beeps and displays a tooltip in red with the message invalid character when the user enters an invalid character.

Input masks are made up one mandatory part and three optional parts, and each part is separated by a semicolon (;). If a part should use the semicolon (;) it must uses the \; instead

The purpose of each part is as follows:

1. The first part (pattern) is mandatory. It includes the mask characters or string (series of characters) along with placeholders and literal data such as, parentheses, periods, and hyphens.

   The following table lists the placeholder and literal characters for an input mask and explains how it controls data entry:

   - ***#**, a digit, +, - or space (entry not required).*
   - ***0**, a digit (0 through 9, entry required; plus [+] and minus [-] signs not allowed).*
   - ***9**, a digit or space (entry not required; plus and minus signs not allowed).*
   - ***x**, a lower case hexa character, [0-9],[a-f] (entry required)*
   - ***X**, an upper case hexa character, [0-9],[A-F] (entry required)*
   - ***A**, any letter, digit (entry required).*
   - ***a**, any letter, digit or space (entry optional).*
   - ***L**, any letter (entry require).*
   - ***?**, any letter or space (entry optional).*
   - ***&**, any character or a space (entry required).*
   - ***C**, any character or a space (entry optional).*
   - ***>**, any letter, converted to uppercase (entry required).*
   - ***<**, any letter, converted to lowercase (entry required).*
   - ****, any characters combinations*
   - ***{ min,max }** (Range), indicates a number range. The syntax {min,max} (Range), masks a number in the giving range. The min and max values should be positive integers. For instance the mask {0,255} masks any number between 0 and 255.*
   - ***[...]** (Alternative), masks any characters that are contained in the [] brackets. For instance, the [abcdA-D] mask any character: a,b,c,d,A,B,C,D*
   - *\, indicates the escape character*
   - ***⌐**, (ALT + 175) causes the characters that follow to be converted to uppercase,*

until Ť(ALT + 174) is found.

- o **Ť**, (ALT + 174) causes the characters that follow to be converted to lowercase, until ť(ALT + 175) is found.
- o **!**, causes the input mask to fill from right to left instead of from left to right.

Characters enclosed in double quotation ("" or ``) marks will be displayed literally. If this part should display/use the semicolon (;) character is should be included between double quotation ("" or ``) characters or as \; (escape).

2. The second part is optional and refers to the embedded mask characters and how they are stored within the field. If the second part is set to 0 (default, exClipModeLiteralsNone), all characters are stored with the data, and if it is set to 1 (exClipModeLiteralsInclude), the literals are stored, not including the masking/placeholder characters, if 2 (exClipModeLiteralsExclude), just typed characters are stored, if 3(exClipModeLiteralsEscape), optional, required, editable and escaped entities are included. No double quoted text is included.
3. The third part of the input mask is also optional and indicates a single character or space that is used as a placeholder. By default, the field uses the underscore (_). If you want to use another character, enter it in the third part of your mask. Only the first character is considered. If this part should display/use the semicolon (;) character is should be \; (escape) (MaskChar property)
4. The forth part of the input, indicates a list of options that can be applied to input mask, separated by comma(,) character.

The known options for the forth part are:

- o **float**, indicates that the field is edited as a decimal number, integer. The first part of the input mask specifies the pattern to be used for grouping and decimal separators, and - if negative numbers are supported. If the first part is empty, the float is formatted as indicated by current regional settings. For instance, "##;;;float" specifies a 2 digit number in float format. The grouping, decimal, negative and digits options are valid if the float option is present. Use the *MaskFloat* property to mask floating point numbers including digit grouping.

- o **grouping**=value, Character used to separate groups of digits to the left of the decimal. Valid only if float is present. For instance ";;;float,grouping=" indicates that no grouping is applied to the decimal number (LOCALE_STHOUSAND)
- o **decimal**=value, Character used for the decimal separator. Valid only if float is present. For instance ";;;float,grouping= ,decimal=\," indicates that the decimal

number uses the space for grouping digits to the left, while for decimal separator the comma character is used (LOCALE_SDECIMAL)

- ○ **negative**=value, indicates whether the decimal number supports negative numbers. The value should be 0 or 1. 1 means negative numbers are allowed. Else 0 or missing, the negative numbers are not accepted. Valid only if float is present.
- ○ **digits**=value, indicates the max number of fractional digits placed after the decimal separator. Valid only if float is present. For instance, ";;;float,digits=4" indicates a max 4 digits after decimal separator (LOCALE_IDIGITS)
- ○ **password**[=value], displays a black circle for any shown character. For instance, ";;;password", specifies that the field to be displayed as a password. If the value parameter is present, the first character in the value indicates the password character to be used. By default, the * password character is used for non-TrueType fonts, else the black circle character is used. For instance, ";;;password=*", specifies that the field to be displayed as a password, and use the * for password character. If the value parameter is missing, the default password character is used. The Password property specifies whether the control the control displays a black circle for any shown character. The PasswordChar property specifies the character to be displayed when the control displays a password.
- ○ **right**, aligns the characters to the right. For instance, "(999) 999-9999;;;right" displays and masks a telephone number aligned to the right. The Right property specifies whether the control displays the characters aligned to the right.
- ○ **readonly**, the editor is locked, user can not update the content, the caret is available, so user can copy the text, excepts the password fields. The ReadOnly property specifies whether the control is read-only. The BackColorReadOnly property specifies the control's background color when the field is locked (ReadOnly property is True). The ForeColorReadOnly property specifies the control's foreground color when the field is locked (ReadOnly property is True).
- ○ **inserttype**, indicates that the field enters in insert-type mode, if this is the first option found. If the forth part includes also the overtype option, it indicates that the user can toggle the insert/over-type mode using the Insert key. For instance, the "##:##;;0;inserttype,overtype", indicates that the field enter in insert-type mode, and over-type mode is allowed. The "##:##;;0;inserttype", indicates that the field enter in insert-type mode, and over-type mode is not allowed. The InsertMode property specifies the control's insertion mode. The AllowToggleInsertMode property specifies whether the control toggles the insertion mode when the user presses the Insert key.
- ○ **overtype**, indicates that the field enters in over-type mode, if this is the first option found. If the forth part includes also the inserttype option, it indicates that the user can toggle the insert/over-type mode using the Insert key. For instance,

the *"##:##;;0;overtype,inserttype"*, indicates that the field enter in over-type mode, and insert-type mode is allowed. The *"##:##;;0;overtype"*, indicates that the field enter in over-type mode, and insert-type mode is not allowed. The *InsertMode* property specifies the control's insertion mode. The *AllowToggleInsertMode* property specifies whether the control toggles the insertion mode when the user presses the Insert key.

- ○ **nocontext**, *indicates that the field provides no context menu when user right clicks the field. For instance, ";;;password,nocontext" displays a password field, where the user can not invoke the default context menu, usually when a right click occurs. The AllowContextMenu property indicates that the field provides no context menu when user right clicks the field.*

- ○ **beep**, *indicates whether a beep is played once the user enters an invalid character. For instance, "00:00;;;beep" plays a beep once the user types in invalid character, in this case any character that's not a digit.*

- ○ **warning**=value, *indicates the html message to be shown when the user enters an invalid character. For instance, "00:00:00;;;warning=invalid character" displays a "invalid character" tooltip once the user types in invalid character, in this case any character that's not a digit. The <%mask%> keyword in value, substitute the current mask of the field, while the <%value%> keyword substitutes the current value (including the literals). If this option should display/use the semicolon (;) character is should be \; (escape)*

- ○ **invalid**=value, *indicates the html message to be displayed when the user enters an inappropriate value for the field. If the value is missing or empty, the option has no effect, so no validation is performed. If the value is a not-empty value, the validation is performed. If the value is single space, no message is displayed and the field is keep opened while the value is inappropriate. For instance, "!(999) 000 0000;;;invalid=The value you entered isn't appropriate for the input mask '<%mask%>' specified for this field." displays the "The value you entered isn't appropriate for the input mask '...' specified for this field." tooltip once the user leaves the field and it is not-valid (for instance, the field includes entities required and uncompleted). The <%mask%> keyword in value, substitute the current mask of the field, while the <%value%> keyword substitutes the current value (including the literals). If this option should display/use the semicolon (;) character is should be \; (escape). This option can be combined with empty, validateas.*

- ○ **validateas**=value, *specifies the additional validation is done for the current field. If value is missing or 0 (exValidateAsNone), the option has no effect. The validateas option has effect only if the invalid option specifies a not-empty value. Currently, the value can be 1 (exValidateAsDate), which indicates that the field is validated as a date. For instance, having the mask "!00/00/0000;;0;empty,validateas=1,invalid=Invalid date!,warning=Invalid*

character!,select=4,overtype", indicates that the field is validate as date (validateas=1). The ValidateAs property indicates the additional validation makes once the user leaves the field.

