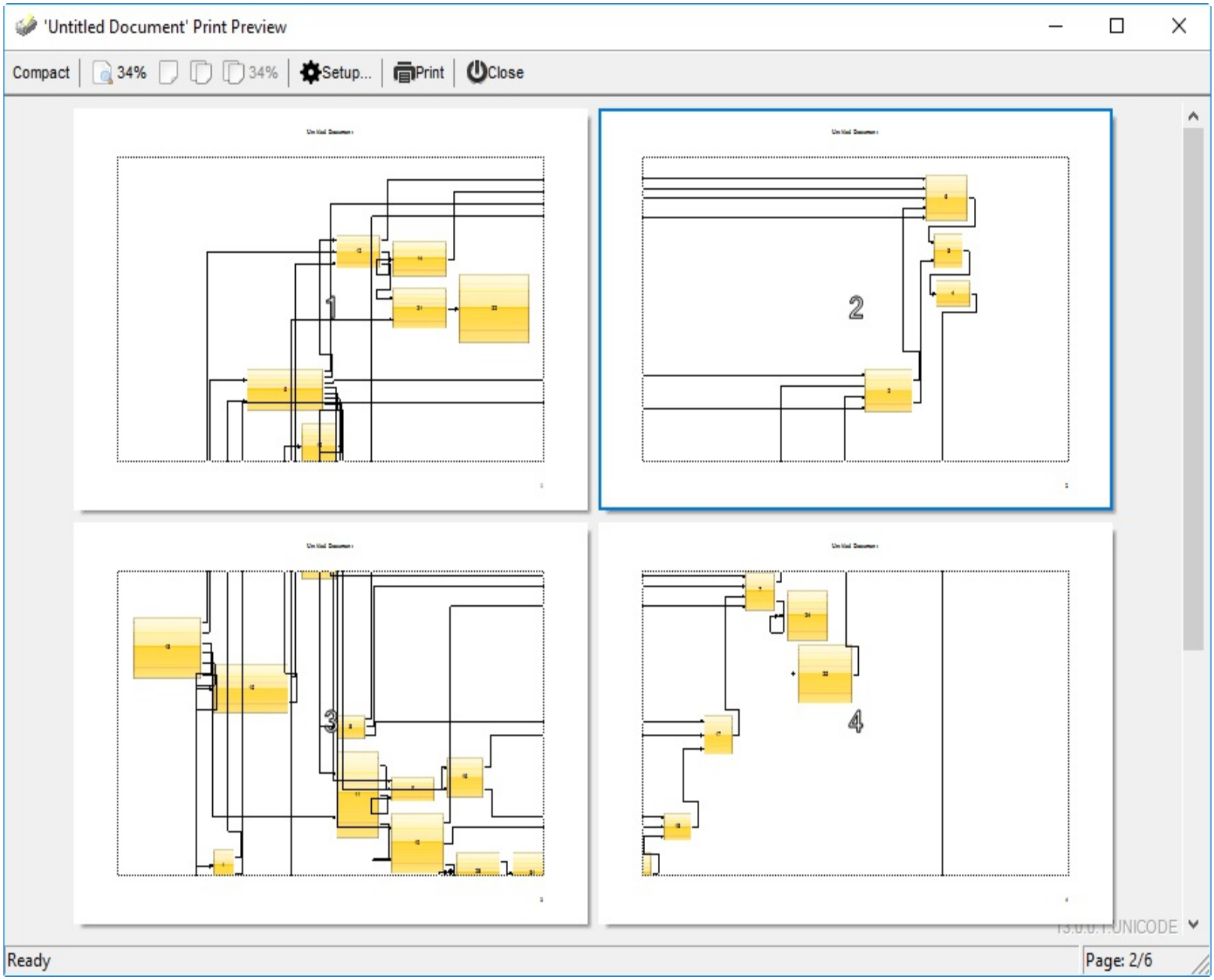
ExPrint

The Exontrol ExPrint component is an advanced printing system specifically to bring your User Interface to the printed page. The ExPrint adds powerful print and preview capabilities to your forms, dialog boxes or other windows. The ExPrint component provides Print and Print Preview features for components like: exG2antt, exGantt, exGrid, exTree, exList, exCalendar, exComboBox, exPropertiesList, exFileView, exEdit, exOrgChart , exSchedule, exPivot, eXSurface, eXSwimLane, eXHTML and so on. (the IPrintExt interface is automatically implemented by these components, and you don't need to implement it). The ExPrint component provides the ability to print the entire document, selected pages or user selected area.

If you want to use the ExPrint/COM component as a separate component, you need to implement the IPrintExt interface. The ExPrint/NET component is installed by most of our .NET assemblies. The eXPrint/COM provides Print and Print Preview for /COM controls, since the /NET assembly provides the Print and Print Preview for /NET and /WPF assemblies.



How to get support?

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

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

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

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

Regards,
Exontrol Development Team

https://www.exontrol.com

constants AlignmentEnum

Specifies the object's alignment. Specifies how the caption of the document is aligned on the printed page. Specifies how the page numbers filed is aligned on the printed page.

Name	Value	Description
exLeft	0	The field is aligned to the left of the printed page.
exCenter	1	The field is centered in the printed page.
exRight	2	The field is aligned to the right of the printed page.
exInside	3	The field is aligned to the left of an odd page, and it is aligned to the right of the even page.
exOutside	4	The field is aligned to the right of an even page, and it is aligned to the right of the odd page.

constants ContentAlignmentEnum

The ContentAlignmentEnum type specifies the alignment of the page numbers in the pages (PreviewSettings(exPreviewShowPageNumbers)). The ContentAlignmentEnum type supports the following values:

Name	Value	Description
exTopLeft	0	Content is vertically aligned at the top, and horizontally aligned on the left.
exTopCenter	1	Content is vertically aligned at the top, and horizontally aligned at the center.
exTopRight	2	Content is vertically aligned at the top, and horizontally aligned on the right.
exMiddleLeft	16	Content is vertically aligned in the middle, and horizontally aligned on the left.
exMiddleCenter	17	Content is vertically aligned in the middle, and horizontally aligned at the center.
exMiddleRight	18	Content is vertically aligned in the middle, and horizontally aligned on the right.
exBottomLeft	32	Content is vertically aligned at the bottom, and horizontally aligned on the left.
exBottomCenter	33	Content is vertically aligned at the bottom, and horizontally aligned at the center.
exBottomRight	34	Content is vertically aligned at the bottom, and horizontally aligned on the right.

constants FieldsEnum

The FieldsEnum type defines the list of fields that user can change before previewing or printing an object. Use the Settings property to access the printer field's value. Use the Preview method to preview an object. Use the DoPrint method to print an object.

Name	Value	Description
exPaperSize	0	Selects the size of the paper to print on. The exPaperSize member can be one the bellow predefined values. For instance, the .Settings(exPaperSize) = 9, indicates an A4 page.
exPaperSource	1	Specifies the paper source. The member supports one of the following values: DMBIN_AUTO (7), DMBIN_CASSETTE (14), DMBIN_ENVELOPE (5), DMBIN_ENVMANUAL (6), DMBIN_FORMSOURCE (15), DMBIN_LARGECAPACITY (11), DMBIN_LARGEFMT (10), DMBIN_LOWER (2), DMBIN_MANUAL (4), DMBIN_MIDDLE (3), DMBIN_ONLYONE (1), DMBIN_TRACTOR (8), DMBIN_SMALLFMT (9)
exPageOrientation	2	Selects the orientation of the paper. This member can be either DMORIENT_PORTRAIT (1) or DMORIENT_LANDSCAPE (2). It
exPrinterName	3	Specifies the "friendly" name of the printer or display; for example, "PCL/HP LaserJet" in the case of PCL/HP LaserJetŽ. This string is unique among device drivers. Use the Printers property to retrieve the list of installed printers. If changing, the exPrinterName must be called before setting any other option for the printer.
exPrinterCopies	4	Selects the number of copies printed if the device supports multiple-page copies.
exPrintQuality	5	Specifies the printer resolution. There are four predefined device-independent values: DMRES_HIGH (-4), DMRES_MEDIUM (-3), DMRES_LOW (-2), DMRES_DRAFT (-1). If a positive value is specified, it specifies the number of dots per inch (DPI) and is therefore device dependent.
		Specifies the width of the left margin. The thousandths of inches are the unit of measurement

exLeftMargin	6	for margins and paper size, if exDisplayInch property is 0, else the measurement are in millimeters. For instance, the .Settings(exLeftMargin) = 1500, indicates 1.5 inch.
exTopMargin	7	Specifies the height of the top margin. The thousandths of inches are the unit of measurement for margins and paper size, if exDisplayInch property is 0, else the measurement are in millimeters. For instance, the .Settings(exTopMargin) = 1500, indicates 1.5 inch.
exRightMargin	8	Specifies the width of the right margin. The thousandths of inches are the unit of measurement for margins and paper size, if exDisplayInch property is 0, else the measurement are in millimeters. For instance, the .Settings(exRightMargin) = 1500, indicates 1.5 inch.
exBottomMargin	9	Specifies the height of the bottom margin. The thousandths of inches are the unit of measurement for margins and paper size, if exDisplayInch property is 0, else the measurement are in millimeters. For instance, the .Settings(exBottomMargin) = 1500, indicates 1.5 inch.
exDisplayInch	10	Specifies whether the print dialog prints inches or millimeters. The exDisplayInch setting MUST be called before any other settings, else it will not have any effect. By default, the exDisplayInch property is 0 that means that the measurements are done using inches, else the measurement are done using millimeters. Valid values are 0 (Inches), 1 (Millimeters).
exFormName	11	Specifies the name of the form.
exPaperWidth	12	Indicates the width of the paper. Use the ClientWidth property to determine the width in pixels, of the drawing area of the print page.
exPaperHeight	13	Indicates the height of the paper. Use the ClientHeight property to determine the width in pixels, of the drawing area of the print page.
exAllFields	256	Gets or sets the values for all fields. Use the Settings(exAllFields) property to save and restore the printer settings.

The exPaperSize field supports one of the following predefined values:

```
DMPAPER_LETTER          1  /* Letter 8 1/2 x 11 in          */
DMPAPER_LETTERSMALL     2  /* Letter Small 8 1/2 x 11 in    */
DMPAPER_TABLOID         3  /* Tabloid 11 x 17 in            */
DMPAPER_LEDGER          4  /* Ledger 17 x 11 in             */
DMPAPER_LEGAL           5  /* Legal 8 1/2 x 14 in           */
DMPAPER_STATEMENT       6  /* Statement 5 1/2 x 8 1/2 in    */
DMPAPER_EXECUTIVE       7  /* Executive 7 1/4 x 10 1/2 in   */
DMPAPER_A3              8  /* A3 297 x 420 mm               */
DMPAPER_A4              9  /* A4 210 x 297 mm               */
DMPAPER_A4SMALL        10  /* A4 Small 210 x 297 mm         */
DMPAPER_A5             11  /* A5 148 x 210 mm               */
DMPAPER_B4             12  /* B4 (JIS) 250 x 354            */
DMPAPER_B5             13  /* B5 (JIS) 182 x 257 mm         */
DMPAPER_FOLIO          14  /* Folio 8 1/2 x 13 in           */
DMPAPER_QUARTO         15  /* Quarto 215 x 275 mm           */
DMPAPER_10X14          16  /* 10x14 in                      */
DMPAPER_11X17          17  /* 11x17 in                      */
DMPAPER_NOTE           18  /* Note 8 1/2 x 11 in            */
DMPAPER_ENV_9          19  /* Envelope #9 3 7/8 x 8 7/8     */
DMPAPER_ENV_10         20  /* Envelope #10 4 1/8 x 9 1/2    */
DMPAPER_ENV_11         21  /* Envelope #11 4 1/2 x 10 3/8   */
DMPAPER_ENV_12         22  /* Envelope #12 4 \276 x 11      */
DMPAPER_ENV_14         23  /* Envelope #14 5 x 11 1/2       */
DMPAPER_CSHEET         24  /* C size sheet                  */
DMPAPER_DSHEET         25  /* D size sheet                  */
DMPAPER_ESHEET         26  /* E size sheet                  */
DMPAPER_ENV_DL         27  /* Envelope DL 110 x 220mm       */
DMPAPER_ENV_C5         28  /* Envelope C5 162 x 229 mm      */
DMPAPER_ENV_C3         29  /* Envelope C3  324 x 458 mm     */
DMPAPER_ENV_C4         30  /* Envelope C4  229 x 324 mm     */
DMPAPER_ENV_C6         31  /* Envelope C6  114 x 162 mm     */
DMPAPER_ENV_C65        32  /* Envelope C65 114 x 229 mm     */
DMPAPER_ENV_B4         33  /* Envelope B4  250 x 353 mm     */
DMPAPER_ENV_B5         34  /* Envelope B5  176 x 250 mm     */
DMPAPER_ENV_B6         35  /* Envelope B6  176 x 125 mm     */
```

```
DMPAPER_ENV_ITALY          36  /* Envelope 110 x 230 mm            */
DMPAPER_ENV_MONARCH        37  /* Envelope Monarch 3.875 x 7.5 in    */
DMPAPER_ENV_PERSONAL       38  /* 6 3/4 Envelope 3 5/8 x 6 1/2 in    */
DMPAPER_FANFOLD_US         39  /* US Std Fanfold 14 7/8 x 11 in      */
DMPAPER_FANFOLD_STD_GERMAN  40  /* German Std Fanfold 8 1/2 x 12 in   */
DMPAPER_FANFOLD_LGL_GERMAN  41  /* German Legal Fanfold 8 1/2 x 13 in */
DMPAPER_ISO_B4             42  /* B4 (ISO) 250 x 353 mm            */
DMPAPER_JAPANESE_POSTCARD   43  /* Japanese Postcard 100 x 148 mm     */
DMPAPER_9X11               44  /* 9 x 11 in                        */
DMPAPER_10X11              45  /* 10 x 11 in                       */
DMPAPER_15X11              46  /* 15 x 11 in                       */
DMPAPER_ENV_INVITE         47  /* Envelope Invite 220 x 220 mm       */
DMPAPER_RESERVED_48        48  /* RESERVED--DO NOT USE             */
DMPAPER_RESERVED_49        49  /* RESERVED--DO NOT USE             */
DMPAPER_LETTER_EXTRA       50  /* Letter Extra 9 \275 x 12 in        */
DMPAPER_LEGAL_EXTRA        51  /* Legal Extra 9 \275 x 15 in         */
DMPAPER_TABLOID_EXTRA      52  /* Tabloid Extra 11.69 x 18 in        */
DMPAPER_A4_EXTRA           53  /* A4 Extra 9.27 x 12.69 in           */
DMPAPER_LETTER_TRANSVERSE   54  /* Letter Transverse 8 \275 x 11 in   */
DMPAPER_A4_TRANSVERSE      55  /* A4 Transverse 210 x 297 mm         */
DMPAPER_LETTER_EXTRA_TRANSVERSE 56 /* Letter Extra Transverse 9\275 x 12 in */
DMPAPER_A_PLUS             57  /* SuperA/SuperA/A4 227 x 356 mm      */
DMPAPER_B_PLUS             58  /* SuperB/SuperB/A3 305 x 487 mm      */
DMPAPER_LETTER_PLUS        59  /* Letter Plus 8.5 x 12.69 in         */
DMPAPER_A4_PLUS            60  /* A4 Plus 210 x 330 mm             */
DMPAPER_A5_TRANSVERSE      61  /* A5 Transverse 148 x 210 mm         */
DMPAPER_B5_TRANSVERSE      62  /* B5 (JIS) Transverse 182 x 257 mm   */
DMPAPER_A3_EXTRA           63  /* A3 Extra 322 x 445 mm            */
DMPAPER_A5_EXTRA           64  /* A5 Extra 174 x 235 mm            */
DMPAPER_B5_EXTRA           65  /* B5 (ISO) Extra 201 x 276 mm        */
DMPAPER_A2                 66  /* A2 420 x 594 mm                  */
DMPAPER_A3_TRANSVERSE      67  /* A3 Transverse 297 x 420 mm         */
DMPAPER_A3_EXTRA_TRANSVERSE 68  /* A3 Extra Transverse 322 x 445 mm   */
DMPAPER_DBL_JAPANESE_POSTCARD 69 /* Japanese Double Postcard 200 x 148 mm
*/
DMPAPER_A6                 70  /* A6 105 x 148 mm                  */
DMPAPER_JENV_KAKU2         71  /* Japanese Envelope Kaku #2        */
```

```
DMPAPER_JENV_KAKU3          72  /* Japanese Envelope Kaku #3       */
DMPAPER_JENV_CHOU3          73  /* Japanese Envelope Chou #3       */
DMPAPER_JENV_CHOU4          74  /* Japanese Envelope Chou #4       */
DMPAPER_LETTER_ROTATED      75  /* Letter Rotated 11 x 8 1/2 11 in */
DMPAPER_A3_ROTATED          76  /* A3 Rotated 420 x 297 mm         */
DMPAPER_A4_ROTATED          77  /* A4 Rotated 297 x 210 mm         */
DMPAPER_A5_ROTATED          78  /* A5 Rotated 210 x 148 mm         */
DMPAPER_B4_JIS_ROTATED      79  /* B4 (JIS) Rotated 364 x 257 mm   */
DMPAPER_B5_JIS_ROTATED      80  /* B5 (JIS) Rotated 257 x 182 mm   */
DMPAPER_JAPANESE_POSTCARD_ROTATED 81 /* Japanese Postcard Rotated 148 x
100 mm */
DMPAPER_DBL_JAPANESE_POSTCARD_ROTATED 82 /* Double Japanese Postcard
Rotated 148 x 200 mm */
DMPAPER_A6_ROTATED          83  /* A6 Rotated 148 x 105 mm         */
DMPAPER_JENV_KAKU2_ROTATED  84  /* Japanese Envelope Kaku #2 Rotated */
DMPAPER_JENV_KAKU3_ROTATED  85  /* Japanese Envelope Kaku #3 Rotated */
DMPAPER_JENV_CHOU3_ROTATED  86  /* Japanese Envelope Chou #3 Rotated */
DMPAPER_JENV_CHOU4_ROTATED  87  /* Japanese Envelope Chou #4 Rotated */
DMPAPER_B6_JIS              88  /* B6 (JIS) 128 x 182 mm           */
DMPAPER_B6_JIS_ROTATED      89  /* B6 (JIS) Rotated 182 x 128 mm   */
DMPAPER_12X11               90  /* 12 x 11 in                      */
DMPAPER_JENV_YOU4           91  /* Japanese Envelope You #4        */
DMPAPER_JENV_YOU4_ROTATED   92  /* Japanese Envelope You #4 Rotated*/
DMPAPER_P16K                93  /* PRC 16K 146 x 215 mm            */
DMPAPER_P32K                94  /* PRC 32K 97 x 151 mm             */
DMPAPER_P32KBIG             95  /* PRC 32K(Big) 97 x 151 mm        */
DMPAPER_PENV_1              96  /* PRC Envelope #1 102 x 165 mm    */
DMPAPER_PENV_2              97  /* PRC Envelope #2 102 x 176 mm    */
DMPAPER_PENV_3              98  /* PRC Envelope #3 125 x 176 mm    */
DMPAPER_PENV_4              99  /* PRC Envelope #4 110 x 208 mm    */
DMPAPER_PENV_5              100 /* PRC Envelope #5 110 x 220 mm    */
DMPAPER_PENV_6              101 /* PRC Envelope #6 120 x 230 mm    */
DMPAPER_PENV_7              102 /* PRC Envelope #7 160 x 230 mm    */
DMPAPER_PENV_8              103 /* PRC Envelope #8 120 x 309 mm    */
DMPAPER_PENV_9              104 /* PRC Envelope #9 229 x 324 mm    */
DMPAPER_PENV_10             105 /* PRC Envelope #10 324 x 458 mm   */
DMPAPER_P16K_ROTATED        106 /* PRC 16K Rotated                 */
```

```
DMPAPER_P32K_ROTATED        107 /* PRC 32K Rotated              */
DMPAPER_P32KBIG_ROTATED     108 /* PRC 32K(Big) Rotated         */
DMPAPER_PENV_1_ROTATED      109 /* PRC Envelope #1 Rotated 165 x 102 mm */
DMPAPER_PENV_2_ROTATED      110 /* PRC Envelope #2 Rotated 176 x 102 mm */
DMPAPER_PENV_3_ROTATED      111 /* PRC Envelope #3 Rotated 176 x 125 mm */
DMPAPER_PENV_4_ROTATED      112 /* PRC Envelope #4 Rotated 208 x 110 mm */
DMPAPER_PENV_5_ROTATED      113 /* PRC Envelope #5 Rotated 220 x 110 mm */
DMPAPER_PENV_6_ROTATED      114 /* PRC Envelope #6 Rotated 230 x 120 mm */
DMPAPER_PENV_7_ROTATED      115 /* PRC Envelope #7 Rotated 230 x 160 mm */
DMPAPER_PENV_8_ROTATED      116 /* PRC Envelope #8 Rotated 309 x 120 mm */
DMPAPER_PENV_9_ROTATED      117 /* PRC Envelope #9 Rotated 324 x 229 mm */
DMPAPER_PENV_10_ROTATED     118 /* PRC Envelope #10 Rotated 458 x 324 mm */
```

constants PageFrameStyleEnum

The PageFrameStyleEnum type specifies the styles of frame a page can show. The PageFrameStyle property specifies the frame to be shown on printed page.

Name	Value	Description
exNoPageFrame	0	The page shows no frame.
exPageFrameSolid	1	The page shows solid frame.
exPageFrameDot	2	The page shows dotted frame.
exPageFrameDash	3	The page shows dash frame.
exPageFrameDashDot	4	The page shows dash-dot frame.
exPageFrameDashDotDot	5	The page shows dash-dot-dot frame.

constants PageOrientationEnum

Specifies the page's orientation.

Name	Value	Description
exPortrait	1	exPortrait
exLandscape	2	exLandscape

constants PositionEnum

Specifies whether the field is displayed on the header or footer of the page.

Name	Value	Description
exHeader	0	The field is displayed on page's header.
exFooter	1	The field is displayed on page's footer.

constants PreviewFieldsEnum

The PreviewFieldsEnum type defines the settings you can change in the preview workspace. The PreviewSettings property specifies the settings you can change in the preview mode. The PreviewFieldsEnum type supports the following values:

Name	Value	Description
exPreviewShowPageNumbers	0	Shows or hides the page number in page's content. By default, the exPreviewShowPageNumbers property is False. Use the exPreviewPageNumberFormat and exPreviewPageNumbersAlignment properties to define the format and alignment of the page numbers to be shown on the preview. For instance, you can set the exPreviewShowPageNumbers property on True, when viewing the pages in compact mode (exPreviewShowCompact). *(Boolean expression)*
		A String expression that defines the HTTML format to display the page number on pages. By default, the exPreviewPageNumberFormat property is "", which indicates that PageNumberFormat property is used instead. The exPreviewPageNumberFormat property supports the following predefined values: • **<%page%>** specifies the current page • **<%count%>** indicates the number of pages in the document. Also the exPreviewPageNumberFormat property supports the HTML format as: • ** ... ** displays the text in **bold** • ** ... ** displays the text in *italics* • **<u> ... </u>** underlines the text • **<s> ... </s>** ~~Strike~~-through text • **<a id;options> ... ** displays an anchor element that can be clicked. An anchor is a piece of text or some other object (for example an image) which marks the beginning and/or the end of a hypertext link.The <a> element is

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

- ** ... ** displays portions of text with a different font and/or different size. For instance, the "bit" draws the bit text using the Tahoma font, on size 12 pt. If the name of the font is missing, and instead size is present, the current font is used with a different size. For instance, "bit" displays the bit text using the current font, but with a different size.
- **<fgcolor rrggbb> ... </fgcolor>** or <fgcolor=rrggbb> ... </fgcolor> displays text with a specified foreground color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<bgcolor rrggbb> ... </bgcolor>** or <bgcolor=rrggbb> ... </bgcolor> displays text with a specified background color. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<solidline rrggbb> ... </solidline>** or <solidline=rrggbb> ... </solidline> draws a solid-line on the bottom side of the current text-line, of specified RGB color. The <solidline> ... </solidline> draws a black solid-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.
- **<dotline rrggbb> ... </dotline>** or <dotline=rrggbb> ... </dotline> draws a dot-line on the bottom side of the current text-line, of specified RGB color. The <dotline> ... </dotline> draws a black dot-line on the bottom side of the current text-line. The rr/gg/bb represents the red/green/blue values of the color in hexa values.

exPreviewPageNumberFormat1	• **<upline> ... </upline>** draws the line on the top side of the current text-line (requires <solidline> or <dotline>). • **<r>** right aligns the text • **<c>** centers the text • ** ** forces a line-break • **<img>number[:width]</img>** inserts an icon inside the text. The number indicates the index of the icon being inserted. Use the Images method to assign a list of icons to your chart. The last 7 bits in the high significant byte of the number expression indicates the identifier of the skin being used to paint the object. Use the Add method to add new skins to the control. If you need to remove the skin appearance from a part of the control you need to reset the last 7 bits in the high significant byte of the color being applied to the part. The width is optional and indicates the width of the icon being inserted. Using the width option you can overwrite multiple icons getting a nice effect. By default, if the width field is missing, the width is 18 pixels. • **<img>key[:width]</img>** inserts a custom size picture into the text being previously loaded using the HTMLPicture property. The Key parameter indicates the key of the picture being displayed. The Width parameter indicates a custom size, if you require to stretch the picture, else the original size of the picture is used. • **&** glyph characters as **&** (&), **<** (<), **>** (>), **&qout;** (") and **&#number;** (the character with specified code), For instance, the € displays the EUR character. The **&** ampersand is only recognized as markup when it is followed by a known letter or a #character and a digit. For instance if you want to display bold in HTML caption you can use bold • **<off offset> ... </off>** defines the vertical offset to display the text/element. The offset parameter defines the offset to display the

