Form Printing DOS-Style

by Paul Warren

I think we could be forgiven if we had put our DOS tools and libraries away and forgotten about them. Delphi and Windows make those old character-based apps look pretty outdated. I even archived my Borland Pascal code to tape.

Recently, though, I was asked to add some features to a statistical process control application called SPC I wrote for DOS years ago. Being an obliging fellow, I moved Borland Pascal back to my hard drive and set about my task.

I think I stared at the old IDE for a good minute waiting for the form designer to appear. When I finally realized it wasn't going to, I had a mild anxiety attack. After another 5 minutes of browsing through basically comment-less code I decided porting to Delphi was the only answer.

Copying the user interface was easy, just as you would expect. Porting the statistical engine wasn't too hard either because I had the foresight (or luck perhaps) to write it as an object. Some minor syntax changes and it worked well.

Printed output, unfortunately, was another matter. Back when I wrote SPC I used pre-drawn forms, done with ASCII line drawing characters, as templates. I inserted data into the forms at runtime based on an index of offsets into the form. Finally, I copied the filled-out form into a print stream.

Since the forms are fairly complex (Figure 1 is a *simple* one!) I didn't feel like drawing them with lots of `MoveTo` and `LineTo` commands. After some thought I decided that, except for the print stream, this scheme could be made to work in Delphi.

Form Files

In the context of this article a form file is a text file which may or may not make use of the extended ASCII character set, ie those characters above #128 used for line drawing. A form file will have areas, fields if you like, where you can insert data prior to printing.

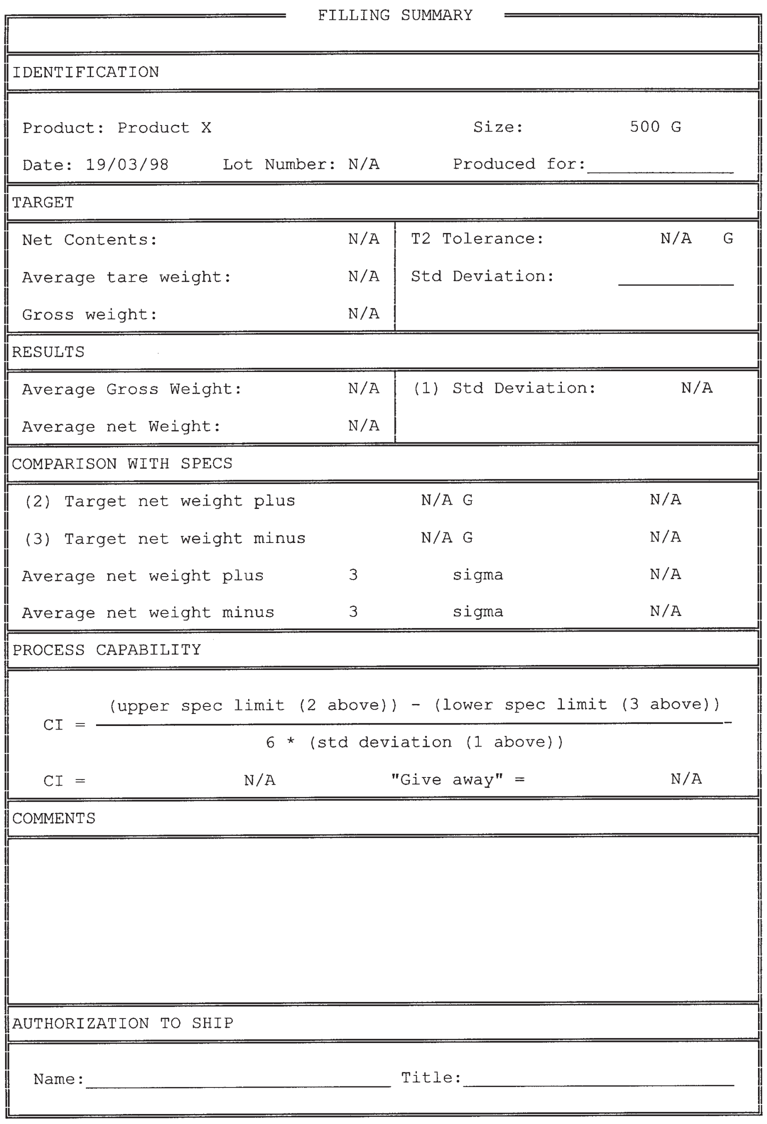
Under DOS, the ASCII character set was almost certainly loaded in your default code page; if you wanted to use other character sets you had to change the code page explicitly.