- **empty**, *indicates whether the field supports empty values. This option can be used with invalid flag, which indicates that the user can leave the field if it is empty. If empty flag is present, the field displays nothing if no entity is completed (empty). Once the user starts typing characters the current mask is displayed. For instance, having the mask "!(999) 000 0000;;;empty,select=4,overtype,invalid=invalid phone number,beep", it specifies an empty or valid phone to be entered. The AllowEmptyValue property specifies whether the field supports empty values.*

- **select**=*value, indicates what to select from the field when it got the focus. The value could be 0 (nothing, exSelectNoGotFocus), 1 (select all, exSelectAllGotFocus), 2 (select the first empty and editable entity of the field, exSelectEditableGotFocus), 3 (moves the cursor to the beginning of the first empty and editable entity of the field, exMoveEditableGotFocus), 4 (select the first empty, required and editable entity of the field, exSelectRequiredEditableGotFocus), 5 (moves the cursor to the beginning of the first empty, required and editable entity of the field, exMoveRequiredEditableGotFocus). For modes 2 and 4 the entire field is selected if no matching entity is found. For instance, "`Time:`XX:XX;;;select=1" indicates that the entire field (including the Time: prefix) is selected once it get the focus. The "`Time:`XX:XX;;;select=3", moves the cursor to first X, if empty, the second if empty, and so on*

Experimental:
multiline, *specifies that the field supports multiple lines.*
rich, *specifies that the field displays a rich type editor. By default, the standard edit field is shown*
disabled, *shows as disabled the field.*

Prior to version **7.1** the Mask property supports the following special characters.

Here's the list of all rules and masking characters.

- **#** *(Digit), Masks a digit character, [0-9]*
- **x** *(Hexa Lower), Masks a lower case hexa character, [0-9],[a-f]*
- **X** *(Hexa Upper), Masks an upper case hexa character, [0-9],[A-F]*
- **A** *(AlphaNumeric), Masks a letter or a digit. [0-9], [a-z], [A-Z]*
- **?** *(Alphabetic), Masks a letter. [a-z],[A-Z]*
- **<** *(Alphabetic lower), Masks a lower case letter. [a-z]*
- **>** *(Alphabetic upper), Masks an upper case letter. [A-Z]*

- *(Any), Masks any combination of characters.*
- *\ (Literal Escape), Displays any masking characters. The following combinations are valid: \#, \x, \X, \A, \?, \<, \>, \\, \{, \[*
- *{nMin,nMax} (Range), Masks a number in a range. The nMin and nMax values should be numbers. For instance the mask {0,255} will mask any number between 0 and 255.*
- *[…] (Alternative), Masks any characters that are contained in the [] brackets. For instance, the [abcA-C] mask any character: a,b,c,A,B,C*

The following sample shows how to mask an IP address:

MaskEdit1.Mask = "{0,255}\.{0,255}\.{0,255}\.{0,255}"

property MaskEdit.MaskChar as Integer

Retrieves or sets a value that determines masking character.

Type	Description
Integer	A character expression that indicates the character used for masking characters

By default, the masking character is "_". Use the MaskChar property to change the masking character. Use the [Mask](#) property to assign a single mask to the control. Use the [Masks](#) property to assign multiple masks to the control. Use the [MaskFloat](#) property to mask floating point numbers without displaying the masking character.

For instance, the following sample changes the masking character to "0":

```
With MaskEdit1
    .MaskChar = Asc("0")
    .Mask = "[0-9][0-9][0-9][0-9]"
End With
```

property MaskEdit.MaskFloat as Boolean

Specifies whether the Mask property masks a floating point number.

Type	Description
Boolean	A boolean expression that indicates whether the Mask property masks a floating point number.

By default, the MaskFloat property is False. Use the MaskFloat property to mask floating point numbers including digit grouping. Use the [Mask](#) property to specify the mask for floating point numbers. If the MaskFloat property is True, and Mask property is empty, the control filters the input characters using your regional options. If the MaskFloat property is True, the control doesn't display the masking character that's referred by the [MaskChar](#) property.

If the MaskFloat property is True, the Mask property may indicate the followings:

- **negative number**: if the first character in the mask is - (minus) the control supports negative numbers. Pressing the - key will toggle the sign of the number. The + sign is never displayed.
- **decimal symbol**: the last character that's different than # (digit), or 0 (zero) indicates the decimal symbol. If it is not present the control mask a floating point number without decimals.
- **thousand symbol**: the thousand symbol is the last character that's not a # (digit), 0 (zero) or it is not the decimal symbol as explained earlier, if present.
- the maximum **number of decimals** in the number (the # or 0 character after the decimal symbol)
- the maximum number of digits in the integer part (the number of # or 0 character before decimal symbol)
- the **0** character indicates **a leading-zero**. The count of 0 (zero) characters before decimal character indicates the leading-zero for integer part of the control, while the count of 0 (zero) characters after the decimal separator indicates the leading-zero for decimal part of the control. For instance, the Mask on "-###,###,##0.00", while the control's Text property is 1, the control displays 1.00, if 1.1 if displays 1.10, and if empty, the 0.00 is displayed.

If the Mask property is empty, the control takes the settings for the regional options like: Decimal Symbol , No. of digits after decimal, Digit grouping symbol.

Here are few samples:

The mask "**-###.###.##0,00**" filter floating point numbers a number for German settings ("," is the decimal sign, "." is the thousands separator). This format displays leading-zeros.

The mask "**-###.###.###,##**" filter floating point numbers a number for German settings (","
is the decimal sign, "." is the thousands separator)

The mask "**-###,###,###.##**" filter floating point numbers a number for English settings ("."
is the decimal sign, "," is the thousands separator)

The mask "**####**" indicates a max-4 digit number (positive) without a decimal symbol and
without digit grouping

The mask "**-##.#**" filters a floating point number from the -99.9 to 99.9 ("." is the decimal
sign, no thousands separator)

The mask "**#,###.##**" filters a floating point number from the 0 to 9,999.99 with digit
grouping ("." is the decimal sign, "," is the thousands separator).

property MaskEdit.Masks as String

Retrieves or sets a value that determines all possible masks that user can use at runtime.

Type	Description
String	A string expression that defines the list of masks in the control.

Use the Masks property to assign multiple masks to the control. Use the Mask property to assign a single mask to the control. The MaskChange event is fired when user changes the current mask. The Masks property is composed by pairs description and mask. The list of masks is delimited by "|" character. Use the VisibleMasks property to specify the number of visible items in the control's masks list.

The following sample adds multiple masks to the control:

VBA (MS Access, Excell...)

```
With MaskEdit1
    .InsertMode = 1
    .SelectGotFocus = 1
    .VisibleMasks = 6
    .TextIncludeLiterals = 2
    .Masks = "Local;!000-0000|Domestic;!(999) 000-0000|International;!`+1` 999-000-
0000|Dialed in the US;!`1`" & _
" 999-000-0000|Dialed from Germany;!`001` 999-000-0000|Dialed from France;!`191`
999-000-0000"
    .Text = "845 0287"
    .ActiveMask = 1
End With
```

VB6

```
With MaskEdit1
    .InsertMode = exEditOvertypeMode
    .SelectGotFocus = exSelectAllGotFocus
    .VisibleMasks = 6
    .TextIncludeLiterals = exClipModeLiteralsExclude
    .Masks = "Local;!000-0000|Domestic;!(999) 000-0000|International;!`+1` 999-000-
0000|Dialed in the US;!`1`" & _
```

```
" 999-000-0000|Dialed from Germany;!`001` 999-000-0000|Dialed from France;!`191`
999-000-0000"
    .Text = "845 0287"
    .ActiveMask = 1
End With
```

VB.NET

```
With Exmaskedit1
    .InsertMode = exontrol.EXMASKEDITLib.InsertModeEnum.exEditOvertypeMode
    .SelectGotFocus =
exontrol.EXMASKEDITLib.SelectGotFocusEnum.exSelectAllGotFocus
    .VisibleMasks = 6
    .TextIncludeLiterals =
exontrol.EXMASKEDITLib.ClipModeLiteralsEnum.exClipModeLiteralsExclude
    .Masks = "Local;!000-0000|Domestic;!(999) 000-0000|International;!`+1` 999-000-
0000|Dialed in the US;!`1`" & _
" 999-000-0000|Dialed from Germany;!`001` 999-000-0000|Dialed from France;!`191`
999-000-0000"
    .Text = "845 0287"
    .ActiveMask = 1
End With
```

VB.NET for /COM

```
With AxMaskEdit1
    .InsertMode = EXMASKEDITLib.InsertModeEnum.exEditOvertypeMode
    .SelectGotFocus = EXMASKEDITLib.SelectGotFocusEnum.exSelectAllGotFocus
    .VisibleMasks = 6
    .TextIncludeLiterals =
EXMASKEDITLib.ClipModeLiteralsEnum.exClipModeLiteralsExclude
    .Masks = "Local;!000-0000|Domestic;!(999) 000-0000|International;!`+1` 999-000-
0000|Dialed in the US;!`1`" & _
" 999-000-0000|Dialed from Germany;!`001` 999-000-0000|Dialed from France;!`191`
999-000-0000"
    .Text = "845 0287"
    .ActiveMask = 1
End With
```