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

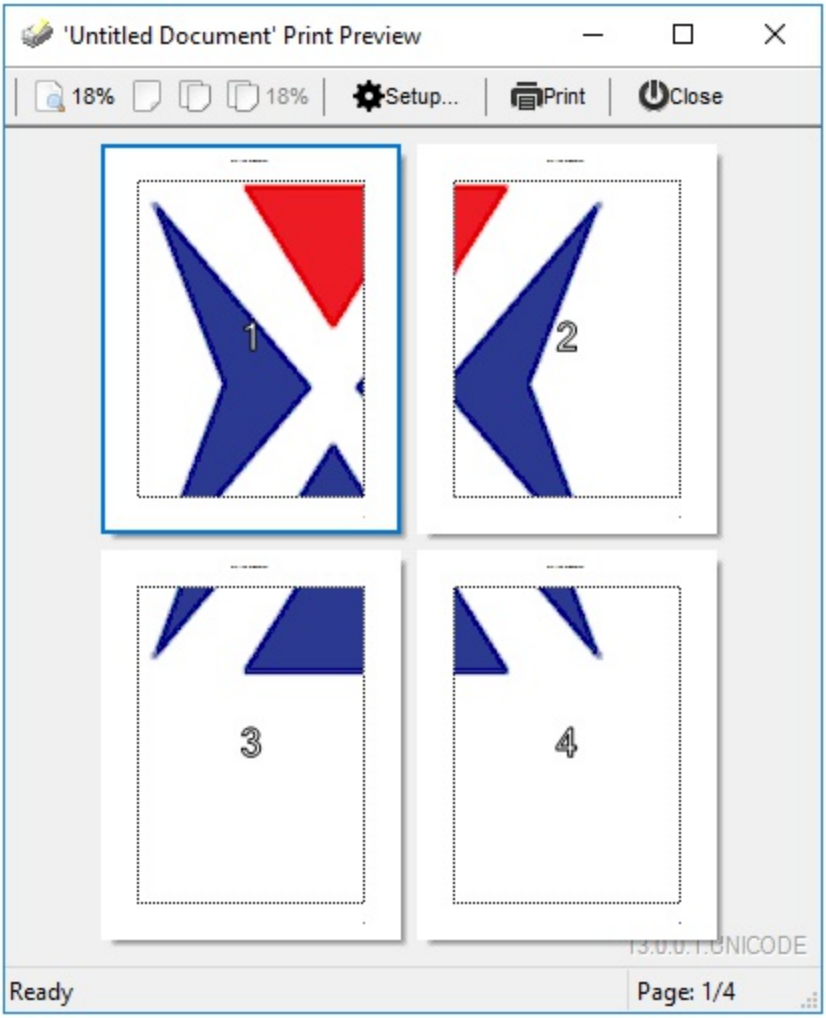
- **<gra rrggbb;mode;blend> ... </gra>** defines a gradient text. The text color or <fgcolor> defines the starting gradient color, while the rr/gg/bb represents the red/green/blue values of the ending color, 808080 if missing as gray. The mode is a value between 0 and 4, 1 if missing, and blend could be 0 or 1, 0 if missing. The HTML tag can be used to define the height of the font. Any of the rrggbb, mode or blend field may not be specified. The <gra> with no fields, shows a vertical gradient color from the current text color to gray (808080). For instance the "<**gra** FFFFFF;1;1>gradient-center</**gra**>" generates the following picture:
- **<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: *(String expression)*
exPreviewPageNumbersAlignment	2	Specifies the alignment of page numbers in the page. By default, the exPreviewPageNumbersAlignment property is exMiddleCenter. *(ContentAlignmentEnum expression)*
exPreviewShowCompact	3	Specifies whether the preview shows the pages in a compact mode. By default, the exPreviewShowCompact property is exPreviewShowPageCompact. *(PreviewShowCompactEnum expression)*
exPreviewBackColor	4	Specifies the preview's mainframe background color. By default, the exPreviewBackColor property is the default system color for the menu. *(Color expression)*
exPreviewAllowWheelScroll	5	Indicates whether the user can scroll pages by rotating the mouse wheel. By default, the exPreviewAllowWheelScroll property is True. *(Boolean expression)*

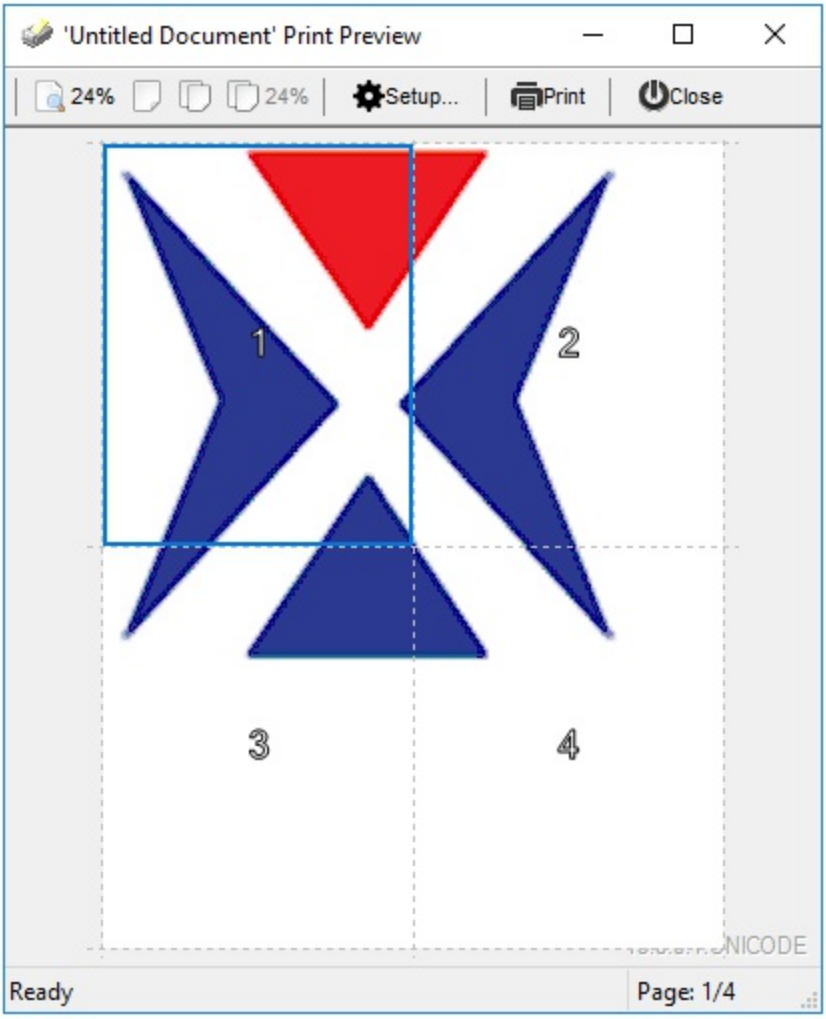
exPreviewAllowDragScroll	6	Specifies whether the user can scroll pages by clicking and dragging the left mouse button. By default, the exPreviewAllowDragScroll property is True. *(Boolean expression)*
exPreviewAllowUnprintPage	7	Specifies whether the user can select/unselect pages/area using the right mouse button (CTRL key), to prevent them from printing. By default, the exPreviewAllowUnprintPage property is True. *(Boolean expression)*
exPreviewAllowToggleZoom	8	Indicates whether the user can toggle the zoom by clicking the focused page. By default, the exPreviewAllowToggleZoom property is True. *(Boolean expression)*
exPreviewAllowMiddleZoom	9	Specifies whether the user can zoom the pages by clicking and dragging the middle mouse button. By default, the exPreviewAllowMiddleZoom property is True. *(Boolean expression)*

constants PreviewShowCompactEnum

The PreviewShowCompactEnum type specifies how the preview mainframe displays pages. The PreviewSettings(exPreviewShowCompact) property specifies whether the pages in preview mode are displayed in compact mode (no margins). The PreviewShowCompactEnum type supports the following values:

Name	Value	Description
exPreviewShowPageDefault	0	The pages in preview, show margins. 
		The pages in preview, show no margins.

exPreviewShowPageCompact-1



constants PreviewStateEnum

The PreviewStateEnum type specifies the visual state of the Print and Print Preview mainframe. The PreviewState property returns or sets the visual state of preview mainframe at runtime. You can use the PreviewState property to programmatically maximize the preview window. The PreviewStateEnum type supports the following values:

Name	Value	Description
exPreviewStateNormal	0	Activates and displays the Print and Print Preview mainframe. If the window is minimized or maximized, the system restores it to its original size and position. By default, the Print and Preview mainframe's size and position is saved once it is closed, and restored at the next Preview call.
exPreviewStateMinimized	1	Activates the window and displays it as a maximized window.
exPreviewStateMaximized	2	Activates the window and displays it as a minimized window.

constants ItemCaptionEnum

The ItemCaptionEnum type defines different strings that are shown in the toolbar or print and print preview workspace. Use the ItemCaption property to change the captions/fields in the print preview window. Use the Images/ReplaceIcon method to add new icons to the control. Use the HTMLPicture property to add custom sized pictures. The ToolBarFormat property specifies the format to display the preview's toolbar.

The ItemCaptionEnum type supports the following values:

Name	Value	Description
exSetup	0	Changes the caption of the '**Setup**' button. (Obsolete, replaced by exToolBarSetup)
exPrint	1	Changes the caption of the '**Print**' button. (Obsolete, replaced by exToolBarPrint)
exClose	2	Changes the caption of the '**Close**' button. (Obsolete, replaced by exToolBarClose)
exPageWidth	3	Changes the caption of the '**PageWidth**' item, in the zoom field.
exWholePage	4	Changes the caption of the '**WholePage**' item, in the zoom field.
exTwoPage	5	Changes the caption of the '**TwoPage**' item, in the zoom field.
exReady	6	Changes the '**Ready**' string.
exPage	7	Changes the '**Page**' string.
exPrinting	8	Changes the '**Printing**' string.
exCancel	9	Changes the message that's displayed when control printing the object. By default, the exCancel is "**, press ESC to cancel the current printing job.**"
exPrintPreview	10	Changes the '**Print Preview**' message in the preview's workspace window.
exAdjustMargin	11	Specifies the '**Adjust Margins**' title being displayed when the cursor hovers a margin of the page.
exSetupPrinter	12	Specifies the '**Printer...**' caption within the page-setup dialog.
exToolBarMagnify	100	Specifies the caption of the '**Zoom**' button. By default, this option is "<img>1</img>%%". The %% indicates the current zooming factor.

exToolBarOnePage	101	Specifies the caption of the '**One Page**' button. By default, this option is "<img>2</img>".
exToolBarTwoPage	102	Specifies the caption of the '**Multiple Pages**' button. By default, this option is "<img>3</img>".
exToolBarSetup	103	Specifies the caption of the '**Setup...**' button. By default, this option is "<img>4</img>Setup...".
exToolBarPrint	104	Specifies the caption of the '**Print**' button. By default, this option is "<img>5</img>Print".
exToolBarClose	105	Specifies the caption of the '**Close**' button. By default, this option is "<img>6</img>Close".
exToolBarTwoPageFixed	106	Specifies the caption of the '**Multiple Pages (Fixed)**' button. By default, this option is " <img>3</img> %%". The %% indicates the current zooming factor.

ExPrint object

The Exontrol ExPrint component is an advanced printing system specifically to bring your User Interface to the printed page. The ExPrint adds powerful print and preview capabilities to your forms, dialog boxes or other windows. The ExPrint component provides Print and Print Preview features for components like: eXGantt, eXG2antt, eXMLGrid, eXGrid, eXTree, eXList, eXCalendar, eXComboBox, eXPropertiesList, eXEdit, eXFileView, eXOrgChart, eXSchedule, eXPivot and so on. The ExPrint component provides the ability to print the entire document or user selected area. Here's the list of supported properties and methods.

Name	Description
AsScreen	Specifies whether the control creates the page's preview as it would be displayed on the screen.
AttachTemplate	Attaches a script to the current object, including the events, from a string, file, a safe array of bytes.
AutoRelease	Specifies whether the print object is released automatically.
Caption	Specifies the document's caption.
CaptionAlignment	Specifies the alignment of document's alignment.
CaptionPosition	Specifies the position of document's caption.
ClientHeight	Retrieves the height in pixels, of the drawing area of the printer page.
ClientWidth	Retrieves the width in pixels, of the drawing area of the printer page.
CopyTo	Copies the pages to EMF files.
Debug	Displays debug information.
Decode64Text	Decodes the giving string, from base64 format (compressed).
Decode64TextA	Decodes (and decompress) the giving string, from base64 format to ANSI string.
Decode64TextW	Decodes (and decompress) the giving string, from base64 format to UNICODE string.
DoPrint	Prints the document.
Encode64	Encodes and compress the picture/file to a BASE64 encoded string.
	Encodes and compress a list of icons to a BASE64

Encode64Icons	encoded string.
Encode64Text	Encodes and compress the giving string, to base64 format.
Encode64TextA	Encodes (ANSI) and compress the giving string, to base64 format.
Encode64TextW	Encodes (UNICODE) and compress the giving string, to base64 format.
ExecuteTemplate	Executes a template and returns the result.
ExtraCaption	Adds or removes an additional caption.
Font	Retrieves or sets the control's font.
Foreground	Brings the Preview window on the foreground and activates it.
FormatABC	Formats the A,B,C values based on the giving expression and returns the result.
HTMLPicture	Adds or replaces a picture in HTML captions.
hWnd	Retrieves the handle of the print preview main frame.
Images	Sets at runtime the print's image list. The Handle should be a handle to an Image List Control.
ImageSize	Retrieves or sets the size of icons the control displays.
ItemCaption	Specifies a value that indicates the caption for specified item.
ItemToolTip	Specifies a value that indicates the tooltip for specified item.
Options	Specifies the document's options.
PageFrameColor	Specifies the color of frame to be shown on printed pages.
PageFrameStyle	Specifies the style of frame to be shown on printed pages.
PageNumberFormat	Specifies the format to display the number of page.
PageNumbersAlignment	Specifies the alignment of page numbers in the document.
PageNumbersPosition	Specifies the position of page numbers in the document.
PageOrientation	Specifies the default page's orientation.
PageRange	Specifies the pages being printed.
PagesCount	Returns the number of pages.

Preview	Invokes the print preview main frame.
PreviewSettings	Sets or gets a value that defines a setting for preview mode.
PreviewState	Returns or sets the visual state of preview mainframe at runtime.
Printers	Retrieves a list of installed printers.
PrintExt	Specifies an object that implements the IPrintExt interface.
PrintExts	Specifies a collection of objects that implement the IPrintExt interface.
Refresh	Refreshes the print preview.
ReplaceIcon	Adds a new icon, replaces an icon or clears the print's image list.
RuntimeKey	Specifies a runtime key to be used for the component.
Settings	Sets or gets a value that indicates the value for specified field.
ShowMargins	Retrieves or sets a value that specifies whether the page displays its margins so the user can resize the margins of the page at runtime.
ShowPageNumbers	Specifies whether the page numbers are shown or hidden.
StartPageNumber	Specifies the number to start page numbering.
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.
ToolBarFont	Retrieves or sets the toolbar's font.
ToolBarFormat	Specifies the CRD format to arrange the buttons inside the print's toolbar.
UILimitPagesCount	Specifies the limit of pages the control can load before a message box to continue shows up.
UILimitPagesCountMessage	Specifies the continue message to show up, when the limit of pages has been reached.
Version	Retrieves the control's version.

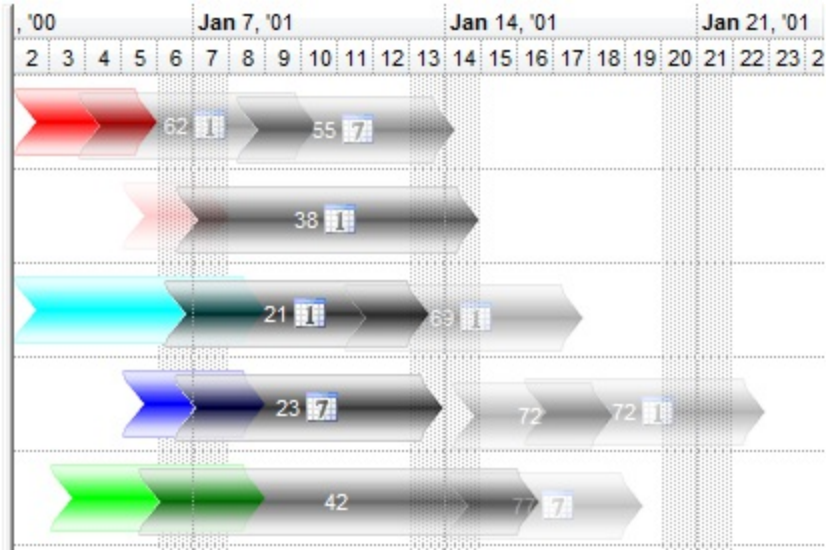
property ExPrint.AsScreen as Boolean

Specifies whether the control creates the page's preview as it would be displayed on the screen.

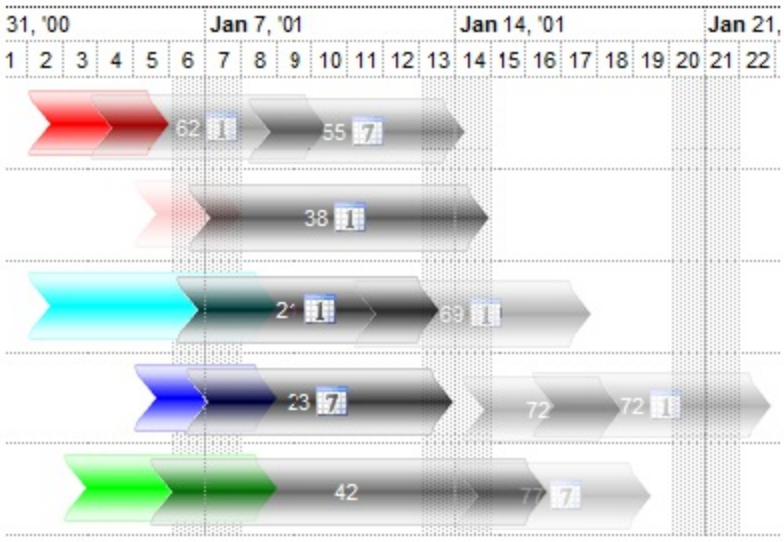
Type	Description
Boolean	A Boolean expression that specifies whether the page's preview shows as on the screen.

By default, the AsScreen property is False. Use the AsScreen property to have the page's preview closer with what displayed on the screen. This property is useful, when the control displays semi-transparent objects and displaying the preview may differ on what you have seen on the screen.

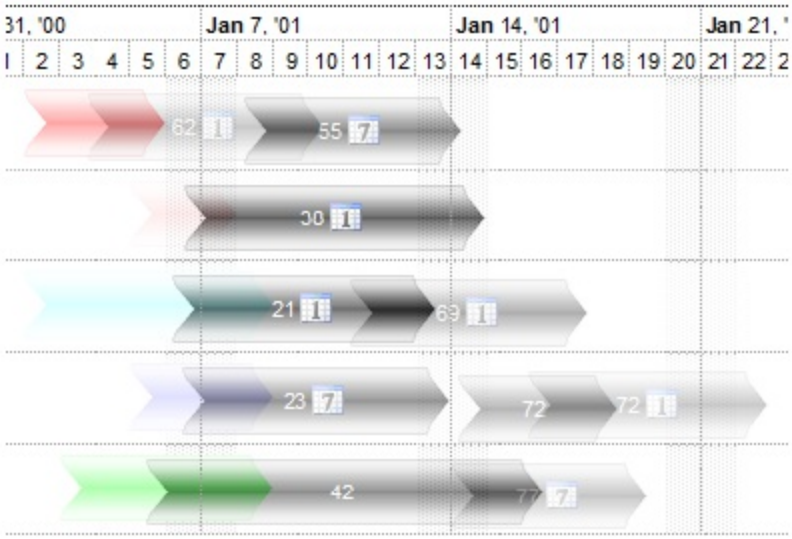
For instance, the following picture shows the control with multiple semi-transparent bars:



If the AsScreen propetry is True, the preview of the bars looks as follow:

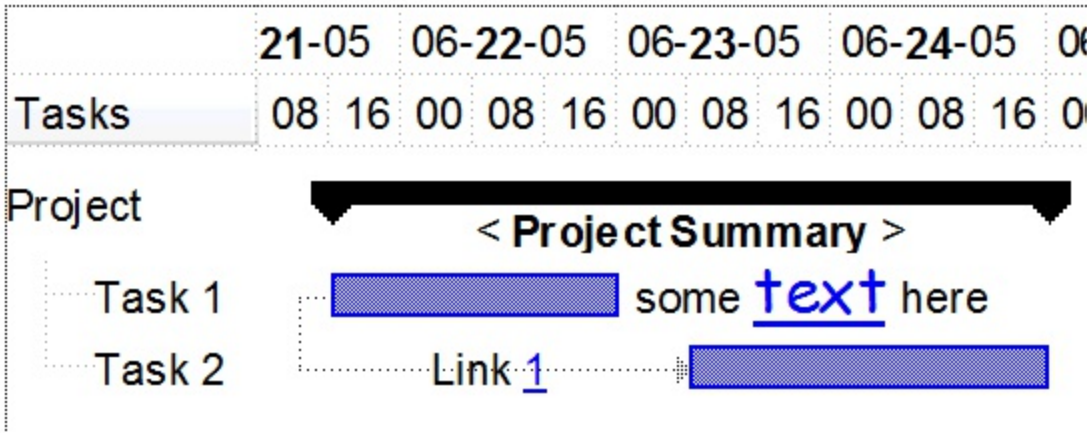


If the AsScreen propetry is False, the preview of the bars looks as follow:

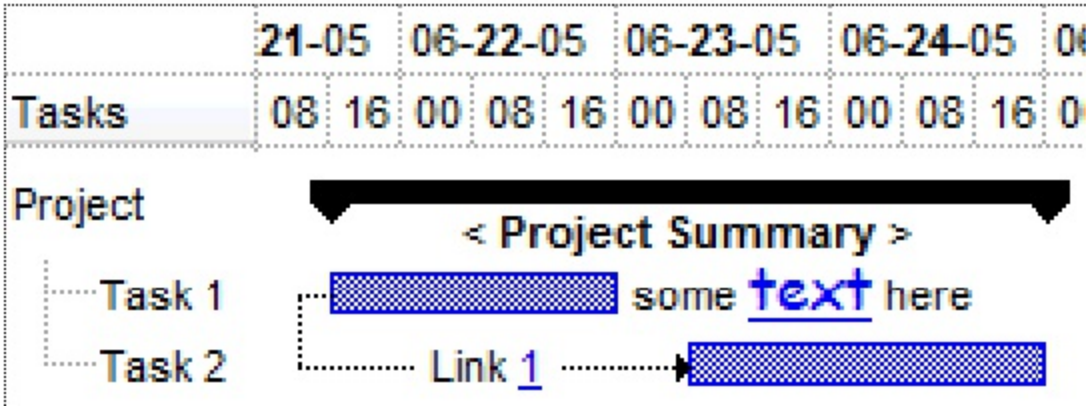


Another difference that may be observed when using the AsScreen property is zooming the preview ie if the AsScreen property is True, the magnified preview may look stretched, as you can see in the following screen shots:

The following screen shot shows the preview of 200% when AsScreen property is False:



The following screen shot shows the preview of 200% when AsScreen property is True:



method ExPrint.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 Print1_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 ExPrint.AutoRelease as Boolean

Specifies whether the print object is released automatically.

Type	Description
Boolean	A Boolean expression that specifies whether the Print object is released automatically when it is not required anymore.

By default, the AutoRelease property is True. If the AutoRelease property is True, the print object is released as soon as the control to be printed is not available anymore, of the user closes the Preview's frame. The AutoRelease on False, indicates that the print object is released as soon as the user closes the Preview's mainframe. Setting the AutoRelease property on False, has effect only if called before Preview method.

The AutoRelease property on False, is useful, if you create the Print object locally at runtime. If created locally, the Print object is released when it is out of scope, or when the procedure ends.

For instance, in the following VB6 sample the user creates a Print object as a local variable, and so the object is released as soon as the Form_Load ends. In other words, you will notice that the Preview window is opened and closed quickly, as it is closed because the Exontrol.Print object is released as soon as the Form_Load function ends.

```
Private Sub Form_Load()
   With CreateObject("Exontrol.Print")
      .PrintExt = G2antt1.Object
      .Preview
   End With
End Sub
```

This problem can be solved in 2 ways:

- Use the AutoRelease property on False, before calling the Preview method
- Using a public member of Exontrol.Print type

as shown in the following samples:

Let's use the AutoRelease property, so the code is like follows. This time, if running the sample, the Preview window will be shown, and the Exontrol.Print object is released as soon as the user closes the Preview's mainframe.

```
Private Sub Form_Load()
```

```
  With CreateObject("Exontrol.Print")
    .AutoRelease = False
    .PrintExt = G2antt1.Object
    .Preview
  End With
End Sub
```

And another solution is declaring a public member of Exontrol.Print type, so it will be available, as form lives as described bellow. This time, if running the sample, the Preview window will be shown, not closed, as the object p is released ONLY when closing the form.

```
Dim p As Object
Private Sub Form_Load()
  Set p = CreateObject("Exontrol.Print")
  With p
    .PrintExt = G2antt1.Object
    .Preview
  End With
End Sub
```

property ExPrint.Caption as String

Specifies the document's caption.

Type	Description
String	A string expression that indicates the document's caption, that support built-in HTML tags like shown bellow.

Use the Caption property to specify the printed document's caption. Use the CaptionAlignment property to align the caption in the header or footer of the printed page. Use the CaptionPosition property to specify whether the document's caption is displayed on the header or footer of the printed page. Use the ShowPageNumbers property to show or hide the page number filed. Use the ExtraCaption property to add extra captions to your document. Use the Font property to assign a different font for the caption printed on the document. Use the <br> to break lines in the caption, and so you can display multiple-lines captions. Use the Images/ReplaceIcon method to add new icons to the control. Use the HTMLPicture property to add custom sized pictures.

Starting from the version 15.0, the Caption property and Caption parameter of ExtraCaption method supports expressions, that can defines the HTML caption of each page based on different fields such as index of the page, total number of pages, the index of the object being printed / previewed, the index of the page relative to the object being printed, and the number of relative pages. *If the Caption's expression is not valid, the Caption itself is displayed as HTML, be replacing the **<%page%>** and **<%count%>** fields with the current page, and number of pages, as explained on the bottom of the this page.*

The Caption property / parameter supports the following keywords:

- **object**, indicates the index of the object being printed, as a numeric value. The object, opages and opage fields are useful in case you are printing multiple-objects at the same time, using the PrintExts method. For instance, PrintExts = Array(Grid1, Grid2) specifies that two controls should be sent to the printer, and in this case the **object** will indicate 0 for the Grid1, and 1 for the Grid2. For instance, Caption = "`<b>` + (object array (`<s>first-grid`,`<i>second-grid`))" defines an expression-caption that displays the caption "first-grid" in bold and strikeout, for the first object of the PrintExts, and displays the "second-grid" in bold and italic, for the second object of the PrintExts method.
- **opages** defines the number of pages the current object is being previewed / printed, as a numeric value.
- **opage** indicates the relative-index of the page being printed / previewed, as a numeric value.
- **pages** defines the total number of pages being previewed / printed, as a numeric value.

- **page** which specifies the index of the current page, from the total number of pages, as a numeric value. For instance, ExtraCaption("logo","page mod 2 ? `` : `<img>logo</img>`") adds a logo picture on every second-page.

