Printing in Delphi: Printing To Scale

by Xavier Pacheco

Last month I introduced you to the TPrinter class and illustrated some simple techniques for printing text, rich text and bitmaps. This month I'm going to show you how to print items to scale. This entails writing code that is not specific to a particular printing device's resolution. Additionally, this enables you to write code that uses units of measurement other than pixels, such as inches and centimetres, when using various TPrinter methods.

Mapping Mode	Logical Unit Size	Orientation (X,Y)
MM_ANISOTROPIC	arbitrary (x <> y) or (x = y)	Right/Up
MM_HIENGLISH	0.001 inch	Right/Up
MM_HIMETRIC	0.001 mm	Right/Up
MM_ISOTROPIC	arbitrary (x = y)	Right/Up
MM_LOENGLISH	0.01 inch	Right/Up
MM_LOMETRIC	0.1 mm	Right/Up
MM_TEXT	1 pixel	Right/Down
MM_TWIPS	1/1440 inch	Right/Up

➤ *Table 1: Mapping modes*

Why Print To Scale

Consider if we were to write code to draw a centimetre ruler on any given canvas. There are basically three ways to do this.

First, there's the really hard and absurd way: you place an actual ruler against your computer screen and change the parameters to your drawing methods until the measurements actually show up right. I kid you not, I've seen people do stuff like this. Well, the problem is that although your code might work fine (after hours of trial and error), you'll have to completely re-set the parameters when drawing your ruler to the printer because of the difference between screen and printer resolutions.

A second way would be to first determine how many pixels make up a centimetre on a given device context such as the computer screen or the printer. This can be determined by calling the GetDeviceCaps Win32 API function. I'll show how to do this later. This second method is fine, except that you have to make all the conversions from centimetres to pixels when you use the TCanvas drawing methods.

A third and simpler method is to change the *mapping mode* of the device context to which the drawing will be performed. By changing the mapping mode of the device context, you can use more representative units of measurement with the drawing routines.

More About Mapping Modes

I need to sidestep for a moment to give you some background information on mapping modes. I won't go into complete detail of mapping modes and coordinate systems, as this warrants a whole article to itself. Instead, I'll let you know what you need to do for this article. You'll find more detailed information on mapping modes in Delphi's online help and in many third party books *[such as one by a certain Xavier Pacheco and Steve Teixeira, by any chance? Editor]*.

When performing any type of graphic output, you normally specify a set of coordinates to indicate the location on the device context where drawing is to be performed. These coordinates are based on a pre-determined unit of measurement. For screen drawing this unit is, by default, pixels. By using different mapping modes for a device context, you can use a different unit of measurement such as inches or centimetres for the drawing coordinates.

Mapping modes allow you to define two attributes for the device context to which you wish to do your drawing routines. These attributes are: the unit of measurement for logical coordinates, and the orientation of the X,Y axis for the device context. For this article, we'll focus primarily on the units of measurement.

Table 1 shows the different mapping modes available to the Win32 system with their respective logical units of measure and orientation.

Notice that each mapping mode uses a different logical unit size. By default, the mapping mode for all device contexts is MM_TEXT. Notice that the unit size for the MM_TEXT mapping mode is 1 pixel and the orientation is the same as how you read text (left to right and down the page). Depending on the drawing operations you wish to perform, it might be more convenient to use a different mapping mode. Let's say, for example, that you want to draw a 2x2 inch rectangle to your printer. You'll probably want to use MM_LOENGLISH as your mapping mode which will change the unit of measure to 1/100th of an inch. You would do this by first using the SetMapMode API function. For example, the following code changes the mapping mode for the printer's device context to MM_LOENGLISH and then restores the old mapping mode after the drawing routines have finished (see Listing 1).

There's quite a bit to mapping modes that can be discussed and as I said that is not really the focus of this article. With what I've shown you, you'll be able to see how we can use mapping modes to perform somewhat more complex printing routines without having to worry about printer resolution. The following examples will illustrate this further.

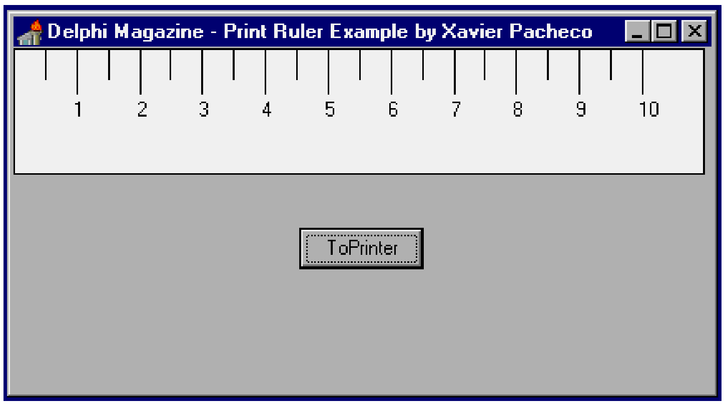
Printing The Ruler

Earlier I mentioned the possibility of printing a centimetre based ruler like that shown on the form in Figure 1. By using mapping modes, we can use the same routine used to draw the ruler on the form to draw the ruler to your printer. Listing 2 is a unit that draws a ruler to both the form's `Canvas` and to the printer's `Canvas` by pressing a button. This example is also included on this month's disk.

