Printing In Delphi: Printing An Invoice Report

by Xavier Pacheco

Last month I showed you how to print items to scale, a useful technique when your printed output must look the same on printers of different resolutions. This month, I'm going to show you how to print a report using only Object Pascal code. Specifically, I'm going to add a unit to the MastApp demo that ships with Borland Delphi 2.0. This unit will enable you to print a professional looking invoice for orders in Borland's sample application. All you have to do is add the unit to the application and change the code on a single event handler.

The unit is included on the disk with this issue. I didn't include the entire application because it belongs to Borland; you should, however, have this application in the \DELPHI 2.0\DEMOS\DB\MASTAPP directory. Also, I want to thank Joe Hecht at Borland's Developer Support who reviewed this article and offered some excellent advice which I included.

Setting Up The PtInvoice Unit

As stated in the introduction, I decided to use the MastApp demo to demonstrate how to print a real-world report. This is primarily because this demo is a realistic inventory application. What was missing from this demo was the ability to actually print an invoice when an order is made from the Order Form shown in Figure 1.

You'll see that the right-most button on the Order Form is a print button. All this button does, however, is print an image of the form. Therefore, I decided to create a unit that would print a professional looking invoice like that shown in Figure 2.

To set up the `PtInvoice` unit with the MastApp demo program, you only need to modify the `EdOrdersc` unit. First, you have to add `PtInvoice` to the uses clause of `EdOrders` as shown below:

```
implementation
uses DataMod, SrchDlg,
  Pickdate, PtInvoce;
```

Next, you have to modify the `PrintBtnClick` event handler for the `TSpeedButton` component on the `TEdOrderForm` to read as shown in Listing 1.

What the `PrintBtnClick` method does is instantiate a `TInvoiceReport` object, calls its `PrintInvoice` method and then frees that object. The `TInvoiceReport` class is defined in the `PtInvoice` unit and contains all the code that performs the actual printing.

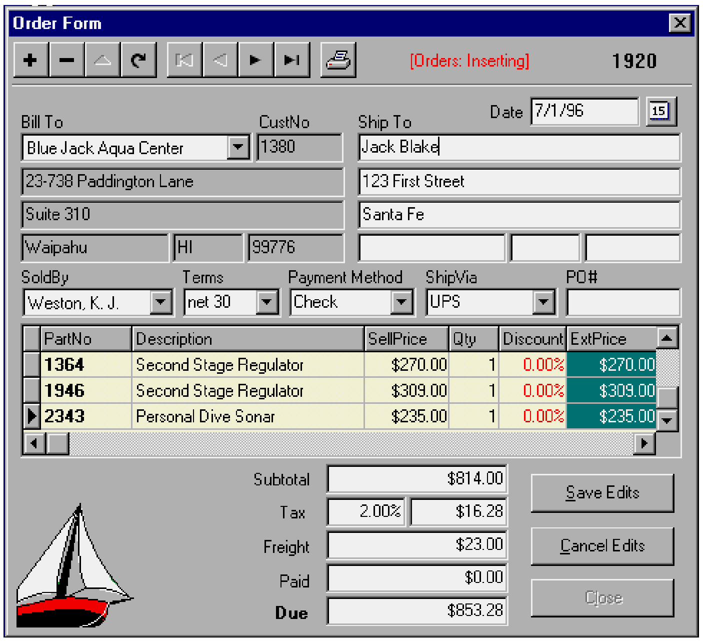
The TInvoiceReport Class

The `TInvoiceReport` class is shown in Listing 2. This class contains the public method `PrintReport` and several private methods that perform printing tasks. The reason I encapsulate the printing code into a class is because I tend to think in terms of objects by habit. Certainly, I could have created procedures to do the same, but I find that it is easier to maintain tasks which are encapsulated in their own class.

Each method of `TInvoiceReport` performs a different part of the report. In other words, I broke up the invoice report as a whole into several sub-tasks.