This property/method supports predefined constants and operators/functions as described here.

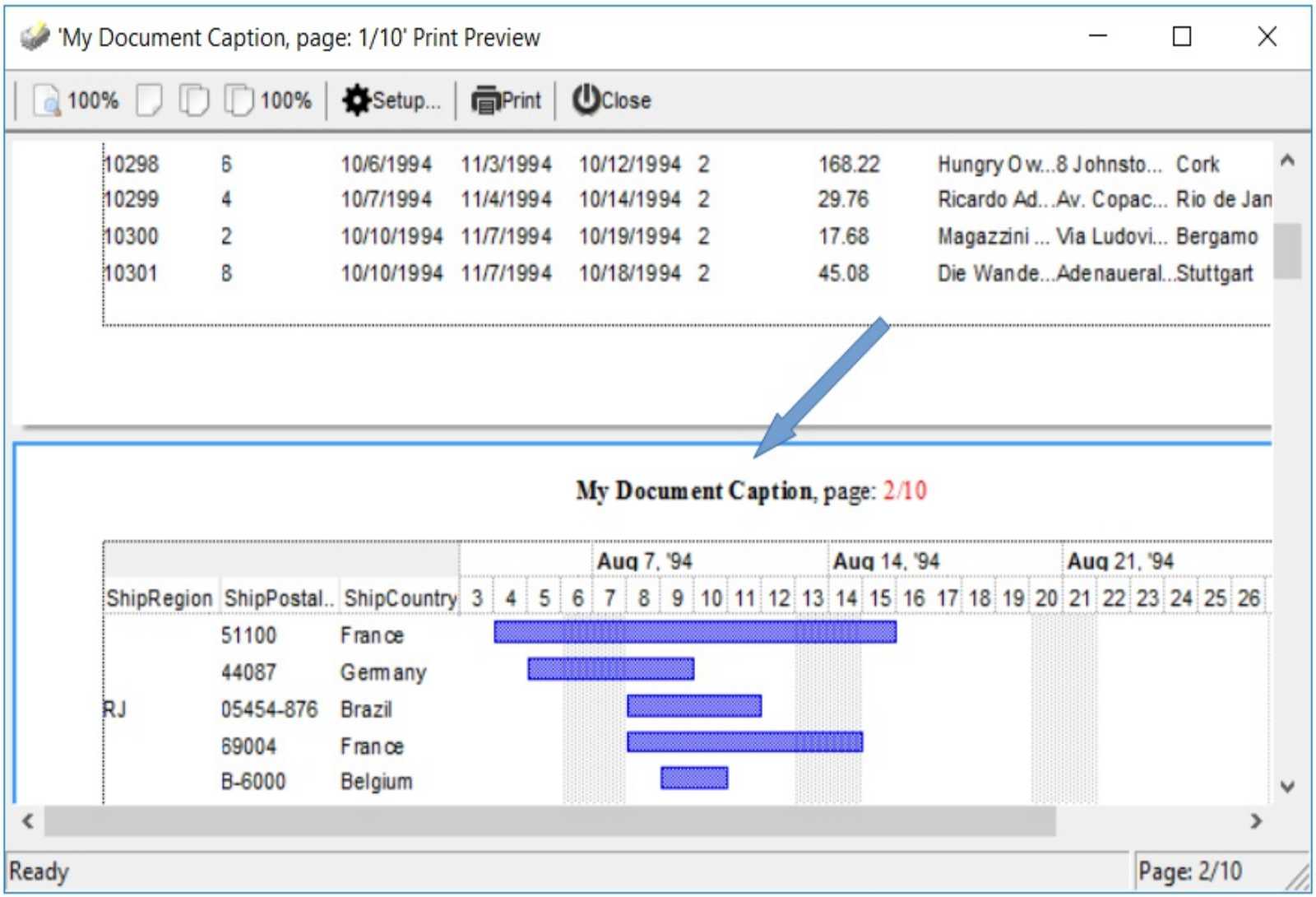
For instance, Caption can be (as a valid-expression):

- "`My Document Title`", the "My Document Title" is what you get on every page. Please pay attention that we used the ` character to quote the string.
- "`My Document <b>Title</b>`", the "My Document **Title**" is what you get on every page. Please pay attention that we used the ` character to quote the string.
- "`Page: ` + page + ` from ` + pages", displays Page: 1 from 100, Page: 2 from 100, on each page.
- "page mod 2 ? `odd` : `even`", displays odd on odd pages, and even on even pages.
- "page mod 2 ? `` : `<img>logo</img>`", displays the logo picture on every second page (even pages). The HTMLPicture property adds or replaces a picture in HTML captions.
- "page = pages ? `last` : ((page - 1) array(`first`,`second`,`third`,`forth`,`fifth`,`sixth`,`seventh`,`eighth`,`ninth`,`tenth`))", displays the last for the last page, first for the page with the index 1, second for the page with the index 2, and so on.

For instance, Caption can be (as a non-expression):

- "My Document Title", the "My Document Title" is what you get on every page
- "My Document Title", the "My Document **Title**" is what you get on every page
- "Page: <%page%> from <%count%>", displays Page: 1 from 100, Page: 2 from 100, on each page

The following screen shot shows where he caption mat be displayed:



Currently, the Caption property may include the following built-in HTML elements:

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

color in hexa values.

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

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

- **<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:

Use the <br> to break the lines, use the <font> element to specify a new face for the font, or to change the size of the font.

Also the Caption property supports the following predefined values:

- **<%page%>** specifies the current page
- **<%count%>** indicates the number of pages in the document.

property ExPrint.CaptionAlignment as AlignmentEnum

Specifies the alignment of document's alignment.

Type	Description
AlignmentEnum	An AlignmentEnum expression that indicates the alignment of the document's caption.

Use the CaptionAlignment property to align the document's caption in the page. Use the CaptionPosition property to specify whether the document's caption is display in the header or footer of the page.

property ExPrint.CaptionPosition as PositionEnum

Specifies the position of document's caption.

Type	Description
PositionEnum	A PositionEnum expression that indicates whether the document's name is displayed on the header or footer of the page.

Use the CaptionPosition property to specify whether the document's caption is displayed on the header or footer of the page. Use the CaptionAlignment property to align the document's caption in the printed page.

property ExPrint.ClientHeight as Long

Retrieves the height in pixels, of the drawing area of the printer page.

Type	Description
Long	A long expression that indicates the height in pixels, of the drawing area of the printer page.

Use the ClientHeight property to retrieve the height in pixels of the print page. Use the PageOrientation property to specify whether the page is landscape oriented. Use the Settings property to specify different options for the printer before previewing or printing. Use the ClientWidth property to determine the width of the printed page, so you can implement the print to page feature for your application.

property ExPrint.ClientWidth as Long

Retrieves the width in pixels, of the drawing area of the printer page.

Type	Description
Long	A long expression that indicates the width in pixels, of the drawing area of the printer page.

Use the ClientWidth property to determine the width of the printed page, so you can implement the print to page feature for your application. Use the PageOrientation property to specify whether the page is landscape oriented. Use the Settings property to specify different options for the printer before previewing or printing. Use the ClientHeight property to retrieve the height in pixels of the print page.

method ExPrint.CopyTo (Path as String)

Copies the pages to EMF files.

Type	Description
Path as String	A String expression that specifies the path and the file names to generate the files. The Path should/may include the **%i** which indicates the index of the page being printed to the specified format. The index starts from 1 for the first page, 2 for second page, and so on. For instance, the C:\Temp\Print\Page%i.emf exports the pages to files such as C:\Temp\Print\Page1.emf, C:\Temp\Print\Page2.emf, and so on depending on the number of pages. If the Path parameter is not empty, the extension (characters after last dot) determines the graphical/ format of the file to be saved as follows: • ***.bmp *.dib *.rle**, exports the pages to **BMP** format. • ***.jpg *.jpe *.jpeg *.jfif**, exports the pages to **JPEG** format. • ***.gif**, , exports the pages to **GIF** format. • ***.tif *.tiff**, exports the pages to **TIFF** format. • ***.png**, exports the pages to **PNG** format. • ***.pdf**, exports the pages to PDF format. The **%i** flag has no effect if using PDF format, as all pages are saved to a single document. • ***.emf** or any other extension determines the control to save the pages in **EMF** format.

Return	Description
Long	0 meaning that no pages where generated or created (due some errors like write error, not setting the PrintExt property before and so on). Any value greater than 0 indicates the number of files being created.

The CopyTo method copies/exports the control's view to BMP, PNG, JPG, GIF, TIFF, PDF or EMF graphical files. The CopyTo returns 0, if any error occurs, or it returns the number of files being created. The Path parameter should include a %i expression that's replaced with the index of the page being generated. If no %i sequence is included in the Path parameter the last page is saved to specified file. The files can be opened with any

application that knows how to open or handle the EMF files, such as Microsoft Office Picture Manager, Microsoft PaintBrush, and so on. During CopyTo method any exiting file (with the generated name) is deleted before created. The PrintExt property must be specified before calling the CopyTo method, as you would do a previewing using the Preview method. If the PrintExt property is empty before calling the CopyTo method, it returns 0, and no files are created.

- The **BMP** file format, also known as bitmap image file or device independent bitmap (DIB) file format or simply a bitmap, is a raster graphics image file format used to store bitmap digital images, independently of the display device (such as a graphics adapter)
- The **JPEG** file format (seen most often with the .jpg extension) is a commonly used method of lossy compression for digital images, particularly for those images produced by digital photography.
- The **GIF** (Graphics Interchange Format) is a bitmap image format that was introduced by CompuServe in 1987 and has since come into widespread usage on the World Wide Web due to its wide support and portability.
- The **TIFF** (Tagged Image File Format) is a computer file format for storing raster graphics images, popular among graphic artists, the publishing industry, and both amateur and professional photographers in general.
- The **PNG** (Portable Network Graphics) is a raster graphics file format that supports lossless data compression. PNG was created as an improved, non-patented replacement for Graphics Interchange Format (GIF), and is the most used lossless image compression format on the Internet
- The **PDF** (Portable Document Format) is a file format used to present documents in a manner independent of application software, hardware, and operating systems. Each PDF file encapsulates a complete description of a fixed-layout flat document, including the text, fonts, graphics, and other information needed to display it.
- The EMF (Enhanced Metafile Format) is a 32-bit format that can contain both vector information and bitmap information. This format is an improvement over the Windows Metafile Format and contains extended features, such as the following

  Built-in scaling information
  Built-in descriptions that are saved with the file
  Improvements in color palettes and device independence

  The EMF format is an extensible format, which means that a programmer can modify the original specification to add functionality or to meet specific needs. You can paste this format to Microsoft Word, Excel, Front Page, Microsoft Image Composer and any application that know to handle EMF formats.

The following VB6 sample generates EMF files instead printing the Exontrol's ExG2antt/COM content:

```
With Print1
   Set .PrintExt = G2antt1.Object
   Debug.Print "Files generated: " & .CopyTo("C:\Temp\Print\Page%i.emf")
End With
```

The following VB/NET sample generates EMF files instead printing the Exontrol's ExG2antt/NET content:

```
With Exprint1
   .PrintExt = Exg2antt1
   System.Diagnostics.Debug.WriteLine("Files generated: " &
.CopyTo("C:\Temp\Print\Page%i.emf"))
End With
```

property ExPrint.Debug as String

Displays debug information.

Type	Description
String	A string expression that holds debug information.

Reserved for internal use only.

property ExPrint.Decode64Text (Text64 as String) as String

Decodes the giving string, from base64 format (compressed).

Type	Description
Text64 as String	A String expression (the string to be decoded) that defines the string being encoded using the Encode64Text property.
String	A String expression (the original string) that indicates the decoded text.

The Decode64Text property returns the original string (previously encoded using the Encoded64Text property). The Text64 parameter must indicates the result of the Encode64Text property. For instance, the Filter attribute of the PivotColumns property of the eXPivot component, returns the list of values for filtering in encoding format. You can use the Encode64Text/Decode64Text property to encoded/decode the value of the Filter attribute. The encoded string looks like: "gBUNBpOYggggMIgMRpOggN5mEBINxjN5kMpkEB0Mp4OgAgI=". Always, the equation Decode64Text(Encode64Text(Text)) = Text is the true.

The Decode64Text property returns the same result as Decode64TextA property, if running the ANSI version, or Decode64TextW property is running the UNICODE version of the eXPrint component. The Version property of the control specifies whether you are running the ANSI or UNICODE version. If the Version property includes the UNICODE string, it means that you are running the UNICODE version, else it is the ANSI version.

property ExPrint.Decode64TextA (Text64 as String) as String

Decodes (and decompress) the giving string, from base64 format to ANSI string.

Type	Description
Text64 as String	A String expression (the string to be decoded) that defines the string being encoded using the Encode64TextA property.
String	A String expression (the original string) that indicates the decoded text.

The Decode64TextA property returns the original string/ANSI (previously encoded using the Encoded64TextA property). The Text64 parameter must indicates the result of the Encode64TextA property. The encoded string looks like: "gBUNBpOYggggMIgMRpOggN5mEBlNxjN5kMpkEB0Mp4OgAgI=". Always, the equation Decode64TextA(Encode64TextA(Text)) = Text is the true.

The Decode64Text property returns the same result as Decode64TextA property, if running the ANSI version, or Decode64TextW property is running the UNICODE version of the eXPrint component. The Version property of the control specifies whether you are running the ANSI or UNICODE version. If the Version property includes the UNICODE string, it means that you are running the UNICODE version, else it is the ANSI version.

property ExPrint.Decode64TextW (Text64 as String) as String

Decodes (and decompress) the giving string, from base64 format to UNICODE string.

Type	Description
Text64 as String	A String expression (the string to be decoded) that defines the string being encoded using the Encode64TextW property.
String	A String expression (the original string) that indicates the decoded text.

The Decode64TextW property returns the original string/UNICODE (previously encoded using the Encoded64TextW property). The Text64 parameter must indicates the result of the Encode64TextW property. The encoded string looks like: "gBUNBpOYggggMIgMRpOggN5mEBlNxjN5kMpkEB0Mp4OgAgI=". Always, the equation Decode64TextW(Encode64TextW(Text)) = Text is the true. For instance, most of our components, provide a Layout property that helps you to store and restore the control's layout, like position of the columns, sorted columns, and so on. In other words, you can use the Decode64TextW property to decode the value that the Layout property returns.

The Decode64Text property returns the same result as Decode64TextA property, if running the ANSI version, or Decode64TextW property is running the UNICODE version of the eXPrint component. The Version property of the control specifies whether you are running the ANSI or UNICODE version. If the Version property includes the UNICODE string, it means that you are running the UNICODE version, else it is the ANSI version.

method ExPrint.DoPrint ([ShowUI as Variant])

Prints the document.

Type	Description
ShowUI as Variant	A boolean expression that indicates whether control displays the print dialog setup before printing. The ShowUI parameter is optional. By default, the ShowUI argument is True.

Use the DoPrint method to print a document. Use the Preview method to show the print preview mainframe. The DoPrint method fails if the PrintExt/PrintExts property is nothing. So, before calling Preview or DoPrint method the PrintExt property must be set with the object being printed. The PageRange property specifies the pages being printed.

property ExPrint.Encode64 (Picture as Variant) as String

Encodes a picture/file to a BASE64 encoded string.

Type	Description
Picture as Variant	A Picture/IPictureDisp object or a string expression that indicates the path to the picture or the file, to be encoded. For instance, you can encode *bmp, *.jpg, *.gif picture files, or EBN files
String	A String expression that indicates the BASE64 representation of the picture or file being loaded. If empty string is returned nothing was encoded.

The Encode64 property compress and encodes programmatically a picture or a file. Use the eXImages tool to generate your BASE64 strings from pictures. Use the Encode64Icons property to compress and encode a list of icons to be used in Images methods.

The following VB sample encodes the "zapotec.bmp" picture file:

```
MsgBox Print1.Encode64("c:\winnt\zapotec.bmp")
```

The following VB.NET sample encodes the "zapotec.bmp" picture file:

```
MsgBox(AxPrint1.get_Encode64("c:\winnt\zapotec.bmp"))
```

The following C# sample encodes the "zapotec.bmp" picture file:

```
MessageBox(AxPrint1.get_Encode64("c:\winnt\zapotec.bmp"))
```

The following C++ sample encodes the "zapotec.bmp" picture file:

```
MessageBox(m_print.get_Encode64(COleVariant("c:\winnt\zapotec.bmp")))
```

The following VFP sample encodes the "zapotec.bmp" picture file:

```
wait window nowait thisform.Print1.Encode64("c:\winnt\zapotec.bmp")
```

property ExPrint.Encode64Icons (Icons as Variant) as String

Encodes a list of icons to a BASE64 encoded string.

Type	Description
Icons as Variant	A String expression that indicates a list of icons being encoded. The list is delimited by "," comma character.
String	A String expression that indicates the BASE64 representation of the list of icons. If empty string is returned nothing was encoded.

Use the Encode64Icons property to compress and encode a list of icons to be used in Images methods. Use the eXImages tool to generate your BASE64 strings from pictures. The Encode64 property compress and encodes programmatically a picture or a file. For instance, the following sample encodes a list of two icons: Encode64Icons("d:\temp\icons\Calendar.ico,d:\temp\icons\Contacts.ico"). Each icon file must provide the full path to the icon, else the method will not be able to find it.

property ExPrint.Encode64Text (Text as String) as String

Encodes and compress the giving string, to base64 format.

Type	Description
Text as String	A String expression (the string to be encoded) that defines the string to be encoded
String	A String expression (the encoded string) that indicates the encoded string, which can be decoded using the Decode64Text property.

The Encode64Text property compress and encodes the giving string, in BASE64 format. The result of the Encode64Text property can be decoded using the Decode64Text property. For instance, the Filter attribute of the PivotColumns property of the eXPivot component, returns the list of values for filtering in encoding format. You can use the Encode64Text/Decode64Text property to encoded/decode the value of the Filter attribute. The encoded string looks like: "gBUNBpOYggggMIgMRpOggN5mEBINxjN5kMpkEB0Mp4OgAgI=". Always, the equation Decode64Text(Encode64Text(Text)) = Text is the true.

The Encode64Text property returns the same result as Encode64TextA property, if running the ANSI version, or Encode64TextW property is running the UNICODE version of the eXPrint component. The Version property of the control specifies whether you are running the ANSI or UNICODE version. If the Version property includes the UNICODE string, it means that you are running the UNICODE version, else it is the ANSI version.

property ExPrint.Encode64TextA (Text as String) as String

Encodes (ANSI) and compress the giving string, to base64 format.

Type	Description
Text as String	A String expression (the string to be encoded) that defines the string to be encoded
String	A String expression (the encoded string) that indicates the encoded string, which can be decoded using the Decode64TextA property.

The Encode64TextA property compress and encodes the giving string (as ANSI), in BASE64 format. The result of the Encode64TextA property can be decoded using the Decode64TextA property. The encoded string looks like: "gBUNBpOYggggMIgMRpOggN5mEBlNxjN5kMpkEB0Mp4OgAgI=". Always, the equation Decode64TextA(Encode64TextA(Text)) = Text is the true.

The Encode64Text property returns the same result as Encode64TextA property, if running the ANSI version, or Encode64TextW property is running the UNICODE version of the eXPrint component. The Version property of the control specifies whether you are running the ANSI or UNICODE version. If the Version property includes the UNICODE string, it means that you are running the UNICODE version, else it is the ANSI version.

property ExPrint.Encode64TextW (Text as String) as String

Encodes (UNICODE) and compress the giving string, to base64 format.

Type	Description
Text as String	A String expression (the string to be encoded) that defines the string to be encoded
String	A String expression (the encoded string) that indicates the encoded string, which can be decoded using the Decode64TextW property.

The Encode64TextW property compress and encodes the giving string (as UNICODE), in BASE64 format. The result of the Encode64TextW property can be decoded using the Decode64TextW property. The encoded string looks like: "gBUNBpOYggggMIgMRpOggN5mEBlNxjN5kMpkEB0Mp4OgAgI=". Always, the equation Decode64TextW(Encode64TextW(Text)) = Text is the true.

The Encode64Text property returns the same result as Encode64TextA property, if running the ANSI version, or Encode64TextW property is running the UNICODE version of the eXPrint component. The Version property of the control specifies whether you are running the ANSI or UNICODE version. If the Version property includes the UNICODE string, it means that you are running the UNICODE version, else it is the ANSI version.

method ExPrint.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 displays the value of the Settings(exPaperSize) property:

```
Debug.Print Print1.ExecuteTemplate("Settings(0)")
```

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.*

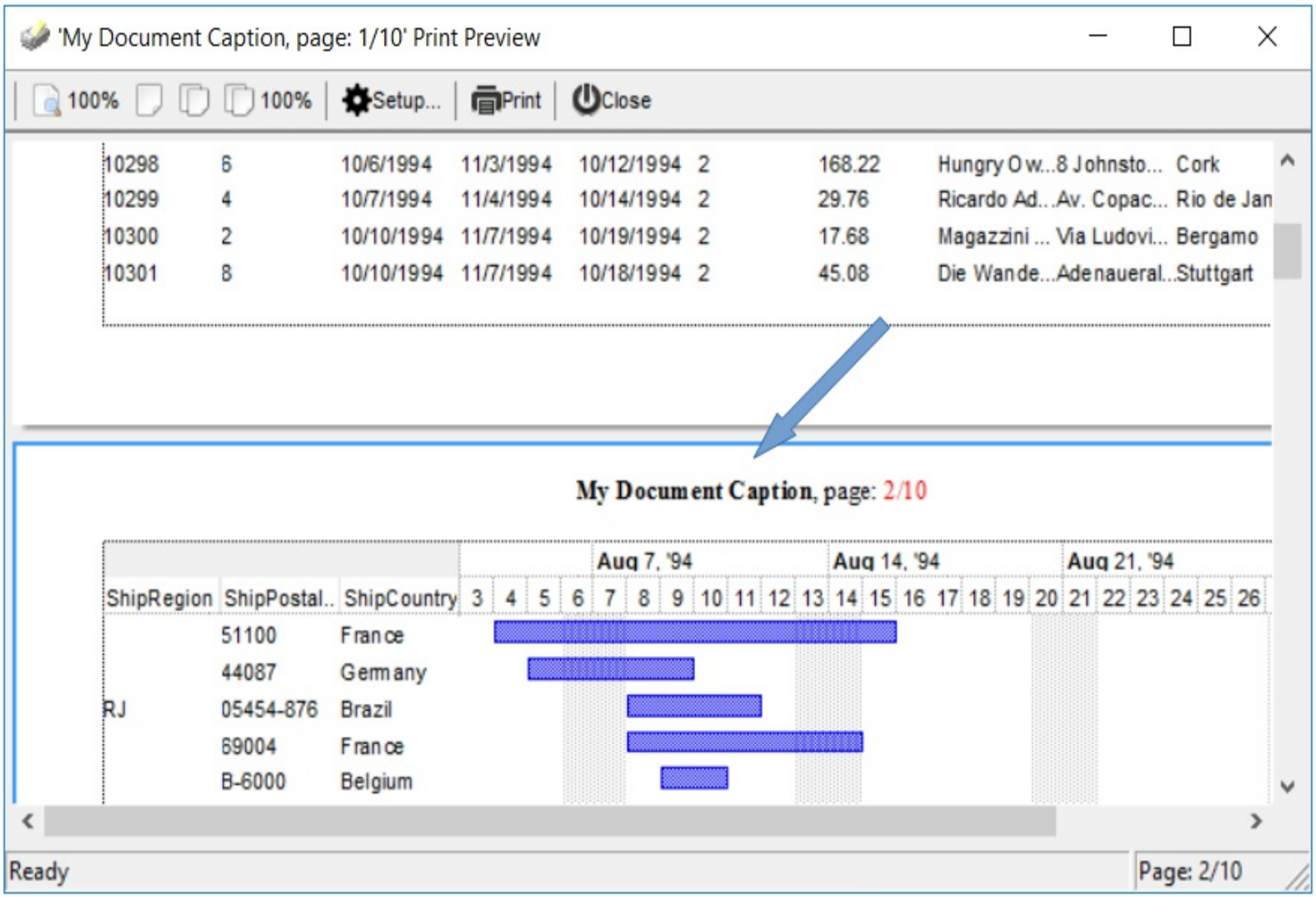
method ExPrint.ExtraCaption (Key as Variant, [Caption as Variant], [Position as Variant], [Alignment as Variant])

Adds or removes an additional caption on the page.

Type	Description
Key as Variant	A string or a numeric expression that indicates an unique key for the extra caption being added or removed. If missing, or empty, whole extra captions are removed.
Caption as Variant	A string expression that indicates the extra caption. If missing, or empty, the extra caption with the Key is removed.
Position as Variant	A PositionEnum expression that indicates the position of the extra caption in the page. If missing, the exFooter value is used (extra caption is displayed on the bottom of the page)
Alignment as Variant	A AlignmentEnum expression that indicates the alignment of the extra caption in the page. If missing, the exLeft value is used (extra caption is displayed on the left margin).

Use the ExtraCaption method to add an extra caption to the page. By default, only a single caption may be added to your page (Caption property, for instance, you can add an extra caption to the bottom of the page to print the current date). The ExtraCaption method may be called to add extra caption to any or all margins of the page. Use the Caption property to specify the name of the document being printed. Use the CaptionAlignment property to specify the alignment of the caption in the page. Use the CaptionPosition to specify the position of the caption in the page. Use the ShowPageNumbers property to specify whether the document displays or hides the page number. Use the Font property to assign a different font for the caption printed on the document. Use the <br> to break lines in the caption, and so you can display multiple-lines captions. Use the Images/ReplaceIcon method to add new icons to the control. Use the HTMLPicture property to add custom sized pictures.

The following screen shot shows where he caption mat be displayed:



Starting from the version 15.0, the Caption property and Caption parameter of ExtraCaption method supports expressions, that can defines the HTML caption of each page based on different fields such as index of the page, total number of pages, the index of the object being printed / previewed, the index of the page relative to the object being printed, and the number of relative pages. *If the Caption's expression is not valid, the Caption itself is displayed as HTML, be replacing the* ***<%page%>*** *and* ***<%count%>*** *fields with the current page, and number of pages, as explained on the bottom of the this page.*

The Caption property / parameter supports the following keywords:

- **object**, indicates the index of the object being printed, as a numeric value. The object, opages and opage fields are useful in case you are printing multiple-objects at the same time, using the PrintExts method. For instance, PrintExts = Array(Grid1, Grid2) specifies that two controls should be sent to the printer, and in this case the **object** will indicate 0 for the Grid1, and 1 for the Grid2. For instance, Caption = "`<b>` + (object array (`<s>first-grid`,`<i>second-grid`))" defines an expression-caption that displays the caption "first-grid" in bold and strikeout, for the first object of the PrintExts, and displays the "second-grid" in bold and italic, for the second object of the PrintExts method.