In Listing 2, the `PrintRuler` procedure is where all the work is done. Notice that we use the `SetMapMode` function as we illustrated earlier to change the mapping mode for the `Canvas` passed in as a parameter to `MM_LOMETRIC`. With this mapping mode, each unit of measure is 1/10th of a millimetre or 1/100th of a centimetre. This accounts for the `UnitsInCent` constant which I use as a multiplier in the drawing routines. The drawing is made up of `MoveTo` and `LineTo` method calls. These calla are used just as before, with the exception of one item.

You'll notice the final parameters for these methods are negated. The reason for this is that by changing the mapping mode for the printer's `Canvas`, we've also changed the orientation. This means that drawing doesn't occur from left to right and downwards as with the `MM_TEXT` mapping mode. Instead, drawing occurs from left to right and *upwards*. So, you must

➤ *Figure 1*



➤ *Listing 1*

```
var
  OldMapMode: Integer;
begin
  OldMapMode := SetMapMode(Printer.Canvas.Handle, MM_LOENGLISH);
  try
    Printer.Canvas.Rectangle(10, -10, 210, -210);
  finally
    SetMapMode(Printer.Canvas.Handle, OldMapMode);
  end;
end;
```

➤ *Listing 2*

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics,
  Controls, Forms, Dialogs, ExtCtrls, StdCtrls;
type
  TForm1 = class(TForm)
    ToPrinter: TButton;
    procedure Form1Paint(Sender: TObject);
    procedure ToPrinterClick(Sender: TObject);
  private
  public
  end;
var Form1: TForm1;

implementation
uses Printers;
{$R *.DFM}

procedure PrintRuler(pCanvas: TCanvas);
const
  { In MM_LOMETRIC each unit is 1/100th of a centimetre.
    Therefore, we need a multiplier to be used with the
    drawing routines }
  UnitsInCent = 100;
var
  i: integer;
  OldMapMode: integer;
begin
  { First set the mapping mode to MM_LOMETRIC and then
    draw a bounding rectangle }
  OldMapMode := SetMapMode(pCanvas.Handle, MM_LOMETRIC);
  try
    pCanvas.Rectangle(0, 0, 1100, -200);
    { Draw each line to represent 1cm and 0.5cm }
    for i := 1 to 10 do begin
      pCanvas.MoveTo(i*UnitsInCent, -0);
      pCanvas.LineTo(i*UnitsInCent, -75);
      pCanvas.MoveTo(
        i*UnitsInCent-round(UnitsInCent / 2), -0);
      pCanvas.LineTo(
        i*UnitsInCent-round(UnitsInCent / 2), -50);
      pCanvas.TextOut(
        i*UnitsInCent-5, -80, IntToStr(i));
    end;
  finally
    { Restore the old mapping mode }
    SetMapMode(pCanvas.Handle, OldMapMode);
  end;
end;

procedure TForm1.Form1Paint(Sender: TObject);
{ This is Form1's OnPaint event handler. It calls the
  PrintRuler procedure and passes the form's Canvas as
  the parameter }
begin
  PrintRuler(Form1.Canvas);
end;

procedure TForm1.ToPrinterClick(Sender: TObject);
{ This is the OnClick method for the button. It calls
  the PrintRuler procedure but passes the printer's
  Canvas as the parameter after setting up the printer
  for output. }
begin
  Printer.BeginDoc;           // Initiate a print job
  PrintRuler(Printer.Canvas); // Print the ruler
  Printer.EndDoc;             // Terminate the print job
end;
end.
```

negate the vertical coordinates in your drawing routines for the output to be displayed accordingly.

Printing A Calendar

This next example performs the same type of technique but is slightly more complex. It prints a calendar based on the month for a TCalendar component. Listing 3 shows the PRNTCAL.PAS unit containing the PrintCalendar procedure which takes as parameters a TCalendar component and a printer orientation .

The PrintCalendar procedure uses both the MM_LOENGLISH and MM_TEXT mapping modes to perform the calendar printing. Under the MM_LOENGLISH mapping mode, the specified unit of measure is used in the drawing routines. These routines are primarily for drawing the bounding rectangle and the boxes to represent calendar days.

Under the MM_TEXT mapping mode, I illustrate how to use the GetDeviceCaps API function to retrieve information for the output device. In particular, I retrieve the number of pixels per inch along the vertical and horizontal axis of the printer device. This is how you would perform your own translation of pixels and inches when not using either the MM_LOENGLISH or MM_HIENGLISH mapping modes. The rest of the PrintCalendar procedure's functionality is explained in the code's comments. An example of its usage is included on the disk with this issue.

Conclusion

This concludes this month's installment. Next month, I'll use the techniques I've already shown you to print something that's really useful. I'll show you how to print master/detail reports by printing invoices from the database tables which ship with Delphi 2. I won't be using any report writers, but I'll show you how you can hard code the printing routines yourself.

Xavier Pacheco is a Field Consulting Engineer with Borland International and co-author of *Delphi 2.0 Developer's Guide*. You can reach him by email at xpacheco@wpo.borland.com or on Compuserve at 76711,666

➤ *Listing 3*

