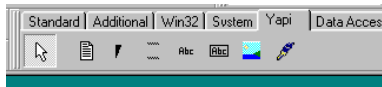


Yapi Programmers Manual

Introduction

Yapi (yet another printer interface) is a set of components to produce printouts (reports) from Delphi programs. It is lightweight and simple to use, does not assume use of a database, and can be used in a wide range of different Delphi applications.

It consists of seven components as follows:



These are (in order)

yapiPaper	A component to control the overall printout format.
yapiTab	A component to set a tab position on the paper
yapiHeaderFooter	A component to make headers and footers on the page
yapiText	A components to control generated text (e.g. fonts etc)
yapiGridText	A component to make grids (tables) of textual information
yapiImage	A component to print a bitmap
yapiPaintBox	A component to that has a “Paint” event

Principles of operation

Yapi creates printout by setting up the font styles, margins, tabs etc using a visual interface (wysiwyg). The actual printed content is created with Pascal code. Normal printing operations such as print preview are handled with simple method calls.

The steps in the creation of a printout are as follows:

1. Visual (wysiwyg) set-up:

- a) Place a yapiPaper component on your form. This component will NOT be visible at run time.
- b) Set up the margins and options properties of yapiPaper.
- c) If required, add a print dialogue box to your form and attach the yapiPaper to it.
- d) Place yapiTabs on the yapiPaper as required, and set their positions.
- e) Place yapiHeaderFooters on the yapiPaper as required, and set their positions and contents.
- f) Place a yapiText on the yapiPaper for every font, style, and colour required in the final report.
- g) Place yapiGridText components on the yapiPaper if grids are required.
- h) Place yapiImage components on the yapiPaper if bitmaps (e.g. for logos) are required.
- i) Place yapiPaintbox components on the yapiPaper if these are required.

2. Programming

Printed content is created using a single block of code. This ‘linear’ programming model greatly simplifies the creation of complex reports compared to event based program where the complexity is spread amongst a lot of small routines.

The typical format of programming is just:

```
Paper.clear;           // Clear the paper
Body.writeln( .... ) ; // Put data in the printout
Paper.preview;         // Preview – print.
```

Paper is a component of type TyapiPaper;
Body is a component of type TyapiText.

The clear method clears any existing printer contents. The writeln methods add text to the printout using the font and style of the yapiText component. The preview method activates the print preview screen. Printing can be done either from the print preview screen, or by directly using a print method.

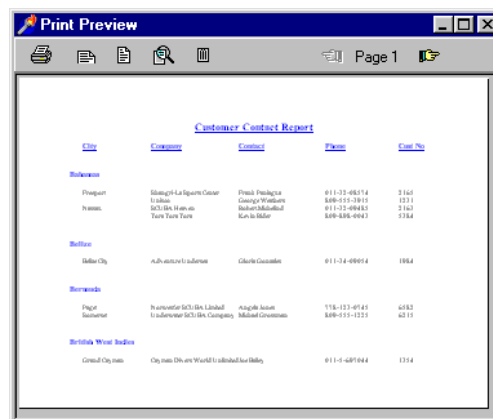
Only the yapiPaintBox uses an event. This is called by the system to get graphics for both the print preview, and the final printing. Program code in this event will have to take account of the size and position of the paint box.

Technical Note:

A report is built and printed by the yapiPaper component. Printed content is mostly generated by yapiText components. Any yapiText component will put its contents into the yapiPaper on which it resides (its parent). Placing any of the yapi components on anything other than a yapiPaper is not productive.

3. Using the Preview

The Yapi system includes a preview screen that gives a lot of control to the end user.



In the Preview screen, the user may:

- Control the end point of each page to remove widows and orphans
- Adjust the page margins either individually page by page or across the whole report.
- Adjust the tab positions either individually page by page or across the whole report.
- Print multiple copies and/or select a range of pages to print.

These features significantly enhance the programs using Yapi. As a programmer is it sometimes impossible to exactly control report output due to the variable nature of the reported data. The adjustment of the final report by the user means that your programs can be used to produce better-looking reports.

5 Minute Start

It is recommended that the reader read the YapiOverview pdf, and try the 5-minute start tutorial before reading further.