- **opages** defines the number of pages the current object is being previewed / printed, as a numeric value.
- **opage** indicates the relative-index of the page being printed / previewed, as a numeric value.
- **pages** defines the total number of pages being previewed / printed, as a numeric value.
- **page** which specifies the index of the current page, from the total number of pages, as a numeric value. For instance, ExtraCaption("logo","page mod 2 ? `` : `<img>logo</img>`") adds a logo picture on every second-page.

This property/method supports predefined constants and operators/functions as described here.

For instance, Caption can be (as a valid-expression):

- "`My Document Title`", the "My Document Title" is what you get on every page. Please pay attention that we used the ` character to quote the string.
- "`My Document <b>Title</b>`", the "My Document **Title**" is what you get on every page. Please pay attention that we used the ` character to quote the string.
- "`Page: ` + page + ` from ` + pages", displays Page: 1 from 100, Page: 2 from 100, on each page.
- "page mod 2 ? `odd` : `even`", displays odd on odd pages, and even on even pages.
- "page mod 2 ? `` : `<img>logo</img>`", displays the logo picture on every second page (even pages). The HTMLPicture property adds or replaces a picture in HTML captions.
- "page = pages ? `last` : ((page - 1) array(`first`,`second`,`third`,`forth`,`fifth`,`sixth`,`seventh`,`eighth`,`ninth`,`tenth`))", displays the last for the last page, first for the page with the index 1, second for the page with the index 2, and so on.

For instance, Caption can be (as a non-expression):

- "My Document Title", the "My Document Title" is what you get on every page
- "My Document Title", the "My Document **Title**" is what you get on every page
- "Page: <%page%> from <%count%>", displays Page: 1 from 100, Page: 2 from 100, on each page

Currently, the ExtraCaption property may include the following built-in HTML tags:

- ** ... ** displays the text in **bold**
- ** ... ** displays the text in *italics*
- **<u> ... </u>** underlines the text
- **<s> ... </s>** ~~Strike~~-through text
- **<a id;options> ... ** displays an anchor element that can be clicked. An anchor is a

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

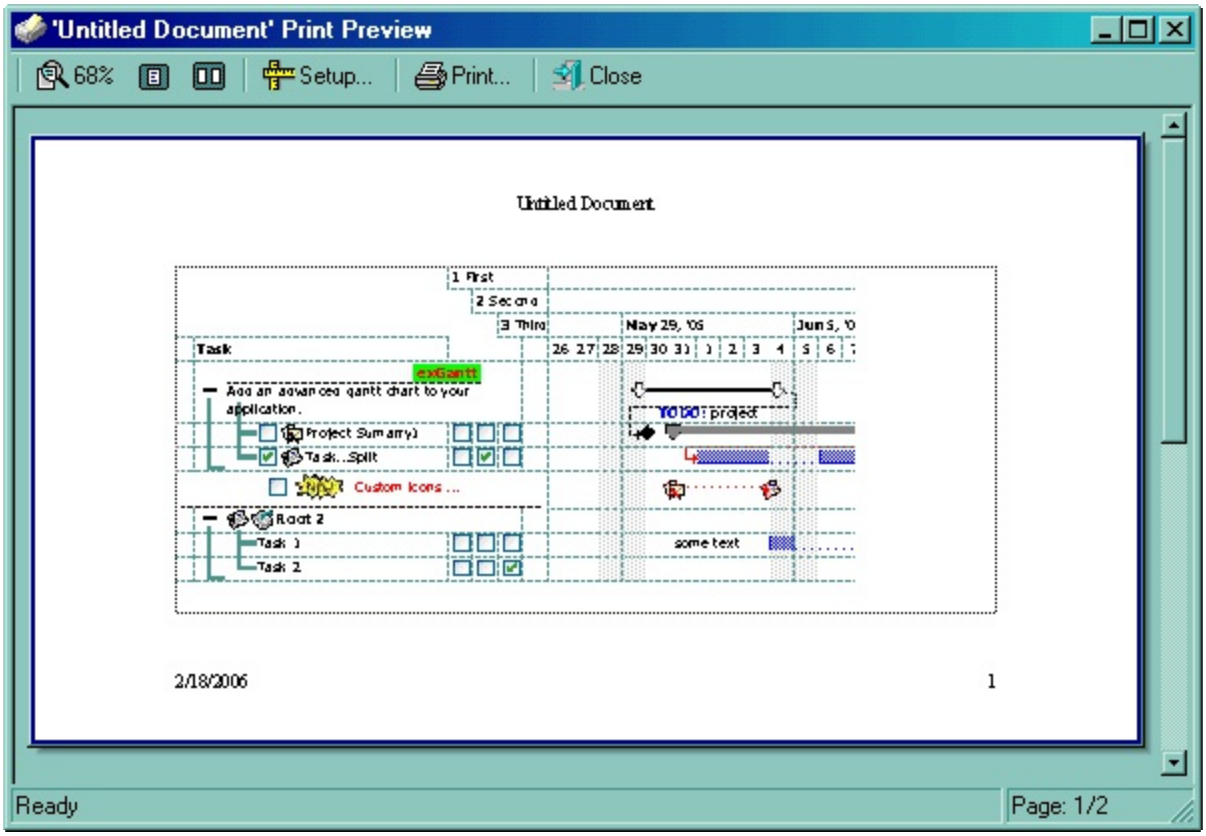
- ** ... ** displays portions of text with a different font and/or different size. For instance, the "bit" draws the bit text using the Tahoma font, on size 12 pt. If the name of the font is missing, and instead size is present, the current font is used with a different size. For instance, "bit" displays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

- **&** glyph characters as **&** (&), **<** (<), **>** (>), **&qout;** (") and **&#number;** (the character with specified code), For instance, the € displays the EUR character. The **&** ampersand is only recognized as markup when it is followed by a known letter or a #character and a digit. For instance if you want to display bold in HTML caption you can use bold
- **<off offset> ... </off>** defines the vertical offset to display the text/element. The offset parameter defines the offset to display the element. This tag is inheritable, so the offset is keep while the associated </off> tag is found. You can use the <off offset> HTML tag in combination with the to define a smaller or a larger font to be displayed. For instance: "Text with <**off** 6>subscript" displays the text such as: Text with $_{\text{subscript}}$ The "Text with <**off** -6>superscript" displays the text such as: Text with ^{subscript}
- **<gra rrggbb;mode;blend> ... </gra>** defines a gradient text. The text color or <fgcolor> defines the starting gradient color, while the rr/gg/bb represents the red/green/blue values of the ending color, 808080 if missing as gray. The mode is a value between 0 and 4, 1 if missing, and blend could be 0 or 1, 0 if missing. The HTML tag can be used to define the height of the font. Any of the rrggbb, mode or blend field may not be specified. The <gra> with no fields, shows a vertical gradient color from the current text color to gray (808080). For instance the "<**gra** FFFFFF;1;1>gradient-center</**gra**>" generates the following picture:
- **<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:

Also the ExtraCaption property supports the following predefined values:

- <%page%> specifies the current page
- <%count%> indicates the number of pages in the document.



The following VB sample adds an extra caption to the bottom of the page that shows the current date:

```
With Print1
        .ExtraCaption "Date", Date, EXPRINTLibCtl.PositionEnum.exFooter,
EXPRINTLibCtl.AlignmentEnum.exLeft
End With
```

The following VB sample adds three extra fields to the page:

```
With Print1
    .ExtraCaption "A", "Header-Right", EXPRINTLibCtl.PositionEnum.exHeader,
EXPRINTLibCtl.AlignmentEnum.exRight
    .ExtraCaption "B", "Header-Left", EXPRINTLibCtl.PositionEnum.exHeader,
EXPRINTLibCtl.AlignmentEnum.exLeft
    .ExtraCaption "C", "Footer-Left", EXPRINTLibCtl.PositionEnum.exFooter,
EXPRINTLibCtl.AlignmentEnum.exLeft
End With
```

The following C++ sample adds an extra caption to the bottom of the page that shows the

current date:

```
COleDateTime date = COleDateTime::GetCurrentTime();
COleVariant vtDate;
V_VT( &vtDate ) = VT_DATE;
V_DATE( &vtDate ) = date.m_dt;
m_print.ExtraCaption( COleVariant("Date"), vtDate,
COleVariant((long)1/*EXPRINTLibCtl.PositionEnum.exFooter*/),
COleVariant((long)0/*EXPRINTLibCtl.AlignmentEnum.exLeft*/) );
```

The following VB.NET sample adds an extra caption to the bottom of the page that shows the current date:

```
With AxPrint1
   .ExtraCaption("Date", DateTime.Today, EXPRINTLib.PositionEnum.exFooter,
EXPRINTLib.AlignmentEnum.exLeft)
End With
```

The following C# sample adds an extra caption to the bottom of the page that shows the current date:

```
axPrint1.ExtraCaption("Date", DateTime.Today, EXPRINTLib.PositionEnum.exFooter,
EXPRINTLib.AlignmentEnum.exLeft);
```

The following VFP sample adds an extra caption to the bottom of the page that shows the current date:

```
with thisform.Print1
   .ExtraCaption("Date",Date(),1,0)
endwith
```

property ExPrint.Font as IFontDisp

Retrieves or sets the control's font.

Type	Description
IFontDisp	A Font object that indicates the font to paint the Caption, PageNumbers or ExtraCaption properties

Use the Font property to change the font to display the Caption, PageNumbers or ExtraCaption properties. The control being printed defines its own font, using the control's font. Use the <b>, <i>, <u>, or <s> built-in HTML tags, to display portions of text with different font attributes.

property ExPrint.Foreground as Long

Brings the Preview window on the foreground and activates it.

Type	Description
Long	A long expression that specifies the z-order of the Preview window.

By default, the Foreground property is 0 (Places the Preview window at the top of the Z order.). The Foreground property has effect only at next Preview call, so always call the Foreground property before Preview method. For instance, use the Foreground property on 1, on Clarion environment so the Preview window will be bring on top when the Preview method is called. The PreviewState property returns or sets the visual state of preview mainframe at runtime.

The Foreground property supports the following values:

- **0 (top)**, Places the Preview window at the top of the Z order (by default).
- **1 (bottom)**, Places the Preview window at the bottom of the Z order.
- **-1 (always-on-top)**, Places the Preview window above all non-topmost windows. The window maintains its topmost position even when it is deactivated.
- **-2 (no-topmost)**, Places the Preview window above all non-topmost windows (that is, behind all topmost windows). This flag has no effect if the window is already a non-topmost window.

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

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

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

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

For instance:

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

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

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

This property/method supports predefined constants and operators/functions as described here.

property ExPrint.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 ToolBarFormat property to add new buttons, icons , pictures or any HTML caption to the eXPrint's toolbar. The ItemCaption property specifies the button's caption that may include icons, pictures if the <img> HTML built-in element is included. The Caption property specifies the name of the document, and may display icons or pictures as well, if <img> HTML tag is included. The ExtraCaption property adds custom captions to each page, and may display icons or pictures as well, if <img> HTML tag is included.

property ExPrint.hWnd as Long

Retrieves the handle of the print preview main frame.

Type	Description
Long	A long expression that indicates the window handle of the print preview main frame.

Use the hWnd property to get the window handle of the print preview main frame. Use the hWnd property to verify whether the print preview main frame is visible or hidden. The following sample displays the print preview main frame if it is hidden, it closes the print preview main frame if it is visible:

```
Private Sub Command1_Click()
   With Print1
   If .hWnd = 0 Then
      Set .PrintExt = Me
      .Preview
   Else
      Set .PrintExt = Nothing
   End If
   End With
End Sub
```

method ExPrint.Images (Handle as Variant)

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

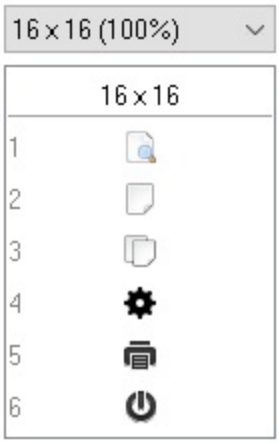
Type	Description
Handle as Variant	The Handle parameter can be: • A string expression that specifies the ICO file to add. The ICO file format is an image file format for computer icons in Microsoft Windows. ICO files contain one or more small images at multiple sizes and color depths, such that they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

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

Use the Images method to attach a image list to the control. At runtime, the user can use the Images and ReplaceIcon method to change the Images collection. The Images collection is 1 based. The Images collection does not affect the content of a page, only the toolbar or the margins of the paper. For instance, the ExPrint1.Caption = "<img>5<img> Print" will print the 5'th icon and the Print label on the each page. Use the HTMLPicture property to display custom size pictures. The ImageSize property defines the size (width/height) of the icons within the control's Images collection.

Use the ToolBarFormat property to add new buttons, icons , pictures or any HTML caption to the eXPrint's toolbar. The ItemCaption property specifies the button's caption that may include icons, pictures if the <img> HTML built-in element is included. The Caption property specifies the name of the document, and may display icons or pictures as well, if <img> HTML tag is included. The ExtraCaption property adds custom captions to each page, and may display icons or pictures as well, if <img> HTML tag is included.

By default, the control loads the following icons to be shown on the control's toolbar:



property ExPrint.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 ExPrint.ItemCaption(Item as ItemCaptionEnum) as String

Specifies a value that indicates the caption for specified item.

Type	Description
Item as ItemCaptionEnum	An ItemCaptionEnum expression that indicates the caption/field being changed.
String	A string expression that indicates the caption or field's value.

Use the ItemCaption property to change captions in the Preview window. The ItemToolTip property specifies the button's tooltip. Use the ToolBarFormat property to add new buttons, icons, pictures or HTML captions to eXPrint's toolbar. The control fires the Click event when the user clicks a button in the eXPrint's toolbar. For exToolBar... values (any value greater that 100), # character splits the button's label and it's identifier (SelectedID parameter). For instance, if the ItemCaption(200) = "Letter#1", the button with the identifier 200 displays the "Letter" label, while the 1 is carried to SelectedID parameter of the Click event when it is fired. If the ItemCaption property includes vbCrLF ("\r\n" sequence), the associated buttons displays a drop down field.

The ItemCaption property supports the following HTML elements:

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

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

HTML tag can be used to define the height of the font. Any of the rrggbb, mode or blend field may not be specified. The <gra> with no fields, shows a vertical gradient color from the current text color to gray (808080). For instance the "<**gra** FFFFFF;1;1>gradient-center</**gra**>" generates the following picture:

- **<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:

The following sample VB changes the caption for "Close" button:

```
With Print1
   .ItemCaption(exClose) = "<img>6</img>Inchide"
   .ItemCaption(exPrint) = "<img>5</img>Listeaza"
   Set .PrintExt = XXX.Object
   .Preview
End With
```

Where XXX is the object being printed.

property ExPrint.ItemToolTip(Item as ItemCaptionEnum) as String

Specifies a value that indicates the tooltip for specified item.

Type	Description
Item as ItemCaptionEnum	An ItemCaptionEnum expression that indicates the caption/field being changed.
String	A string expression that indicates the button's tooltip.

Use the ToolBarFormat property to add new buttons, icons, pictures or HTML captions to eXPrint's toolbar. The control fires the Click event when the user clicks a button in the eXPrint's toolbar. The ItemToolTip property supports the following HTML elements:

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

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

Specifies the document's options.

Type	Description
Variant	A Variant expression that indicates a document extra data used when document is printing or previewing.

The Options property is passed to all methods of IPrintExt interface. The Options property has affect only in the object's preview. Use the Settings property to change print settings like paper size, orientation and so on.

property ExPrint.PageFrameColor as Color

Specifies the color of frame to be shown on printed pages.

Type	Description
Color	A Color expression that specifies the color to show the frame on printed pages.

By default, the PageFrameColor property is black. The PageFrameStyle property indicates whether the margins of the page are shown on the paper. The PageFrameColor property specifies the color to show the margins on the printed paper.

property ExPrint.PageFrameStyle as PageFrameStyleEnum

Specifies the style of frame to be shown on printed pages.

Type	Description
PageFrameStyleEnum	A PageFrameStyleEnum expression that specifies the style of the frame to be shown on the page.

By default, the PageFrameStyle property is exNoPageFrame, so the printed page, will not show its margins. Use the PageFrameStyle property to display the margins of the page on printed paper. The PageFrameColor property specifies the color to show the margins on the printed paper.

property ExPrint.PageNumberFormat as String

Specifies the format to display the number of page.

Type	Description
String	A String expression that defines the format to display the page number on pages.

By default, the PageNumberFormat property is "<%page%>" which means that the current number of page is being displayed. The PageNumberFormat property specifies the format to display the page number. If the property PageNumberFormat is empty the page number are not shown. Use the PageNumbersAlignment property to align the page number field in the header or footer of the page. Use the ShowPageNumbers property to hide the page number field. Use the PageNumbersPositon property to indicate whether the page number field is displayed in the header or footer of the page. Use the Caption property to specify a caption for the document being printed on all pages. Use the ExtraCaption property to specify a additional captions for the document being printed on the pages. Use the StartPageNumber property to define the start number for page numbering.

The PageNumberFormat property supports the following predefined values:

- **<%page%>** specifies the current page
- **<%count%>** indicates the number of pages in the document.

Also the PageNumberFormat property supports the HTML format as:

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

color in hexa values.

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

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

- **<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:

For instance, the "Page <%page%> of <%count%>" format displays pages as Page x from **y**.

property ExPrint.PageNumbersAlignment as AlignmentEnum

Specifies the alignment of page numbers in the document.

Type	Description
AlignmentEnum	An AlignmentEnum property that specify the alignment of the page numbers in the page.

Use the PageNumbersAlignment property to align the page number field in the header or footer of the page. Use the PageNumbersPosition property to specify whether the page number field is displayed on the header or footer of the page. Use the ShowPageNumbers property to hide the page number field. Use the PageNumberFormat property specifies the format to display the page number on each page.

property ExPrint.PageNumbersPosition as PositionEnum

Specifies the position of page numbers in the document.

Type	Description
PositionEnum	A PositionEnum expression that indicates whether the page number field is displayed on the header or footer of the printed page.

Use the PageNumbersPositon property to indicate whether the page number field is displayed in the header or footer of the page. Use the PageNumbersAlignment property to align the page number field in the header or footer of the page. Use the ShowPageNumbers property to hide the page number field. Use the PageNumberFormat property specifies the format to display the page number on each page.

property ExPrint.PageOrientation as PageOrientationEnum

Specifies the default page's orientation.

Type	Description
PageOrientationEnum	A PageOrientationEnum expression that indicates the page's orientation.

The PageOrientation property specifies the page orientation. The PageOrientation property must be called before any call of Settings property. Use the Settings property to change settings in the printer dialog.

The following sample changes specifies an A4 Landscape page, and specifies that margins must be measured in millimeters:

```
With Print1
   Set .PrintExt = Grid1.Object
   .PageOrientation = exLandscape
   .Settings(exDisplayInch) = 1
   .Settings(exPaperSize) = 9
   .Preview
End With
```

property ExPrint.PageRange as String

Specifies the pages being printed.

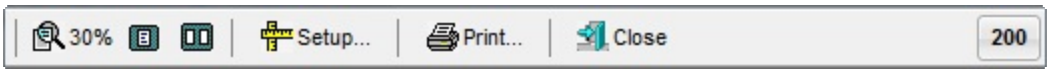
Type	Description
String	A String expression that specifies the pages being printed. It supports page numbers and/or page ranges separated by commas. For instance: "1,5-7,9-". If empty, all pages are printed.

By default, the PageRange property is empty, in other words the print command will send all pages to the printer. The pages being printed shows in white, while the pages being not printed shows as disabled as in the following screen shots. You can print only a selection from a page by selecting the area using the RIGHT click, and moving the cursor. You can specify a page to be not printed by clicking the page and keeping the CTRL key down, or if it is already not-printed, toggle it's state. If you require only a single page to be printed, you can click the page and press CTRL + SHIFT key. Use the ToolBarFormat property to add new controls to the preview's toolbar. The Click event notifies your application once the user clicks a button in the preview's toolbar. Use the Refresh event to update the buttons' captions when pages are refreshed. The ItemCaption property specifies the button's caption in the preview's toolbar.

The following VB sample adds a button PageRange to the preview's toolbar, with the identifier 200

```
Print1.ToolBarFormat = Print1.ToolBarFormat + ",|,200"
```

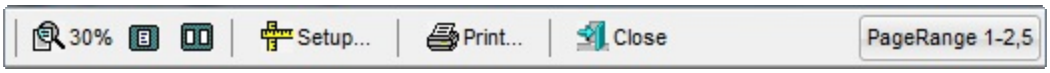
At this point, the control's Preview displays a 200 button in the right side of the toolbar as shown bellow:



Now update the newly button's caption with "PageRange" caption, and eventually the current page range value using the Refresh event as follows:

```
Private Sub Print1_Refresh()
  With Print1
    .ItemCaption(200) = "PageRange"
    If Len(.PageRange) > 0 Then
      .ItemCaption(200) = .ItemCaption(200) + " " + .PageRange
    End If
  End With
```

```
End Sub
```

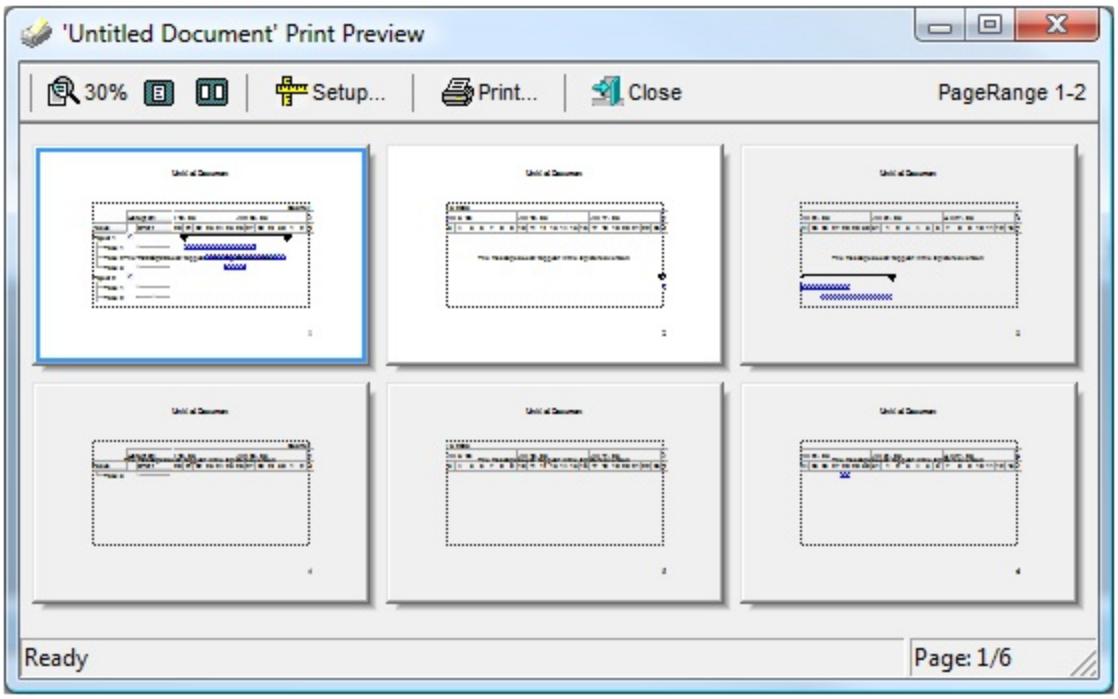


Next, we need to handle the Click event, so once the button is clicked we can select a new selection for pages being printed, using the PageRange property:

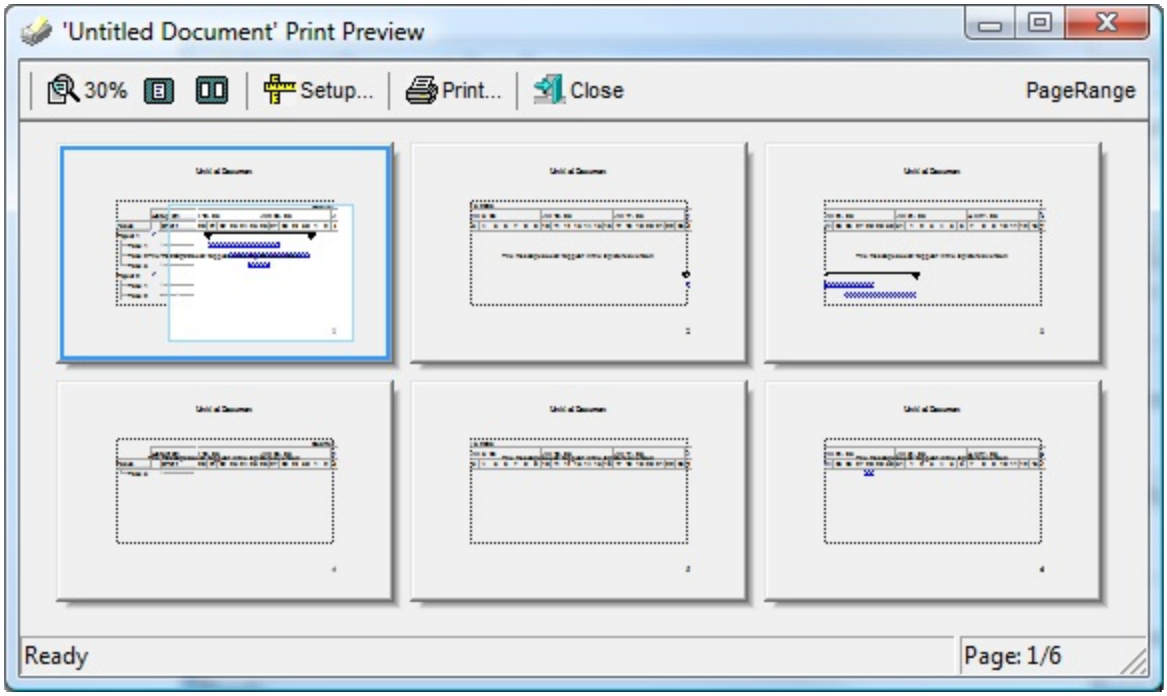
```
Private Sub Print1_Click(ID As Long, SelectedID As Long)
   If (ID = 200) Then
      Print1.PageRange = InputBox("Specifies the page range", "PageRange",
Print1.PageRange)
      Print1.Refresh
   End If
End Sub
```

Using the Refresh event, the PageRange button is updated as soon as you click a page while the CTRL or CTRL+SHIFT combination is pressed.