C++

```cpp
/*
    Copy and paste the following directives to your header file as
    it defines the namespace 'EXMASKEDITLib' for the library: 'ExMaskEdit 7.1 Control
Library'

    #import <MaskEdit.dll>
    using namespace EXMASKEDITLib;
*/
EXMASKEDITLib::IMaskEditPtr spMaskEdit1 = GetDlgItem(IDC_MASKEDIT1)->GetControlUnknown();
spMaskEdit1->PutInsertMode(EXMASKEDITLib::exEditOvertypeMode);
spMaskEdit1->PutSelectGotFocus(EXMASKEDITLib::exSelectAllGotFocus);
spMaskEdit1->PutVisibleMasks(6);
spMaskEdit1->PutTextIncludeLiterals(EXMASKEDITLib::exClipModeLiteralsExclude);
spMaskEdit1->PutMasks(_bstr_t("Local;!000-0000|Domestic;!(999) 000-0000|International;!`+1` 999-000-0000|Dialed in the US;!`1`") +
" 999-000-0000|Dialed from Germany;!`001` 999-000-0000|Dialed from France;!`191` 999-000-0000");
spMaskEdit1->PutText(L"845 0287");
spMaskEdit1->PutActiveMask(1);
```

C++ Builder

```cpp
MaskEdit1->InsertMode = Exmaskeditlib_tlb::InsertModeEnum::exEditOvertypeMode;
MaskEdit1->SelectGotFocus = Exmaskeditlib_tlb::SelectGotFocusEnum::exSelectAllGotFocus;
MaskEdit1->VisibleMasks = 6;
MaskEdit1->TextIncludeLiterals = Exmaskeditlib_tlb::ClipModeLiteralsEnum::exClipModeLiteralsExclude;
MaskEdit1->Masks = TVariant(String("Local;!000-0000|Domestic;!(999) 000-0000|International;!`+1` 999-000-0000|Dialed in the US;!`1`") +
" 999-000-0000|Dialed from Germany;!`001` 999-000-0000|Dialed from France;!`191` 999-000-0000");
MaskEdit1->Text = L"845 0287";
```

```
MaskEdit1->ActiveMask = 1;
```

C#

```
exmaskedit1.InsertMode =
exontrol.EXMASKEDITLib.InsertModeEnum.exEditOvertypeMode;
exmaskedit1.SelectGotFocus =
exontrol.EXMASKEDITLib.SelectGotFocusEnum.exSelectAllGotFocus;
exmaskedit1.VisibleMasks = 6;
exmaskedit1.TextIncludeLiterals =
exontrol.EXMASKEDITLib.ClipModeLiteralsEnum.exClipModeLiteralsExclude;
exmaskedit1.Masks = "Local;!000-0000|Domestic;!(999) 000-0000|International;!`+1`
999-000-0000|Dialed in the US;!`1`" +
" 999-000-0000|Dialed from Germany;!`001` 999-000-0000|Dialed from France;!`191`
999-000-0000";
exmaskedit1.Text = "845 0287";
exmaskedit1.ActiveMask = 1;
```

JavaScript

```
<OBJECT classid="clsid:43F80262-F652-11D3-AD39-00C0DFC59237"
id="MaskEdit1"></OBJECT>

<SCRIPT LANGUAGE="JScript">
    MaskEdit1.InsertMode = 1;
    MaskEdit1.SelectGotFocus = 1;
    MaskEdit1.VisibleMasks = 6;
    MaskEdit1.TextIncludeLiterals = 2;
    MaskEdit1.Masks = "Local;!000-0000|Domestic;!(999) 000-0000|International;!`+1`
999-000-0000|Dialed in the US;!`1`" +
    " 999-000-0000|Dialed from Germany;!`001` 999-000-0000|Dialed from
France;!`191` 999-000-0000";
    MaskEdit1.Text = "845 0287";
    MaskEdit1.ActiveMask = 1;
</SCRIPT>
```

C# for /COM

```
axMaskEdit1.InsertMode = EXMASKEDITLib.InsertModeEnum.exEditOvertypeMode;
axMaskEdit1.SelectGotFocus =
EXMASKEDITLib.SelectGotFocusEnum.exSelectAllGotFocus;
axMaskEdit1.VisibleMasks = 6;
axMaskEdit1.TextIncludeLiterals =
EXMASKEDITLib.ClipModeLiteralsEnum.exClipModeLiteralsExclude;
axMaskEdit1.Masks = "Local;!000-0000|Domestic;!(999) 000-0000|International;!`+1`
999-000-0000|Dialed in the US;!`1`" +
" 999-000-0000|Dialed from Germany;!`001` 999-000-0000|Dialed from France;!`191`
999-000-0000";
axMaskEdit1.Text = "845 0287";
axMaskEdit1.ActiveMask = 1;
```

X++ (Dynamics Ax 2009)

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

    super();

    exmaskedit1.InsertMode(1/*exEditOvertypeMode*/);
    exmaskedit1.SelectGotFocus(1/*exSelectAllGotFocus*/);
    exmaskedit1.VisibleMasks(6);
    exmaskedit1.TextIncludeLiterals(2/*exClipModeLiteralsExclude*/);
    var_s = "Local;!000-0000|Domestic;!(999) 000-0000|International;!`+1` 999-000-
0000|Dialed in the US;!`1` ";
    var_s = var_s + "999-000-0000|Dialed from Germany;!`001` 999-000-0000|Dialed
from France;!`191` 999-000-0000";
    exmaskedit1.Masks(var_s);
    exmaskedit1.Text("845 0287");
    exmaskedit1.ActiveMask(1);
}
```

Delphi 8 (.NET only)

```
with AxMaskEdit1 do
begin
   InsertMode := EXMASKEDITLib.InsertModeEnum.exEditOvertypeMode;
   SelectGotFocus := EXMASKEDITLib.SelectGotFocusEnum.exSelectAllGotFocus;
   VisibleMasks := 6;
   TextIncludeLiterals :=
EXMASKEDITLib.ClipModeLiteralsEnum.exClipModeLiteralsExclude;
   Masks := 'Local;!000-0000|Domestic;!(999) 000-0000|International;!`+1` 999-000-
0000|Dialed in the US;!`1` ' +
   '999-000-0000|Dialed from Germany;!`001` 999-000-0000|Dialed from
France;!`191` 999-000-0000';
   Text := '845 0287';
   ActiveMask := 1;
end
```

Delphi (standard)

```
with MaskEdit1 do
begin
   InsertMode := EXMASKEDITLib_TLB.exEditOvertypeMode;
   SelectGotFocus := EXMASKEDITLib_TLB.exSelectAllGotFocus;
   VisibleMasks := 6;
   TextIncludeLiterals := EXMASKEDITLib_TLB.exClipModeLiteralsExclude;
   Masks := 'Local;!000-0000|Domestic;!(999) 000-0000|International;!`+1` 999-000-
0000|Dialed in the US;!`1` ' +
   '999-000-0000|Dialed from Germany;!`001` 999-000-0000|Dialed from
France;!`191` 999-000-0000';
   Text := '845 0287';
   ActiveMask := 1;
end
```

VFP

```
with thisform.MaskEdit1
   .InsertMode = 1
   .SelectGotFocus = 1
```

```
    .VisibleMasks = 6
    .TextIncludeLiterals = 2
    var_s = "Local;!000-0000|Domestic;!(999) 000-0000|International;!`+1`999-000-
0000|Dialed in the US;!`1`"
    var_s = var_s + "999-000-0000|Dialed from Germany;!`001`999-000-0000|Dialed
from France;!`191`999-000-0000"
    .Masks = var_s
    .Text = "845 0287"
    .ActiveMask = 1
endwith
```

dBASE Plus

```
local oMaskEdit

oMaskEdit = form.Activex1.nativeObject
oMaskEdit.InsertMode = 1
oMaskEdit.SelectGotFocus = 1
oMaskEdit.VisibleMasks = 6
oMaskEdit.TextIncludeLiterals = 2
oMaskEdit.Masks = "Local;!000-0000|Domestic;!(999) 000-0000|International;!`+1`
999-000-0000|Dialed in the US;!`1`999-000-0000|Dialed from Germany;!`001`999-
000-0000|Dialed from France;!`191`999-000-0000"
oMaskEdit.Text = "845 0287"
oMaskEdit.ActiveMask = 1
```

XBasic (Alpha Five)

```
Dim oMaskEdit as P

oMaskEdit = topparent:CONTROL_ACTIVEX1.activex
oMaskEdit.InsertMode = 1
oMaskEdit.SelectGotFocus = 1
oMaskEdit.VisibleMasks = 6
oMaskEdit.TextIncludeLiterals = 2
oMaskEdit.Masks = "Local;!000-0000|Domestic;!(999) 000-0000|International;!`+1`
999-000-0000|Dialed in the US;!`1`999-000-0000|Dialed from Germany;!`001`999-
```

```
000-0000|Dialed from France;!`191` 999-000-0000"
oMaskEdit.Text = "845 0287"
oMaskEdit.ActiveMask = 1
```

Visual Objects