Component Reference

A programmer's reference for all of the Yapi components follows:

Enumerated Types

The following enumerated types are defined. They are used in component properties, and can be set at runtime. They may also be for some method calls.

```
TyapiJustify=(yapLeft,yapCenter,yapRight); For controlling justification
```

```
TyapiPaperType=(yapA4,yapLetter,yapCustom); Type of paper in printer
```

```
TyapiUnits=(yapInches,yapMillimeters);
```

```
TyapiOrientation=(yapPortrait,yapLandscape);
```

```
TyapiHeaderType=(yapHeader,yapFooter); Used to specify header or footer
```

```
TyapiPreviewOptions=set of  
  yapShowMarginleft, yapShowMarginright, yapShowMargintop,  
  yapShowMarginBottom, yapShowMarginAll, yapShowTabs,  
  yapShowPageBreak, yapAdjustMarginleft, yapAdjustMarginright,  
  yapAdjustMargintop, yapAdjustMarginBottom,  
  yapAdjustMarginAll, yapAdjustTabs);
```

This set controls exactly what margins and tabs the user is allowed to see and adjust in the print preview screen.

```
TyapiImageDirection=(yapUpFromLineTop,yapUpFromLineBottom,  
  yapDownFromLineTop,yapDownFromLineBottom,yapAsPlaced);
```

Used in for controlling Image and Paint box positioning

```
TyapiStatistics=record  
  pageNumber:Integer; // page of next entry  
  position:double;    // position on page  
end;
```

This record is used in a method of yapPaper to establish the current position on the page.

yapiPaper (inherits from TcustomPanel)

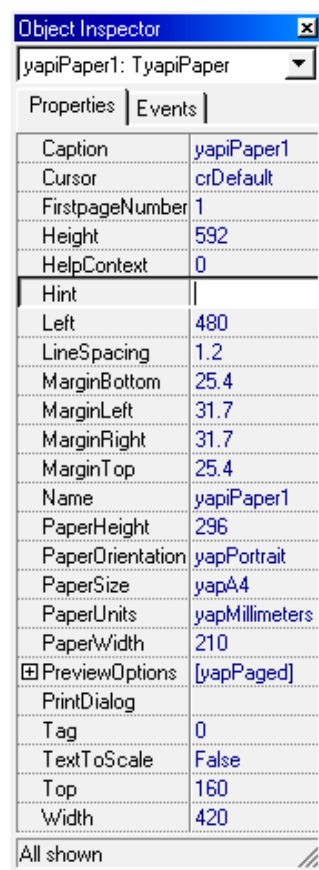
The paper component performs two functions.

- It specifies the general parameters of the printout such as margins etc.
- It is used as a container for all the generated output such as text and graphics.

It is possible to have two (or more) paper components on a single form. Text generated by call to yapiText components will put their data into the paper component on which they are dropped. It is not possible to use any of the other yapi components unless they are dropped on a Paper component.

Paper components are invisible at run time.

Properties of yapiPaper:



Object Inspector	
yapiPaper1: T yapiPaper	
Properties	Events
Caption	yapiPaper1
Cursor	crDefault
FirstpageNumber	1
Height	592
HelpContext	0
Hint	
Left	480
LineSpacing	1.2
MarginBottom	25.4
MarginLeft	31.7
MarginRight	31.7
MarginTop	25.4
Name	yapiPaper1
PaperHeight	296
PaperOrientation	yapPortrait
PaperSize	yapA4
PaperUnits	yapMillimeters
PaperWidth	210
PreviewOptions	[yapPaged]
PrintDialog	
Tag	0
TextToScale	False
Top	160
Width	420
All shown	

- Component Caption - defaults to the same as Name property.
- Inherited property – not used.
- The first page number. Used in the preview plus header or footers.
- Height of the component on the form.
- Inherited property – not used.
- Inherited property – not used.
- Position of the component on the form.
- Controls the vertical spacing of lines.
- The margin at the bottom of the page.
- The margin at the left of the page.
- The margin at the right of the page.
- The margin at the top of the page.
- The component Name.
- The height of the paper, in inches or millimetres.
- Controls portrait or landscape printing.
- Set to A4 or Letter, or custom for other paper sizes.
- Set to Inches or Millimetres.
- The width of the paper, in inches or millimetres.
- Set of preview option flags. Controls the print preview.
- Can be set to reference a print dialog component.
- Inherited property – not used.
- If true, then the components on the page are scaled.
- Position of the component on the form.
- Width of the component on the form.

Notes:

1. The above screen capture shows the default values for each of the properties.
2. The Pages are numbered from FirstpageNumber. The page numbers are used both in the preview screen, and also for page numbering if used in the headers and footers.
3. Increasing LineSpacing increases the space allowed for each line without adjusting the sizes of the characters on the lines. The paper operates on the principle of that the height of a font (in inches) is the font size divided by 72. A LineSpacing of one tends to make the lines too

cramped in normal viewing. The LineSpacing property is used to adjust for this, and a value of 1.2 gives a reasonable output.

