Printing in Delphi: Getting Printer Information

by Xavier Pacheco

In this month's article, I will illustrate how you can retrieve information about a printer device. This includes physical characteristics (number of bins, paper sizes supported etc), as well as the printer's text and graphics drawing capabilities. The demo provided on this month's disk contains comments to further explain the code used. As before, the code is written for Delphi 2, but some of the information is also relevant to those developing in Delphi 1.

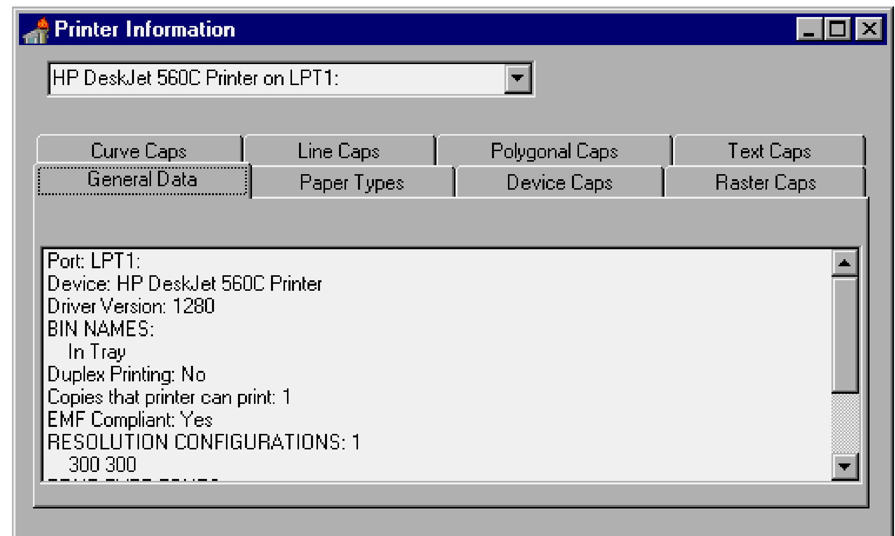
There are different reasons why you would want to get printer information. For example, you might need to know whether or not your printer supports a particular capability. A typical example is to determine if the printer supports banding, which is a process that can improve printing speed and disk space requirements for printers with limited memory. There are certain API calls specific to banding, so you can first determine if the printer will support banding and if so use these API calls, otherwise, you can avoid them.

GetDeviceCaps And DeviceCapabilities

The two functions which I'll primarily be discussing are `GetDeviceCaps` and `DeviceCapabilities`. The example provided with this article, the key parts of which are shown in Listing 1, makes extensive use of both functions.

`GetDeviceCaps` allows you to obtain information about devices such as printers, plotters, screens, etc. Generally, these are devices which have a device context. You use `GetDeviceCaps` by supplying it a handle to a device context and an index which specifies the information that you want to retrieve.

`DeviceCapabilities` is specific to printers. In fact, the information obtained from `DeviceCapabilities` is provided by the printer driver for a specified printer. You use `DeviceCapabilities` by supplying it with strings identifying the printer device and an index specifying the data you want to retrieve.



Sometimes you will need two calls to `DeviceCapabilities` to retrieve certain data. The first call is made to determine how much memory you must allocate for the data to be retrieved. The second call stores the data in the memory block which you've allocated. I will illustrate this in more details when I discuss the Printer Information example program.

One thing you should know is that most of the drawing capabilities that aren't supported by a particular printer will still work if you use them. For example, when `GetDeviceCaps` or `DeviceCapabilities` indicates that `BitBlt`, `StretchBlt` or printing True Type fonts are not supported, you can still use them since the GDI will simulate these functions for you (though that doesn't mean you can print graphics on a daisywheel printer!).

Printer Information Example

The main form for the example program is shown in the screen shot above. It comprises a (Delphi 2) `TPageControl` component which contains eight tab sheets. Each tab sheet shows different printer capabilities or other printer specific information for the printer selected by the `TComboBox` at the top of the form.