```
oDCOCX_Exontrol1:InsertMode := exEditOvertypeMode
oDCOCX_Exontrol1:SelectGotFocus := exSelectAllGotFocus
oDCOCX_Exontrol1:VisibleMasks := 6
oDCOCX_Exontrol1:TextIncludeLiterals := exClipModeLiteralsExclude
oDCOCX_Exontrol1:Masks := "Local;!000-0000|Domestic;!(999) 000-
0000|International;!`+1` 999-000-0000|Dialed in the US;!`1` 999-000-0000|Dialed
from Germany;!`001` 999-000-0000|Dialed from France;!`191` 999-000-0000"
oDCOCX_Exontrol1:Text := "845 0287"
oDCOCX_Exontrol1:ActiveMask := 1
```

PowerBuilder

```
OleObject oMaskEdit

oMaskEdit = ole_1.Object
oMaskEdit.InsertMode = 1
oMaskEdit.SelectGotFocus = 1
oMaskEdit.VisibleMasks = 6
oMaskEdit.TextIncludeLiterals = 2
oMaskEdit.Masks = "Local;!000-0000|Domestic;!(999) 000-0000|International;!`+1`
999-000-0000|Dialed in the US;!`1` 999-000-0000|Dialed from Germany;!`001` 999-
000-0000|Dialed from France;!`191` 999-000-0000"
oMaskEdit.Text = "845 0287"
oMaskEdit.ActiveMask = 1
```

Visual DataFlex

```
Procedure OnCreate
```

```
    Forward Send OnCreate
    Set ComInsertMode to OLEexEditOvertypeMode
    Set ComSelectGotFocus to OLEexSelectAllGotFocus
    Set ComVisibleMasks to 6
    Set ComTextIncludeLiterals to OLEexClipModeLiteralsExclude
    Set ComMasks to "Local;!000-0000|Domestic;!(999) 000-0000|International;!`+1`
999-000-0000|Dialed in the US;!`1` 999-000-0000|Dialed from Germany;!`001` 999-
000-0000|Dialed from France;!`191` 999-000-0000"
    Set ComText to "845 0287"
    Set ComActiveMask to 1
End_Procedure
```

property MaskEdit.MultiLine as Boolean

Retrieves or sets a value that determines whether the control can accept multiple lines of text.

Type	Description
Boolean	A boolean expression that determines whether the control can accept multiple lines of text.

Use the MultiLine property to allow multiple lines to the control.

property MaskEdit.Password as Boolean

Displays all characters as an asterisk (*)

Type	Description
Boolean	A Boolean expression that specifies whether the control displays (*) characters instead.

By default, the Password property is False. The Password property specifies whether the control the control displays a black circle for any shown character. For instance, Mask on ";;;password", specifies that the field to be displayed as a password. If the value parameter is present, the first character in the value indicates the password character to be used. By default, the * password character is used for non-TrueType fonts, else the black circle character is used. For instance, Mask on ";;;password=*", specifies that the field to be displayed as a password, and use the * for password character. If the value parameter is missing, the default password character is used. The PasswordChar property specifies the character to be displayed instead the black circle character.

property MaskEdit.PasswordChar as Integer

Retrieves or sets a value that determines password character.

Type	Description
Integer	A Short expression that specifies the code of the character to be displayed when the control's [Password](#) property is True.

By default, the PasswordChar property is -107, which indicates the code for the black circle character. The [Password](#) property specifies whether the control the control displays a black circle for any shown character. For instance, Mask on ";;;password", specifies that the field to be displayed as a password. If the value parameter is present, the first character in the value indicates the password character to be used. By default, the * password character is used for non-TrueType fonts, else the black circle character is used. For instance, Mask on ";;;password=*", specifies that the field to be displayed as a password, and use the * for password character. If the value parameter is missing, the default password character is used.

property MaskEdit.ReadOnly as Boolean

Retrieves or sets a value that determines whether the control is read only.

Type	Description
Boolean	A boolean expression that determines whether the control is read only.

By default, the ReadOnly property is False. The ReadOnly property indicates that the editor is locked, user can not update the content, the caret is available, so user can copy the text, excepts the password fields. Use the ReadOnly property to make your control read-only. Use Enabled property to disable the control. If the control is read-only, the user can change the cursor position. If the control is disabled, the cursor position cannot be changed. The BackColorReadOnly property specifies the control's background color when the field is locked (ReadOnly property is True). The ForeColorReadOnly property specifies the control's foreground color when the field is locked (ReadOnly property is True).

method MaskEdit.Refresh ()

Refreshes the control's field.

Type	Description

Use the Refresh method to programmatically refresh visually the control's content. Use the Text property to specify the control's text. Use the Mask property to change the control's mask.

method MaskEdit.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 long expression that indicates the icon's handle.
Index as Variant	A long expression that indicates the index where icon is inserted.

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

Use the 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 a image list to the control. The ImageSize property defines the size (width/height) of the icons within the control's Images collection.

The following VB sample adds a new icon to control's images list:

 i = ExMaskEdit1.ReplaceIcon(LoadPicture("d:\icons\help.ico").Handle), i specifies the index where the icon is added

The following VB sample replaces an icon into control's images list::

 i = ExMaskEdit1.ReplaceIcon(LoadPicture("d:\icons\help.ico").Handle, 0), i is zero, so the first icon is replaced.

The following VB sample removes an icon from control's images list:

 ExMaskEdit1.ReplaceIcon 0, i, i specifies the index of icon removed.

The following VB clears the control's icons collection:

 ExMaskEdit1.ReplaceIcon 0, -1

property MaskEdit.Right as Boolean

Right aligns text in a single-line or multiline edit control.

Type	Description
Boolean	A Boolean expression that specifies whether the control aligns right the text

By default, the Right property is False. Use the Right property to align the control's text to the right.

method MaskEdit.Select (Start as Long, End as Long)

Selects the text between Start and End

Type	Description
Start as Long	A long expression that indicates the starting position.
End as Long	A long expression that indicates the ending position.

The Select method selects the text inside the control. The Start and End parameters determine the starting and ending point of the selection. 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 SelEnd property returns or sets the ending point of the text selected. The SelectGotFocus property indicates what the control should select once the control gets the focus.

property MaskEdit.SelectGotFocus as SelectGotFocusEnum

Indicates whether the entire text is selected once the field receives the focus.

Type	Description
SelectGotFocusEnum	A SelectGotFocusEnum expression that specifies what to select when the control gets the focus.

By default, The SelectGotFocus property is exMoveEditableGotFocus. The SelectGotFocus property indicates what the control should select once the control gets the focus. For instance, use the SelectGotFocus property to highlight the un-completed part of the field when control gets the focus. The TextIncludeLiteralsLoseFocus property specifies the control's text to be displayed when the control loses the focus. The Select method selects programmatically a part of the text inside the control.

property MaskEdit.SelEnd as Long

Returns or sets the ending point of text selected.

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

The SelEnd property returns or sets the ending point of the text selected. The SelStart property returns or sets the starting point of text selected; indicates the position of the insertion point if no text is selected. Use the Select method to select a part of the control's text. The SelectGotFocus property indicates what the control should select once the control gets the focus.

property MaskEdit.SelStart as Long

Returns or sets the starting point of text selected.

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

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 [SelEnd](#) property returns or sets the ending point of the text selected. Use the [Select](#) method to select a part of the control's text. The [SelectGotFocus](#) property indicates what the control should select once the control gets the focus.

property MaskEdit.ShowImageList as Boolean

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

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

By default, the ShowImageList property is True. Use the ShowImageList property to hide the control's images list window. The control's images list window is visible only at design time. Use the Images method to associate an images list control to the control. Use the RepaceIcon method to add, remove or clear icons in the control's images collection. Use the <img> HTML tag to display an icon to HTML tooltips.

method MaskEdit.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**
- **<i> ... </i>** displays the text in *italics*
- **<u> ... </u>** <u>underlines</u> the text
- **<s> ... </s>** ~~Strike~~-through text
- **<a id;options> ... ** displays an <u>anchor</u> 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 _{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:

- **<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 MaskEdit.Template as String

Specifies the control's template.

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

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

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

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

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

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

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

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

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

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

The x-script may uses constant expressions as follow:

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

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

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

property MaskEdit.TemplateDef as Variant

Defines inside variables for the next Template/ExecuteTemplate call.

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

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

So, calling the TemplateDef property should be as follows:

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

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

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

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

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

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

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

```
local Control,var_Column

Control = form.Activex1.nativeObject
// Control.Columns.Add("Column 1").Def(4) = 255
var_Column = Control.Columns.Add("Column 1")
with (Control)
   TemplateDef = [Dim var_Column]
   TemplateDef = var_Column
   Template = [var_Column.Def(4) = 255]
endwith
Control.Columns.Add("Column 2")
Control.Items.AddItem(0)
Control.Items.AddItem(1)
Control.Items.AddItem(2)
```

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

```
Dim Control as P
Dim var_Column as P

Control = topparent:CONTROL_ACTIVEX1.activex
' Control.Columns.Add("Column 1").Def(4) = 255
var_Column = Control.Columns.Add("Column 1")
Control.TemplateDef = "Dim var_Column"
Control.TemplateDef = var_Column
Control.Template = "var_Column.Def(4) = 255"