4. The units for margins and ALL other metrics are either inches or millimetres, depending on the setting of the property PaperUnits. Tab indents are similarly set.
5. The PrintDialog property can be set to reference any print dialog component on your form. If this property is set, then the print preview screen will use that print dialog screen as a print control. This will allow the end user to preview the whole of the document, and (for example), while previewing, choose to print only some pages, print multiple copies etc.
6. If TextToScale is true, then the components on the page will be scaled to their relative sizes. If false then they will remain in the default sizes.
7. It is usual to rename the yapiPaper1 component to "Paper".
8. It is recommended that the width of the Paper be set to a value related to its true width. The default width of the paper component is 420 pixels, which is 2 pixels per millimetre for A4 paper. Having the component width closely related to the actual width makes for greater ease in setting tab positions, and also setting widths and heights of yapiImage and yapiPaintBox components.

Preview Options

The preview option flags are as follows:

yapPaged	Always true for yapiPaper property
yapShowMarginleft	Shows the left margin indicator
yapShowMarginright	Shows the right margin indicator
yapShowMargintop	Shows the top margin indicator
yapShowMarginBottom	Shows the bottom margin indicator
yapShowMarginAll	Shows the all margin indicators
yapShowTabs	Shows the tabs
yapShowPageBreak	Always false for yapiPaper property
yapAdjustMarginleft	Allows the user to adjust the left margin in the preview
yapAdjustMarginright	Allows the user to adjust the right margin in the preview
yapAdjustMargintop	Allows the user to adjust the top margin in the preview
yapAdjustMarginBottom	Allows the user to adjust the bottom margin in the preview
yapAdjustMarginAll	Allows the user to adjust the all margins in the preview

There is some interaction between these options

The items non settable in the properties can be used to the call to the placeoncanvas method described below.

Common Methods of yapiPaper

In general, methods of yapiPaper control the operation of the paper, except for the placing of data on it. Placing of data is controlled by yapiText etc.

procedure clear	Clears the contents of the paper (all text and images).
procedure preview	Activates the preview screen.
procedure print	Prints without using the preview screen.
procedure newpage	Creates a new page in the printout (see note below).
procedure backup	Backs up across the line see Note below.
procedure drawline(Font:TFont)	Draws a horizontal line from left margin to right margin. The thickness of the line is set by the thickness of the underline bar of a text of the size of the Font parameter (increase font to increase thickness).

<pre> procedure settabspacing (values:array of double;leftmargin:double=-1;rightmargin:double=-1); </pre>	<p>Overrides the design time tab positioning e.g. <code>paper.settabspacing ([1,2.5,1.5])</code> sets the tab separations to 1 inch, 2.5 inches, and 1.5 inches (or millimetres). The new tabs remain in position for all lines until the next call to “settabs” or “resettabs”. The leftmargin and rightmargin parameters are optional, and, if set to a positive value, will change the position of the left hand and right hand margins.</p>
<pre> procedure settabpositions(values:array of double;leftmargin:double=-1;rightmargin:double=-1); </pre>	<p>Overrides the design time tab positioning as above, but the values are the positions of the tabs across the page. The values should be increasing in size e.g. <code>settabpositions([1,3.5,5])</code> is equivalent to the above call to settabspacing.</p>
<pre> procedure resettab; </pre>	<p>Removes programmed tabs. The tabs revert back to the wysiwyg designed tabs.</p>
<pre> procedure everypage; </pre>	<p>Any text added to the screen after this call will appear on the top of every page (see below).</p>
<pre> procedure otherpages; </pre>	<p>Any text added to the screen after this call will appear on the top of every page except the first page (see below).</p>
<pre> procedure thispage; </pre>	<p>Any text added to the screen after this call will appear on only the current page. This method is default and ends the actions of “everypage” and “otherpages”.</p>
<pre> procedure cancelEveryPage </pre>	<p>All the text added by everypage and otherpages will be cancelled from this position on in the report.</p>
<pre> procedure linecolor(col:Tcolor) </pre>	<p>Set the colour under the text of the current line.</p>
<pre> function getXPositon:double </pre>	<p>Returns the X position that the next item will print.</p>
<pre> function getTabPosition(tabNumber:Integer):double </pre>	<p>Returns the position of the tab.</p>

Notes:

1. “backup” returns the cursor to the start of the previous output. An example of its use would be.

```

Body.Write('Test 1');
Paper.backup;
Body.Write('XXXX')

```

Produces a result like : ~~Test1~~

2. “newpage” produces a "soft" page break. If the user adjusts this break, then it becomes the same as any other automatically generated page break.
3. Thispage, everypage, and otherpages are procedures that may be used to control columns headings. When rows of data are listed in columns (eg a spreadsheet or a database fetch), the usual practice is to put the names of the columns on top. If the rows run to several pages then the names should be on the top of each page. The calls operate like:

Problem:

We have a stringgrid on the form, and we wish to print this to paper. The string grid is long, and the printout will span several pages in the report. The top row of the string grid is a greyed fixed row, and contains the titles of each column.

