Yapi

If you use Delphi and want to create printout then you need yapi.

Yapi is Yet Another Printer Interface. It can be used for simple printouts or complex reports. It is very simple to use. It can be used for any kind of reports, from complex database reports to simple printouts for non-database applications.

Yapi provides:

- The printing of text with any mixtures of font or color
- The printing of grids like spreadsheets
- The printing of bitmaps
- The printing of graphical information.

Yapi includes all the facilities that your customers expect including print preview and printer selection.

Yapi is both very simple to use and very powerful. - This PDF you are reading was produced by yapi.

The yapi web site is currently http://yapi.webjump.com

How this PDF was created

This PDF is an output from yapi. To create it, a Pascal program was written. This is a relatively simple program, and a listing of it is included. The program was run. The print button was pressed. The yapi preview screen was used to minimize widows and orphans. It was then printed to laser printer and checked. It was then printed to a Printer Driver that produces a PDF output. You are reading that output.

If you wish to print this document you may print this PDF. As a better alternative you may down load the executable that created from the yapi web site. This executable is called YapiIntro.exe, and will print this document to the best resolution of your printer.

The programs include some screen captures (using Lview Pro) and this text was prepared in Word and then pasted into the memo component on the program. This program is available as a demo and can be run.

Background to the design of yapi

As a programmer I had been using Delphi for a few years. 90% of the applications I write do not use databases. Printing was always the weakest part of Delphi, and consumed and inordinate amount of time, typically working with the printer canvas.

In 1999 I worked on a traffic analysis program. This project was heavy in data processing - but in RAM and not using a DBMS. I inherited this project from another engineer who had purchased Report Printer Pro. I did not use this tool, as the non-database operations were not easy. Instead I developed some initial yapi type components

Recently we started building a banking system for a credit union (based on SQL server). A team member evaluated QReport system, and found that it was not sufficiently flexible. The complex nature of the system added to communication constraints meant we chose not to use data aware components. This also ruled out most of the QReport components. Also, the general ledger report for this system is very complex and defies the use of data aware components. Yapi was born for this project and was used for all of the reports for the banking system, with very good feed back from all team members.

How does yapi operate

Yapi is a reversion to procedural programming principles. This is the most appropriate for report generation as this kind of programming is essentially procedural. The generation of the report "proceeds" from start to finish.

To create a yapi report a "paper" component is dropped on a form. This is used to set up the printout format (egg margins paper size etc.). "Text" component is then dropped on the paper component. These are used to set up fonts, size colour etc. The contents of the report are generated with "write" and "writeln" calls. It's that simple. Yapi then provides the print preview, printer management, and page range, number of copies etc. facilities that users expect these days.

If you wish to try this process, read the 5-minute start PDF at the Yapi web page. This takes you through the process of producing some printout. It will also work with the free yapi set.

Samples and Examples

There are a set of example and samples on the yapi web page.

The samples are PDF files of yapi output from real projects using yapi. Each is a PDF file and a brief description. Yapi users may submit samples that may be displayed with credits.

The examples are all supplied as zip files. These contain pas and .dcu files. All of these examples are single form programs so it is easy to compile and run them. They should be compatible with both Delphi 4 and 5. The examples are accompanied by executables and PDF outputs so they can view without running, or alternatively, run without compilation.

If you're here just browsing, take a moment to click on a few of the PDFs.

Working with Grids

As well as straight text output, an obvious requirement is to print grids (like string grids or spreadsheet output). One of the problems with specifying grid output is what to do at page breaks. Grids often run across multiple pages.

The yapi approach is to treat grid data like ordinary text in a box. The boxes are designed to line up with adjacent boxes. This means that when grid-text is printed in rows and columns a grid appears. The nice thing about it is that page breaks are handled without any problems at all.

The sample below shows some simple grid work.