The following screen shot shows the pages to be printed in white (1,2), while the others being excluded as disabled:



The following screen shot shows the user selection to be printed (white rectangle) (RIGHT click the cursor and moves the cursor to select the area):



In C# the sample will be like:

```
private void exprint1_RefreshEvent(object sender)
{
   exontrol.EXPRINTLib.ItemCaptionEnum idPageRange =
(exontrol.EXPRINTLib.ItemCaptionEnum)200;
   exprint1.set_ItemCaption(idPageRange, "PageRange");
   if (exprint1.PageRange.Length > 0)
      exprint1.set_ItemCaption(idPageRange, exprint1.get_ItemCaption(idPageRange) + " "
+ exprint1.PageRange);

}

private void exprint1_Click(object sender, int ID, int SelectedID)
{
   if (ID == 200)
   {
      exprint1.PageRange = Microsoft.VisualBasic.Interaction.InputBox("Specifies the page
range", "PageRange", exprint1.PageRange, 0, 0);
      exprint1.Refresh();
   }
}
```

In VB/NET the sample will be like:

```
Private Sub Exprint1_RefreshEvent(ByVal sender As System.Object) Handles
Exprint1.RefreshEvent
    With Exprint1
        .set_ItemCaption(200, "PageRange")
        If Len(.PageRange) > 0 Then
            .set_ItemCaption(200, .get_ItemCaption(200) + " " + .PageRange)
        End If
    End With

End Sub

Private Sub Exprint1_Click(ByVal sender As System.Object, ByVal ID As System.Int32, ByVal
SelectedID As System.Int32) Handles Exprint1.Click
    If (ID = 200) Then
        With Exprint1
            .PageRange = InputBox("Specifies the page range", "PageRange", .PageRange)
            .Refresh()
        End With
    End If
End Sub
```

property ExPrint.PagesCount as Long

Returns the number of pages.

Type	Description
Long	A long expression that specifies the number of pages being loaded.

The PagesCount property specifies the number of pages being displayed. The number of loaded pages is also displayed in the right part of the preview's status bar as "Page: 1/8". The PageRange property specifies the pages, or the range of pages being printed. A page being not printed shows disabled in the print preview. The object being printed determines the number of paged being loaded.

method ExPrint.Preview ()

Invokes the print preview main frame.

Type	Description

Use the Preview method to invoke the Print Preview Main Frame, that allows you to preview the object being printed on the paper. Use the hWnd property to get the window handle of the Print Preview Main Frame. The Preview method fails if the PrintExt/PrintExts property is not set before. The PageRange property specifies the pages being printed. A page that is not printed shows as disabled. The ToolBarFormat property formats the layout of the preview's toolbar. The control fires the Click event when the user clicks a button in the preview's toolbar. Use the ItemCaption property to specify the caption being displayed in the preview's toolbar. The ItemToolTip property specifies tooltip being displayed when the cursor hovers a button in the preview's toolbar. Use the AutoRelease property on False, to prevent closing the Preview's window when releasing the Print object. The Brings the Preview window on the foreground and activates it. The Foreground property brings the Preview window on the foreground and activates it. The PreviewState property returns or sets the visual state of preview mainframe at runtime.

The ExPrint component provides Print and Print Preview features for components like: eXGantt, eXG2antt, eXMLGrid, eXGrid, eXTree, eXList, eXCalendar, eXComboBox, eXPropertiesList, eXEdit, eXFileView, eXOrgChart and so on.

Tip You can copy and paste the page's content to word, excel, mspaint, ... (applications that supports OLE Clipboard Mechanism) by pressing the **CTRL + C** keys combination in preview mode (click a page in preview, so it gets selected, and then press CTRL + C, so the clipboard will contain the picture of current page).

Tip In preview mode, you can select the pages to be printed, by keeping the **CTRL** key down while **click** the page. If the page's background is white, it will be sent to the printer, else it will not.

Tip In preview mode, you can select only a part of the page to be printed, by **RIGHT clicking** the page and drag the area to be printed. The white part of the page, will be sent to the printer, as soon as you click the Print command (right click the page where will be the starting point and drag the mouse to define the ending point of the part to be printed

Tip If the ShowMargins property is True, you can **click** the "**One Page**" command button in the toolbar, and so the page will show the margins, so the user can resizes the margins of the paper at runtime.

Tip **Double clicking** the **margin** of the paper displays the Setup dialog.

Tip **CTRL + Double clicking** the **margin** of the page, will make the margin to be the same as the opposite margin.

The following samples show in different programming languages how to provide the print-preview capabilities for eXG2antt control:

- **Access**. The Object method of the G2antt1 object gets the native or the original component.

```
Private Sub callPreview()
   With Print1
      Set .PrintExt = G2antt1.Object
      .preview
   End With
End Sub
```

- **VB6**. The Object method of the G2antt1 object gets the native or the original component.

```
Private Sub callPreview()
   With Print1
      Set .PrintExt = G2antt1.Object
      .Preview
   End With
End Sub
```

- **VFP**. The Object method of the G2antt1 object gets the native or the original component.

```
with thisform.Print1
   .PrintExt = thisform.G2antt1.Object
   .Preview
endwith
```

- **VB.NET** (ActiveX version). The GetOcx method of the G2antt1 object gets the native or the original component.

```
Private Sub callPreview()
```

```
  With AxPrint1
    .PrintExt = AxG2antt1.GetOcx()
    .Preview()
  End With
End Sub
```

- **VB.NET** (/NET or /WPF Assembly).

```
Private Sub callPreview()
  With Exprint1
    .PrintExt = Exg2antt1
    .Preview()
  End With
End Sub
```

- **C#** (ActiveX version). The GetOcx method of the G2antt1 object gets the native or the original component.

```
private void callPreview()
{
  axPrint1.PrintExt = axG2antt1.GetOcx();
  axPrint1.Preview();

}
```

- **C#** (/NET or /WPF Assembly).

```
private void callPreview()
{
  exprint1.PrintExt = exg2antt1;
  exprint1.Preview();
}
```

- **C++ (6.0)**. The m_print member of the CWindowMFCDlg class, is of CExPrint type (which has been defined by the class wizard). The m_g2antt member is of CG2antt type that has been defined by the class wizard. The GetControlUnknown method retrieves a pointer to IUnknown interface being implemented by the original component, which is the pointer required by the PrintExt method before calling the Preview or DoPrint method of the eXPrint component.

```
void CWindowMFCDlg::callPreview()
{
   m_print.SetPrintExt( m_g2antt.GetControlUnknown() );
   m_print.Preview();
}
```

- **C++ (2005,2008)**. The m_print member of the CWindowMFCDlg class, is of CExPrint type (which has been defined by the class wizard). The m_g2antt member is of CG2antt type that has been defined by the class wizard. The GetControlUnknown method retrieves a pointer to IUnknown interface being implemented by the original component, which is the pointer required by the PrintExt method before calling the Preview or DoPrint method of the eXPrint component.

```
void CWindowMFCDlg::callPreview()
{
   m_print.put_PrintExt( m_g2antt.GetControlUnknown() );
   m_print.Preview();
}
```

- **C++ Builder (2009)**. The DefaultDispatch function of the TOleControl retrieves the original object, so it can be passed to PrintExt function of the eXPrint in order to print or print preview the component, as seen in the callpreview function:

```
void TForm1::callPreview()
{
   Print1->PrintExt = G2antt1->DefaultDispatch;
   Print1->Preview();
}
```

- **Delphi (2009).** The DefaultDispatch function of the TOleControl retrieves the original object, so it can be passed to PrintExt function of the eXPrint in order to print or print preview the component, as seen in the callpreview function:

```
procedure callPreview(P : TPrint; G : TG2antt);
begin
with P do
begin
 PrintExt := G.DefaultDispatch;
 Preview();
end;
```

```
end;
```

```
callPreview( Print1, G2antt1 );
```

- **Clarion.** Code under ?PrintThis button, with some properties:

```
 ?Print{'Settings(10)'} = 1
 ?Print{'PrintExt'} = ?Gantt{PROP:Object}            ! If we going to print ExGantt
 ?Print{'AsScreen'} = False                           ! Sometimes makes problems under
Terminal services
 ?Print{'PageOrientation'} = 2
 ?Print{'Caption'} = 'Some text'
 ?Print{'Settings(6)'} = 1500
 ?Print{'Settings(7)'} = 1500
 ?Print{'Settings(8)'} = 1500
 ?Print{'Settings(9)'} = 1500
 ?Print{'Font.Name'}   = 'Arial'
 ?Print{'Font.Size'}   = 9
 ?Print{'Preview'}
```

The following VB sample implements the IPrintExt interface:

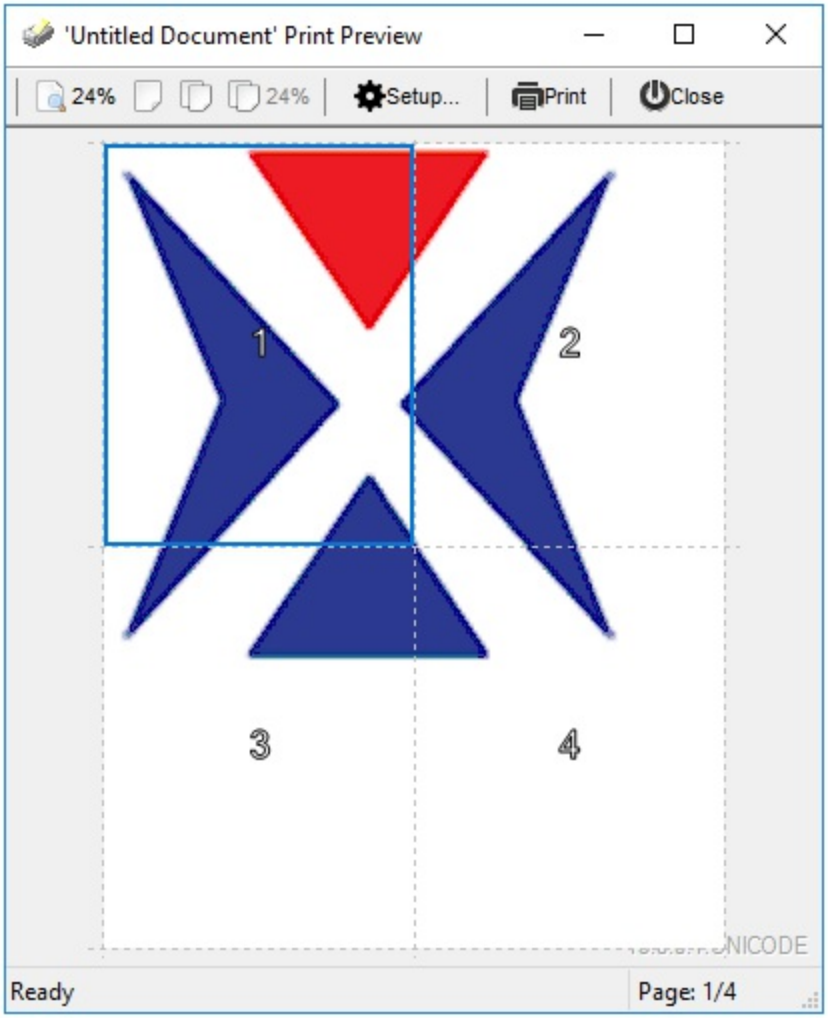
```
Option Explicit
Implements IPrintExt

Private Sub Command1_Click()
   With Print1
      Set .PrintExt = Me
      .Preview
   End With
End Sub

Private Sub IPrintExt_DrawPage(ByVal Options As Variant, ByVal hDC As Long, ByVal Page
As EXPRINTLibCtl.IPage, pbContinue As Boolean)
End Sub

Private Property Get IPrintExt_PageCount(ByVal Options As Variant) As Long
   IPrintExt_PageCount = 1
End Property
```

property ExPrint.PreviewSettings(Field as PreviewFieldsEnum) as Variant

Sets or gets a value that defines a setting for preview mode.

Type	Description
Field as PreviewFieldsEnum	A PreviewFieldsEnum expression that defines the setting to be defined.
Variant	A VARIANT expression that specifies the value of giving field.

The PreviewSettings property defines different settings for the control's Print Preview mode. For instance you can use the PreviewSettings(exPreviewShowCompact) property to show the pages in compact mode. The following screen shot shows how the preview is displayed in compact mode:



property ExPrint.PreviewState as PreviewStateEnum

Returns or sets the visual state of preview mainframe at runtime.

Type	Description
PreviewStateEnum	A PreviewStateEnum expression that specifies the visual state of preview mainframe at runtime.

By default, The PreviewState property is exPreviewStateNormal. The PreviewState property returns or sets the visual state of preview mainframe at runtime. For instance, you can use the PreviewState property to programmatically maximize the preview window. The Foreground property brings the Preview window on the foreground and activates it. Use the Preview method to invoke the Print Preview Main Frame, that allows you to preview the object being printed on the paper. The PrintExt/PrintExts property specifies the object being previewed / printed.

The following VB sample shows how you can programmatically maximize the Print and Print preview window:

```
Set p = CreateObject("Exontrol.Print")
With p
   .PrintExt = G2antt1.Object
   .PreviewState = exPreviewStateMaximized
   .Preview
End With
```

property ExPrint.Printers as String

Retrieves a list of installed printers.

Type	Description
String	A String expression that indicates the list of installed printers. Each printer is found on a new line.

Use the Printers property to list the printers installed on the system. The Printers property retrieves the list of printers separated by new line and carriage characters (\r\n). The exPrinterName property gets the name of the default printer.

property ExPrint.PrintExt as IUnknown FAR*

Specifies an object that implements the IPrintExt interface.

Type	Description
IUnknown FAR*	An object that implements the IPrintExt interface. Most of our UI components implements the IPrintExt interface so automatically provides the print and print-preview capabilities.

If you have an object that needs to provide Print and Print Preview capabilities the object needs to implement the IPrintExt interface. The ExPrint component communicates with an object being printed using the IPrintExt interface. The methods like Preview and DoPrint fail if the PrintExt property is not set. You can use the PrintExts property to send multiple objects/components to the same document. Use the PrintExt property to send a single object to the printer.

The ExPrint component provides Print and Print Preview features for components like: eXGantt, eXG2antt, eXMLGrid, eXGrid, eXTree, eXList, eXCalendar, eXComboBox, eXPropertiesList, eXEdit, eXFileView, eXOrgChart, eXSchedule, eXPivot and so on.

The following samples show in different programming languages how to provide the print-preview capabilities for eXG2antt control:

- **Access**. The Object method of the G2antt1 object gets the native or the original component.

```
Private Sub callPreview()
  With Print1
    Set .PrintExt = G2antt1.Object
    .preview
  End With
End Sub
```

- **VB6**. The Object method of the G2antt1 object gets the native or the original component.

```
Private Sub callPreview()
  With Print1
    Set .PrintExt = G2antt1.Object
    .Preview
  End With
```

```
End Sub
```

- **VFP**. The Object method of the G2antt1 object gets the native or the original component.

```
with thisform.Print1
   .PrintExt = thisform.G2antt1.Object
   .Preview
endwith
```

- **VB.NET** (ActiveX version). The GetOcx method of the G2antt1 object gets the native or the original component.

```
Private Sub callPreview()
   With AxPrint1
      .PrintExt = AxG2antt1.GetOcx()
      .Preview()
   End With
End Sub
```

- **VB.NET** (/NET or /WPF Assembly).

```
Private Sub callPreview()
   With Exprint1
      .PrintExt = Exg2antt1
      .Preview()
   End With
End Sub
```

- **C#** (ActiveX version). The GetOcx method of the G2antt1 object gets the native or the original component.

```
private void callPreview()
{
   axPrint1.PrintExt = axG2antt1.GetOcx();
   axPrint1.Preview();


}
```

- **C#** (/NET or /WPF Assembly).

```
private void callPreview()
{
   exprint1.PrintExt = exg2antt1;
   exprint1.Preview();
}
```

- **C++ (6.0)**. The m_print member of the CWindowMFCDlg class, is of CExPrint type (which has been defined by the class wizard). The m_g2antt member is of CG2antt type that has been defined by the class wizard. The GetControlUnknown method retrieves a pointer to IUnknown interface being implemented by the original component, which is the pointer required by the PrintExt method before calling the Preview or DoPrint method of the eXPrint component.

```
void CWindowMFCDlg::callPreview()
{
   m_print.SetPrintExt( m_g2antt.GetControlUnknown() );
   m_print.Preview();
}
```

- **C++ (2005,2008)**. The m_print member of the CWindowMFCDlg class, is of CExPrint type (which has been defined by the class wizard). The m_g2antt member is of CG2antt type that has been defined by the class wizard. The GetControlUnknown method retrieves a pointer to IUnknown interface being implemented by the original component, which is the pointer required by the PrintExt method before calling the Preview or DoPrint method of the eXPrint component.

```
void CWindowMFCDlg::callPreview()
{
   m_print.put_PrintExt( m_g2antt.GetControlUnknown() );
   m_print.Preview();
}
```

- **C++ Builder (2009)**. The DefaultDispatch function of the TOleControl retrieves the original object, so it can be passed to PrintExt function of the eXPrint in order to print or print preview the component, as seen in the callpreview function:

```
void TForm1::callPreview()
{
   Print1->PrintExt = G2antt1->DefaultDispatch;
   Print1->Preview();
```

```
}
```

- **Delphi (2009).** The DefaultDispatch function of the TOleControl retrieves the original object, so it can be passed to PrintExt function of the eXPrint in order to print or print preview the component, as seen in the callpreview function:

```
procedure callPreview(P : TPrint; G : TG2antt);
begin
with P do
begin
 PrintExt := G.DefaultDispatch;
 Preview();
end;
end;
```

```
callPreview( Print1, G2antt1 );
```

- **Clarion.** Code under ?PrintThis button, with some properties:

```
?Print{'Settings(10)'} = 1
?Print{'PrintExt'} = ?Gantt{PROP:Object}
?Print{'AsScreen'} = False
?Print{'PageOrientation'} = 2
?Print{'Caption'} = 'Some text'
?Print{'Settings(6)'} = 1500
?Print{'Settings(7)'} = 1500
?Print{'Settings(8)'} = 1500
?Print{'Settings(9)'} = 1500
?Print{'Font.Name'}   = 'Arial'
?Print{'Font.Size'}   = 9
?Print{'Preview'}
```

property ExPrint.PrintExts as Variant

Specifies a collection of objects that implement the IPrintExt interface.

Type	Description
Variant	An Object to be printed, or a safe array of Objects to send to the printer. An Object can be printed if it implements the IPrintExt interface. Most of our UI components implements the IPrintExt interface so automatically provides the print and print-preview capabilities.

The PrintExts property sends one or more objects to the printer, in a single document.

Please notice that starting from version 6.1, there are two methods to associate one or more objects, as listed:

- PrintExt property, prints a single object
- PrintExts property, prints one or more objects, in the same document.

Use the PrintExt property to print a single object into a single document. The ExPrint component communicates with an object being printed using the IPrintExt interface. The methods like Preview and DoPrint fail if the PrintExt/PrintExts property is not set. The ExPrint component provides Print and Print Preview features for components like: eXGantt, eXG2antt, eXMLGrid, eXGrid, eXTree, eXList, eXCalendar, eXComboBox, eXPropertiesList, eXEdit, eXFileView, eXOrgChart, eXSchedule, eXPivot and so on.

ExPrint/COM The following VB6 sample sends the Grid1 and Grid2 to the printer, in the same document:

```
With Print1
   .PrintExts = Array(Grid1, Grid2)
   .Preview
End With
```

The following VB6 sample sends the Grid1 to the printer:

```
With Print1
   .PrintExts = Grid1
   .Preview
End With
```

This sample is equivalent with:

```
With Print1
   Set .PrintExt = Grid1.Object
   .Preview
End With
```

ExPrint/NET The following VB.NET sample sends the Exgrid1 and Exgrid2 to the printer, in the same document:

```
With Exprint1
   .PrintExts = New Object() {Exgrid1, Exgrid2}
   .Preview()
End With
```

The following VB.NET sample sends the Exgrid1 to the printer:

```
With Exprint1
   .PrintExts = Exgrid1
   .Preview()
End With
```

method ExPrint.Refresh ()

Refreshes the print preview.

Type	Description

The Refresh method refreshes the pages for printing or previewing. The Refresh event is called once the pages are refreshed. For instance, if you are changing some properties of the printed object, you can call the Refresh to re-get the printed pages to reflect the new changes in the printed pages too. The Settings property specifies different settings for the current printer.

method ExPrint.ReplaceIcon ([Icon as Variant], [Index as Variant])

Adds a new icon, replaces an icon or clears the print'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. Use the HTMLPicture property to display custom size pictures. Use the ToolBarFormat property to add new buttons, icons , pictures or any HTML caption to the eXPrint's toolbar. The ItemCaption property specifies the button's caption that may include icons, pictures if the <img> HTML built-in element is included. The Caption property specifies the name of the document, and may display icons or pictures as well, if <img> HTML tag is included. The ExtraCaption property adds custom captions to each page, and may display icons or pictures as well, if <img> HTML tag is included.

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

i = ExPrint1.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 = ExPrint1.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:

ExPrint1.ReplaceIcon 0, i, i specifies the index of icon removed.

The following VB clears the control's icons collection:

ExPrint1.ReplaceIcon 0, -1

property ExPrint.RuntimeKey as String

Specifies a runtime key to be used for the component.

Type	Description
String	A String expression that specifies the control's runtime key.

The RuntimeKey property is required only if you are using the control to print your pages, not using any of our UI components, or when using the eXPrint/COM component in the Isolated Application.

Isolated applications are self-describing applications installed with manifests. Isolated applications can use both private assemblies and shared assemblies.

An application is considered fully isolated if all of its components are either shared side-by-side assemblies or private assemblies. It is called partially isolated if it uses some components that are not side-by-side assemblies. Note that if an application uses some components that are not side-by-side assemblies, or uses private assemblies, the application may be affected by the installation or removal of other applications on the system. For more information, see Side-by-side Assembly Sharing.

Developers are encouraged to design isolated applications and to update existing applications into isolated applications for the following reasons:

- Isolated applications are more stable and reliably updated because they are unaffected by the installation, removal, or upgrading of other applications on the system.
- Isolated applications can be designed so that they always run using the same assembly versions with which they were built and tested.
- Isolated applications can use functionality provided by the side-by-side assemblies made available by Microsoft. For more information, see Supported Microsoft Side-by-side Assemblies.
- Isolated applications are not tied to the shipping schedule of their side-by-side assemblies because applications and administrators can update the configuration after deployment without having to reinstall the application. This would not apply in the case where only one version of the assembly is being made available.
- A fully isolated application may be installed by using the xcopy command. Windows Installer can also be used to install an isolated application without impact to the registry. For more information, see Installation of Win32 Assemblies.

Shortly, the Isolated COM allows your application to use ActiveX components without having to register them.

property ExPrint.Settings(Field as FieldsEnum) as Variant

Sets or gets a value that indicates the value for specified field.

Type	Description
Field as FieldsEnum	A Field being changed
Variant	A long expression or a string expression that indicates the field's value.

Use the Settings property to initialize fields like Paper Size, Orientation, Margins, Number of Copies, Print Quality, and so on before previewing or printing the object. *If you intent to change the printer's name, you must call the Setting(exPrinterName) property before changing any other property.* Use the PageOrientation property or Settings(exPageOrientation) to specify the page's orientation. Use the Settings(exAllFields) property to save and restore all printer settings.

Use the Preview method to preview an object. The DoPrint method prints an object.

The following sample changes the PageSize field to A4 format, before previewing an exGrid control:

```
Private Sub Command1_Click()
   With Print1
      Set .PrintExt = Grid1.Object
      .Settings(exPaperSize) = 9
      .Preview
   End With
End Sub
```

The exDisplayInch value must be changed before any other Settings like in the following sample. The sample specifies an A4 Landscape format, and printer dialog displays the millimeters instead inches:

```
With Print1
   Set .PrintExt = Grid1.Object
   .PageOrientation = exLandscape
   .Settings(exDisplayInch) = 1
   .Settings(exPaperSize) = 9
   .Preview
End With
```

property ExPrint.ShowMargins as Boolean

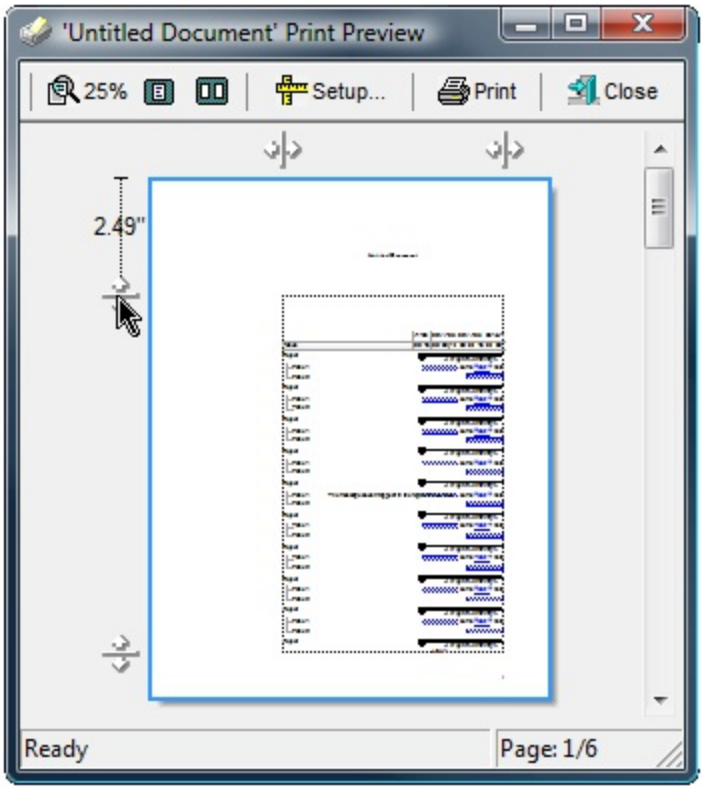
Retrieves or sets a value that specifies whether the page displays its margins so the user can resize the margins of the page at runtime.

Type	Description
Boolean	A boolean expression that specifies whether the margins of the page are shown so the user can resize the page's margins at runtime.