We wish the printout to have the text “Sales Report” on every page. We also wish that the column titles be printed on each page, immediately over the printout of the stringgrid itself. A black line will separate the column titles and the grid.

This code does that:

```
paper.clear;
paper.everypage;

// The code below following prints at the top of every page.
// First write the name of the report:
Header.writeln('Sales Report');
// write the titles of the columns contained in stringgrid Row zero .
for j:=0 to stringgrid1.colcount-1 do
    Body.writecentre(stringgrid1[j,0],j); // centered!
Body.writeln;
paper.drawline;//a decorative line - under the column headings on every page
paper.thispage;
// The code below following prints page by page
for i:=1 to stringgrid1.rowcount-1 do begin
    for j:=0 to stringgrid1.colcount-1 do
        Grid.writeat(stringgrid1[j,i],j);
    Grid.writeln;
end;
paper.preview;
```

Otherpages offers the ability to customise the first page of a report.

- a) Create the first page heading information using thispage (default)
- b) Use otherpages to create the heading information for all other pages.
- c) Use thispage to create the bulk of the report.

Uncommon Methods of yapiPaper

procedure ScaleFont(Font:TFont); Scales a font – see YapipaintBox.

constructor Create(AOwner: TComponent);

destructor Destroy;

procedure getstatistics(var stats:TyapiStatistics);overload;
call to: getstatistics(stats,[yapPaged])

procedure getstatistics(var stats:TyapiStatistics;const options:TyapiPreviewOptions);overload;
Gets the statistics about the current page and position
Use this call if you want to manage widows and orphans
in your own program code.
If option are [] (empty set) then the whole report will be
on one page.

procedure placeoncanvas(canvas:TCanvas;canvaswidth:integer;pagenumber:Integer);overload;
call to: placeoncanvas(canvas,canvaswidth,pagenumber,[yappaged]);

placeoncanvas(canvas:TCanvas;canvaswidth:integer;pagenumber:Integer;const
options:TyapiPreviewOptions);overload;

The “placeoncanvas” method is used by both the yapi’s print preview, and actual printing process. It scales the data in the printout and renders it onto the canvas. It can be used for your own print preview, or any other Tcanvas oriented processing. The parameters are:

Canvas	- receiving canvas
Canvaswidth	- width of canvas. Both horizontal and vertical scaling use this for scaling
Pagenumber	- which page to render.
Options	- useful values are [yapPaged] and [] empty set
	- if paged then only 1 page will be rendered, else the whole of the report
	- will be placed continually on the canvas.

yapiTab (inherits from TGraphicControl)

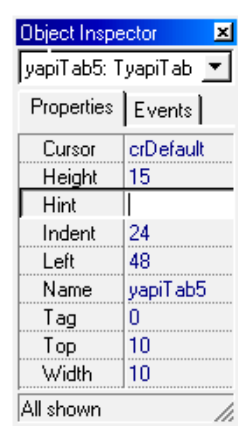
The tab component provides a visual setting of tabs. They are dropped on to the paper component and dragged into position using the mouse. Alternatively, their positions can be adjusted by setting the Indent property.

Text is placed at the tab positions with writeattab methods.

The tabs can be placed in any order. They are automatically reordered at program run time such that the left most tab is tab 1, then next leftmost is tab 2 etc. Tab 0 always refers to the left margin.

Do not place tabs on, or to the left of, the left margin.

Properties of yapiTab:



Inherited property – not used
Height of the component on the form – not used
Inherited property – not used
Position (in inches or millimetres) of the tab
Position of the component on the form
The name of the component.
Inherited property – not used
Position of the component on the form
Width of the component on the form

Notes:

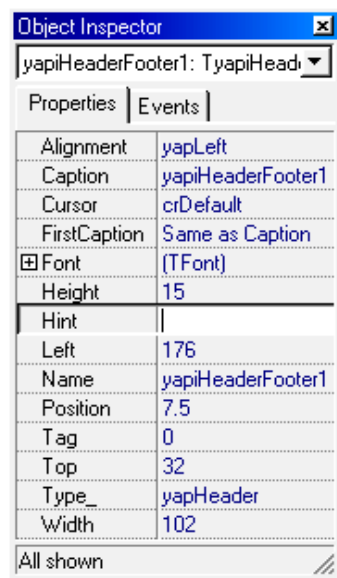
1. Indent is the important property. It specifies the position of the tab across the paper.
2. Indent and Left are linked. Changing one changes the other.