Control.Columns.Add("Column 2")
Control.Items.AddItem(0)
Control.Items.AddItem(1)
Control.Items.AddItem(2)
```

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

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

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

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

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

The x-script may uses constant expressions as follow:

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

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

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

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

method MaskEdit.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.

property MaskEdit.Text as String

Retrieves or sets the text contained in the control.

Type	Description
String	A string expression that indicates the text contained in the control.

Use the Text property to get/set the contained control's text, containing masking or non masking characters. The TextIncludeLiterals property specifies the way the Text property get/set the control's text. For instance, if you do not need to get the placeholders characters in the Text property, set the TextIncludeLiterals property on exClipModeLiteralsInclude before calling the Text property. The Value property is equivalent with Text/TextIncludeLiterals property. The control fires the Change event if the user changes the control's text. The TextIncludeLiteralsLoseFocus property specifies how the control's field is shown when it loses the focus. The AllowEmptyValue property specifies whether the control handles empty values.

property MaskEdit.TextIncludeLiterals as ClipModeLiteralsEnum

Determines the way the Text property returns or set the value of the field

Type	Description
ClipModeLiteralsEnum	A ClipModeIncludeLiteralEnum expression that specifies the way the control's text is retrieved/set.

By default, the TextIncludeLiterals property is exClipModeLiteralsNone. The TextIncludeLiterals property specifies the way the Text property get/set the control's text. For instance, if you do not need to get the placeholders characters in the Text property, set the TextIncludeLiterals property on exClipModeLiteralsInclude before calling the Text property. The TextIncludeLiteralsLoseFocus property specifies how the control's field is shown when it loses the focus. The Value property is equivalent with Text/TextIncludeLiterals property. The control fires the Change event if the user changes the control's text. The AllowEmptyValue property specifies whether the control handles empty values.

property MaskEdit.TextIncludeLiteralsLoseFocus as ClipModeLiteralsEnum

Determines how the field shows its content once it loses the focus.

Type	Description
ClipModeLiteralsEnum	A ClipModeLiteralsEnum expression that specifies the way the control displays its content once it loses the focus.

By default, the TextIncludeLiteralsLoseFocus property is exClipModeLiteralsNone. The TextIncludeLiteralsLoseFocus property specifies how the control's field is shown when it loses the focus. The TextIncludeLiterals property specifies the way the Text property get/set the control's text. For instance, if you do not need to get the placeholders characters in the Text property, set the TextIncludeLiterals property on exClipModeLiteralsInclude before calling the Text property. The Value property is equivalent with Text/TextIncludeLiterals property. The control fires the Change event if the user changes the control's text. The AllowEmptyValue property specifies whether the control handles empty values. The SelectGotFocus property indicates how the control specifies the selection once the control gets the focus.

property MaskEdit.Type as TypeEnum

Retrieves or sets a value that determines the based window used by control.

Type	Description
TypeEnum	A TypeEnum expression that indicates the based window type used.

By default, the Type property is Edit. Use the Type = RichEdit to allow coloring when the masked value is not valid. Use Valid property to check if the masked value is valid or invalid

property MaskEdit.Valid as Boolean

Retrieves a value indicating whether the edit contains a valid value.

Type	Description
Boolean	A boolean expression that indicates whether the control's mask contains a valid or invalid value.

Use the Valid property to check whether the user has completed entering the value into the masked edit. The following sample shows how to use Valid property:

```
Private Sub MaskEdit1_Change()
    Debug.Print Iif(Not MaskEdit1.Valid, "...uncompleted", MaskEdit1.MaskValue())
End Sub
```

property MaskEdit.ValidateAs as ValidateAsEnum

Indicates the additional validation is performed, once the user leaves the field.

Type	Description
[ValidateAsEnum](#)	A ValidateAsEnum expression that specifies the type of validation it should be performed once the field is completed.

By default, the ValidateAs property is exValidateAsNone, so no further validation is performed. For instance, you can use the ValidateAs property to validate a valid date your user enters in the field. This property has effect only if the Invalid property is not empty. The [Invalid](#) property is not empty, indicates the html message to be displayed when the user enters an inappropriate value for the field. If the value is missing or empty, the option has no effect, so no validation is performed. If the value is a not-empty value, the validation is performed. If the value is single space, no message is displayed and the field is keep opened while the value is inappropriate. For instance, "!(999) 000 0000;;;invalid=The value you entered isn't appropriate for the input mask '<%mask%>' specified for this field." displays the "The value you entered isn't appropriate for the input mask '...' specified for this field." tooltip once the user leaves the field and it is not-valid (for instance, the field includes entities required and uncompleted). The <%mask%> keyword in value, substitute the current mask of the field, while the <%value%> keyword substitutes the current value (including the literals). If this option should display/use the semicolon (;) character is should be \; (escape). This option can be combined with empty, validateas.

The following samples validates the field as a date:

VBA (MS Access, Excell...)

```
With MaskEdit1
    .SelectGotFocus = 4
    .Mask = "!99/99/9999;; ;select=4,overtype"
    .Text = #1/2/2001#
    .Warning = "Invalid character!"
    .Invalid = "Invalid date!"
    .ValidateAs = 1
    .AllowEmptyValue = True
End With
```

VB6

```
With MaskEdit1
```

```
            .SelectGotFocus = exSelectRequiredEditableGotFocus
            .Mask = "!99/99/9999;; ;select=4,overtype"
            .Text = #1/2/2001#
            .Warning = "Invalid character!"
            .Invalid = "Invalid date!"
            .ValidateAs = exValidateAsDate
            .AllowEmptyValue = True
        End With
```

VB.NET

```
With Exmaskedit1
    .SelectGotFocus =
exontrol.EXMASKEDITLib.SelectGotFocusEnum.exSelectRequiredEditableGotFocus
    .Mask = "!99/99/9999;; ;select=4,overtype"
    .Text = #1/2/2001#
    .Warning = "Invalid character!"
    .Invalid = "Invalid date!"
    .ValidateAs = exontrol.EXMASKEDITLib.ValidateAsEnum.exValidateAsDate
    .AllowEmptyValue = True
End With
```

VB.NET for /COM

```
With AxMaskEdit1
    .SelectGotFocus =
EXMASKEDITLib.SelectGotFocusEnum.exSelectRequiredEditableGotFocus
    .Mask = "!99/99/9999;; ;select=4,overtype"
    .Text = #1/2/2001#
    .Warning = "Invalid character!"
    .Invalid = "Invalid date!"
    .ValidateAs = EXMASKEDITLib.ValidateAsEnum.exValidateAsDate
    .AllowEmptyValue = True
End With
```

C++

```
/*
```

```
    Copy and paste the following directives to your header file as
    it defines the namespace 'EXMASKEDITLib' for the library: 'ExMaskEdit 7.1 Control
Library'

    #import <MaskEdit.dll>
    using namespace EXMASKEDITLib;
*/
EXMASKEDITLib::IMaskEditPtr spMaskEdit1 = GetDlgItem(IDC_MASKEDIT1)-
>GetControlUnknown();
spMaskEdit1-
>PutSelectGotFocus(EXMASKEDITLib::exSelectRequiredEditableGotFocus);
spMaskEdit1->PutMask(L"!99/99/9999;; ;select=4,overtype");
spMaskEdit1->PutText(L"1/2/2001");
spMaskEdit1->PutWarning(L"Invalid character!");
spMaskEdit1->PutInvalid(L"Invalid date!");
spMaskEdit1->PutValidateAs(EXMASKEDITLib::exValidateAsDate);
spMaskEdit1->PutAllowEmptyValue(VARIANT_TRUE);
```

C++ Builder

```
MaskEdit1->SelectGotFocus =
Exmaskeditlib_tlb::SelectGotFocusEnum::exSelectRequiredEditableGotFocus;
MaskEdit1->Mask = L"!99/99/9999;; ;select=4,overtype";
MaskEdit1->Text = L"TDateTime(2001,1,2).operator double()";
MaskEdit1->Warning = L"Invalid character!";
MaskEdit1->Invalid = L"Invalid date!";
MaskEdit1->ValidateAs = Exmaskeditlib_tlb::ValidateAsEnum::exValidateAsDate;
MaskEdit1->AllowEmptyValue = true;
```

C#

```
exmaskedit1.SelectGotFocus =
exontrol.EXMASKEDITLib.SelectGotFocusEnum.exSelectRequiredEditableGotFocus;
exmaskedit1.Mask = "!99/99/9999;; ;select=4,overtype";
exmaskedit1.Text =
Convert.ToDateTime("1/2/2001",System.Globalization.CultureInfo.GetCultureInfo("en-
```

```
US")).ToString();
exmaskedit1.Warning = "Invalid character!";
exmaskedit1.Invalid = "Invalid date!";
exmaskedit1.ValidateAs =
exontrol.EXMASKEDITLib.ValidateAsEnum.exValidateAsDate;
exmaskedit1.AllowEmptyValue = true;
```

JavaScript

```
<OBJECT classid="clsid:43F80262-F652-11D3-AD39-00C0DFC59237"
id="MaskEdit1"></OBJECT>