If you attempt to use the `DeviceCapabilities` function, defined in WINDOWS.PAS, you will not be able to run your program because this function is not defined in GDI32.DLL as WINDOWS.PAS so indicates. Instead, this function in GDI32.DLL is `DeviceCapabilitiesEx`. However, even if you define this function's prototype as:

```
function DeviceCapabilitiesEx(
  pDevice, pPort: Pchar;
  fwCapability: Word; pOutput:
  Pchar; DevMode: PdeviceMode):
  Integer; stdcall;
  external 'Gdi32.dll';
```

it will not work as expected and returns erroneous results. It turns out that two functions, `DeviceCapabilitiesA` (for Ansi) and `DeviceCapabilitiesW` (for wide) are defined in WINSPOOL.DRV which is the Win32 print spooler interface. This function is the correct one to use

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, ComCtrls;
type
  TForm1 = class(TForm)
    PageControl1: TPageControl;
    PaperInfo: TTabSheet;
    ListBox1: TListBox;
    Gen: TTabSheet;
    ListBox2: TListBox;
    Label4: TLabel;
    Label5: TLabel;
    PrinterCombo: TComboBox;
    Characteristics: TTabSheet;
    DevCapsLB: TListBox;
    TabSheet1: TTabSheet;
    RasterCapLB: TListBox;
    TabSheet2: TTabSheet;
    CurveCapLB: TListBox;
    TabSheet3: TTabSheet;
    TabSheet4: TTabSheet;
    TabSheet5: TTabSheet;
    LineCapLB: TListBox;
    PolyCapLB: TListBox;
    TextCapLB: TListBox;
    GenDataLB: TListBox;
    procedure FormCreate(Sender: TObject);
    procedure PrinterComboChange(Sender: TObject);
   private
    Device, Driver, Port: array[0..255] of char;
    ADevMode: THandle;
   public
    procedure GetBinNames;
    procedure GetDuplexSupport;
    procedure GetCopies;
    procedure GetEMFStatus;
    procedure GetResolutions;
    procedure GetTrueTypeInfo;
    procedure GetCasePaperNames;
    procedure GetDevCapsPaperNames;
    procedure GetDevCaps;
    procedure GetRasterCaps;
    procedure GetCurveCaps;
    procedure GetLineCaps;
    procedure GetPolyCaps;
    procedure GetTextCaps;
  end;
var Form1: TForm1;
{ DeviceCapabilitiesA exists in WINSPOOL.DRV although you'll
  find that most documentation says it exists in GDI32.DLL }
function DeviceCapabilitiesA(pDevice, pPort: Pchar;
  fwCapability: Word; pOutput: Pchar; DevMode: PdeviceMode):
  Integer; stdcall; external 'winspool.drv';
implementation
uses Printers;
type
  { Types for holding bin names }
  TBinName = array[0..23] of char;
  TBinNames = array[0..0] of TBinName; // set $R-
  { Types for holding paper names }
  TPName = array[0..63] of char;
  TPNames = array[0..0] of TPName;     // set $R-
  { Types for holding resolutions }
  TResolution = array[0..1] of integer;
  TResolutions = array[0..0] of TResolution;   // set $R-
  { Type for holding array of pages sizes (word types) }
  TPageSizeArray = Array[0..0] of word;        // set $R-
var Rslt: Integer;
{$R *.DFM}
procedure TForm1.GetBinNames;
var
  BinNames: Pointer;
  i: integer;
begin
{$R-} // Range checking must be turned off here.
  { First determine how many bin names are available. }
  Rslt := DeviceCapabilitiesA(
    Device, Port, DC_BINNAMES, nil, nil);
  if Rslt > 0 then begin
    { Each bin name is 24 bytes long. Therefore,
      allocate Rslt*24 bytes to hold the bin names. }
    GetMem(BinNames, Rslt*24);
    try
      { Now retrieve bin names in allocated block of memory }
      if DeviceCapabilitiesA(Device, Port, DC_BINNAMES,
        BinNames, nil) = -1 then
        raise Exception.Create('DevCap Error');
      { Add the information to the appropriate list box. }
      GenDataLB.Items.Add('BIN NAMES:');
      for i := 0 to Rslt - 1 do
        GenDataLB.Items.Add('     '+
          StrPas(TBinNames(BinNames^)[i]));
    finally
      FreeMem(BinNames, Rslt*24); // free allocated memory
    end;
  end;
  {$R+} // Turn range checking back on.