There are no events or methods for yapiTab

The tab positions created with tab components can be overwritten at run time with the settabspacing or the settabpositions methods. They can be restored with the resettab method.

YapiHeaderFooter (inherits from TCustomLabel)

Properties of YapiHeaderFooter:



Object Inspector	
YapiHeaderFooter1: TCustomLabel	
Properties	Events
Alignment	yapLeft
Caption	YapiHeaderFooter1
Cursor	crDefault
FirstCaption	Same as Caption
Font	TFont
Height	15
Hint	
Left	176
Name	YapiHeaderFooter1
Position	7.5
Tag	0
Top	32
Type_	YapHeader
Width	102
All shown	

Controls alignment of text. Left, centre or right allowed
The text to be placed in the header or footer (change this)
Inherited property – not used
Caption to be used in the First page instead of Caption
Font for both design and printing
Inherited property - not used
Inherited property - not used
Position of the component on the form
The name of the component.
The position on the paper (in mm or inches).
Inherited property – not used
Position of the component on the form
Either YapiHeader or YapiFooter
Width of the component on the form

Notes:

1. The Position is measured from the top for a header, or from the bottom for a footer.
2. The caption may include %d which will be replaced by a page number.
3. Leave FirstCaption to 'Same as Caption' to make first page header footers same as rest
4. A printout can have many headers and footers (one left justified one for right etc.)

There are no events or methods for YapiHeaderFooter

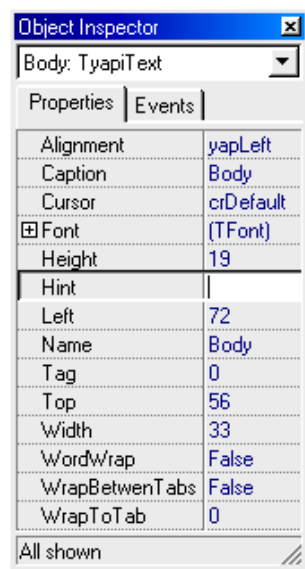
All headers and footer components are placed on every page of the report. No programming is required.

yapiText (inherits from TyapiBaseText which inherits from TCustomLabel)

The Text component is the workhorse of the yapi system. The bulk of the text is placed on the report using instances of this component. The most important property of this component is the Font (including colour). All text created using the write and writeln methods take their font from the Text component used to generate them.

Other aspect of the output such as centre and right alignment can be set, as can word wrapping.

Properties of yapiText:



Alignment yapLeft Controls alignment of text. Left, Center or Right.
Caption Body Component Caption - defaults to the same as Name property
Cursor crDefault Inherited property – not used
Font (TFont) The font (including typeface, size, colour, underlining etc).
Height 19 Height of the component on the form
Hint Inherited property – not used
Left 72 Position of the component on the form
Name Body The name of the component.
Tag 0 Inherited property – not used
Top 56 Inherited property – not used
Width 33 Width of the component on the form
WordWrap False If true text will wrap at the right hand margin
WrapBetweenTabs False If true text will wrap at the next tab position
WrapToTab 0 Word wrapped text will start at this tab.

There is an additional public but unpublished property “NextWidth”. See below.

Notes:

1. The font property (including typeface, size, colour, underlining etc) controls both how the component appears on your form, as well as the appearance of printout produced by it.
2. Alignment is normally Left, but Center and Right allow for creation of titles etc. Alignment is ignored for calls to tabs (eg writeattab and writecentre etc.). Alignment is used for the write and writeln methods only.
3. It is normal to rename yapiText components to “Title”, “SubTitle”, “Normal”, “Body” etc.

Methods of yapiText

procedure write(text:string)	Puts a string into the printout
procedure writeln(text:string)	Puts a string in a printout and goes to next line
procedure writeln	Goes to next line only.
procedure writeAtTab(text:string;tabNumber:Integer)	Puts a string at a tab position.
procedure writeLeft(text:string;tabNumber:Integer)	Puts a string at a tab (same as writeAtTab).
procedure writeCentre(text:string;tabNumber:Integer)	Puts a string centred between tabs.
procedure writeRight(text:string;tabNumber:Integer)	Puts a string right justified to the next tab.
function getWidth(s:string):double	Gets the width (in inches or mm) of the specified text.
property NextWidth:double	Sets the printable width of the next write.