<SCRIPT LANGUAGE="JScript">
  MaskEdit1.SelectGotFocus = 4;
  MaskEdit1.Mask = "!99/99/9999;; ;select=4,overtype";
  MaskEdit1.Text = "1/2/2001";
  MaskEdit1.Warning = "Invalid character!";
  MaskEdit1.Invalid = "Invalid date!";
  MaskEdit1.ValidateAs = 1;
  MaskEdit1.AllowEmptyValue = true;
</SCRIPT>
```

C# for /COM

```
axMaskEdit1.SelectGotFocus =
EXMASKEDITLib.SelectGotFocusEnum.exSelectRequiredEditableGotFocus;
axMaskEdit1.Mask = "!99/99/9999;; ;select=4,overtype";
axMaskEdit1.Text =
Convert.ToDateTime("1/2/2001",System.Globalization.CultureInfo.GetCultureInfo("en-
US")).ToString();
axMaskEdit1.Warning = "Invalid character!";
axMaskEdit1.Invalid = "Invalid date!";
axMaskEdit1.ValidateAs = EXMASKEDITLib.ValidateAsEnum.exValidateAsDate;
axMaskEdit1.AllowEmptyValue = true;
```

X++ (Dynamics Ax 2009)

```
public void init()
{
  ;

  super();

  exmaskedit1.SelectGotFocus(4/*exSelectRequiredEditableGotFocus*/);
  exmaskedit1.Mask("!99/99/9999;; ;select=4,overtype");
  exmaskedit1.Text(str2Date("1/2/2001",213));
  exmaskedit1.Warning("Invalid character!");
  exmaskedit1.Invalid("Invalid date!");
  exmaskedit1.ValidateAs(1/*exValidateAsDate*/);
  exmaskedit1.AllowEmptyValue(true);
}
```

Delphi 8 (.NET only)

```
with AxMaskEdit1 do
begin
  SelectGotFocus :=
EXMASKEDITLib.SelectGotFocusEnum.exSelectRequiredEditableGotFocus;
  Mask := '!99/99/9999;; ;select=4,overtype';
  Text := '1/2/2001';
  Warning := 'Invalid character!';
  Invalid := 'Invalid date!';
  ValidateAs := EXMASKEDITLib.ValidateAsEnum.exValidateAsDate;
  AllowEmptyValue := True;
end
```

Delphi (standard)

```
with MaskEdit1 do
begin
  SelectGotFocus := EXMASKEDITLib_TLB.exSelectRequiredEditableGotFocus;
  Mask := '!99/99/9999;; ;select=4,overtype';
  Text := '1/2/2001';
  Warning := 'Invalid character!';
  Invalid := 'Invalid date!';
```

```
    ValidateAs := EXMASKEDITLib_TLB.exValidateAsDate;
    AllowEmptyValue := True;
end
```

VFP

```
with thisform.MaskEdit1
    .SelectGotFocus = 4
    .Mask = "!99/99/9999;; ;select=4,overtype"
    .Text = {^2001-1-2}
    .Warning = "Invalid character!"
    .Invalid = "Invalid date!"
    .ValidateAs = 1
    .AllowEmptyValue = .T.
endwith
```

dBASE Plus

```
local oMaskEdit

oMaskEdit = form.Activex1.nativeObject
oMaskEdit.SelectGotFocus = 4
oMaskEdit.Mask = "!99/99/9999;; ;select=4,overtype"
oMaskEdit.Text = Str("01/02/2001")
oMaskEdit.Warning = "Invalid character!"
oMaskEdit.Invalid = "Invalid date!"
oMaskEdit.ValidateAs = 1
oMaskEdit.AllowEmptyValue = true
```

XBasic (Alpha Five)

```
Dim oMaskEdit as P

oMaskEdit = topparent:CONTROL_ACTIVEX1.activex
oMaskEdit.SelectGotFocus = 4
oMaskEdit.Mask = "!99/99/9999;; ;select=4,overtype"
oMaskEdit.Text = {01/02/2001}
```

```
oMaskEdit.Warning = "Invalid character!"
oMaskEdit.Invalid = "Invalid date!"
oMaskEdit.ValidateAs = 1
oMaskEdit.AllowEmptyValue = .t.
```

Visual Objects

```
oDCOCX_Exontrol1:SelectGotFocus := exSelectRequiredEditableGotFocus
oDCOCX_Exontrol1:Mask := "!99/99/9999;; ;select=4,overtype"
oDCOCX_Exontrol1:Text := AsString(SToD("20010102"))
oDCOCX_Exontrol1:Warning := "Invalid character!"
oDCOCX_Exontrol1:Invalid := "Invalid date!"
oDCOCX_Exontrol1:ValidateAs := exValidateAsDate
oDCOCX_Exontrol1:AllowEmptyValue := true
```

PowerBuilder

```
OleObject oMaskEdit

oMaskEdit = ole_1.Object
oMaskEdit.SelectGotFocus = 4
oMaskEdit.Mask = "!99/99/9999;; ;select=4,overtype"
oMaskEdit.Text = String(2001-01-02)
oMaskEdit.Warning = "Invalid character!"
oMaskEdit.Invalid = "Invalid date!"
oMaskEdit.ValidateAs = 1
oMaskEdit.AllowEmptyValue = true
```

Visual DataFlex

```
Procedure OnCreate
    Forward Send OnCreate
    Set ComSelectGotFocus to OLEexSelectRequiredEditableGotFocus
    Set ComMask to "!99/99/9999;; ;select=4,overtype"
```

```
    Set ComText to "1/2/2001"
    Set ComWarning to "Invalid character!"
    Set ComInvalid to "Invalid date!"
    Set ComValidateAs to OLEexValidateAsDate
    Set ComAllowEmptyValue to True
End_Procedure
```

property MaskEdit.Value ([ClipModeIncludeLiteral as Variant]) as String

Retrieves the control's value with or without literals.

Type	Description
ClipModeIncludeLiteral as Variant	A ClipModeIncludeLiteralEnum expression that specifies the way the control's text is retrieved.
String	A String expression that defines the control's value base on the ClipModeIncludeLiteral parameter.

The Value property indicates the control's text or value. The Mask property specifies the control's mask. Use the Text property to change programmatically the control's text. The TextIncludeLiterals property specifies the way the Text property get/set the control's text. For instance, if you do not need to get the placeholders characters in the Text property, set the TextIncludeLiterals property on exClipModeLiteralsInclude before calling the Text property. The Value property is equivalent with Text/TextIncludeLiterals property. The control fires the Change event if the user changes the control's text. The TextIncludeLiteralsLoseFocus property specifies how the control's field is shown when it loses the focus. The AllowEmptyValue property specifies whether the control handles empty values.

property MaskEdit.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. The Version property is read-only.

property MaskEdit.VisibleMasks as Long

Retrieves or sets a value that indicates the number of visible items in the control masks list.

Type	Description
Long	A long expression that indicates the number of visible items in the control masks list.

The VisibleMasks property indicates the number of visible items in the control masks list. Use the Masks property to assign multiple masks to the control.
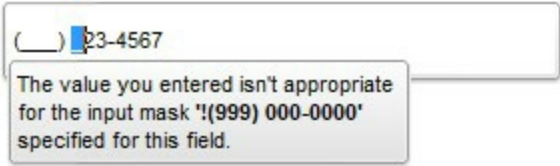
property MaskEdit.VisualAppearance as Appearance
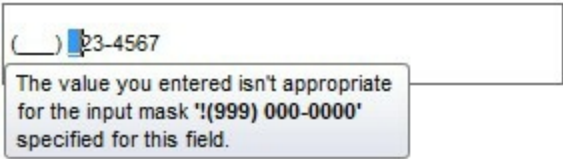
Retrieves the control's appearance.

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 apply a skin to a part of the control. Use the Appearance property to change the visual appearance of the control's frame.

The following screen shot shows the control's with visual appearance changed:

The following screen shot shows the control's with no visual appearance changed:
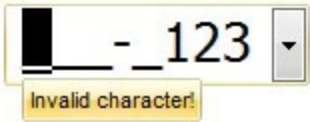
property MaskEdit.Warning as String

Indicates the html-tooltip message to be shown when the user enters an invalid character.

Type	Description
String	A String expression that defines the built-in HTML tooltip to be shown at cursor position when user enters any invalid character.

By default, the Warning property is "". Use the Warning property to specify a HTML tooltip to be shown at the cursor position when user enters an invalid character. The Warning property has no effect if the property is empty or the control is not masked. Use the Invalid property to specify a HTML tooltip to keep the control focused while the user enters an inappropriate value for the field. Use the AllowBeep property to let control beeps once the user enters any invalid character.

The following screen shot shows the Warning once the user enters an invalid character:

The control uses built-in HTML tags to display the caption using HTML format. The control supports the following HTML tags:

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

with a specified <mark>background</mark> 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](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 ₛᵤᵦₛ꜀ᵣᵢₚₜ 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

ExMaskEdit events

The ExMaskEdit control supports the following events:

Name	Description
Change	Fired while the user changes the control's text.
Click	Occurs when the user presses and then releases the left mouse button over the control.
DblClick	Occurs when the user dblclk the left mouse button over an object.
Event	Notifies the application once the control fires an event.
KeyDown	Occurs when the user presses a key while an object has the focus.
KeyPress	Occurs when the user presses and releases an ANSI key.
KeyUp	Occurs when the user releases a key while an object has the focus.
MaskChange	Occurs when the user changes the control's mask.
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.
RClick	Occurs once the user right clicks the control.
ValidateValue	Validates the field's value, once the user leaves the field.

event Change ()

Fired while the user changes the control's text.

Type	Description

Indicates that the control's contents is changed. Occurs when a DDE link updates data, when a user changes the mask, or when you change the [Text](#) property setting through code. The Change event procedure can synchronize or coordinate data display among controls. The [MaskChange](#) property is fired once the user changes the control's [Mask](#) property.

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;
```