Value	Sine	Cos
0	0.00	1.00
20	0.34	0.94
40	0.64	0.77
60	0.87	0.50
80	0.98	0.17
100	0.98	-0.17
120	0.87	-0.50
140	0.64	-0.77
160	0.34	-0.94
180	0.00	-1.00
200	-0.34	-0.94
220	-0.64	-0.77
240	-0.87	-0.50
260	-0.98	-0.17
280	-0.98	0.17
300	-0.87	0.50
320	-0.64	0.77
340	-0.34	0.94
360	0.00	1.00
380	0.34	0.94
Totals	0.34	1.94

As well as grid-text being used as above, they may be used as in the shadowed box in the previous heading above. The shadowing of the box is specified in the grid text properties box.

Images

Yapi supports images. Images are treated just like text in that they are placed in the report by a specific "write" instruction. This may seem a little strange, but it does allow a lot of control. The programmer can choose to include or exclude the image (write or not write). They can also include images wherever in the report they require (as for the bit maps in the document). They can also use software to create many images, even from a single yapiImage.

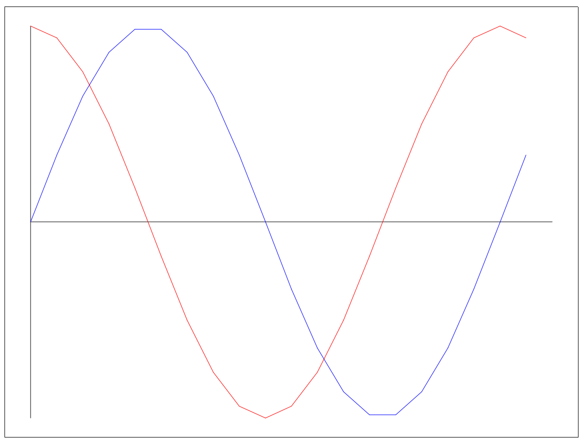
The bullets in this PDF are by yapi Images, as is the picture at the top of this PDF, and the screen captures

For another example of bitmap outputs see the fishfact example. This is based on the sample program "Fishfact" provided with the Delphi Demos (in the DB directory). The original program has been extended to produce a report on all of the fish in the database, together with the pictures of them.

PaintBox

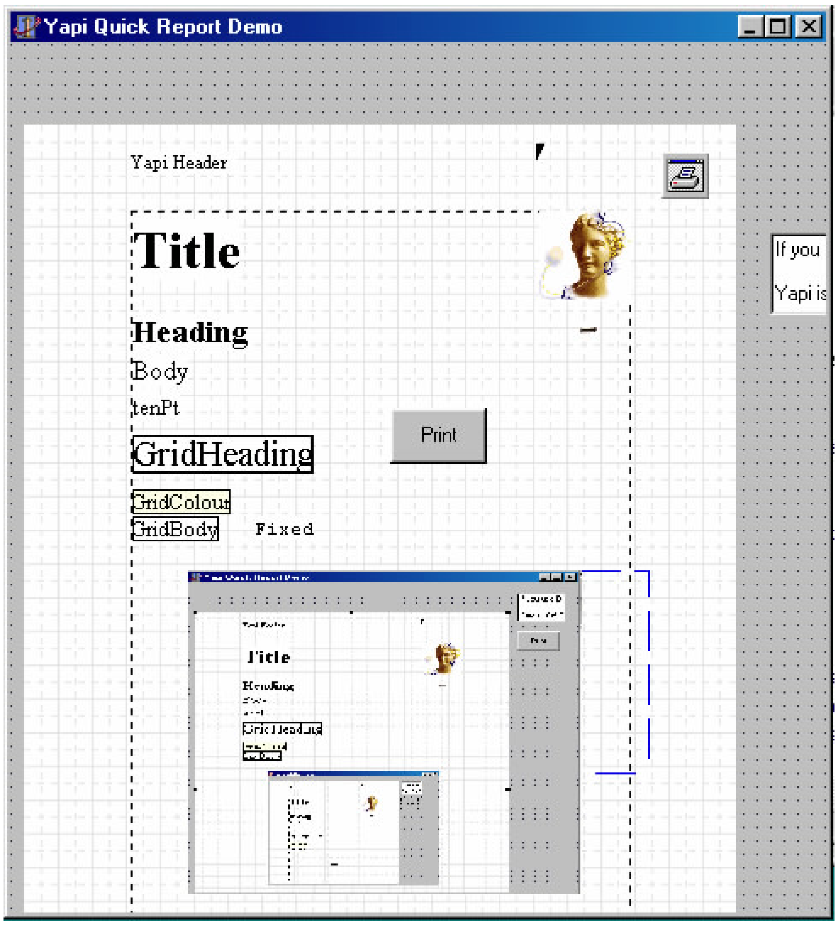
The paint box allows you to create graphics with a paint event. This is equivalent to the Delphi PaintBox component. Programmers who have used the printer canvas directly will be able to use the yapi paintbox for placing their graphics on the printer.