Under Windows the situation is different and rather obscure (at least it is to me). For example, according to the Windows help, you should be able to construct a font with the line drawing characters by changing the font character set. Unfortunately this doesn't seem to work. Luckily there are fonts which do have the extended ASCII characters, for example MS LineDraw and Terminal, which, with a little care, make it possible to create, view and print form files.

Creating Form Files

There are dedicated editors for creating form files, for example Formtool for DOS (IMSI software), but I suspect these applications are getting rare. If you don't have FormTool your Delphi editor will work. You need to change the font to Terminal or MS LineDraw and enter the extended characters with `Alt-xxxx` (where xxxx is the key code). Alternatively, you can use Windows Character Map.

➤ *Figure 1*



```
                    FILLING SUMMARY

IDENTIFICATION

 Product: Product X                          Size:         500 G

 Date: 19/03/98     Lot Number: N/A       Produced for:______________

TARGET

 Net Contents:                     N/A   T2 Tolerance:          N/A   G

 Average tare weight:              N/A   Std Deviation:      ___________

 Gross weight:                     N/A

RESULTS

 Average Gross Weight:             N/A   (1) Std Deviation:      N/A

 Average net Weight:               N/A

COMPARISON WITH SPECS

 (2) Target net weight plus           N/A G               N/A

 (3) Target net weight minus          N/A G               N/A

 Average net weight plus         3       sigma            N/A

 Average net weight minus        3       sigma            N/A

PROCESS CAPABILITY

         (upper spec limit (2 above)) - (lower spec limit (3 above))
  CI = ----------------------------------------------------------------
                       6 * (std deviation (1 above))

  CI =              N/A           "Give away" =                N/A

COMMENTS




AUTHORIZATION TO SHIP

  Name:______________________________  Title:___________________________
```

Indexing Text Files

Once you have your form files you have to create an index of the offsets where data is to be entered. In its simplest form the index is a file of LongInt with each entry representing a sequential offset into the form file.

For this simplest case to work you need to mark the locations where data is to appear with a character not used elsewhere in the form. Generally an @ symbol will be fine. Next, you need to read the file, or page the file, into a buffer and check each character. When you find an @ character you write the offset to the index file. Listing 1 shows a crude routine that performs this task.

Note that since this indexing need only be done once I'm not too concerned about speed.

Adding Data To The Form

Now that you have both a form file and a list of offsets at which to insert your data it's pretty simple to create a filled-in form. Listing 2 shows the code to enter data into the form.

Keep in mind that in this simple case you are responsible for formatting the data to the correct length for each data field. You must pad data that is too short in order to erase previous data and you must trim data that is too long.

There are more sophisticated twists on this technique and I'm sure you have already thought of some. One thing that comes to mind is to save both the offset where the data is to be entered and the length of the data in a file of record structure. The trick here would be to put multiple @ characters where the data is to go and scan the file for the offset and length before appending the record to the file.

What I really wanted, though, was a visual tool to index form files. I felt this would be both quicker and easier. A TMemo component already has all the functionality needed in the LoadFromFile method and SelStart and SelLength properties. All you need to do is load a form file into the TMemo and highlight the desired area for each data field and write the properties to the index file. I have provided both a visual indexing tool and a demo form printing project on this month's disk.

```
type
  PBuffer = ^TBuffer;
  TBuffer = array[0..65200] of char;
var
  Form: file;
  Index: file of LongInt;
  i: LongInt;
  Buf: PBuffer;
begin
  AssignFile(Form, 'LABELS.FRM');
  Reset(Form, 1);
  try
    AssignFile(Index, 'LABELS.NDX');
    Rewrite(Index);
    GetMem(Buf, FileSize(Form));
    try
      BlockRead(Form, Buf^, FileSize(Form));
      for i := 0 to FileSize(Form) do
        if Buf^[i] = '@' then begin
          Write(Index, i);
          ListBox1.Items.Add(IntToStr(i));
        end;
    finally
      FreeMem(Buf, FileSize(Form));
      CloseFile(Index);
    end;
  finally
    CloseFile(Form);
  end;
end;
```