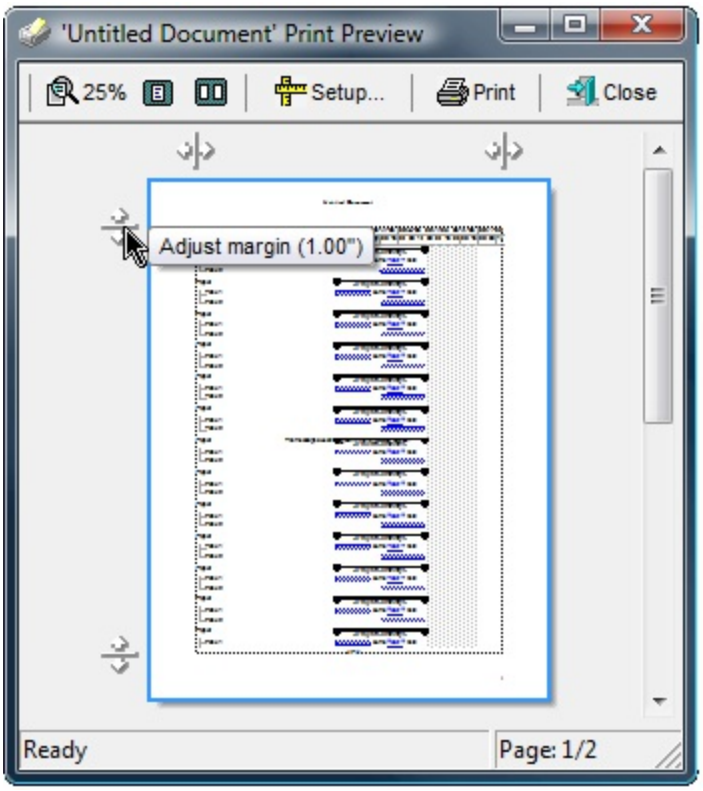
By default, the ShowMargins property is True, and so the page's margins are shown. The value of the page's margin is being shown when the cursor hovers it. User can click and resize the page margins at runtime as shown in the following screen shot. Use the Settings property to access settings from the printer's page, as exLeftMargin, exTopMargin, exRightMargin and exBottomMargin. Use the exDisplayInch setting to specify whether the control displays inches or millimeters. The Refresh event notifies your application once the preview/print job is being executed again.

The page margins are shown only if

- the ShowMargins property is true,
- a single page is displayed,
- and the entire page fits the client area of the print's maintframe.



The ItemToolTip property specifies the tooltip being shown when the cursor hovers the page's margins as shown in the following screen shot:


'Untitled Document' Print Preview
25%
Setup...
Print
Close
Adjust margin (1.00")
Ready
Page: 1/2

property ExPrint.ShowPageNumbers as Boolean

Specifies whether the page numbers are shown or hidden.

Type	Description
Boolean	A boolean expression that indicates whether the page number fiels is visible or hidden in the printed page.

The ShowPageNumbers property shows or hides the page number field in the page. Use the PageNumberFormat property specifies the format to display the page number on each page. Use the PageNumbersAlignment property to align the page number field in the page. Use the PageNumbersPositon property to indicate whether the page number field is displayed in the header or footer of the page. Use the Font property to assign a different font for the caption printed on the document. Use the Images/ReplaceIcon method to add new icons to the control. Use the HTMLPicture property to add custom sized pictures.

property ExPrint.StartPageNumber as Long

Specifies the number to start page numbering.

Type	Description
Long	A long expression that specifies the number to start page numbering.

By default, the StartPageNumber property is 1 and it means that the page number starts at 1. Use the StartPageNumber property to change the number to start page numbering. The PageNumberFormat property specifies the format to display the page numbers. Use the PageNumbersAlignment property to align the page number field in the header or footer of the page. Use the ShowPageNumbers property to hide the page number field. Use the PageNumbersPositon property to indicate whether the page number field is displayed in the header or footer of the page. Use the Caption property to specify a caption for the document being printed on all pages. Use the ExtraCaption property to specify a additional captions for the document being printed on the pages. The StartPageNumber property affects the <%page%> and <%count%> predefined values whenever the PageNumberFormat, Caption or ExtraCaption properties use these values.

property ExPrint.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 ExPrint.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 / TemplatePut property has been added to allow programming languages such as dBASE Plus to set control's properties with multiple parameters (TemplateDef and TemplatePut are equivalents except that the TemplateDef is a property with no parameters, while the TemplatePut is a method with a single parameter). It is known that programming languages such as **dBASE Plus, 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 methods. 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]
   TemplatePut( var_Column )
   Template = [var_Column.Def(4) = 255]
endwith
```

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

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

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

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

```
    .Items.AddItem 0
    .Items.AddItem 1
    .Items.AddItem 2
End With
```

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

```
local Control,var_Column

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

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

```
Dim Control as P
Dim var_Column as P

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

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

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

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

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

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

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

The x-script may uses constant expressions as follow:

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

Sample: #31/12/1971# indicates the December 31, 1971

- *string* expression is delimited by **"** or ` characters. If using the ` character, please make sure that it is different than ' which allows adding comments inline. *Sample: "text" indicates the string text.*

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

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

method ExPrint.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 TemplateDef / TemplatePut property has been added to allow programming languages such as dBASE Plus to set control's properties with multiple parameters (TemplateDef and TemplatePut are equivalents except that the TemplateDef is a property with no parameters, while the TemplatePut is a method with a single parameter). It is known that programming languages such as **dBASE Plus, 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 methods. 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]
   TemplatePut( var_Column )
   Template = [var_Column.Def(4) = 255]
endwith
```

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

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

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

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

```
    .Items.AddItem 0
    .Items.AddItem 1
    .Items.AddItem 2
End With
```

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

```
local Control,var_Column

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

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

```
Dim Control as P
Dim var_Column as P

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

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

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

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

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

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

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

The x-script may uses constant expressions as follow:

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

Sample: #31/12/1971# indicates the December 31, 1971

- *string* expression is delimited by **"** or ` characters. If using the ` character, please make sure that it is different than ' which allows adding comments inline. *Sample: "text" indicates the string text.*

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

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

property ExPrint.ToolBarFont as IFontDisp

Retrieves or sets the toolbar's font.

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

The ToolBarFont property retrieves or sets the toolbar's font. The ToolBarFormat property defines the CRD format to arrange the buttons inside the print's toolbar.

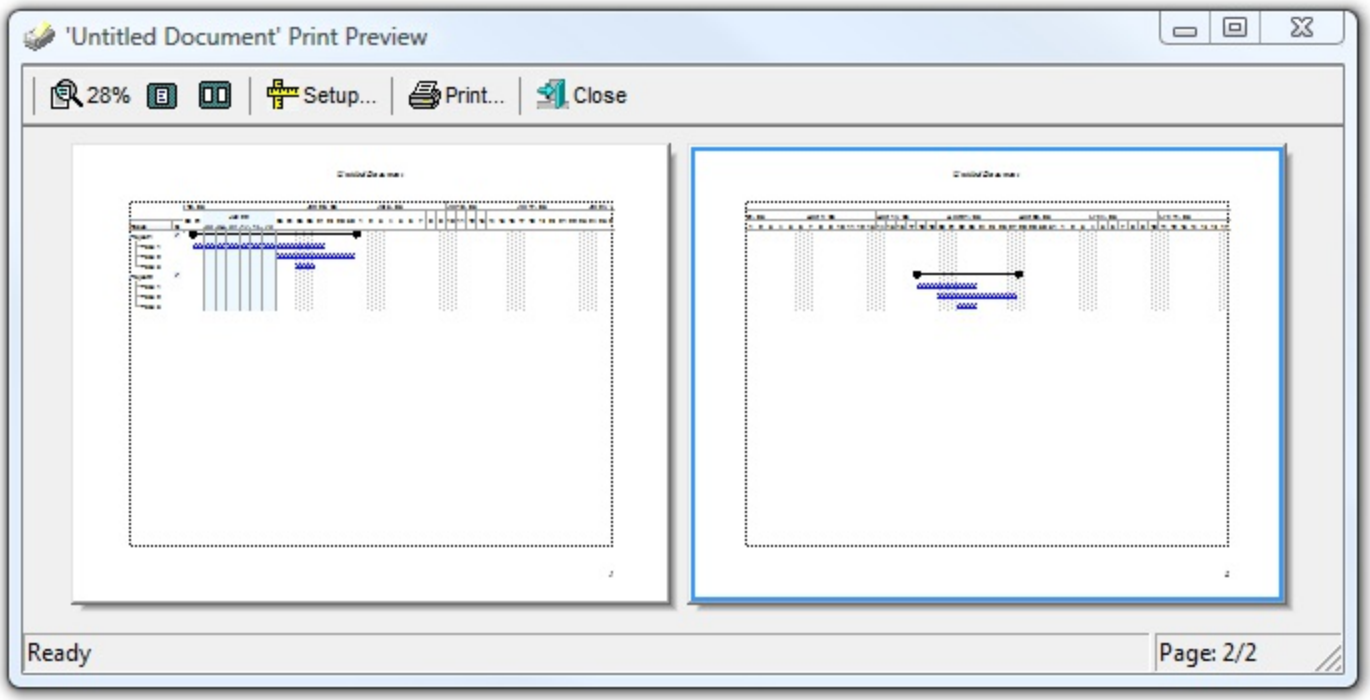
property ExPrint.ToolBarFormat as String

Specifies the CRD format to arrange the buttons inside the print's toolbar.

Type	Description
String	A String expression that specifies the CRD format to eXPrint's toolbar.

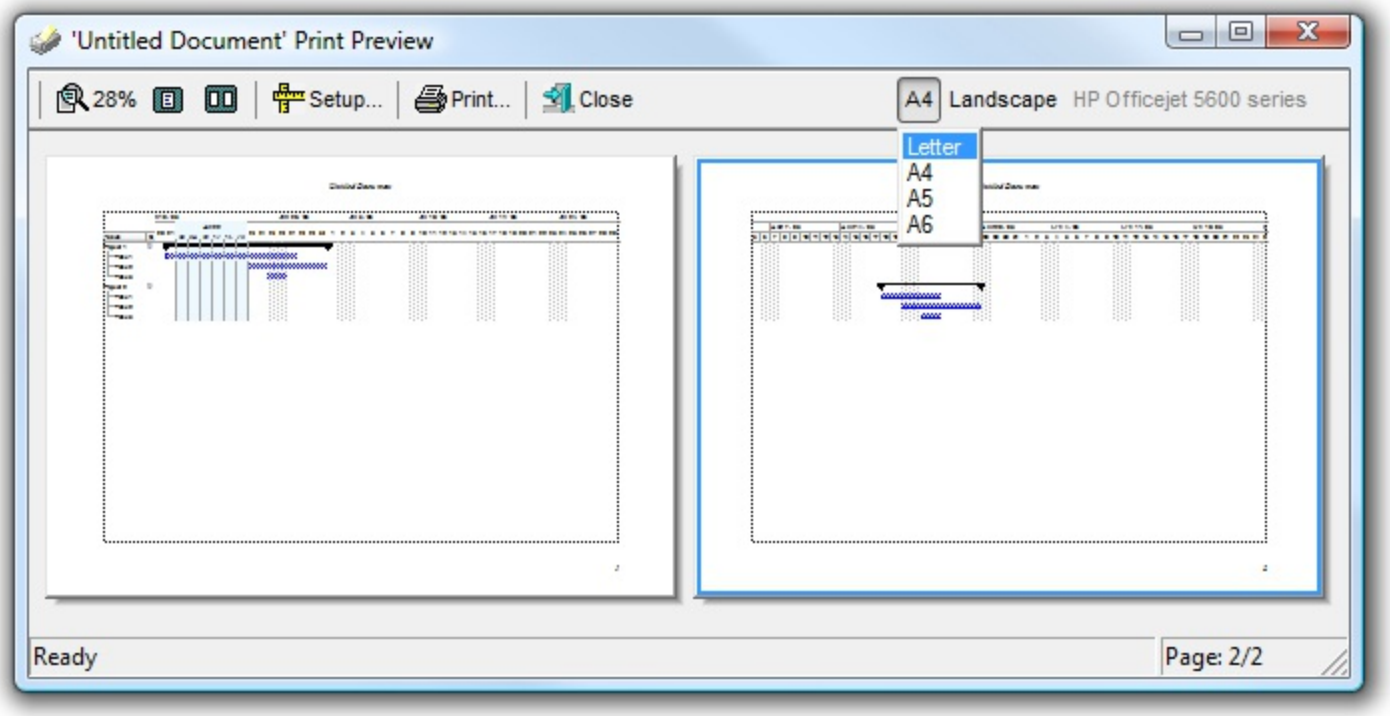
By default, the ToolBarFormat property is "-1,100,101,102,106,-1,103,-1,104,-1,105" and arranges the toolbar as in the following screen shot. If empty, the eXPrint's toolbar displays no buttons. The ToolBarFont property retrieves or sets the toolbar's font.

The following screen shot shows how toolbar shows up:



so, by default, the toolbar displays the Magnify (100), One Page (101), Two Page (102), Setup (103), Print (104) and Close (105) buttons. Use the ToolBarFormat property to add new buttons, to display icons, pictures, or any other HTML caption. The ItemCaption property specifies the caption of the button. The ItemToolTip property specifies the button's tooltip. The control fires the Click event when the user clicks a button in the eXPrint's toolbar. The control fires the AnchorClick event when the user clicks an hyperlink element.

The following VB sample adds three buttons to the right side of the toolbar to display and edit the paper size, the paper orientation, and to display the current printer (as shown in the following screen shot):




```
With Print1
    .ToolBarFormat = .ToolBarFormat & ",|,(201,200,-201):224"
End With
```

```
Private Sub Print1_Click(ID As Long, SelectedID As Long)
    With Print1
        If (ID = 200) Then
            .PageOrientation = SelectedID
            .Refresh
        End If
        If (ID = 201) Then
            .Settings(exPaperSize) = SelectedID
            .Refresh
        End If
    End With
End Sub

Private Sub Print1_Refresh()
    With Print1
        .ItemCaption(-201) = "<fgcolor=808080>" & .Settings(exPrinterName) & "
</fgcolor>"
        .ItemToolTip(-201) = .Settings(exPrinterName)
        .ItemCaption(200) = IIf(.PageOrientation = exLandscape, "Landscape#1", "Portrait#2")
```

```
        .ItemToolTip(200) = "Page Orientation"
        .ItemCaption(201) = .Settings(exFormName) & vbCrLf & "Letter#1" & vbCrLf &
"A4#9" & vbCrLf & "A5#11" & vbCrLf & "A6#70"
        .ItemToolTip(201) = "Paper Size"
    End With
End Sub
```

The following VB/NET sample adds three buttons to the right side of the toolbar to display and edit the paper size, the paper orientation, and to display the current printer (as shown in the following screen shot):

```
With Exprint1
    .ToolBarFormat = .ToolBarFormat & ",|,(201,200,-201):224"
End With
```

```
Private Sub Exprint1_Click(ByVal sender As System.Object, ByVal ID As System.Int32, ByVal
SelectedID As System.Int32) Handles Exprint1.Click
    With Exprint1
        If (ID = 200) Then
            .PageOrientation = SelectedID
            .Refresh()
        End If
        If (ID = 201) Then
            .set_Settings(exontrol.EXPRINTLib.FieldsEnum.exPaperSize, SelectedID)
            .Refresh()
        End If
    End With
End Sub

Private Sub Exprint1_RefreshEvent(ByVal sender As System.Object) Handles
Exprint1.RefreshEvent
    With Exprint1
        .set_ItemCaption(-201, "<fgcolor=808080>" &
.get_Settings(exontrol.EXPRINTLib.FieldsEnum.exPrinterName) & "</fgcolor>")
        .set_ItemToolTip(-201, .get_Settings(exontrol.EXPRINTLib.FieldsEnum.exPrinterName))
        .set_ItemCaption(200, IIf(.PageOrientation =
exontrol.EXPRINTLib.PageOrientationEnum.exLandscape, "Landscape#1", "Portrait#2"))
        .set_ItemToolTip(200, "Page Orientation")
```

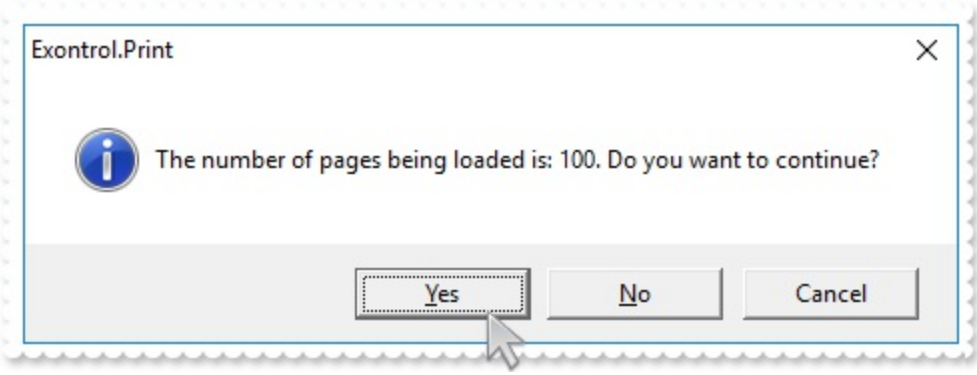
```
        .set_ItemCaption(201, .get_Settings(exontrol.EXPRINTLib.FieldsEnum.exFormName) &
vbCrLf & "Letter#1" & vbCrLf & "A4#9" & vbCrLf & "A5#11" & vbCrLf & "A6#70")
        .set_ItemToolTip(201, "Paper Size")
    End With
End Sub
```

property ExPrint.UILimitPagesCount as Long

Specifies the limit of pages the control can load before a message box to continue shows up.

Type	Description
Long	A Long expression that defines the limit of pages the control can load before a message box to continue shows up.

By default, the UILimitPagesCount property is 100, which indicates that the following message shows up when pages being loaded reach the specified limit. You can disable it, by setting the UILimitPagesCount property on -1. The UILimitPagesCountMessage defines the message to be shown when limit is reached.

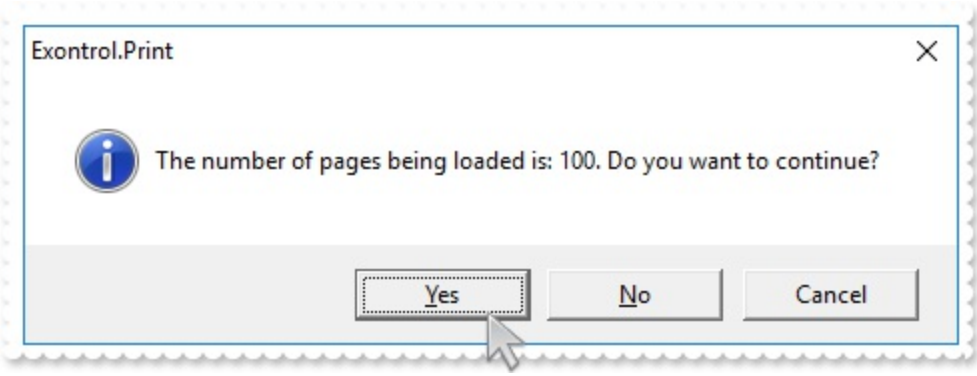


property ExPrint.UILimitPagesCountMessage as String

Specifies the continue message to show up, when the limit of pages has been reached.

Type	Description
String	A String expression that specifies the continue message to show up, when the limit of pages has been reached.

By default, The UILimitPagesCountMessage property is: "The number of pages being loaded is: <%count%>. Do you want to continue?", where the <%count%> indicates the current count of loaded pages. The UILimitPagesCountMessage property defines the message to be shown when limit is reached. You can disable it, by setting the UILimitPagesCount property on -1, or set the UILimitPagesCountMessage property on "" .



property ExPrint.Version as String

Retrieves the control's version.

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

Specifies the control's version.

Page object

The Page object holds information about printed page. The DrawPage method of the IPrintExt interface carries a Page object each time when a new page is printing or previewing. The Page object support the following properties and methods:

Name	Description
Height	Specifies the height of page's client area excluding the margins, in inches coordinates.
Index	Retrieves the index of the page being printed.
Left	Specifies the left margin of the page, in inches coordinates.
PageHeight	Specifies the height of the page in inches coordinates.
PageWidth	Specifies the width of the page, in inches coordinates.
Top	Specifies the top margin of the page, in inches coordinates..
Width	Specifies the width of page's client area excluding the margins, in inches coordinates.

property Page.Height as Double

Specifies the height of page's client area excluding the margins, in inches coordinates.

Type	Description
Double	A double expression that indicates the height of the page's client area, in inches coordinates.

Use the Left, Top, Height and Width properties to identify the client are of the printed page. Use the PageHeight and PageWidth properties to get the page size including the margins. In case you need the coordinates of the page's client are in pixels you should use the GetClipBox API function. The exPrint control sets the clipping area of the device context being the page's client area, before firing the DrawPage callback.

property Page.Index as Long

Retrieves the index of the page being printed.

Type	Description
Long	A long expression that indicates the index of the page being printed.

The Index property specify the index of the page being printed.

property Page.Left as Double

Specifies the left margin of the page, in inches coordinates.

Type	Description
Double	A double expression that indicates the width of the page's client area. The page's client area excludes the margins of the page.

Use the Left, Top, Height and Width properties to identify the client are of the printed page. Use the PageHeight and PageWidth properties to get the page size including the margins. In case you need the coordinates of the page's client are in pixels you should use the GetClipBox API function. The exPrint control sets the clipping area of the device context being the page's client area, before firing the DrawPage callback.

property Page.PageHeight as Double

Specifies the height of the page in inches coordinates.

Type	Description
Double	A double expression that indicates the height of the printed page (including the size of margins).

Use the Left, Top, Height and Width properties to identify the client are of the printed page. Use the PageHeight and PageWidth properties to get the page size including the margins. In case you need the coordinates of the page's client are in pixels you should use the GetClipBox API function. The exPrint control sets the clipping area of the device context being the page's client area, before firing the DrawPage callback.

property Page.PageWidth as Double

Specifies the width of the page, in inches coordinates.

Type	Description
Double	A double expression that indicates the width of the page (including the size of margins).

Use the Left, Top, Height and Width properties to identify the client are of the printed page. Use the PageHeight and PageWidth properties to get the page size including the margins. In case you need the coordinates of the page's client are in pixels you should use the GetClipBox API function. The exPrint control sets the clipping area of the device context being the page's client area, before firing the DrawPage callback.

property Page.Top as Double

Specifies the top margin of the page, in inches coordinates..

Type	Description
Double	A double expression that indicates the top margin of the page, in inches coordinates.

Use the Left, Top, Height and Width properties to identify the client are of the printed page. Use the PageHeight and PageWidth properties to get the page size including the margins. In case you need the coordinates of the page's client are in pixels you should use the GetClipBox API function. The exPrint control sets the clipping area of the device context being the page's client area, before firing the DrawPage callback.

property Page.Width as Double

Specifies the width of page's client area excluding the margins, in inches coordinates.

Type	Description
Double	A double expression that indicates the width of the page's client area in inches coordinates.

Use the Left, Top, Height and Width properties to identify the client are of the printed page. Use the PageHeight and PageWidth properties to get the page size including the margins. In case you need the coordinates of the page's client are in pixels you should use the GetClipBox API function. The exPrint control sets the clipping area of the device context being the page's client area, before firing the DrawPage callback.

ExPrint events

The eXPrint component supports the following events:

Name	Description
AnchorClick	Occurs when an anchor element is clicked.
Click	Occurs when the user clicks a button in the toolbar.
Refresh	Notifies your application when the pages are refreshed.
Refreshing	Notifies your application when the pages are about to be refreshed.

event AnchorClick (AnchorID as String, Options as String)

Occurs when an anchor element is clicked.

Type	Description
AnchorID as String	A string expression that indicates the identifier of the anchor
Options as String	A string expression that specifies options of the anchor element.

The control fires the AnchorClick event to notify that the user clicks an anchor element. 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 AnchorClick event is fired only if prior clicking the control it shows the hand cursor. Use the ToolBarFormat property to add anchor elements to your toolbar. The Click event notifies your application once the user clicks a button in the preview's toolbar.

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

C#

```
private void AnchorClick(object sender,string AnchorID,string Options)
{
}
```

VB

```
Private Sub AnchorClick(ByVal sender As System.Object,ByVal AnchorID As
String,ByVal Options As String) Handles AnchorClick
End Sub
```

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

C#

```
private void AnchorClick(object sender,
AxEXPRINTLib._IExPrintEvents_AnchorClickEvent e)
{
}
```

C++

```
void OnAnchorClick(LPCTSTR AnchorID,LPCTSTR Options)
{
}
```

C++ Builder

```
void __fastcall AnchorClick(TObject *Sender,BSTR AnchorID,BSTR Options)
{
```

```
}
```

Delphi

```
procedure AnchorClick(ASender: TObject; AnchorID : WideString;Options :
WideString);
begin
end;
```

Delphi 8 (.NET only)

```
procedure AnchorClick(sender: System.Object; e:
AxEXPRINTLib._IExPrintEvents_AnchorClickEvent);
begin
end;
```

Powe…

```
begin event AnchorClick(string AnchorID,string Options)
end event AnchorClick
```

VB.NET

```
Private Sub AnchorClick(ByVal sender As System.Object, ByVal e As
AxEXPRINTLib._IExPrintEvents_AnchorClickEvent) Handles AnchorClick
End Sub
```

VB6

```
Private Sub AnchorClick(ByVal AnchorID As String,ByVal Options As String)
End Sub
```

VBA

```
Private Sub AnchorClick(ByVal AnchorID As String,ByVal Options As String)
End Sub
```

VFP

```
LPARAMETERS AnchorID,Options
```

Xbas…

```
PROCEDURE OnAnchorClick(oPrint,AnchorID,Options)
RETURN
```

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

Java…

```
<SCRIPT EVENT="AnchorClick(AnchorID,Options)" LANGUAGE="JScript">
</SCRIPT>
```

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
Function AnchorClick(AnchorID,Options)
```

```
End Function
</SCRIPT>
```

Visual Data...