```
unit Prntcal;
interface
uses
 Printers, Windows, Classes, Sysutils, Dialogs, Calendar;
const
  { In the MM_LOENGLISH mapping mode, each unit is
    1/100th of an inch. Therefore, this constant is used
    as a multiplier in this mapping mode. }
  UnitsInInch = 100;
  { Constant array holding strings for days of the week }
  Days: array[1..7] of string = ('Sunday', 'Monday',
   'Tuesday', 'Wednesday', 'Thursday', 'Friday',
   'Saturday');
  { Constant array holding strings for months of year }
  Months: array[1..12] of string = ('January',
    'February', 'March', 'April', 'May', 'June', 'July',
    'August', 'September', 'October', 'November',
    'December');
procedure PrintCalendar(Calendar: TCalendar;
pOrientation: TPrinterOrientation);
implementation
procedure PrintCalendar(Calendar: TCalendar;
  pOrientation: TPrinterOrientation);
var
  R: TRect;
  i: integer;
  LinePos: double;
  PixInInchX, PixInInchY: integer;
  TwnthOfInchX, TwnthOfInchY: integer;
  StrHeight: integer;
  X, Y, XPos, YPos: integer;
begin
  { Determine outer rectangle coordinates for calendar }
  R := Rect(1*UnitsInInch, 1*UnitsInInch, 8*UnitsInInch,
          Round(6.5*UnitsInInch));
  with Printer do begin
    Orientation := pOrientation; // Set orientation
    BeginDoc;                    // Start the print job
    { Change the mapping mode of the printer device
      context to MM_LOENGLISH }
    SetMapMode(Printer.Canvas.Handle, MM_LOENGLISH);
    { Draw the outer rectangle of the calendar to the
      printer's Canvas }
    Canvas.Rectangle(R.Left, -R.Top, R.Right, -R.Bottom);
    { Now draw the horizontal and vertical lines inside
      of calendar which represent days of the month }
    for i := 1 to 5 do begin
      LinePos := 0.5 + i;
      Canvas.MoveTo(1*UnitsInInch,
        -Round(LinePos*UnitsInInch));
      Canvas.LineTo(8*UnitsInInch,
        -Round(LinePos*UnitsInInch));
    end;
    for i := 2 to 7 do begin
      Canvas.MoveTo(i*UnitsInInch, -1*UnitsInInch);
      Canvas.LineTo(i*UnitsInInch,
        -round(6.5*UnitsInInch));
    end;
    { Now we're going to output text so set the mapping
      mode back to MM_TEXT }
    SetMapMode(Canvas.Handle, MM_TEXT);
    { In the MM_TEXT mapping mode, units are no longer
      represented as 1/100th of an inch as with the
      MM_LOENGLISH mapping mode. Instead, they are
      represented as pixels. Therefore, we must
      determine the number of pixels per inch along the
      device context's horizontal and vertical axis by
      using the GetDeviceCaps function }
    PixInInchX :=
      GetDeviceCaps(Printer.Canvas.Handle,  LOGPIXELSX);
    PixInInchY :=
      GetDeviceCaps(Printer.Canvas.Handle,  LOGPIXELSY);
    { Calculate 1/20th of an inch along the X and Y axes
      to be used for positioning of text }
    TwnthOfInchX := PixInInchX div 20;
    TwnthOfInchY := PixInInchY div 20;
    { Draw the day titles in their appropriate block }
    for i := 1 to 7 do begin
      StrHeight := Canvas.TextHeight(Days[i]);
      Canvas.TextOut(i*PixInInchY+TwnthOfInchY,
        round(1.5*PixInInchX)-(StrHeight+TwnthOfInchX),
        Days[i]);
    end;
    { Now draw the day numbers where the block in which
      they belong }
    for Y := 0 to 4 do begin
      for X := 0 to 6 do begin
        XPos := round((X+1)*PixInInchX)+TwnthOfInchX;
        YPos := round((Y+1.5)*PixInInchY)+TwnthOfInchY;
        Canvas.TextOut(XPos, YPos,
          Calendar.CellText[X, Y+1]);
      end;
    end;
    { Draw the month/year string on the
      upper left side of the calendar }
    StrHeight :=
      Canvas.TextHeight(Months[Calendar.Month]);
    Canvas.TextOut(PixInInchX, PixInInchY-StrHeight,
       Months[Calendar.Month]+ ' '+
       IntToStr(Calendar.Year));
    EndDoc;   // End the print job
  end;
end;
end.
```