➤ *Listing 1*

```
procedure TEdOrderForm.PrintBtnClick(Sender: TObject);
var
  InvoiceReport: TInvoiceReport;
begin
  InvoiceReport := TInvoiceReport.Create;
  try
    InvoiceReport.PrintInvoice;
  finally
    InvoiceReport.Free;
  end;
end;
```



[Orders: Inserting] 1920

Bill To	CustNo	Ship To	Date 7/1/96
Blue Jack Aqua Center	1380	Jack Blake	
23-738 Paddington Lane		123 First Street	
Suite 310		Santa Fe	
Waipahu	HI 99776		

SoldBy	Terms	Payment Method	ShipVia	PO#
Weston, K. J.	net 30	Check	UPS	

PartNo	Description	SellPrice	Qty	Discount	ExtPrice
1364	Second Stage Regulator	$270.00	1	0.00%	$270.00
1946	Second Stage Regulator	$309.00	1	0.00%	$309.00
2343	Personal Dive Sonar	$235.00	1	0.00%	$235.00

Subtotal		$814.00
Tax	2.00%	$16.28
Freight		$23.00
Paid		$0.00
Due		$853.28

➤ *Figure 1: MastApp demo program on which this month's example is based*

Breaking Up The Report Into Sub-Tasks

This is an approach that can be used to print any type of report. As with any programming task, you look at where you can break down the entire task into several independent sub-tasks. From Listing 1 you can see that TInvoiceReport contains several methods that perform the actual sub-tasks. I will be discussing each of these tasks separately. Key parts of the remainder of the PtInvoice unit are shown in Listing 3 (at the end of the article).

Print Initialization

The public method PrintInvoice takes care of initializing the print job and terminating it after calling the StartPrinting method, which is where several default field values are initialized that will be used by the other methods. StartPrinting also calls the other methods which handle various sub-tasks. Table 1 shows the purpose of the various fields in TInvoiceReport.

You will notice that I used the Win32 API function GetDeviceCaps to retrieve the pixels per inch along the X and Y axis of the printer device with the following two lines of code:

```
FPixInInchX := GetDeviceCaps(
  Printer.Canvas.Handle,
  LOGPIXELSX);
FPixInInchY := GetDeviceCaps(
  Printer.Canvas.Handle,
  LOGPIXELSY);
```

GetDeviceCaps returns much more information about the printer device that you might find useful (look up the online help for more information).

You'll notice that StartPrinting calls a procedure SetFontSize, which is required because of an anomaly that comes up when setting the font size with different printer types and resolutions. The line of code shown below fixes this problem:

```
Printer.Canvas.Font.PixelsPerInch:=
 GetDeviceCaps(
   Printer.Canvas.Handle,
   LOGPIXELSY);
```

➤ *Above Figure 2: The printed report*

➤ *Below: Listing 2*

```
TInvoiceReport = class
  private
    FPageNum: Integer;    // Keeps track of pages
    FPixInInchX,          // # of pixels per horizontal inch
    FPixInInchY: Integer; // # of pixels per vertical inch
    FAmountPrintedY: Integer; // Keeps track of pixels used along Y axis
    FPageCenterX: Integer;    // Center of the page, used for alignment
    FLeftMargin: Integer;     // Left Border Margin
    FDiscountAlignPos: Integer;  // Used for alignment
    FSellPriceAlignPos: Integer; // Used for alignment
    FExtPriceAlignPos: Integer;  // Used for alignment
    procedure StartPrinting;     // Starts the printing process
    procedure PrintBorders;      // Prints the border
    procedure PrintHeading;      // Prints heading information
    procedure PrintCustomerInfo; // Prints Bill To information
    procedure PrintShipToInfo;   // Prints Ship To information
    procedure PrintOrderInfo;    // Prints Order data
    procedure PrintTitles;       // Prints item titles
    procedure PrintRecords;      // Prints ordered items
    procedure PrintTotals;       // Prints totals
    procedure PrintFooter;       // Prints footer and page number
    function PageDone: Boolean;  // Determines if there is vertical space left
    function GetLineHeight: Integer; // Calculates a line height
    procedure SetFontSize(Size: Integer); // Sets the font size
    procedure WriteDecimalAlign(R: TRect; S: String; RAlignPos: Integer);
    { Writes floating point number based on specific decimal alignment position }
  public
    procedure PrintInvoice; { Public procedure to print the invoice }
  end;
```

In Delphi 1.0, this value was never set correctly. It has been experienced that Delphi 2.0 occasionally does not set this value either. For example, if you switch from a 300 dpi printer to a 720 dot printer, `Printer.Canvas.Font.PixelsPerInch` will remain at 300. The font will either print too large or too small. As stated above, the line shown fixes this problem. Also, although I don't show this here, you can even create your own font and set the font's width and height so that you can account for printers that don't have a 1:1 aspect ratio, eg 300x600 dpi. Delphi will not properly handle fonts on such printers.