```
Procedure OnComAnchorClick String llAnchorID String llOptions
   Forward Send OnComAnchorClick llAnchorID llOptions
End_Procedure
```

Visual Objects

```
METHOD OCX_AnchorClick(AnchorID,Options) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_AnchorClick(str _AnchorID,str _Options)
{
}
```

XBasic

```
function AnchorClick as v (AnchorID as C,Options as C)
end function
```

dBASE

```
function nativeObject_AnchorClick(AnchorID,Options)
return
```

The following VB sample displays a message that specifies the hyperlink being clicked:

```
Private Sub Print1_AnchorClick(ByVal AnchorID As String, ByVal Options As String)
   MsgBox AnchorID
End Sub
```

The ToolBarFormat property is set as:

```
Print1.ToolBarFormat = "-1,100,101,102,-1,103,-1,104,-1,105,|,""
<aexontrol>https://www.exontrol.com</a>""[a=18]:164"
```

The following VB/NET sample displays a message that specifies the hyperlink being clicked:

```
Private Sub Exprint1_AnchorClick(ByVal sender As System.Object, ByVal AnchorID As
System.String, ByVal Options As System.String) Handles Exprint1.AnchorClick
   MsgBox(AnchorID)
End Sub
```

The ToolBarFormat property is set as:

```
Exprint1.ToolBarFormat = "-1,100,101,102,-1,103,-1,104,-1,105,|,""
<aexontrol>https://www.exontrol.com</a>""[a=18]:164"
```

event Click (ID as Long, SelectedID as Long)

Occurs when the user clicks a button in the toolbar.

Type	Description
ID as Long	A long expression that specifies the identifier of the button being clicked.
SelectedID as Long	A long expression that specifies the identifier being selected. (the identifier being specified by the second part of the ItemCaption property [separated by # character]). For instance, if the ItemCaption property is "Letter#1234" the button displays the "Letter" label, the SelectedID paremeter is 1234 if the user clicks the button or selects the item in a drop down field.

The Click event notifies your application when the user clicks a button in the preview's toolbar. Use the ToolBarFormat property to add new buttons, pictures or HTML captions to eXPrint's toolbar. The ItemCaption property specifies the caption of the buttons inside the toolbar. The ItemToolTip property specifies the tool tip for buttons inside the toolbar. The Settings property specifies different printer settings. The Refresh event is called once the pages are refreshed.

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

C#
```
private void Click(object sender,int ID,int SelectedID)
{
}
```

VB
```
Private Sub Click(ByVal sender As System.Object,ByVal ID As Integer,ByVal
SelectedID As Integer) Handles Click
End Sub
```

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

C#
```
private void ClickEvent(object sender, AxEXPRINTLib._IExPrintEvents_ClickEvent e)
{
}
```

C++
```
void OnClick(long ID,long SelectedID)
{
}
```

C++ Builder

```
void __fastcall Click(TObject *Sender,long ID,long SelectedID)
{
}
```

Delphi

```
procedure Click(ASender: TObject; ID : Integer;SelectedID : Integer);
begin
end;
```

Delphi 8 (.NET only)

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

Powe…

```
begin event Click(long ID,long SelectedID)
end event Click
```

VB.NET

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

VB6

```
Private Sub Click(ID As Long,SelectedID As Long)
End Sub
```

VBA

```
Private Sub Click(ByVal ID As Long,ByVal SelectedID As Long)
End Sub
```

VFP

```
LPARAMETERS ID,SelectedID
```

Xbas…

```
PROCEDURE OnClick(oPrint,ID,SelectedID)
RETURN
```

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

Java…

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

VBSc…

```
<SCRIPT LANGUAGE="VBScript">
```

```
Function Click(ID,SelectedID)
End Function
</SCRIPT>
```

Visual Data...

```
Procedure OnComClick Integer llID Integer llSelectedID
    Forward Send OnComClick llID llSelectedID
End_Procedure
```

Visual Objects

```
METHOD OCX_Click(ID,SelectedID) CLASS MainDialog
RETURN NIL
```

X++

```
void onEvent_Click(int _ID,int _SelectedID)
{
}
```

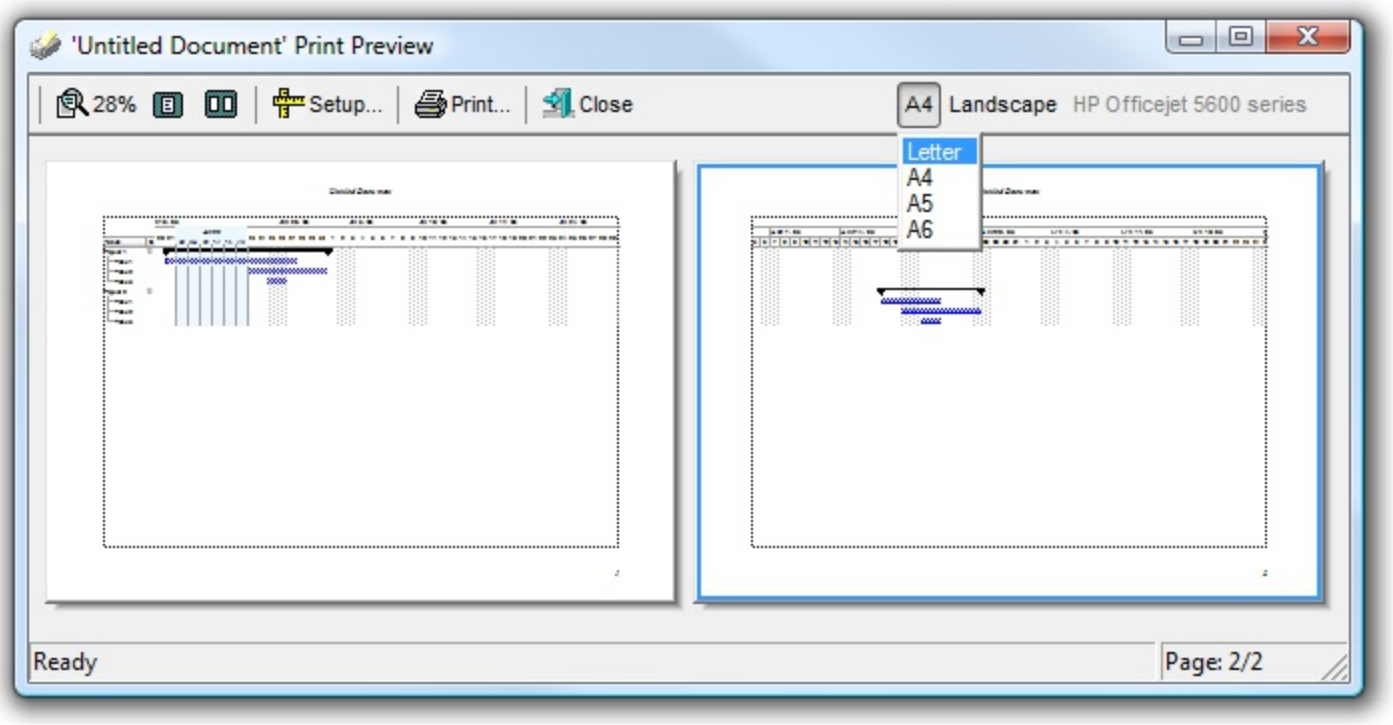
XBasic

```
function Click as v (ID as N,SelectedID as N)
end function
```

dBASE

```
function nativeObject_Click(ID,SelectedID)
return
```

The following VB sample adds three buttons to the right side of the toolbar to display and edit the paper size, the paper orientation, and to display the current printer (as shown in the following screen shot):



```
With Print1
   .ToolBarFormat = .ToolBarFormat & ",|,(201,200,-201):224"
End With
```

```
Private Sub Print1_Click(ID As Long, SelectedID As Long)
   With Print1
      If (ID = 200) Then
         .PageOrientation = SelectedID
         .Refresh
      End If
      If (ID = 201) Then
         .Settings(exPaperSize) = SelectedID
         .Refresh
      End If
   End With
End Sub

Private Sub Print1_Refresh()
   With Print1
      .ItemCaption(-201) = "<fgcolor=808080>" & .Settings(exPrinterName) & "
</fgcolor>"
      .ItemToolTip(-201) = .Settings(exPrinterName)
      .ItemCaption(200) = IIf(.PageOrientation = exLandscape, "Landscape#1", "Portrait#2")
      .ItemToolTip(200) = "Page Orientation"
      .ItemCaption(201) = .Settings(exFormName) & vbCrLf & "Letter#1" & vbCrLf &
"A4#9" & vbCrLf & "A5#11" & vbCrLf & "A6#70"
      .ItemToolTip(201) = "Paper Size"
   End With
End Sub
```

The following VB/NET sample adds three buttons to the right side of the toolbar to display and edit the paper size, the paper orientation, and to display the current printer (as shown in the following screen shot):

```
With Exprint1
   .ToolBarFormat = .ToolBarFormat & ",|,(201,200,-201):224"
End With
```

```
Private Sub Exprint1_Click(ByVal sender As System.Object, ByVal ID As System.Int32, ByVal
SelectedID As System.Int32) Handles Exprint1.Click
   With Exprint1
      If (ID = 200) Then
         .PageOrientation = SelectedID
         .Refresh()
      End If
      If (ID = 201) Then
         .set_Settings(exontrol.EXPRINTLib.FieldsEnum.exPaperSize, SelectedID)
         .Refresh()
      End If
   End With
End Sub

Private Sub Exprint1_RefreshEvent(ByVal sender As System.Object) Handles
Exprint1.RefreshEvent
   With Exprint1
      .set_ItemCaption(-201, "<fgcolor=808080>" &
.get_Settings(exontrol.EXPRINTLib.FieldsEnum.exPrinterName) & "</fgcolor>")
      .set_ItemToolTip(-201, .get_Settings(exontrol.EXPRINTLib.FieldsEnum.exPrinterName))
      .set_ItemCaption(200, IIf(.PageOrientation =
exontrol.EXPRINTLib.PageOrientationEnum.exLandscape, "Landscape#1", "Portrait#2"))
      .set_ItemToolTip(200, "Page Orientation")
      .set_ItemCaption(201, .get_Settings(exontrol.EXPRINTLib.FieldsEnum.exFormName) &
vbCrLf & "Letter#1" & vbCrLf & "A4#9" & vbCrLf & "A5#11" & vbCrLf & "A6#70")
      .set_ItemToolTip(201, "Paper Size")
   End With
End Sub
```

event Refresh ()

Notifies your application when the pages are refreshed.

Type	Description

The Refresh event occurs once the pages are updated for previewing or printing. Use the Refresh event to update the buttons in the toolbar while previewing. The ToolBarFormat property specifies the CRD format to display buttons, pictures and your HTML captions in the eXPrint's toolbar. The ItemCaption property specifies the caption of the buttons inside the toolbar. The ItemToolTip property specifies the tool tip for buttons inside the toolbar. The Settings property specifies different printer settings. The Refresh method forces refreshing the print and print preview pages. The Refreshing event notifies your application once the pages are about to be updated for previewing or printing.

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

C#

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

VB

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

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

C#

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

C++

```
void OnRefresh()
{
}
```

C++ Builder

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

Delphi

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

Delphi 8 (.NET only)

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

Powe…

```
begin event Refresh()
end event Refresh
```

VB.NET

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

VB6

```
Private Sub Refresh()
End Sub
```

VBA

```
Private Sub Refresh()
End Sub
```

VFP

```
LPARAMETERS nop
```

Xbas…

```
PROCEDURE OnRefresh(oPrint)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComRefresh
    Forward Send OnComRefresh
End_Procedure
```

Visual Objects

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

X++

```
void onEvent_Refresh()
{
}
```

XBasic

```
function Refresh as v ()
end function
```

dBASE

```
function nativeObject_Refresh()
return
```

The following VB sample adds a button to the right side of the toolbar, and shows the current printer name:

```
With Print1
   .ToolBarFormat = .ToolBarFormat & ",|,-201:128"
End With
```

```
Private Sub Print1_Refresh()
   With Print1
      .ItemCaption(-201) = "<fgcolor=808080>" & .Settings(exPrinterName) & "
</fgcolor>"
   End With
End Sub
```

The following VB/NET sample adds a button to the right side of the toolbar, and shows the current printer name:

```
With Exprint1
   .ToolBarFormat = .ToolBarFormat & ",|,-201:128"
End With

Private Sub Exprint1_RefreshEvent(ByVal sender As System.Object) Handles
Exprint1.RefreshEvent
```

```
    With Exprint1
        .set_ItemCaption(-201, "<fgcolor=808080>" &
.get_Settings(exontrol.EXPRINTLib.FieldsEnum.exPrinterName) & "</fgcolor>")
    End With
End Sub
```

event Refreshing ()

Notifies your application when the pages are about to be refreshed.

Type	Description

(fit-to-page) The Refreshing event notifies your application once the pages are about to be updated for previewing or printing. The Refresh event occurs once the pages are previewed or printed. Use the Refreshing event to prepare the object to be printed, as changing particular properties, options and so on, so the print preview will be updated based on the new values. If you are changing properties of the object to be printed, you can restore their values during the Refresh event, after the pages were updated, when changes are not longer required.

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

C#

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

VB

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

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

C#

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

C++

```
void OnRefreshing()
{
}
```

C++ Builder

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

Delphi

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

Delphi 8 (.NET only)

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

Powe…

```
begin event Refreshing()
end event Refreshing
```

VB.NET

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

VB6

```
Private Sub Refreshing()
End Sub
```

VBA

```
Private Sub Refreshing()
End Sub
```

VFP

```
LPARAMETERS nop
```

Xbas…

```
PROCEDURE OnRefreshing(oPrint)
RETURN
```

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

Java…

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

VBSc…

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

Visual Data…

```
Procedure OnComRefreshing
   Forward Send OnComRefreshing
```

```
End_Procedure
```

Visual Objects

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

X++

```
void onEvent_Refreshing()
{
}
```

XBasic

```
function Refreshing as v ()
end function
```

dBASE

```
function nativeObject_Refreshing()
return
```

For instance, let's say that you need to provide the FitToPage feature for the eXG2antt chart control. The following samples show different implementations.

The following VB sample changes the UnitWidth property of the eXG2ant's Chart object so, the entire chart is printed to the page:

```
With Print1
   Dim l As Long
   With G2antt1.Chart
      l = .UnitWidth
      .UnitWidth = (Print1.ClientWidth - .PaneWidth(False)) / .CountVisibleUnits()
   End With
   Set .PrintExt = G2antt1.Object
   .Preview
   G2antt1.Chart.UnitWidth = l
End With
```

The sample has the disadvantage that once the user changes the Page's setup during Previewing the code is not re-executed, so the chart is displayed as it is on the screen. In order to update the UnitWidth property once the page's setup is changed, we need to handle the Refreshing and Refresh events as shown in the following VB sample:

```
Dim nUnitWidth As Long
```

```
Private Sub Print1_Refreshing()
   With G2antt1.Chart
      nUnitWidth = .UnitWidth
      .UnitWidth = (Print1.ClientWidth - .PaneWidth(False)) / .CountVisibleUnits()
   End With
End Sub

Private Sub Print1_Refresh()
   G2antt1.Chart.UnitWidth = nUnitWidth
End Sub

Private Sub Preview_Click()
   With Print1
      Set .PrintExt = G2antt1.Object
      .Preview
   End With
End Sub
```

The sample changes the UnitWidth property of the Chart during the Refresing event, so the chart fits to page, and restores the UnitWidth's value when the Refresh event is invoked.

The following VB/NET sample changes the UnitWidth property so the chart fits to page:

```
Dim nUnitWidth As Long

Private Sub Exprint1_RefreshingEvent(ByVal sender As System.Object) Handles
Exprint1.RefreshingEvent
   With Exg2antt1.Chart
      nUnitWidth = .UnitWidth
      .UnitWidth = (Exprint1.ClientWidth - .get_PaneWidth(False)) / .CountVisibleUnits()
   End With
End Sub

Private Sub Exprint1_RefreshEvent(ByVal sender As System.Object) Handles
Exprint1.RefreshEvent
   Exg2antt1.Chart.UnitWidth = nUnitWidth
End Sub
```

```
Private Sub Preview_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Preview.Click
    Exprint1.PrintExt = Exg2antt1
    Exprint1.Preview()
End Sub
```

PrintExt object

The IPrintExt interface defines the extension interface that a client needs to implement it in order to provide Print and Print Preview capabilities. The IPrintExt interface definition follows:

```
[
    uuid(9A7B1864-C090-4E42-B41D-4E21712EA23A),
]
interface IPrintExt : IUnknown
{
    [propget, id(1), helpstring("Retrieves the count of pages.")] HRESULT PageCount( [in] VARIANT Options, [out, retval] long *pVal );
    [id(2), helpstring("Prints a page.")] HRESULT DrawPage( [in] VARIANT Options, long hDC, IPage* Page, VARIANT_BOOL* pbContinue );
}
```

The PrintExt/PrintExts property accepts only objects that implement IPrintExt interface. If PrintExt and PrintExts properties are empty, the DoPrint and Preview methods fail.

- How do I implement this interface using VB?
- How do I implement this interface using VC?

Name	Description

Expressions

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

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

Usage examples:

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

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

The constants can be represented as:

- numbers in **decimal** format (where dot character specifies the decimal separator). For instance: -1, 100, 20.45, .99 and so on
- numbers in **hexa-decimal** format (preceded by **0x** or **0X** sequence), uses sixteen distinct symbols, most often the symbols 0-9 to represent values zero to nine, and A, B, C, D, E, F (or alternatively a, b, c, d, e, f) to represent values ten to fifteen. Hexadecimal numerals are widely used by computer system designers and programmers. As each hexadecimal digit represents four binary digits (bits), it allows a more human-friendly representation of binary-coded values. For instance, 0xFF,

0x00FF00, and so so.

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

The predefined constants are:

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

The supported binary arithmetic operators are:

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

The supported unary boolean operators are:

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

The supported binary boolean operators are:

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

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

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

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

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

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

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

  variable := expression

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

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

  =: variable

  where variable is a integer between 0 and 9. You can use the **:=** operator to store the value of any expression (please make the difference between := and =:). For

instance, *(0:=dbl(value)) = 0 ? "zero" : =:0*, stores the value converted to double, and prints zero if it is 0, else the converted number. Please pay attention that the **:=** and **=:** are two distinct operators, the first for storing the result into a variable, while the second for restoring the variable

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

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

  expression ? true_part : false_part

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

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

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

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

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

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

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

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

statement is used, it could take up to 1000 operations to check, so by far, the *in* operator could save time on finding elements within a collection.

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

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

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

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

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

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

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

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

The supported conversion unary operators are:

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

The bitwise operators for numbers are:

- a **bitand** b (binary operator) computes the AND operation on bits of a and b, and returns the unsigned value. For instance, 0x01001000 bitand 0x10111000 returns 0x00001000.
- a **bitor** b (binary operator) computes the OR operation on bits of a and b, and returns the unsigned value. For instance, 0x01001000 bitor 0x10111000 returns 0x11111000.
- a **bitxor** b (binary operator) computes the XOR (exclusive-OR) operation on bits of a and b, and returns the unsigned value. For instance, 0x01110010 bitxor 0x10101010 returns 0x11011000.
- a **bitshift** (b) (binary operator) shifts every bit of a value to the left if b is negative, or to the right if b is positive, for b times, and returns the unsigned value. For instance, 128 bitshift 1 returns 64 (dividing by 2) or 128 bitshift (-1) returns 256 (multiplying by

2)

- **bitnot** (unary operator) flips every bit of x, and returns the unsigned value. For instance, bitnot(0x00FF0000) returns 0xFF00FFFF.

The operators for numbers are:

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

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

  - *NumDigits* - specifies the number of fractional digits, If the flag is missing, the field "No. of digits after decimal" from "Regional and Language Options" is using.
  - *DecimalSep* - specifies the decimal separator. If the flag is missing, the field "Decimal symbol" from "Regional and Language Options" is using.
  - Grouping - indicates the number of digits in each group of numbers to the left of

the decimal separator. Values in the range 0 through 9 and 32 are valid. The most significant grouping digit indicates the number of digits in the least significant group immediately to the left of the decimal separator. Each subsequent grouping digit indicates the next significant group of digits to the left of the previous group. If the last value supplied is not 0, the remaining groups repeat the last group. Typical examples of settings for this member are: 0 to group digits as in 123456789.00; 3 to group digits as in 123,456,789.00; and 32 to group digits as in 12,34,56,789.00. If the flag is missing, the field "Digit grouping" from "Regional and Language Options" indicates the grouping flag.

- *ThousandSep* - specifies the thousand separator. If the flag is missing, the field "Digit grouping symbol" from "Regional and Language Options" is using.
- *NegativeOrder* - indicates the negative number mode. If the flag is missing, the field "Negative number format" from "Regional and Language Options" is using. The valid values are 0, 1, 2, 3 and 4 with the following meanings:
  - 0 - Left parenthesis, number, right parenthesis; for example, (1.1)
  - 1 - Negative sign, number; for example, -1.1
  - 2 - Negative sign, space, number; for example, - 1.1
  - 3 - Number, negative sign; for example, 1.1-
  - 4 - Number, space, negative sign; for example, 1.1 -
- *LeadingZero* - indicates if leading zeros should be used in decimal fields. If the flag is missing, the field "Display leading zeros" from "Regional and Language Options" is using. The valid values are 0, 1

The operators for strings are:

- **len** (unary operator) retrieves the number of characters in the string. For instance, the *len("Mihai")* returns 5.
- **lower** (unary operator) returns a string expression in lowercase letters. For instance, the *lower("MIHAI")* returns "mihai"
- **upper** (unary operator) returns a string expression in uppercase letters. For instance, the *upper("mihai")* returns "MIHAI"
- **proper** (unary operator) returns from a character expression a string capitalized as appropriate for proper names. For instance, the *proper("mihai")* returns "Mihai"
- **ltrim** (unary operator) removes spaces on the left side of a string. For instance, the *ltrim(" mihai")* returns "mihai"
- **rtrim** (unary operator) removes spaces on the right side of a string. For instance, the *rtrim("mihai ")* returns "mihai"
- **trim** (unary operator) removes spaces on both sides of a string. For instance, the *trim(" mihai ")* returns "mihai"
- **reverse** (unary operator) reverses the order of the characters in the string a. For instance, the *reverse("Mihai")* returns "iahiM"
- a **startwith** b (binary operator) specifies whether a string starts with specified string (

0 if not found, -1 if found). For instance *"Mihai" startwith "Mi"* returns -1

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

The operators for dates are:

- **time** (unary operator) retrieves the time of the date in string format, as specified in the control's panel. For instance, the *time(#1/1/2001 13:00#)* returns "1:00:00 PM"

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

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

Exontrol's Custom Row Designer

The Exontrol's Custom Row Designer (**exCRD**) is a WYSWYG tool to build new layouts for cells/nodes, items/rows or columns/fields. The exCRD tool generates CRD strings from the layout you built. The syntax of CRD strings is designed to be easy to build, change and read. Using CRD strings is powerful than preformatted card view, group view formats, nested bands, and so on, since you are free to define the full layout of the cell/node, item/row or a column/field.

Start Here, CRD Syntax

For instance, here are few simple CRD strings:

- The CRD string `**1,2**` divides the cell in two parts, the left side displays the first column, and the right part displays the second column. Similar with horizontally splitting a cell in two pieces.

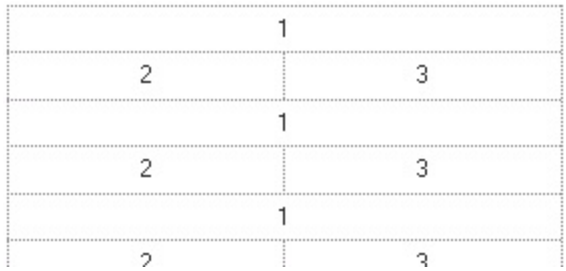
1	2
1	2
1	2
1	2
1	2
1	2

- The CRD string `**1/2**` splits vertically the cell in two parts, where the upper part displays the first column, and the down part displays the second column. Similar with vertically splitting a cell in two pieces.

1
2
1
2
1
2

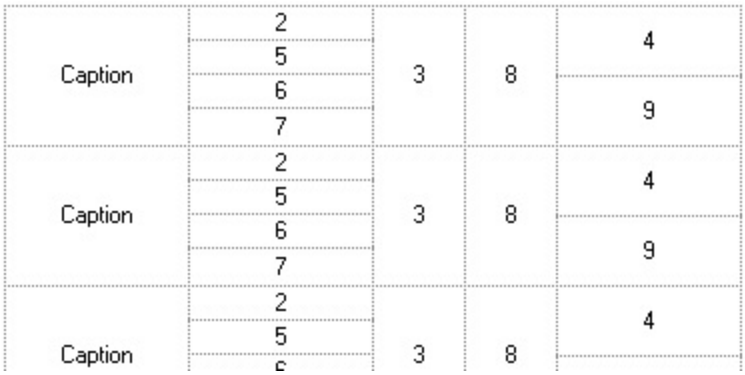
- The CRD string `**1/2,3**` splits a cell in two, the upper part displays the first

column, the bottom part is divided in other two parts, where the left part displays the second column, and the right part displays the third column.



- The CRD string `**18;"Ca<u>pti</u>on"[a=17]/1,(2/3)**` splits vertically the cell in two parts, the upper part displays the "Caption" string aligned on the center, with the height of 18 pixels, the bottom part is divided in other two parts, the left part displays the first column, and the second part is vertically divided in other two parts, where the upper part displays the second column and the bottom part displays the third column.

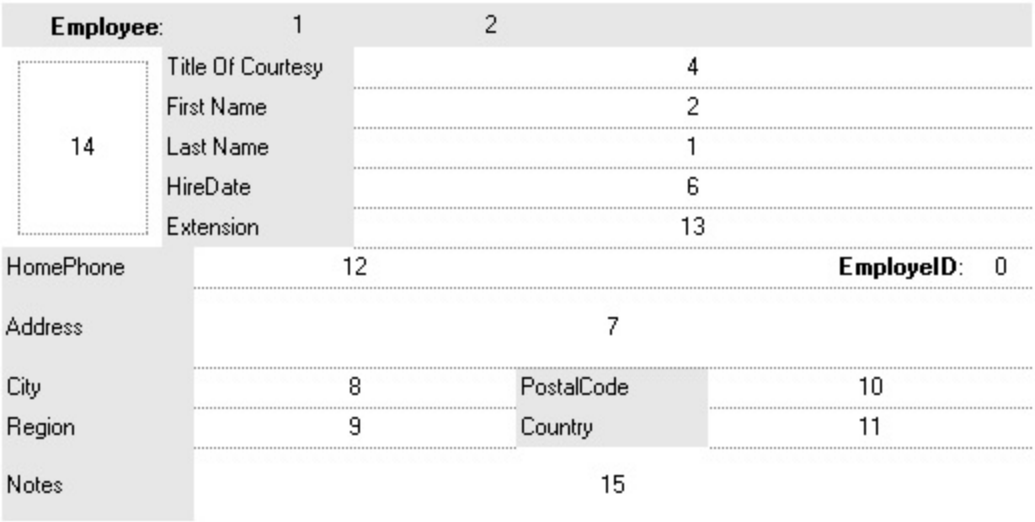
- The CRD string `**"Caption"[a=17],(2/5/6/7),(3,8),(4/9)**`generates the following layout:



- The CRD string

```
[b=0][dgl=1](2;""/("":2,(22;((4;""/("":4,("<img class="calibre2">1</img><b
class="calibre12">Employee</b>:":96,(1:96,(2:96,"")))," ":4)/4;"")[bg=RGB(230,230,
[bg=RGB(230,230,230)]/(100;((" ":8,(""/86;14[b=15]/""),"":8):80,(("Title Of Courtesy'
[bg=RGB(230,230,230)]:96,4[b=4])/(93;"First Name"[bg=RGB(230,230,230)]:96,2[
Name"[bg=RGB(230,230,230)]:96,1[b=4])/("HireDate"
[bg=RGB(230,230,230)]:96,6[b=4])/("Extension"[bg=RGB(230,230,230)]:96,13[b=4
((("HomePhone"[bg=RGB(230,230,230)]:96,12[b=4]),("<b class="calibre12">Emplo
[b=4][a=18],0[b=4]:32))/40;"Address"[bg=RGB(230,230,230)]:96,7[b=4]/(("City"
[bg=RGB(230,230,230)]:96,8[b=4]),("PostalCode"
```

generates the following layout.



The CRD syntax in BNF notation is defined like follows:

```
<CRD> ::= [<Options>] <GroupCRD>
<GroupCRD> ::= <UpPart> [ "|" <DownPart> ]
<UpPart> ::= <Lines>
<DownPart> ::= <Lines>
<Lines> ::= <Line> | "(" <Lines> ")" | <Lines> "/" <Lines>
<Line> ::= [<Height>;] <LeftPart> [ "|" <RightPart> ]
<LeftPart> ::= <Fields>
<RightPart> ::= <Fields>
<Fields> ::= <Field> | "(" <Fields> ")" | <Fields> "," <Fields>
<Field> ::= <Identifier> [<Options>] [ ":" <Width>]
<Identifier> ::= <Index> | <Caption>
<Options> ::= <Options> [ "[" <Option> "]" ]
<Option> ::= <Property> [ "=" <Value> ]
<Property> ::= <letter> | <Property> [ <letter> | <digit> ]
<Value> ::= <Number> | <String>
<Index> ::= <Number>
<Caption> ::= <String>
<Width> ::= <Number>
<Height> ::= <Number>
<Number> ::= <digit><Number>
<String> ::= """<any_character>"""
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

The CRD strings may include the following elements:

- **Index**, represents numbers, a set of digits. For instance, 1 2 3, ... and so on
- **Caption**, represents strings, and are delimitated by " characters. For instance, "" (empty string) "Caption" "test", and so on
- **,** (**field separator**), delimits the elements in the same line. For instance: 1,"Caption",(3/4/5)
- **/**, (**line separator**), delimits the elements in different lines. For instance: 1/"Caption"/(2,3,4)
- **|**, (**divider character**), splits the left and right parts of a line, or top or bottom parts of a group
- **()**, (**groups**), defines a group
- **[]**, (**options**), specifies options for elements in the layout
- **;**, (**line's height separator**), specifies the height of the line or the group
- **:**, (**field's width separator**), specifies the width of the field or the group

The **Index** and **Caption** element may have one or more of the following options:

- **Border** option, [b=<Number>], specifies which borders are shown or hidden. The <Number> may be a sum of one or more values like follows:

  - **1**, top border, draws the top border
  - **2**, right border, draws the right border
  - **4**, bottom border, draws the bottom border
  - **8**, left border, draws the left border

  For instance, the [b=5] means that the element draws the top and the bottom borders. For instance, if the [b=0] is at the beginning of the CRD string, it specifies that by default, no borders are shown.