➤ *Listing 1*

```
var
  i: LongInt;
  j: integer;
  F: TFileStream;
  Index: file of LongInt;
  S: string;
begin
  AssignFile(Index, 'FILLSUM.NDX');
  Reset(Index);
  F := TFileStream.Create('FILLSUM.FRM', fmOpenWrite);
  try
    for j := 0 to FileSize(Index)-1 do begin
      Read(Index, i);
      F.Seek(i, 0);
      case j of
        0: S := Format('%-30s', [Edit1.Text]);
        1: S := Format('%14s', [Edit2.Text]);
        2: S := Format('%-10s', [DateToStr(Date)]);
        3: S := Format('%-8s', ['N/A']);
      else
        S := Format('%10s', ['N/A']);
      end;
      F.Write(S[1], Length(S));
    end;
  finally
    F.Free;
    CloseFile(Index);
  end;
end;
```

➤ *Listing 2*

```
var
  F: TextFile;
  i: integer;
begin
  {...  read form file into Memo1  ...}
  AssignPrn(F);
  Rewrite(F);
  try
    for i := 0 to Memo1.Lines.Count-1 do
      Writeln(F, Memo1.Lines[i]);
  finally
    CloseFile(F);
  end;
end;
```

➤ *Listing 3*

Printing The Form

The last step is printing the form. This is easy. Using the Printer object you simply use writeln() statements to print the form.

Listing 3 is a code snippet showing the general idea. The only thing to keep in mind is whether your chosen font is supported by the printer.

I don't know if I'm alone in this, but I don't completely understand Windows font management. As near as I can figure, it goes like this. If the font you are trying to send to the printer is a raster font, for example Terminal, then if your printer supports Terminal everything is fine. If your printer doesn't support Terminal then Windows substitutes a font of its choice. Unfortunately the character set will not usually be the set we want.

If the font you are using is a True Type font, for example MS LineDraw, then if your printer directly supports MS LineDraw everything is again fine. If your printer doesn't support MS LineDraw then Windows will either download a bitmap font copy of MS LineDraw or a graphic representation of MS LineDraw, depending on a checkbox set in your printer's property sheet. In either case, the output will be correct, it will just print a whole lot slower.

Another point to be aware of is that MS LineDraw looks poor on screen: there are unsightly gaps between rows. The printed output, however, is fine. Why this is I don't know.

Based on these observations I decided to check for the presence of Terminal and MS LineDraw during installation. If they are not present they should be installed (you might want to read the article by Stewart McSporran in Issue 18 for a discussion of font loading techniques). I then display my preview ouput on screen in Terminal.

My program iterates `Printer.Fonts` to see if Terminal is supported. If it is then I output in Terminal, if not I output in MS LineDraw. This way I seem to be able to support most systems. For example, I have successfully used this form printing method on a Canon BJC 4200, an HP Laserjet II, an HP Laserjet 5L and a Panasonic KXP 1123 from Windows 3.1, 3.11 and Win95.

Conclusion

Delphi provides the very flexible `TPrinter` object and there are many fine third-party tools for printing. There are times, though, when an alternative method is desirable. Printing a form is one of these, especially when the form is designed for manual data collection. Why draw an identical form every time your application needs to output to the printer? It makes more sense to create the form once and just copy data into it.

If there is one thing I learned from porting this application to Delphi it is that DOS may be dead as far as Microsoft is concerned but the programming techniques are often just as valid under Windows. My DOS code is now staying on my hard drive until I have distilled every last gem out of it.

Paul Warren runs HomeGrown Software Development in Langley, British Columbia, Canada and can be contacted by email at hg_soft@haven.uniserve.com