The rest of `StartPrinting` calls the various methods to perform the printing sub-tasks which I will now discuss.

Field	Purpose
FPageNum	Keeps track of the current page number.
FPixInInchX	Holds the horizontal pixels per inch for the printer device. This is used as a multiplier for positioning of the printed output since the printing is done relative to inches.
FPixInInchY	Holds the vertical pixels per inch for the printer device. Like the `FPixInInch` field, this is also used as a multiplier for positioning of the printed output relative to inches.
FAmountPrintedY	Keeps track of the current vertical position where subsequent printing will occur.
FPageCenterX	Holds the pixel position of the page center.
FLeftMargin	Holds the pixel position of the left border margin.
FDiscountAlignPos	Used to align a sales item's discount value.
FSellPriceAlignPos	Used to align a sell price value.
FExtPriceAlignPos	Used to align the total price value for a sales item.

➤ *Table 1: TInvoiceReport fields*

Printing The Borders

The `PrintBorders` method is straightforward. It uses the `TCanvas` methods `Rectangle`, `MoveTo` and `LineTo` to draw the border for the invoice that you see in Figure 2. Notice in this method how I use the `FPixInInchX` and `FPixInInchY` fields as multipliers so that I can print using inch measurements. For example, the following line uses these two fields:

```
Rectangle(FLeftMargin,
  1*FPixInInchY,
  FLeftMargin +
  RightBorderMargin*FPixInInchX,
  Round(2.5*FPixInInchY));
```

First, examine how the `FLeftMargin` field was set in the `StartPrinting` method:

```
FLeftMargin := FPageCenterX -
  Round(3.5*FPixInInchX);
```

This sets the left border margin 3.5 inches to the left of the page center. Remember, `FPixInInchX` holds the number of pixels per inch along the X axis of the printer device. Therefore, on a 300dpi printer the statement `3.5*FPixInInchX` would result in the value 1050. Ultimately, this will result in a 7 inch distance between the left and right border margins and also centered on the printed page. So, back to the `tRectangle` method shown above, the left-most position of the rectangle is at the position specified by `FLeftMargin`. The top is set to 1 inch from the top of the page with the statement `1*FPixInInchY`. The right border is set to

```
FLeftMargin +
  RightBorderMargin*FPixInInchX
```

`RightBorderMargin` is a constant which is set to the value 7. By multiplying this by `FPixInInchX`, I get the number of pixels that make up 7 inches on the printer device. Finally, the rectangle is drawn to be 2.5 inches high with the statement `2.5*FPixInInchY`.

Also, notice how I set the `Pen.Width` property:

```
Pen.Width :=
  Round(FPixInInchX / 40);
```

The reason I don't set the `Pen.Width` to something like 1 and 3 is because this property is not device independent. On some high resolution printers (such as imagesetters at 1270dpi or more), a 1 pixel pen would not even show up. Therefore, I set the `Pen.Width` as a fraction of an inch rather than by pixel. I didn't do this in my code, but just as a tip, you might also wish to set the pen's `Width` and `Height` separately based on the x and y pixels per inch of the device if they differ for the printer you are using.

Now, you might recall from my last article that it is possible to use mapping modes to perform this translation of pixels to inches and back. However, for this particular task, I need to be able to draw to fractions of an inch, which seems to be more straightforward by performing the translation myself. I admit, this is probably personal preference.

For very precise printing, you could always set the mapping mode to `MM_TWIPS` in which the logical unit of measurement is converted to one 20th of a point or one 1440th of an inch. The same technique for translating inches to pixels is used throughout the rest of the code.

Printing The Heading

The `PrintHeading` method prints the three string constants:

```
Heading1 =
  'Marine Adventures Inc.';
Heading2 =
  '1234 N. Ocean Blvd.';
Heading3 =
  'Santa Cruz, CA 95066';
```

centered on the upper portion of the page.