- **Background** option, [bg=RGB(,<Number>,<Number>,<Number>)] | [bg= <Number>], specifies the background color of the element.
- **Foreground** option, [fg=RGB(,<Number>,<Number>,<Number>)] | [fg= <Number>], specifies the foreground color of the element. This option has effect only for Caption elements.

The **Caption** element may have one or more of the following options:

- **Alignment** option, [a=<Number>], specifies the alignment of the caption in the element. By default, if the option is missing, the caption is aligned to the left. The <Number> may be one of the values like follows:

  - **0**, TopLeft, Aligns the caption to the top left corner.
  - **1**, TopCenter, Centers the caption on the top edge.

- **2**, TopRight, Aligns the caption to the upper right corner.
- **16**, MiddleLeft, Aligns horizontally the caption on the left side, and centers the caption vertically
- **17**, MiddleCenter, Puts the caption on the center of the element. (Default)
- **18**, MiddleRight, Aligns horizontally the caption on the right side, and centers the caption vertically
- **32**, BottomLeft, Aligns the caption to the lower left corner
- **33**, BottomCenter, Centers the caption on the lower edge
- **34**, BottomLeft, The caption is resized to fit the source

- **WordWrap** option, [ww], specifies whether the caption is wrapping in the element's client area. If the option is present, the text is arranged on multiple lines, else the text is displayed on a single line.

The options for the CRD string may be (these options must be always at the beginning of the CRD string):

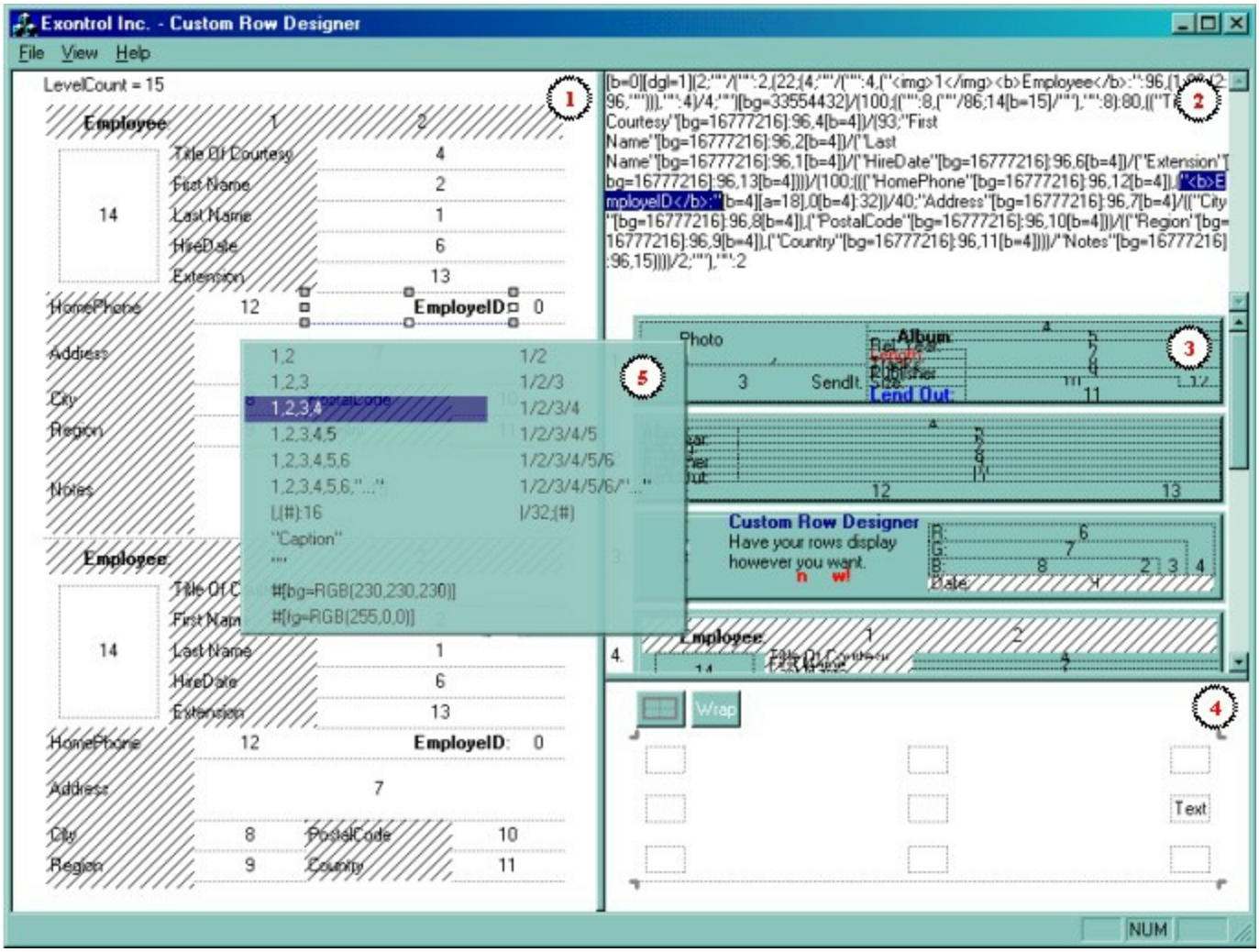
- **Debug** option, [debug], displays debug information when running in the component. Has no effect in the exCRD tool
- **Border** option, [b=<Number>], specifies which borders are shown or hidden, for all elements in the CRD string. The <Number> may be a sum of one or more values like follows:

  - **1**, top border, all elements in the CRD layout draw the top border
  - **2**, right border, all elements in the CRD layout draw the right border
  - **4**, bottom border, all elements in the CRD layout draw the bottom border
  - **8**, left border, all elements in the CRD layout draw the left border

- **DrawGridLines** options, [dgl=0|1|-1], specifies whether the CRD layout draws the grid lines. This option is depending on the component's context.

Introducing the exCRD tool

The exCRD



WYSWYG tool helps you to build CRD strings. The left side (1) of the exCRD tool displays the layout. The right side includes an edit control (2) where the CRD string can be changed or updated, the CRD history (3), and a toolbox (4) to update the borders and the alignment of the text for the selected element in the layout. **Changing the CRD string in the edit control updates automatically the layout in the left side. Also, changing the layout in the left side, updates the CRD string the right side edit control.** The left side's context menu (5) includes options to format the selected element in the layout, such of splitting the element, aligning the element to the right side, or bottom side, changing the element's background or foreground color, and so on. The tool's status bar displays information about the selected item in the context menu.

1. The Layout area divides the client area how the CRD format string specifies. Using the mouse or the keyboard you can select a new element. A ticker frame is drawn around the selected element. Once that the user selects an element in the layout area, the portion of CRD text that defines the element is selected in the CRD (2) area. For instance, if you select a caption element, the name of the element including the " characters are selected in the CRD (2) area, so you can use the keyboard to change the name or the index of the element. The toolbox (4) displays the options that can be changed for the selected element. For instance, if you selected a caption element, you can specify the alignment of the caption in the field, the borders around the element, where the text should be

wrapped and so on. Use the white tickers to resize the element. While resizing the element in the layout area, the CRD (2) area highlights the portion of the CRD text that defines the width or the height being changed. Use the keyboard to type the width or the height of the element. If the width or the height of the element is missing or it is zero, the control resizes it so the child elements fit the parent's client area. Clicking multiple times in the same element, you get highlighted the parent of the selected element and so on. Use the right click to display the layout's context menu (5). Use the ALT + arrow keys to navigate throw the index elements. Use the CTRL+Z to undo the last operation.

2. The CRD area displays and edits CRD strings. The CRD area is a simple edit control, and can be used to change the CRD string. The Layout (1) area is updated automatically when the user alters the CRD string. Use this area to locate a specific element, and change it's name or its options. To clear the entire layout, select the entire text in the CRD area and press the DELETE key.

3. The History list holds the CRD strings being formatted using the exCRD tool. The CRD strings are saved to the "ExCRD.history" file. The "ExCRD.history" file persists in the same folder where the exCRD tool is. Click the CRD history if you need to save the current layout. If the layout was not saved previously the layout is saved, else the selected layout is loaded and displayed. Use the DELETE key to delete a layout in the history.

4. The Toolbox displays options like border, alignment of the caption for the selected element in the layout (1). Changing an option in the Toolbox automatically updates the Layout (1) and the CRD(2) area.

5. Click an element in the Layout (1) area, and then do a right click to invoke the layout's context menu. The current selection in the CRD area is replaced with the format you chose when selecting an item from the layout's context menu. The layout's context menu includes the following:

   - 1,2 - Splits horizontally a field in two parts.
   - 1,2,3 - Splits horizontally a field in three parts.
   - 1,2,3,4 - Splits horizontally a field in four parts.
   - 1,2,3,4,5 - Splits horizontally a field in five parts.
   - 1,2,3,4,5,6 - Splits horizontally a field in six parts.
   - 1,2,3,4,5,6,"..." - Splits horizontally a field in multiple parts.
   - |,(#):16 - Aligns the field to the right.

- "Caption" - Specifies a caption, instead an index.
- "" - Specifies an empty unit.
- #[bg=RGB(230,230,230)] - Specifies the part's background color.
- #[fg=RGB(255,0,0)] - Specifies the part's foreground color.
- 1/2 - Splits vertically a field in two parts.
- 1/2/3 - Splits vertically a field in three parts.
- 1/2/3/4 - Splits vertically a field in four parts.
- 1/2/3/4/5 - Splits vertically a field in five parts.
- 1/2/3/4/5/6 - Splits vertically a field in six parts.
- 1/2/3/4/5/6/"..." - Splits vertically a field in multiple parts.
- |/32;(#) - Aligns the field to the bottom.

Start building CRD strings using the exCRD tool

Let's say that we want to have a view to display fields like in the following screen shot. The first step we need to follow is identifying the fields/indexes we need to display in the layout. In our sample we have the following fields:

- photo, 2
- artist, 4
- album, 5
- release year, 6
- length, 7
- tracks, 8
- publisher, 9

- size, 10
- lend out, 11
- rate, 12
- availability, 13

Once that we identified the fields, we can start building the layout we want to have.

	4		
	Album:	5	
	Rel.Year:	6	
	Length:	7	
2	Tracks:	8	
	Publisher	9	
	Size:	10	
	LendOut:	11	
	12		13

Here's the steps you need to follow in order to build the above layout:

- Empty the old CRD string in the tool. Select the entire text in the CRD area, and press DELETE key
- Right click in the Layout area, and select the **1,2** format.
- Resize the element **1**. Select it, and move the mouse over the right ticker. Click it and start resizing the element. In the same time, you notice that the CRD area highlights the portion of text that defines the width of the element. Type **64**, so we have the element **1** with the width of **64** pixels.
- Splits the element **1** in three parts, to let the photo field in the center. Select the element **1**, right click, and select the **1/2/3** format.
- Specifies the height of the photo field. Click the element **3**, move the mouse over the bottom ticker so we can resize the height of the element. While resizing, the CRD area highlights the portion of text that defines the height of the element. Type **64**, and you have fixed the height of the photo field to **64** pixels.
- Removes extra borders for the element **3**. Click the element **3**, and move the mouse to the toolbox, where you remove the top and bottom borders.
- Empty the element **1** and **4**, because we do not need them. Click the element **1**, and type **""**. Click the element **4**, and type **""** (two " characters).
- We have obtained the following CRD string `**(""[b=11]/64;3[b=10]/""**

[b=14]):64,2`, and imagine that the photo field goes to the element 3.

- Split the element **2** in 9 vertical parts, so we can have the other fields displayed in the layout. Select the element **2**, right click, select **1/2/3/4/5/6/"..."** format. We have only 7 elements, so we need to type the rest of them in the CRD area. Go to the CRD area and select the **"..."** text, and type **9/10/11**. Finally, we got 9 elements.

- Split last element **11** in two pieces so we can display two fields instead. Click the element **11**, right click, and select **1,2** format. Click the element **12**, and move the mouse to the right ticker to resize the field. Click and resize the field. While resizing the CRD tool highlights the portion of text that defines the width of the field. Type **48**, so we defined the element 12 aligned to the right with the width of **48** pixels.

- Currently we got: `**(""[b=11]/64;3[b=10]/""[b=14]):64, (2/4/5/6/7/8/9/10/(11,12:48))**`.

- Select the element **4**, right click, select **1,2**. Select the element **4**, move the mouse over the right ticker, and resize the element. Type **96**. Select the element **4** again, right click, select "Caption". Locate the "Caption" string in the CRD area, and replace it with "Album:"

- Do the same thing as you did for the element **4**, for the elements **5, 6, 7, 8, 9, 10**

- We got: `**(""[b=11]/64;3[b=10]/""[b=14]):64, (2/("Album:":96,13)/("RelYear:":96,4)/("Length:":96,5)/("Tracks:":96,6)/ ("Publisher:":96,7)/("Size:":96,8)/("LendOut:":96,9)/(11,12:48))**`

- The last step we need to follow is to replace the index elements with the numbers in our table. Select the element **3** and type **2**, as the photo field has the identifier 2. Do the same thing for the rest index elements.

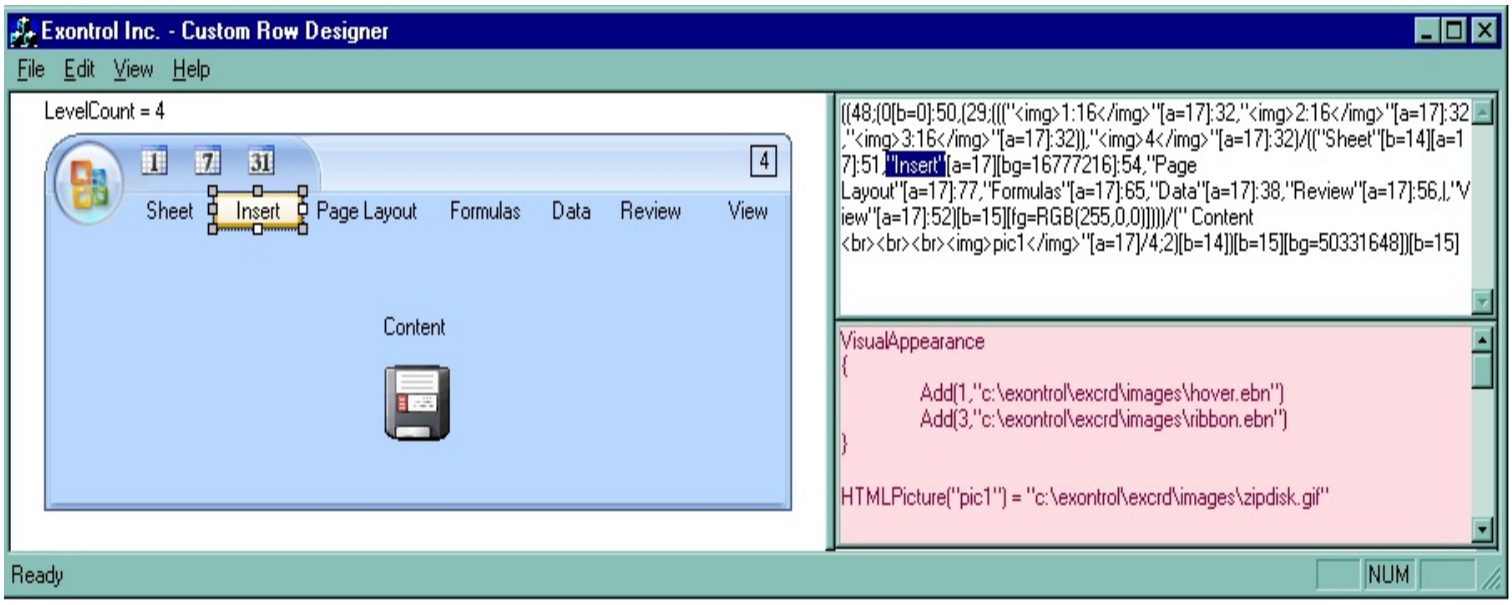
- Finally, we got the CRD string: `**(""[b=11]/64;3[b=10]/""[b=14]):64, (4/("Album:":96,5)/("RelYear:":96,6)/("Length:":96,7)/ ("Tracks:":96,8)/("Publisher:":96,9)/("Size:":96,10)/("LendOut:":96,11)/(12,13**

Display Icons, Images, Pictures and EBN objects

Starting from the version 2.0, the CRD tool includes an Visual-Appearance area to load icons, images, pictures and EBN objects, so the objects in the CRD format may display <img> or <bg> tags. *For instance, the <img>1</img> displays the first icon that was previously loaded using the Images or ReplaceIcon method. The*

<img>pic1</img> displays a custom size picture that was previously loaded using the HTMLPicture property. The [bg=16777216] indicates that the background of the object displays the EBN object with the identifier 1. The 16777216 represents the 0x1RRGGBB, or 0x1000000, where 1 indicates the identifier of the skin object being loaded using the VisualAppearance.Add method.

The following screen shot shows in red the area where visual appearance of the CRD strings may be changed:



The following properties are known:

- **Images(Handle)**, sets the control's image list at runtime, where the handle may be: A long expression that identifies a handle to an Image list (the Handle should be of HIMAGELIST type) or a string expression that indicates the base64 encoded string that holds the icons list. Use the eximages tool to save your icons as base64 encoded format.

- **ReplaceIcon ([Icon], [Index])**, Adds a new icon, replaces an icon or clears the control's image list, where Icon may be: A long expression that indicates the icon's handle. By default, the Icon parameter is 0, if it is missing., and the Index may be: A long expression that indicates the index where icon is inserted. By default, the Index parameter is -1, if it is missing.

- **HTMLPicture(Key)**, Adds or replaces a picture in HTML captions, where the Key may be: 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. The HTMLPicture specifies the picture being associated to a key. It can be one of the followings:

1. a string expression that indicates the path to the picture file, being loaded.
2. 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.
3. 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

- **VisualAppearance.Add(ID,Skin)**, Adds or replaces a skin object to the control, where the ID may be: 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, and the Skin may be: A string expression that indicates:
  1. 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.
  2. a 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 following screen shot shows the control's header when using a "CP:1 -6 -6 6 6", that displays the original skin on larger rectangles
  3. the path to the skin file (*.ebn). The Exontrol's exButton component installs a skin builder that should be used to create new skins
  4. 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

Shortly, the Visual-Appearance area displays and handles x-script code for the Exontrol.CRD object.

How do I implement the IPrintExt interface in VB?

The VB provides the **Implements** keyword that helps you to implement an interface. The **Implements** keyword is used to signify that a class member implements a specific interface. An Implements statement requires a comma-separated list of interface members to be implemented. Generally, only a single interface member is specified, but you can specify multiple members. The specification of an interface member consists of the interface name, which must be specified in an implements statement within the class, a period, and the name of the member function, property or event to be implemented. The name of a member that implements an interface member can use any legal identifier, and is not limited to the InterfaceName_MethodName convention used in earlier versions of Visual Basic.

The following steps will guide you to implement the IPrintExt interface.

1. Add a reference to exPrint.dll file to your project. Select Project menu, then References, and check the "ExPrint 1.0 Control Library" component.
2. Add Implements IPrintExt to the class that defines the IPrintExt interface.
3. Add IPrintExt_PageCount, and IPrintExt_DrawPage methods to your class, by selecting IPrintExt member of the class.

Once that you followed the steps, you need to get something like follows (in this case we have choose to have the Form class the object that implements the IPrintExt interface, of course you can choose another object or class) :

```
Implements IPrintExt

Private Sub Form_Load()

End Sub

Private Sub IPrintExt_DrawPage(ByVal Options As Variant, ByVal hDC As Long, ByVal
Page As EXPRINTLib.IPage, pbContinue As Boolean)

End Sub

Private Property Get IPrintExt_PageCount(ByVal Options As Variant) As Long

End Property
```

Now, all that we need to do is to write the PageCount and DrawPage methods. Here's a

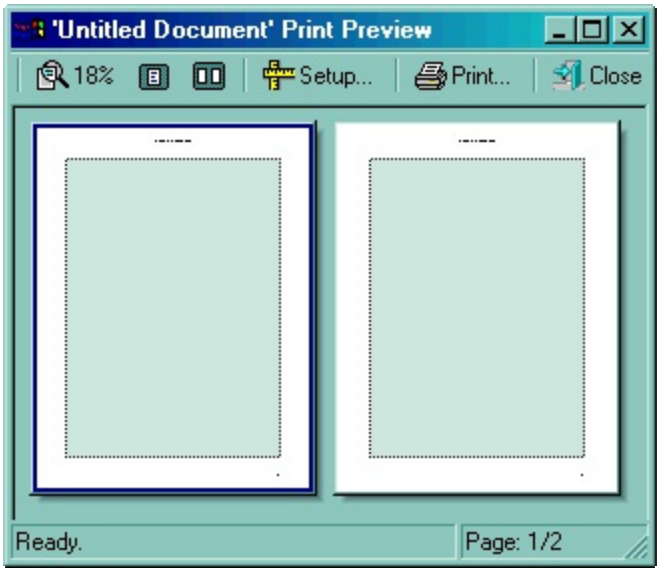
simple code that fills the page's client area:

```
Implements IPrintExt

Private Declare Function GetClipBox Lib "gdi32" (ByVal hdc As Long, lpRect As RECT)
As Long
Private Type RECT
Left As Long
Top As Long
Right As Long
Bottom As Long
End Type
Private Declare Function FillRect Lib "user32" (ByVal hdc As Long, lpRect As RECT,
ByVal hBrush As Long) As Long

Private Sub IPrintExt_DrawPage(ByVal Options As Variant, ByVal hdc As Long, ByVal
Page As EXPRINTLib.IPage, pbContinue As Boolean)
    Dim r As RECT
    GetClipBox hdc, r
    FillRect hdc, r, 1

    ' Add your own drawing code

End Sub

Private Property Get IPrintExt_PageCount(ByVal Options As Variant) As Long
    IPrintExt_PageCount = 2
End Property
```

Now, we have to check the code, by adding a new instance of exPrint component to the form and a new command button to the same form. The code will look like follows:

```
Implements IPrintExt

Private Declare Function GetClipBox Lib "gdi32" (ByVal hdc As Long, lpRect As RECT)
As Long
Private Type RECT
Left As Long
Top As Long
Right As Long
Bottom As Long
End Type
Private Declare Function FillRect Lib "user32" (ByVal hdc As Long, lpRect As RECT,
ByVal hBrush As Long) As Long
```

```
Private Sub IPrintExt_DrawPage(ByVal Options As Variant, ByVal hdc As Long, ByVal
Page As EXPRINTLib.IPage, pbContinue As Boolean)
    Dim r As RECT
    GetClipBox hdc, r
    FillRect hdc, r, 1

    ' Add your own drawing code

End Sub

Private Property Get IPrintExt_PageCount(ByVal Options As Variant) As Long
    IPrintExt_PageCount = 2
End Property

Private Sub Command1_Click()
    With Print1
        Set .PrintExt = Me
        .Preview
    End With
End Sub
```



How do I implement the IPrintExt interface in VC?

In a VC++ project you need to use the #import directive to import the exprint.dll file as a new namespace. Once that the exprint.dll file is imported, the VC++ generates a new namespace where you can find the definition for IPrintExt interface, that must be implemented by C++ clients.

To implement an interface, you must have created a project as an ATL COM application or as an MFC application that contains ATL support. You can use the ATL Project Wizard to create an ATL application, or add an ATL object to your MFC application to implement ATL support for an MFC application. Once you create the project, to implement an interface, you must first add an ATL object. See Adding Objects and Controls to an ATL Project for a list of wizards that add objects to your ATL project. Once you have added the object or control, you can implement other interfaces, located in any available type library, using the Implement Interface Wizard.

To implement an interface

- In Class View, right-click the class name for your ATL object.
- Click Add from the shortcut menu, and then click Implement Interface to display the Implement Interface Wizard.
- Select the interfaces to implement from the appropriate type libraries and click Finish.
- In Class View, expand the object's Bases and Interfaces node to see the interface you have implemented, and then expand the interface's node to see its available properties, methods