`procedure lefttab(text:string;tabNumber:Integer)` Obsolete method.
`procedure writewith(text,spacing:string)` Obsolete method.
`procedure writeatabwith(text:string;tabNumber:Integer;spacetext:string)` Obsolete method.

Notes

1. `NextWidth` is used to control the ‘spill’ of text from one column to the other. It does this by limiting the width of the textual output. A `canvas.TextRect` call is made rather than a `Canvas.TextOut` call.

The value of size must be set immediately before a call to `writeln`. E.g.

```
Body.NextWidth:=1.5;
Body.write('This is very long text that will be clipped')
```

The text object does not remember the size for more than one call to write.

On the preview screen, the text may be clipped half way through a character. Printers don’t exhibit this property.

2. The obsolete method `LeftTab` puts text at a tab position, but extending out to the left of it, rather than to the right. This makes the output right justified from the tab position. It is useful for lists of numbers e.g. financial reports.

```
Lefttab('This Text',9)
Is the same as
WriteRight('This Text',8);
```

3. The obsolete methods `writeWith` are for setting text positions. Setting tabs to control the position is OK, but it has limitations. Two obvious limitations are
 - a) It needs to be done at design time
 - b) Tabs are set for the extent of the report – for all pages.

There are two alternatives.

Use the `paper.settabs` method. This can set the tabs for the whole report, or just for several `writeln`s. This is the preferred method.

“`Writewith`” gives an alternative method of tabulating data. There are two strings passed to it. The first is the string to be printed; the second is a spacing string. – The text cursor is moved as if the second string had been printed.

4. Wrapping can be margin to margin. This is suitable for bulk text – e.g. text taken from memo components. As alternative text can be wrapped between tabs.

Wrapped text by default starts again at the left margin. It is possible however to wrap to any tab position. This allows indented and wrapped paragraphs. – See the Fish Fact demonstration.

yapiGridText (inherits from TyapiBaseText which inherits from TCustomLabel)

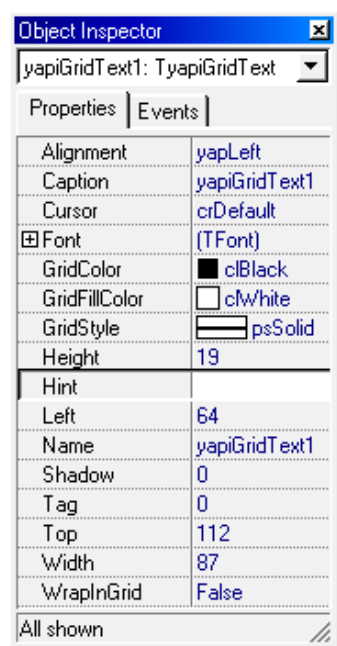
The GridText component is exactly the same as the text component, except that a border is drawn around the text. This border extends from the start position of the text (typically a tab) to the logical end – either the right margin or the next tab.

The design of the border is such that each of the four borderlines will coincide with the border of the adjacent text – adjacent either horizontally or vertically. In this manner, many horizontally and vertically adjacent elements make up a grid, just like a spreadsheet or a string grid.

The methods writeLeft, writeCenter, and WriteRight can be used to put text into the grid of the spreadsheet with the appropriate justification

Properties and methods of yapiGridText are the same as yapiText with the addition to two properties to control the border.

Properties of yapiGridText:



Object Inspector	
yapiGridText1: TyapiGridText	
Properties Events	
Alignment	yapLeft
Caption	yapiGridText1
Cursor	crDefault
Font	(TFont)
GridColor	clBlack
GridFillColor	clWhite
GridStyle	psSolid
Height	19
Hint	
Left	64
Name	yapiGridText1
Shadow	0
Tag	0
Top	112
Width	87
WrapInGrid	False
All shown	

Controls alignment of text. Left, Center or Right.
Component Caption - defaults to the same as Name property.
Inherited property – not used.
The font (including typeface, size, colour, underlining etc).
The colour of the line surrounding the cell.
The colour of the background of the cell.
The line drawing pattern of the line surrounding the text.
Height of the component on the form.
Inherited property – not used.
Position of the component on the form.
The name of the component.
Set for decorative box shadow. Value is % height.
Inherited property – not used.
Position of the component on the form.
Width of the component on the form.
If true, text will wrap between tabs (i.e. in the cell).