One thing I should mention is the use of the line:

```
FAmountPrintedY :=
  FAmountPrintedY +
  GetLineHeight +
  Round(GetLineHeight/20);
```

This adds 1 and one 20th of a line height to FAmountPrintedY based on the currently set font. To do this, I created the function GetLineHeight, which uses the GetTextMetrics Win32 API function which fills a TTextMetric structure with information about a font. All measurements for this font are given in logical units. Since I need to know the height of the text, GetLineHeight returns the tmHeight field of the TTextMetric structure. The TTextMetric structure is documented in the Windows API online help under TEXTMETRIC, or look up the GetTextMetrics function.

Bill To And Ship To Data

The bill to and ship to data is printed in the PrintCustomerInfo and PrintShipToInfo procedures respectively. Each procedure uses methods similar to that previously discussed. Notice that both procedures extract the data relating to customer and shipping from the controls on the EdOrderForm in the MastApp demo. Therefore, it was necessary to add the EdOrders unit to the uses clause for the PtInvoice unit. Additionally, the DataMod unit is also added to the uses clause as I will use the MastData data module from the demo as well. The comments in this procedure explain the code further.

Printing Order Data

Miscellaneous order information is printed in the PrintOrderInfo method. Like the two previously mentioned methods, PrintOrderInfo gets the data from controls on the EdOrderForm with the exception of the order number which is obtained from the TFloatField, OrdersOrderNo, which is in the MastData data module. Also, notice:

```
LeftTextMargin :=
  RightBorderMargin +
  (FPixInInchX*5);
LeftTextMargin2 :=
  LeftTextMargin + TextWidth(
  'Payment Method: ');
```

This allows me to specify a left margin for the order information. You'll notice in Figure 2 that the order information is printed in a columnar fashion. LeftTextMargin is set so that the labels are printed 5 inches from the right border. LeftTextMargin2 allows me to left-align the data just slightly to the right of the label Payment Method:. This is just a simple technique which I use in various places to allow me to align text in the report.

Sales Items And Their Titles

The PrintTitles method prints the titles for the sales item fields. These titles are printed with a bold font so as to distinguish them as titles. This method isn't that different from the others previously mentioned. It is in PrintTitles where I initialize the values for the fields FDiscountAlignPos, FSellPriceAlignPos and FExtPriceAlignPos. These fields are assigned the value of the right most position of the titles to which they apply. I'll use these value for right-aligning decimal values when writing out the sales item records.

The PrintRecords method prints all the sales item fields aligned to their titles. Basically, it just loops through the Items table on the MastData data module and prints the necessary fields. PrintRecords calls the method WriteDecimalAlign which prints floating point values which are decimal aligned. Printing decimal aligned text is simply a matter of printing the integer and fractional portions of the floating point value separately. When printing the integer portion, it is right-aligned by using the SetTextAlign API function. Likewise, when printing the fractional portion, I left align it and print it along with the decimal.

Notice as well that PrintRecords makes use of a function called PageDone, which is responsible for determining whether an additional line will exceed the page height plus some added space which I've specified. If the page length will be exceeded, a new page is started. The borders, header, customer, bill to and other information is printed on the new page as well.

Printing The Totals

PrintTotals prints the total charges for the ordered items. This method first ensures that all 5 lines making up the totals information, plus one vertical inch, will fit on the current page. If not, a new page is started. The test is performed with:

```
if FAmountPrintedY +
  FPixInInchY +
  (GetLineHeight*5) >=
  Printer.PageHeight -
  FPixInInchY then begin
```

PrintTotals also makes use of the WriteDecimalAlign function to decimal align the floating point values. In PrintTotals, you'll notice several lines similar to that shown below:

```
S := TaxRateEdit.Text;
if Pos('%', S) = 0 then
  S := S+'%';
```

The reason for this is because when the TDBEdit controls on EdOrderForm have focus, the literal characters % or $ disappear. This is the designed behavior for the controls. However, since we want these characters to appear in our printed output, the above logic simply ensures their presence.

Printing A Footer

The PrintFooter method is the last sub-task, it simply prints the Footer constant and the page number.

Conclusion

The main point I wanted to make is that writing printing code is simplified when you break down the entire task into several manageable, independent tasks. This also makes printing professional looking documents easier to accomplish. Next month, I'll show you how to get information about the selected printer and how to set various print properties.