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

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

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

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

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

```
LPARAMETERS nop
```

```
PROCEDURE OnChange(oMaskEdit)
RETURN
```

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

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

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

```
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

Here's a VB6 sample that handles the Change event:

```
Private Sub MaskEdit1_Change()
    Debug.Print "The Change event was fired. The new control's mask value is: " &
MaskEdit1.Text()
End Sub
```

event Click ()

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

Type	Description

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

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;``` |

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

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

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

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

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

```
LPARAMETERS nop
```

```
PROCEDURE OnClick(oMaskEdit)
RETURN
```

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

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

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

```
Procedure OnComClick
    Forward Send OnComClick
```

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 dblclk the left mouse button over an object.

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

By default, the control selects the entity from the cursor when the user double clicks the control. The DblClick event notifies your application once the user double clicks the control. The [Click](#) event notifies your application once the user clicks the control's content. The [RClick](#) event notifies once the user right clicks the control.

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

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

```vb
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:

```csharp
C#    private void DblClick(object sender,
      AxEXMASKEDITLib._IMaskEditEvents_DblClickEvent e)
      {
      }
```

```cpp
C++   void OnDblClick(short Shift,long X,long Y)
      {
      }
```

| C++ Builder | ```cpp
void __fastcall DblClick(TObject *Sender,short Shift,int X,int Y)
{
}
``` |
|---|---|

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

| Delphi 8 (.NET only) | ```pascal
procedure DblClick(sender: System.Object; e:
AxEXMASKEDITLib._IMaskEditEvents_DblClickEvent);
begin
end;
``` |
|---|---|

| Powe… | ```
begin event DblClick(integer Shift,long X,long Y)
end event DblClick
``` |
|---|---|

| VB.NET | ```vbnet
Private Sub DblClick(ByVal sender As System.Object, ByVal e As
AxEXMASKEDITLib._IMaskEditEvents_DblClickEvent) Handles DblClick
End Sub
``` |
|---|---|

| VB6 | ```vb
Private Sub DblClick(Shift As Integer,X As Single,Y As Single)
End Sub
``` |
|---|---|

| VBA | ```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(oMaskEdit,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">
``` |
|---|---|

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.MaskEdit.1::OLE_XPOS_PIXELS,Y as
OLE::Exontrol.MaskEdit.1::OLE_YPOS_PIXELS)
end function
```

dBASE

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

event Event (EventID as Long)

Notifies the application once the control fires an event.

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

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

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 exmaskedit1.EventParam(-2).toString();
}
```

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

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

```
"AfterDrawPart/55( 2 , -1962866148 , 0 , 0 , 0 , 0 )" VT_BSTR
"MouseMove/-606( 1 , 0 , 145 , 35 )" VT_BSTR
```

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

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

```
// Notifies the application once the control fires an event.
void onEvent_Event(int _EventID)
{
    ;
    if ( _EventID == 125 ) /*event BarParentChange (Item as HITEM, Key as Variant, NewItem
as HITEM, Cancel as Boolean) */
        exmaskedit1.EventParam( 3 /*Cancel*/, COMVariant::createFromBoolean(true) );
}
```

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

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

```
// Notifies the application once the control fires an event.
void onEvent_Event(int _EventID)
{
    ;
    if ( _EventID == 125 ) /*event BarParentChange (Item as HITEM, Key as Variant, NewItem
as HITEM, Cancel as Boolean) */
        if ( !exmaskedit1.Items().EnableItem( exmaskedit1.EventParam( 2 /*NewItem*/ ) ) )
            exmaskedit1.EventParam( 3 /*Cancel*/, COMVariant::createFromBoolean(true) );
}
```

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

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, AxEXMASKEDITLib._IMaskEditEvents_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: AxEXMASKEDITLib._IMaskEditEvents_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 AxEXMASKEDITLib._IMaskEditEvents_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(oMaskEdit,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)
{
} |

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

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

event KeyDown (KeyCode as Integer, Shift as Integer)

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

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

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

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

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

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

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

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

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

```csharp
C#

private void KeyDownEvent(object sender,
AxEXMASKEDITLib._IMaskEditEvents_KeyDownEvent e)
{
}
```

```cpp
C++

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

```cpp
C++
Builder

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

```delphi
Delphi

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

```delphi
Delphi 8
(.NET
only)

procedure KeyDownEvent(sender: System.Object; e:
AxEXMASKEDITLib._IMaskEditEvents_KeyDownEvent);
begin
end;
```

```
Powe…

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

```vbnet
VB.NET

Private Sub KeyDownEvent(ByVal sender As System.Object, ByVal e As
AxEXMASKEDITLib._IMaskEditEvents_KeyDownEvent) Handles KeyDownEvent
End Sub
```

```vb
VB6

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

```vba
VBA

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

```
VFP

LPARAMETERS KeyCode,Shift
```

PROCEDURE OnKeyDown(oMaskEdit,KeyCode,Shift)
RETURN

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

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

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

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

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

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

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

dBASE
```
function nativeObject_KeyDown(KeyCode,Shift)
return
```

event KeyPress (KeyAscii as Integer)

Occurs when the user presses and releases an ANSI key.

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

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

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

<div>

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

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

</div>

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

`C#`
```csharp
private void KeyPressEvent(object sender,
AxEXMASKEDITLib._IMaskEditEvents_KeyPressEvent e)
{
}
```

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

`C++ Builder`
```cpp
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:
AxEXMASKEDITLib._IMaskEditEvents_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
AxEXMASKEDITLib._IMaskEditEvents_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(oMaskEdit,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>``` |

```
Procedure OnComKeyPress Short llKeyAscii
    Forward Send OnComKeyPress llKeyAscii
End_Procedure
```

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

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

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

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

event KeyUp (KeyCode as Integer, Shift as Integer)

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

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

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

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

```cs
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:

```cs
private void KeyUpEvent(object sender,
AxEXMASKEDITLib._IMaskEditEvents_KeyUpEvent e)
{
}
```

```cpp
void OnKeyUp(short FAR* KeyCode,short Shift)
{
}
```

```cpp
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:
AxEXMASKEDITLib._IMaskEditEvents_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
AxEXMASKEDITLib._IMaskEditEvents_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(oMaskEdit,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 MaskChange ()

Occurs when the user changes the control's mask.

Type	Description

The MaskChange event occurs if user changes the [Mask](#) property. The MaskChange event occurs when user selects a new mask from the control's masks list. Use the [Masks](#) property to assign multiple masks to the control. The [Change](#) event notifies your application once the control's [Text](#) property is changed (the control's content).

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

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

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

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

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

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

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

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

Delphi 8 (.NET only)
```
procedure MaskChange(sender: System.Object; e: System.EventArgs);
begin
end;
```

Powe…	begin event MaskChange() end event MaskChange

VB.NET	Private Sub MaskChange(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MaskChange End Sub

VB6	Private Sub MaskChange() End Sub

VBA	Private Sub MaskChange() End Sub

VFP	LPARAMETERS nop

Xbas…	PROCEDURE OnMaskChange(oMaskEdit) RETURN

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

Java…	<SCRIPT EVENT="MaskChange()" LANGUAGE="JScript"> </SCRIPT>

VBSc…	<SCRIPT LANGUAGE="VBScript"> Function MaskChange() End Function </SCRIPT>

Visual Data…	Procedure OnComMaskChange Forward Send OnComMaskChange End_Procedure

Visual Objects	METHOD OCX_MaskChange() CLASS MainDialog RETURN NIL

X++	void onEvent_MaskChange()

```
    {
    }
```

function MaskChange as v ()
end function

function nativeObject_MaskChange()
return

The following sample displays the current control's mask when the user changes the mask from the control's masks list:

```
Private Sub Form_Load()
    With MaskEdit1
        .Masks = "USA resident;(090)-###-###-###;Canada resident;(091)-###-###-###"
    End With
End Sub


Private Sub MaskEdit1_MaskChange()
    Debug.Print "The current mask is: " & MaskEdit1.Mask
End Sub
```

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](#) event, MouseDown and MouseUp events lets you distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers.

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

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

```vb
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:

```csharp
C#
private void MouseDownEvent(object sender,
AxEXMASKEDITLib._IMaskEditEvents_MouseDownEvent e)
{
```

```
      }
```

C++

```cpp
void OnMouseDown(short Button,short Shift,long X,long Y)
{
}
```

C++ Builder

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

Delphi

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

Delphi 8 (.NET only)

```delphi
procedure MouseDownEvent(sender: System.Object; e:
AxEXMASKEDITLib._IMaskEditEvents_MouseDownEvent);
begin
end;
```

Powe…

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

VB.NET

```vbnet
Private Sub MouseDownEvent(ByVal sender As System.Object, ByVal e As
AxEXMASKEDITLib._IMaskEditEvents_MouseDownEvent) Handles
MouseDownEvent
End Sub
```

VB6

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

VBA

```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
```

```
PROCEDURE OnMouseDown(oMaskEdit,Button,Shift,X,Y)
RETURN
```

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

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

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

```
Procedure OnComMouseDown Short llButton Short llShift OLE_XPOS_PIXELS llX
OLE_YPOS_PIXELS llY
    Forward Send OnComMouseDown llButton llShift llX llY
End_Procedure
```

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

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

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

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

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

Occurs when the user moves the mouse over the control

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

The MouseMove event is generated continually as the mouse pointer moves across objects. Unless another object has captured the mouse, an object recognizes a MouseMove event whenever the mouse position is within its borders.