Notes

1. WrapInGrid is the same as WrapBetweenTabs for yapiText. Wrapping is designed to occur inside the box in a similar way that Word wraps text inside a table.

2. The shadow property creates text like:

Big Title

The value in shadow controls the depth of the shadow. Zero is no shadow, 0.2 is 20% of the height of the box. A value of 0.2 gives a good result. Print spaces on each side of the internal text.

3. Methods of yapiGridText are the same as yapiText.

YapiImage (inherits from TGraphicControl)

The Image component places bitmaps on to the printout. Images can be placed in the following manner:

- In a wysiwyg place on the first page (e.g. a company logo on the first page of a report).
- In a wysiwyg place on the every page (e.g. a logo on the every page).
- Centered in the body of the printout (e.g. bit maps placed with other text above and below)
- Imbedded in the text itself (e.g. really cool little graphical bullets)

Images are treated just like text. Their properties are designed in a wysiwyg way, and their placement in the printout is controlled by write and writeln method calls.

The sources of printed images can be:

- The picture loaded into the component itself (double click the Picture property)
- A bitmap (*.bmp) file specified with a write or writeln method call.
- Another image component (e.g. a DBImage)

Properties of yapImage:

Object Inspector	
yapiImage1: TyapiImage	
Properties	Events
AffectsLineSpacing	True
Alignment	yapLeft
Cursor	crDefault
Height	105
Hint	
ImageDirection	yapDownFromLineTop
ImageScale	yapDesign
Left	240
Name	yapiImage1
Picture	(None)
Tag	0
Top	160
Width	105
Zoom	1
All shown	

- If false, the text line spacing will not include this image.
- Controls alignment of Image. Left, Center or Right.
- Inherited Property not used.
- Height of Component.
- Inherited Property not used.
- Controls how Image is placed on paper.
- Controls scaling of the Image.
- Position of component on paper.
- The name of the component.
- The picture of the component.
- Inherited Property not used.
- Position of component on paper.
- Width of the Component.
- Zoom factor for use when scaled to original.

Notes:

1. The picture can be loaded from file like the TImage component.
2. If AffectsLineSpacing is true, the vertical text cursor will be moved down by the height of the picture. If there is text on the same line, then the line height will be the greater of: the picture height, and the height of the tallest text. The line spacing will always be affected in the same way regardless of the setting of ImageDirection.
3. Alignment is used only for write and writeln, in a similar way to the alignment of yapText.

4. The vertical image placement can be controlled relative to either the top of the current line or the bottom of the current line.

Spacing can be:

DownFromLineTop The image will descend from the top of the current line.
UpFromLineBottom The image will descend from the top of the current line.
UpFromLineTop,yap The image will ascend above the top of the current line.
DownFromLineBottom The image will descend below the bottom of the current line.
AsPlaced The line position (both X and Y) will be totally ignored. The bit map will be placed exactly as designed on the yapIPaper component.

5. The scaling of the image size can be:

YapDesign – the scale of the image on the paper is proportional to the scale of the component on the yapIPaper component. Increasing height and depth of the component increases the size of the final image on the paper.

YapOriginal – the width and height of the yapImage component are ignored. The image is scaled to proportional to the size of the original bitmap in pixels.

6. The Zoom property is ignored in yapDesign mode. In yapOriginal mode, the bitmaps is zoomed by this amount.

Methods of yapImage

procedure write	Puts an image on the paper.
procedure write(filename:string)	Puts an image on the paper from a file.
procedure write(pic:TPicture)	Puts an image on the paper from another Image
procedure writeln	Puts an image on the paper and goes to next line.
procedure writeln(filename:string)	Writeln from file
procedure writeln(pic:TPicture)	Writeln from another Image
procedure writeatab(tabnumber:Integer)	Write at tab
procedure writeatab(tabnumber:Integer;filename:string)	Write at tab from file
procedure writeatab(tabnumber:Integer;pic:TPicture)	Write at tab from another Image.
function getWidth:double	Gets the width (in inches or mm) of the Image.

Notes:

Writing of images is similar to text. The images are placed on the paper in the current line, after the existing text, and may be followed by more text.

To design a logo at the top of a page:

Drop an Image component on to the page.
Load the logo into the picture property of the image (double click it).
Set the ImageScale to yapDesign
Set the Image Direction to yapAsPlaced.
Set AffectsLineSpacing to false.

Use writeln once to put it on the page. Use paper.everypage to put it on the top of all pages.