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

➤ *Below and following page: Listing 3. Because of space limitations some parts of this unit are omitted; however, the full source code is included on the disk with this issue.*

```
unit PtInvoce;
interface
uses Windows, Messages, Dialogs, SysUtils, Classes;
const
  Heading1 = 'Marine Adventures Inc.';
  Heading2 = '1234 N. Ocean Blvd.';
  Heading3 = 'Santa Cruz, CA 95066';
  Footer = 'All sales are final - no returns or exchanges!!';
  RightBorderMargin = 7;
type
  TInvoiceReport = class
  {... see Listing 2 }
implementation
uses Printers, EdOrders, DataMod;
function TInvoiceReport.GetLineHeight: Integer;
{ Determines line height based on currently rendered font }
var TM: TTextMetric;
begin
  GetTextMetrics(Printer.Canvas.Handle, TM);
  Result := TM.tmHeight;
end;
procedure TInvoiceReport.SetFontSize(Size: Integer);
{ This procedure sets the printer font size. }
begin
  Printer.Canvas.Font.PixelsPerInch:=
    GetDeviceCaps(Printer.Canvas.Handle, LOGPIXELSY);
  Printer.Canvas.Font.Size := Size;
end;
procedure TInvoiceReport.PrintInvoice;
{ Starts the print job, called by the TInvoiceReport user }
begin
  Printer.BeginDoc; // Start the print job.
  try
    StartPrinting ;  // Perform the printing operations.
  finally
    Printer.EndDoc; // End the print job.
  end;
end;
procedure TInvoiceReport.StartPrinting;
{ This function calls all of the invoice print function
  after setting up the initial global values. }
begin
  FAmountPrintedY := 0; // Y Position starts at zero
  FPageNum := 1;        // Page 1
  { Get the number of pixels along the X and Y axis by
    calling the GetDeviceCaps Win32 API function }
  FPixInInchX :=
    GetDeviceCaps(Printer.Canvas.Handle, LOGPIXELSX);
  FPixInInchY :=
    GetDeviceCaps(Printer.Canvas.Handle, LOGPIXELSY);
  { Calculate the center of the page }
  FPageCenterX := Round(Printer.PageWidth / 2);
  { The bounding border is 7 inches wide. Therefore, center
    its Left position based on the center of the page }
  FLeftMargin := FPageCenterX - Round(3.5*FPixInInchX);
  { Set a standard font for the report }
  Printer.Canvas.Font.Name := 'Arial';
  SetFontSize(8);     // Set the font size to 8
  PrintBorders;       // Print report borders.
  PrintHeading;       // Print heading information.
  PrintCustomerInfo;  // Print Bill To information.
  PrintShipToInfo;    // Print Ship To information.
  PrintOrderInfo;     // Print Order data.
  PrintTitles;        // Print items titles.
  PrintRecords;       // Print sale items.
  PrintTotals;        // Print cost totals.
  PrintFooter;        // Print footer information.
end;
procedure TInvoiceReport.PrintBorders;
{ This procedure prints out the borders for the report }
begin
  with Printer.Canvas do begin
    Pen.Width := Round(FPixInInchX / 40); // Wider pen width.
    { Draw a rectangle in which the Bill To, Send To and
      order information will be drawn }
    Rectangle(FLeftMargin, 1*FPixInInchY,
      FLeftMargin+RightBorderMargin*FPixInInchX,
      trunc(2.5*FPixInInchY));
    { Rectangle in which the sales items will be drawn. }
    Rectangle(FLeftMargin, trunc(2.5*FPixInInchY),
      FLeftMargin+RightBorderMargin*FPixInInchX,
      Printer.PageHeight-FPixInInchY);
    Pen.Width := Round(FPixInInchX / 80); // Restore pen width
    { Draw a vertical line to separate the totals from the
      rest of the sales item information }
    MoveTo(FLeftMargin+FPixInInchX*6,trunc(2.5*FPixInInchY));
    LineTo(FLeftMargin+FPixInInchX*6,
      Printer.PageHeight-FPixInInchY);
  end;
end;
procedure TInvoiceReport.PrintHeading;
{ Prints the heading information for the invoice report }
var OldSize: Integer;
begin