Most of the properties of the paint box are the same as for the Yapi Image Component. It supports a "Paint" event for the creation of the graphics.



Listing for the Program that created this PDF file

This program was created with a single form program. The form consists of a Yapi paper component, various yapi components on the paper, and a button to print the form. A screen capture of the form is as follows:



The Listing of the program is as follows:

```
unit IntroDemoUnit;


interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, yapi, ExtCtrls, Db, DBTables,printers;

type
  TForm1 = class(TForm)
    TextMemo: TMemo;
    Button1: TButton;
    Paper: TyapiPaper;
    Header: TyapiHeaderFooter;
    yapiHeaderFooter1: TyapiHeaderFooter;
    AthenaImage: TyapiImage;
    yapiTab1: TyapiTab;
    BulletImage: TyapiImage;
    Title: TyapiText;
    Heading: TyapiText;
    Body: TyapiText;
    GridHeading: TyapiGridText;
    GridBody: TyapiGridText;
    GridColour: TyapiGridText;
    tenPt: TyapiText;
    Form: TyapiImage;
    Fixed: TyapiText;
    PaintBox: TyapiPaintBox;
    PrintDialog: TPrintDialog;
    SourceMemo: TMemo;
    procedure Button1Click(Sender: TObject);
    procedure PaintBoxPaint(Canvas: TCanvas; left, top, right,
      bottom: Integer; Ref: TObject; PaintboxNumber, page: Integer;
      Printer: Boolean);
    procedure FormPaint(Sender: TObject);
  private
    { Private declarations }
    lineindex:Integer;
    sinedata,cosdata:array[0..19] of double;
    procedure PutText;
    procedure PlaceGrid;
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}


procedure TForm1.PutText;
{ this procedure puts text from the TextMemo component
 (which has the textual content of the report) on to yapi.
 It operates with a simple markup language
}
var s:string;
i:Integer;
mark:char;
begin
  with textmemo do begin
     i:=lines.count;
     while lineindex<i do begin
        s:=lines[lineindex];
        if (length(s)>1) and (s[1]='-') then begin
           mark:=lines[lineindex][2];
           case mark of
              'H' : // Print a sub heading line
                       Heading.writeln(pchar(s)+3);
              'S' : begin
                       // exit with linindex pointing to the next line
                       inc(lineindex);
                       exit;
                    end;
              'B' : begin
                      body.write('     '); // simple indent
                      BulletImage.write;   // write out our bullet - an image
                      body.write(' ');     // spacing between bullet and text
                      body.writeln(pchar(s)+3);
                      body.writeln;        // next line
                    end;
              'P' : begin
                      paper.newpage;
                    end;

           end;
        end else begin
           body.writeln(lines[lineindex]);
        end;
        inc(lineindex);
     end;
  end;
end;
```

```
procedure TForm1.PlaceGrid;
{ this procedure puts the data in the grid on the report }
var i:Integer;
tot1,tot2,v:double;
begin
   paper.settabspacing([gridbody.getWidth('Value   '),
                        gridbody.getWidth('0.00000'),
                        gridbody.getWidth('0.00000')],80);
   gridcolour.writecentre('Value',0);
   gridcolour.writecentre('Sine',1);
   gridcolour.writecentre('Cos',2);
   gridcolour.writeln;
   tot1:=0;
   tot2:=0;
   for i:=0 to 19 do begin
      gridbody.writecentre(inttostr(i*20),0);
      v:=sin(i*20*3.14159/180);
      sinedata[i]:=v;
      tot1:=tot1+v;
      gridbody.writeright(format('%.2f ',[v]),1);
      v:=cos(i*20*3.14159/180);
      cosdata[i]:=v;
      tot2:=tot2+v;
      gridbody.writeright(format('%.2f ',[v]),2);
      gridbody.writeln;
   end;
   tenPt.write('Totals');
   gridbody.writeright(format('%.2f ',[tot1]),1);