It is an easy matter to create a report from a database including all the images stored in the graphics blobs. The method call write(pic:TPicture) will take a copy of each DBImage bitmap. Watch out for memory usage however, as all bitmaps will have to be stored in memory at the same time, and only released with a paper.clear. See the FishFacts demo.

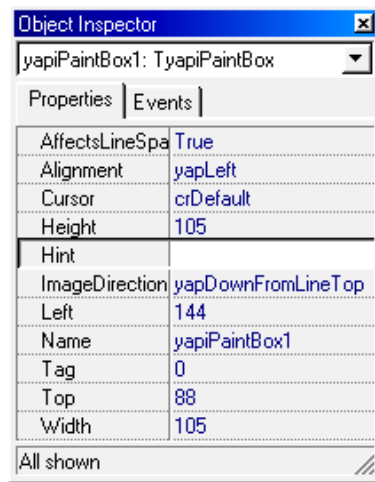
YapiPaintBox (inherits from TGraphicControl)

The PaintBox component allows for the use of program drawn graphics within the printout.

The programmer provide and OnPaint method. This is called whenever yapi needs to draw the object – either in the preview screen, or on the printed paper.

Paintboxes can be used for many purposes, but the most obvious is to place a graph on the printout.

Properties of yapiPaintBox:



If false the text line spacing will not include this paintbox.
Controls alignment of drawing. Left, Center orRight.
Inherited Property not used.
Height of Component.
Inherited Property not used.
Controls how paintBox is placed on paper.
Position of component on paper.
The name of the component.
Inherited Property not used.
Position of component on paper.
Width of the Component.

Notes:

1. AffectsLineSpacing, Alignment, and ImageDirection operate exactly as in the yapiImage component.

Methods of yapiPaintBox

procedure write	Puts an picture on the paper.
procedure write(ref:TObject)	Puts an picture on the paper with a reference.
procedure writeln	Puts an picture on the paper and goes to next line.
procedure writeln(ref:TObject)	Writeln with a reference
procedure writeattab(tabnumber:Integer)	Write at tab
procedure writeattab(tabnumber:Integer; ref:TObject)	Write at tab from file
function getWidth:double	Gets the width (in inches or mm) of the picture.

The reference is past to the OnPaint event. This is a mechanism for when there are many the PaintBoxes. It allows the OnPaint method to locate an object that it is associated with (in a simiar way to the Sender parameter in conventional event).

Event

The paintbox has a single event

```
procedure TForm1.YapiPaintBoxPaint(Canvas: TCanvas; left, top, right,
    bottom: Integer; Ref: TObject; PaintboxNumber, page: Integer);
```

The parameters are as follows:

Canvas. Use this to write the graphics on. It will be either the canvas of the preview screen, or the canvas of the printer.

Left, Top, Right, Bottom. This defines the area in which to paint. You must automatically scale your drawing to fit within this rectangle.

Ref. The object passed to the write method call that corresponds to this event.

PaintBoxNumber. The number of this paint box (first is zero).

Page: The page number on which the paintbox is appearing.

Note the use of the paper method `ScaleFont`. This will scale a font from original size to the proper size. If you use `ScaleFont` before any `Canvas.Textout` calls within the `YapiPaintBoxPaint` method, the text will zoom and print correctly.

Preview options

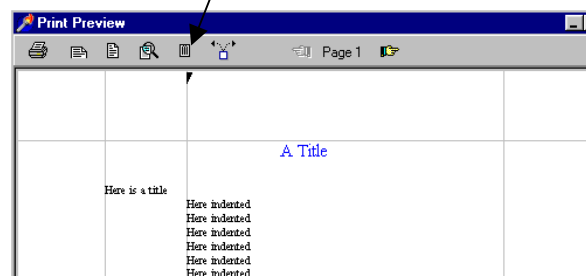
The preview of Yapi contains some unique features that allow the user to adjust the format of the report immediate before it is printed.

These features allow adjustment of the page margins, and limited adjustment of the tab positions. Normally an adjustment affects only the preview page, but the changes can be propagated to all pages.

Probably the most significant aspect of this adjustment feature is the ability of the user to adjust the bottom margin in each page to remove widows and orphans.

The operation of the preview page is best illustrated with the "simpleDemo" program. Compile and run this, Use the "Make Report" button to produce the preview.

The margins can be shown with the button



The margins become marked and may be dragged into new positions with the mouse.

The button next to right will propagate any changes made to the current page to all pages.

The programmer can control which margins are adjustable in the "Preview Options" property of the `yapiPaper` component.