  FAmountPrintedY := 0; // Set Y printing position to zero.
```

```
  with Printer.Canvas do begin
    Font.Style := [fsBold];  // Heading is printed in bold
    OldSize := Font.Size;    // Save the original font size
    SetFontSize(12);         // Set a new font size to 12
    { Print heading constants by centering them on page. }
    TextOut(FPageCenterX-(Round(TextWidth(Heading1) / 2)),
      FAmountPrintedY, Heading1);
    { Increment FAmountPrintedY by 1+1/20th of a line height
      in the current font. }
    FAmountPrintedY := FAmountPrintedY +
      GetLineHeight+(round(GetLineHeight /20));
    { Draw remaining header lines. }
    TextOut(FPageCenterX-(Round(TextWidth(Heading2) / 2)),
      FAmountPrintedY, Heading2);
    FAmountPrintedY := FAmountPrintedY +
      GetLineHeight+(Round(GetLineHeight / 20));
    TextOut(FPageCenterX-(Round(TextWidth(Heading3) / 2)),
      FAmountPrintedY, Heading3);
    FAmountPrintedY := FAmountPrintedY +
      GetLineHeight+(Round(GetLineHeight / 20));
    Font.Style := [];      // Restore font style.
    SetFontSize(OldSize);  // Restore font size.
  end;
end;
procedure TInvoiceReport.PrintCustomerInfo;
{ This procedure prints the Bill To customer information }
var
  LeftTextMargin: Integer;  // Keep track of left text margin
  OldSize: Integer;
begin
  with Printer.Canvas, EdOrderForm do begin
    OldSize := Printer.Canvas.Font.Size;  // Save print font size.
    SetFontSize(10);          // Set printer font size to 10
    { Calculate left margin from which to start printing text. }
    LeftTextMargin := FLeftMargin + (Round(FPixInInchX / 4));
    FAmountPrintedY :=
      1*FPixInInchY + (round(FPixInInchY / 20));
    Printer.Canvas.Font.Style := [fsBold];
    TextOut(LeftTextMargin, FAmountPrintedY, 'BILL TO:');
    { Double distance between the title and information }
    FAmountPrintedY := FAmountPrintedY+(2*GetLineHeight) +
      Round(GetLineHeight / 20);

    { ... see disk for omitted lines ... }

  end;
end;
procedure TInvoiceReport.PrintShipToInfo;
{ This procedure prints the Ship To information }
var
  LeftTextMargin: Integer;
  OldSize: Integer;
begin
  with Printer.Canvas, EdOrderForm do begin
    OldSize := Printer.Canvas.Font.Size; // Save Font size
    SetFontSize(10); // Set Font size to 10
    { Set next margin at 3 inches from original border }
    LeftTextMargin := RightBorderMargin+(FPixInInchX*3);
    FAmountPrintedY :=
      1*FPixInInchY + Round(FPixInInchY / 20);
    Printer.Canvas.Font.Style := [fsBold]; // Font to bold
    TextOut(LeftTextMargin, FAmountPrintedY, 'SHIP TO:');
    { Double distance between the title and the information }
    FAmountPrintedY := FAmountPrintedY+(2*GetLineHeight)+
      Round(GetLineHeight / 20);
    Printer.Canvas.Font.Style := []; // Restore font style
    { Print the Ship To information }
    TextOut(LeftTextMargin, FAmountPrintedY,
      ShipToCompanyEdit.Text);
    FAmountPrintedY := FAmountPrintedY+GetLineHeight+
      Round(GetLineHeight / 20);

    { ... see disk for omitted lines ... }

  end;
end;
procedure TInvoiceReport.PrintOrderInfo;
{ Prints other miscellaneous order information. }
var
  OldSize: Integer;
  LeftTextMargin: Integer;
  LeftTextMargin2: Integer;
begin
  with Printer.Canvas, EdOrderForm, MastData do begin
    OldSize := Printer.Canvas.Font.Size; // Save font size.
    SetFontSize(10); // Set font size to 10
    { Set next margin at 5 inches from the original border }
    LeftTextMargin := RightBorderMargin+(FPixInInchX*5);
    { The string “Payment Method:” is the longest label,
      so align the data along with its data }
    LeftTextMargin2 :=
      LeftTextMargin+TextWidth('Payment Method: ');
    FAmountPrintedY :=
      1*FPixInInchY + Round(FPixInInchY / 20);