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

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

```vb
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:

```cs
C#    private void MouseMoveEvent(object sender,
      AxEXMASKEDITLib._IMaskEditEvents_MouseMoveEvent e)
      {
      }
```

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

```
{
}
```

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

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

```
procedure MouseMoveEvent(sender: System.Object; e:
AxEXMASKEDITLib._IMaskEditEvents_MouseMoveEvent);
begin
end;
```

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

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

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

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

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

```
PROCEDURE OnMouseMove(oMaskEdit,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.MaskEdit.1::OLE_XPOS_PIXELS,Y as OLE::Exontrol.MaskEdit.1::OLE_YPOS_PIXELS)` `end function`

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

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

Occurs when the user releases a mouse button.

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

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

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,
AxEXMASKEDITLib._IMaskEditEvents_MouseUpEvent e)
{
```

```
}
```

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

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

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

Delphi 8 (.NET only)
```delphi
procedure MouseUpEvent(sender: System.Object; e:
AxEXMASKEDITLib._IMaskEditEvents_MouseUpEvent);
begin
end;
```

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

VB.NET
```vbnet
Private Sub MouseUpEvent(ByVal sender As System.Object, ByVal e As
AxEXMASKEDITLib._IMaskEditEvents_MouseUpEvent) Handles MouseUpEvent
End Sub
```

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

VBA
```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(oMaskEdit,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.MaskEdit.1::OLE_XPOS_PIXELS,Y as`
`OLE::Exontrol.MaskEdit.1::OLE_YPOS_PIXELS)`
`end function` |

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

event RClick ()

Occurs once the user right clicks the control.

Type	Description

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

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

<table>
<tr><td>C#</td><td>

```
private void RClick(object sender)
{
}
```

</td></tr>
</table>

<table>
<tr><td>VB</td><td>

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

</td></tr>
</table>

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

<table>
<tr><td>C#</td><td>

```
private void RClick(object sender, EventArgs e)
{
}
```

</td></tr>
</table>

<table>
<tr><td>C++</td><td>

```
void OnRClick()
{
}
```

</td></tr>
</table>

<table>
<tr><td>C++
Builder</td><td>

```
void __fastcall RClick(TObject *Sender)
{
}
```

</td></tr>
</table>

<table>
<tr><td>Delphi</td><td>

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

</td></tr>
</table>

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

Powe…
```
begin event RClick()
end event RClick
```

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

VB6
```
Private Sub RClick()
End Sub
```

VBA
```
Private Sub RClick()
End Sub
```

VFP
```
LPARAMETERS nop
```

Xbas…
```
PROCEDURE OnRClick(oMaskEdit)
RETURN
```

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

Java…
```
<SCRIPT EVENT="RClick()" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…
```
<SCRIPT LANGUAGE="VBScript">
Function RClick()
End Function
</SCRIPT>
```

Visual
Data…
```
Procedure OnComRClick
    Forward Send OnComRClick
```

End_Procedure

METHOD OCX_RClick() CLASS MainDialog
RETURN NIL

```
void onEvent_RClick()
{
}
```

```
function RClick as v ()
end function
```

```
function nativeObject_RClick()
return
```

event ValidateValue (NewValue as Variant, Cancel as Boolean)

Validates the field's value, once the user leaves the field.

Type	Description
NewValue as Variant	A Text expression that specifies the control's text to be validated. The NewValue parameter carries the same value as Value(exClipModeLiteralsInclude)
Cancel as Boolean	A Boolean expression that specifies whether the value is validated or canceled. If True, the Value is not validated, so the field receives again the focus, if False, the user can leave the field.

The ValidateValue event is fired once the user leaves the field and the current value needs to be validated. The ValidateValue event could be user to validate programmatically the value your user enters in the field. The Invalid property specifies the html message to be displayed when the user enters an inappropriate value for the field. For instance, if the user fills the field's content properly, the ValidateValue event gives you a chance to validate more the value, if for instance, it does not fit your requirements. The Valid property specifies whether the field is completed and valid (all required entities are completed, and the ValidateAs option is verified). Use the Text property to gives the control's text as it is displayed.

The ValidateValue event is fired only if

- Invalid property is not empty, indicates the html message to be displayed when the user enters an inappropriate value for the field. If the value is missing or empty, the option has no effect, so no validation is performed. If the value is a not-empty value, the validation is performed. If the value is single space, no message is displayed and the field is keep opened while the value is inappropriate. For instance, "!(999) 000 0000;;;invalid=The value you entered isn't appropriate for the input mask '<%mask%>' specified for this field." displays the "The value you entered isn't appropriate for the input mask '...' specified for this field." tooltip once the user leaves the field and it is not-valid (for instance, the field includes entities required and uncompleted). The <%mask%> keyword in value, substitute the current mask of the field, while the <%value%> keyword substitutes the current value (including the literals). If this option should display/use the semicolon (;) character is should be \; (escape). This option can be combined with empty, validateas.
- the Mask property is **not** empty (the first part in the mask or the pattern). In other words, the pattern is not empty. The pattern includes the mask characters or string (series of characters) along with placeholders and literal data such as, parentheses, periods, and hyphens.
- the ReadOnly is False (by default), the editor is locked, user can not update the

content, the caret is available, so user can copy the text, excepts the password fields.

The [Value](#) property gives the field's content in different formats.

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

C#	private void ValidateValue(object sender,object NewValue,ref bool Cancel)
	{
	}

VB	Private Sub ValidateValue(ByVal sender As System.Object,ByVal NewValue As Object,ByRef Cancel As Boolean) Handles ValidateValue End Sub

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

C#	private void ValidateValue(object sender,
	AxEXMASKEDITLib._IMaskEditEvents_ValidateValueEvent e)
	{
	}

C++	void OnValidateValue(VARIANT NewValue,BOOL FAR* Cancel)
	{
	}

C++
Builder
	void __fastcall ValidateValue(TObject *Sender,Variant NewValue,VARIANT_BOOL * Cancel)
	{
	}

Delphi	procedure ValidateValue(ASender: TObject; NewValue : OleVariant;var Cancel : WordBool);
	begin
	end;

Delphi 8
(.NET
only)
	procedure ValidateValue(sender: System.Object; e: AxEXMASKEDITLib._IMaskEditEvents_ValidateValueEvent);
	begin
	end;

| Powe… | begin event ValidateValue(any NewValue,boolean Cancel)
end event ValidateValue |

| VB.NET | Private Sub ValidateValue(ByVal sender As System.Object, ByVal e As AxEXMASKEDITLib._IMaskEditEvents_ValidateValueEvent) Handles ValidateValue
End Sub |

| VB6 | Private Sub ValidateValue(ByVal NewValue As Variant,Cancel As Boolean)
End Sub |

| VBA | Private Sub ValidateValue(ByVal NewValue As Variant,Cancel As Boolean)
End Sub |

| VFP | LPARAMETERS NewValue,Cancel |

| Xbas… | PROCEDURE OnValidateValue(oMaskEdit,NewValue,Cancel)
RETURN |

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

| Java… | ```
<SCRIPT EVENT="ValidateValue(NewValue,Cancel)" LANGUAGE="JScript">
</SCRIPT>
``` |

| VBSc… | ```
<SCRIPT LANGUAGE="VBScript">
Function ValidateValue(NewValue,Cancel)
End Function
</SCRIPT>
``` |

| Visual Data… | ```
Procedure OnComValidateValue Variant llNewValue Boolean llCancel
    Forward Send OnComValidateValue llNewValue llCancel
End_Procedure
``` |

| Visual Objects | ```
METHOD OCX_ValidateValue(NewValue,Cancel) CLASS MainDialog
RETURN NIL
``` |

| X++ | ```
void onEvent_ValidateValue(COMVariant _NewValue,COMVariant /*bool*/ _Cancel)
{
``` |

```
      }
```

function ValidateValue as v (NewValue as A,Cancel as L)
end function

function nativeObject_ValidateValue(NewValue,Cancel)
return