   gridbody.writeright(format('%.2f ',[tot2]),2);
   gridbody.writeln;
   paper.resettab;
end;

procedure TForm1.Button1Click(Sender: TObject);
// this is the button that does all the printing
var
i:Integer;
begin
   paper.clear;
   lineindex:=0;
   title.writecentre('Yapi',0);     // page title
   AthenaImage.writeattab(1);       // Athena bit map on top of paper.
   title.writeln;                   // do some vertical spacing with empty writeln
   title.writeln;                   // spacing
   title.writeln;                   // spacing
   puttext;                         // place text from the TextMemo onto the printer
   paper.newpage;                   // form feed on the printer
   Gridheading.writeln('     Working with Grids     '); // Shadowed title
   puttext;                         // more text from the TextMemo onto the printer
   placegrid;                       // Put the grid onto the paper
   puttext;                         // more text from the TextMemo onto the printer
   body.writeln;                    // spacing
   body.writeln;                    // spacing
   paintbox.writeln;                // Put the paintbox onto the printer
   paper.newpage;                   // form feed on the printer
   puttext;                         // more text from the TextMemo onto the printer
   form.writeln;                    // Put the screen capture of this form onto the printer
   body.writeln;                    // spacing
   puttext;                         // more text from the TextMemo onto the printer
   body.writeln;                    // spacing
   for i:=0 to sourcememo.lines.count-1 do         // put in onthe paper using a fixed font
     fixed.writeln(sourcememo.lines[i]);
   paper.preview;                   // Print preview and printing
end;
```

```
procedure TForm1.PaintBoxPaint(Canvas: TCanvas; left, top, right,
  bottom: Integer; Ref: TObject; PaintboxNumber, page: Integer;
  Printer: Boolean);
  { This is an event for filling the paintbox.
    It takes the two arrays of sin and cos and plots them. This event will be used
    whenever the graph needs to be displayed - either at the preview time or
    at print time.
    To simplify grahics programing there are routines to convert from a virtual space
    which can be convientient and "fixed" is size and space to the required coordinates
  }
  const
     // define the virtual space in ral coordinates
     MinX=-1;    // left hand side of plot has virtual coordinate of -1
     MinY=-1.1;  // Bottom of plot of plot has virtual coordinate of  -1.1
     MaxX=21;    // Right hand side of Plot has virtual coordinate of  -21
     MaxY=1.1;   // Top of Plot is has virtual coordinate of  1.1
  var i:Integer;
  function getx(val:double):Integer;      // scale coordinates to image space
  begin
     result:=left+round((val-Minx)/(MaxX-Minx)*(right-left));
  end;
  function gety(val:double):Integer;      // scale coordinates to image space
  begin
     result:=bottom-round((val-Miny)/(MaxY-Miny)*(bottom-top));
  end;
  procedure drawline(x1,y1,x2,y2:double); // draws a line in virtual space
  begin
      canvas.moveto(getx(x1),gety(y1));
      canvas.lineto(getx(x2),gety(y2));
  end;
begin
  Canvas.Brush.Color := clBlack;
  canvas.FrameRect(bounds(left,top,right-left,bottom-top));
  Canvas.Brush.Color := clwhite;
  // draw xaxis - use coordinates in virtuyal space
  drawline( 0,0 , 20,0 );
  // draw yaxis - use coordinates in virtual space
  drawline( 0,-1 , 0,1 );
  for i:=0 to 18 do begin
      // draw data in virtual space
      canvas.pen.color:=clBlue;
      drawline( i,sinedata[i] , 1+i,sinedata[i+1] );
      canvas.pen.color:=clRed;
      drawline( i,cosdata[i] , 1+i,cosdata[i+1] );
  end;
end;

procedure TForm1.FormPaint(Sender: TObject);
begin
  textmemo.visible:=false;
  sourcememo.visible:=false;
end;

end.
```