    { Print the additional order information }
    TextOut(LeftTextMargin, FAmountPrintedY, 'Date: ');
    TextOut(LeftTextMargin2, FAmountPrintedY,
      SaleDateEdit.Text);
    FAmountPrintedY := FAmountPrintedY+GetLineHeight+
       Round(GetLineHeight / 20);
    TextOut(LeftTextMargin, FAmountPrintedY, 'Cust No: ');
    TextOut(LeftTextMargin2, FAmountPrintedY,
      CustNoEdit.Text);

 { ... see disk for omitted lines ... }

  end;
end;
procedure TInvoiceReport.PrintTitles;
{ This procedure prints the sales item titles. }
var OldSize: Integer;
begin
  with Printer.Canvas do begin
    OldSize := Printer.Canvas.Font.Size; // Save font size
    SetFontSize(8); // Set font size to 8
    FAmountPrintedY := trunc(2.5*FPixInInchY)+GetLineHeight+
      Round(GetLineHeight / 20);
    Printer.Canvas.Font.Style := [fsBold]; // font to bold.
    { Print out the titles. }
    TextOut(FLeftMargin+
      Round(FPixInInchX / 10), FAmountPrintedY, 'PART NO');
    TextOut(FLeftMargin+FPixInInchX, FAmountPrintedY,
      'DESCRIPTION');
    TextOut(FLeftMargin+trunc(FPixInInchX*3.5),
      FAmountPrintedY, 'DISCOUNT');
    { FDiscountAlignPos will hold alignment value to be used
      in printing records }
    FDiscountAlignPos := FLeftMargin+trunc(FPixInInchX*3.5)+
       TextWidth('DISCOUNT');
    TextOut(FLeftMargin+trunc(FPixInInchX*4.5),
       FAmountPrintedY, 'QTY');
    TextOut(FLeftMargin+FPixInInchX*5, FAmountPrintedY,
       'SELL PRICE');
    { FSellPriceAlignPos will hold alignment value to be
      used in printing records }
    FSellPriceAlignPos := FLeftMargin + FPixInInchX*5 +
      TextWidth('SELL PRICE');
    TextOut(FLeftMargin+FPixInInchX*6 +
      Round(FPixInInchX / 10), FAmountPrintedY, 'EXT PRICE');
    { FExtPriceAlignPos will hold alignment value to be used
      in printing records }
    FExtPriceAlignPos := FLeftMargin+FPixInInchX*6+
      Round(FPixInInchX / 10)+ TextWidth('EXT PRICE');
    { Restore printer information }
    Printer.Canvas.Font.Style := [];
    SetFontSize(OldSize);
  end;
end;
procedure TInvoiceReport.WriteDecimalAlign(R: TRect; S:
String; RAlignPos: Integer);
{ Used to decimal align columns of floating point values,
alignment position is specified by the RAlignPos parameter }
var
  OldAlign: Integer;
  P: Integer;
begin
  with Printer.Canvas, MastData do begin
    { Save the original alignment flag }
    OldAlign := SetTextAlign(Handle, TA_RIGHT);
    P := Pos('.', S); // Determine position of the decimal
    { Draw the integer portion of the value }
    TextRect(R, R.Right, R.Top, Copy(S, 1, P - 1));
    { Recalculate the alignment position }
    R.Left := R.Right;
    R.Right := RAlignPos;
    { Change alignment to left align }
    SetTextAlign(Handle, TA_LEFT);
    { Draw the fractional portion of the decimal along
      with the decimal }
    TextRect(R, R.Left, R.Top, Copy(S, P, 255));
    { Restore the original alignment flag }
    SetTextAlign(Handle, OldAlign);
  end;
end;
procedure TInvoiceReport.PrintRecords;
{ This procedure prints each sales item record }
var
  R: TRect;
  S: String;
begin
  { Recalculate the Y position from which to begin drawing }
  FAmountPrintedY := FAmountPrintedY+(GetLineHeight*2)+
    Round(GetLineHeight / 20);
  with Printer.Canvas, MastData do begin
    Items.First; // go to first item in the database.
    { Loop through sales items and print each record. }
    while (not Items.EOF) and (not PageDone) do begin
      { Print each field from the sales item. Align the data
        accordingly. }
      TextOut(FLeftMargin+Round(FPixInInchX / 10),
        FAmountPrintedY, FloatToStr(ItemsPartNo.Value));
      TextOut(FLeftMargin+FPixInInchX, FAmountPrintedY,
        ItemsDescription.Value);
      { Determine the TRect values for use with the
        WriteDecimalAlign procedure }
      R := Rect(FLeftMargin+trunc(FPixInInchX*3.5),
        FAmountPrintedY, FDiscountAlignPos -
        TextWidth('.00%'), FAmountPrintedY+GetLineHeight);
      { Format the decimal string to pass to
        WriteDecimalAlign }
      S := FormatFloat('0.00%', ItemsDiscount.Value);
      WriteDecimalAlign(R, S, FDiscountAlignPos);
      { Continue printing sales item data }
      TextOut(FLeftMargin+trunc(FPixInInchX*4.5),
        FAmountPrintedY, IntToStr(ItemsQty.Value));
      R := Rect(FLeftMargin+FPixInInchX*5, FAmountPrintedY,
        FSellPriceAlignPos-TextWidth('.00'),
        FAmountPrintedY+GetLineHeight);
      S := FormatFloat('$0.00', ItemsSellPrice.Value);
      WriteDecimalAlign(R, S, FSellPriceAlignPos);
      R := Rect(FLeftMargin + FPixInInchX*6 +
        Round(FPixInInchX / 10), FAmountPrintedY,
        FExtPriceAlignPos-TextWidth('.00'),
        FAmountPrintedY+GetLineHeight);
      S := FormatFloat('$0.00', ItemsExtPrice.Value);
      WriteDecimalAlign(R, S, FExtPriceAlignPos);
      FAmountPrintedY := FAmountPrintedY+GetLineHeight+
        Round(GetLineHeight / 20);
      Items.Next;
      { If next record will not fit, then start a new page. }
      if (not Items.Eof) and PageDone then begin
        PrintFooter;
        Printer.NewPage;
        PrintBorders;
        PrintHeading;
        PrintCustomerInfo;
        PrintShipToInfo;
        PrintOrderInfo;
        PrintTitles;
        FAmountPrintedY := FAmountPrintedY+(GetLineHeight*2)+
          Round(GetLineHeight / 20);
        inc(FPageNum);          // Increment the page number.
      end;
    end;
  end;
end;
procedure TInvoiceReport.PrintTotals;
{ This procedure prints the sales totals }
var
  LeftTextMargin: Integer;
  R: TRect;
  S: String;
begin
  { If the sales totals information + 1 inch will not fit on
    the current page go to a new page. }
  if FAmountPrintedY+FPixInInchY+(GetLineHeight*5) >=
    Printer.PageHeight - FPixInInchY then begin
      PrintFooter;
      Printer.NewPage;
      PrintBorders;
      PrintHeading;
      PrintCustomerInfo;
      PrintShipToInfo;
      PrintOrderInfo;
      PrintTitles;
  end;
  with EdOrderForm, Printer.Canvas do begin
    { Print the totals information }
    LeftTextMargin := FLeftMargin+FPixInInchX*5;
    FAmountPrintedY := FAmountPrintedY+FPixInInchY;
    TextOut(LeftTextMargin, FAmountPrintedY, 'Subtotal');
    { Calculate TRect values to pass to WriteDecimalAlign }
    R := Rect(FLeftMargin+FPixInInchX*6+Round(FPixInInchX /
    10), FAmountPrintedY, FExtPriceAlignPos -
    TextWidth('.00'), FAmountPrintedY+GetLineHeight);
    { If the control on the form has focus the “$” and “%”
    characters are removed. Use the Pos function to ensure
    these characters are written }
    S := TotalEdit.Text;
    if Pos('$', S) = 0 then
      S := '$'+S;
    WriteDecimalAlign(R, S, FExtPriceAlignPos);
    FAmountPrintedY := FAmountPrintedY + GetLineHeight +
      Round(GetLineHeight / 20);

    { ... see disk for omitted lines ... }

  end;
end;
procedure TInvoiceReport.PrintFooter;
{ Prints footer information and page number }
begin
  with Printer.Canvas do begin
    TextOut(FLeftMargin, Printer.PageHeight-
      Round(FPixInInchX / 2), Footer);
    TextOut(RightBorderMargin*FPixInInchX -
       TextWidth('  Page '+IntToStr(FPageNum)),
       Printer.PageHeight-Round(FPixInInchX / 2),
       'Page '+IntToStr(FPageNum));
  end;
end;
function TInvoiceReport.PageDone: Boolean;
{ Determines if more can be printed on the current page. }
begin
  Result := FAmountPrintedY+GetLineHeight >=
    Printer.PageHeight - FPixInInchY;
end;
end.
```