end;
procedure TForm1.GetDuplexSupport;
begin
  { Uses DeviceCapabilitiesA to determine whether printer
    device supports duplex printing. }
  Rslt := DeviceCapabilitiesA(
    Device, Port, DC_DUPLEX, nil, nil);
  if Rslt = 1 then
    GenDataLB.Items.Add('Duplex Printing: Yes')
  else // Rslt should be zero.
    GenDataLB.Items.Add('Duplex Printing: No')
end;
procedure TForm1.GetCopies;
begin
  { Determines how many copies the device can be set to
    print. If the result is not greater then 1 then the
    print logic must be executed multiple times }
  Rslt :=
    DeviceCapabilitiesA(Device, Port, DC_COPIES, nil, nil);
  if Rslt > 0 then // Some devices report a 0 value.
    GenDataLB.Items.Add(
      'Copies that printer can print: '+IntToStr(Rslt))
end;
procedure TForm1.GetEMFStatus;
begin
  { Determines if device supports enhanced metafiles. }
  Rslt := DeviceCapabilitiesA(Device, Port,
    DC_EMF_COMPLIANT, nil, nil);
  if Rslt = 1 then
    GenDataLB.Items.Add('EMF Compliant: Yes')
  else if Rslt = -1 then
    GenDataLB.Items.Add('EMF Compliant: No')
end;
procedure TForm1.GetResolutions;
var Resolutions: Pointer;
    i: integer;
begin
{$R-}  // Range checking must be turned off.
{ Determine how many resolutions are available. }
  Rslt := DeviceCapabilitiesA(Device, Port,
    DC_ENUMRESOLUTIONS, nil, nil);
  if Rslt > 0 then begin
    { Allocate memory to hold different resolutions which
      are represented by integer pairs, ie: 300, 300 }
    GetMem(Resolutions, (SizeOf(Integer)*2)*Rslt);
    try
      { Retrieve the different resolutions. }
      if DeviceCapabilitiesA(Device, Port,
        DC_ENUMRESOLUTIONS, Resolutions, nil) = -1 then
        Raise Exception.Create('DevCaps Error');
      { Add resolution information to appropriate list box. }
      GenDataLB.Items.Add(
        'RESOLUTION CONFIGURATIONS: '+IntToStr(Rslt));
      for i := 0 to Rslt - 1 do
        GenDataLB.Items.Add('    '+
          IntToStr(TResolutions(Resolutions^)[i][0])+
          ' '+IntToStr(TResolutions(Resolutions^)[i][1]));
    finally
      FreeMem(Resolutions, SizeOf(Integer)*Rslt*2);
    end;
  end;
  {$R+}  // Turn range checking back on.
end;
procedure TForm1.GetTrueTypeInfo;
begin
  { Get the True-Type font capabilities of the device
    represented as bitmasks }
  Rslt := DeviceCapabilitiesA(
    Device, Port, DC_TRUETYPE, nil, nil);
  if Rslt <> 0 then
    { Now mask out individual true type capabilities and
      indicate the result in the appropriate list box. }
    with GenDataLB.Items do begin
      Add('TRUE TYPE FONTS:');
      if (Rslt and DCTT_BITMAP) = DCTT_BITMAP then
        Add('   Prints true type fonts as graphics: Yes')
      else Add('   Prints true type fonts as graphics: No');
      if (Rslt and DCTT_DOWNLOAD) = DCTT_DOWNLOAD then
        Add('   Downloads true type fonts: Yes')
      else Add('   Downloads true type fonts: No');
      if (Rslt and DCTT_DOWNLOAD_OUTLINE) =
        DCTT_DOWNLOAD_OUTLINE then
        Add('   Downloads outline true type fonts: Yes')
      else Add('   Downloads outline true type fonts: No');
      if (Rslt and DCTT_SUBDEV) = DCTT_SUBDEV then
        Add('   Substitutes device for True Type fonts: Yes')
      else Add(
        '   Substitutes device for True Type fonts: No')
    end;
end;
procedure TForm1.GetCasePaperNames;
{ Shows paper types available by  getting an array of word
  values that map to DMPAPER_XXXX constants defined in
  WINDOWS.PAS. These constants refer to specific paper
  types. See the method GetDevCapsPaperNames which gets the
  paper names from the DeviceCapabilitiesA function. }
var
  P: Pointer;
  pSize: Integer;
  i: integer;
begin
{$R-} // Range checking must be turned off.

{ CONTINUED OVERLEAF... }
```

➤ *Facing page: Listing 1*

as indicated in the MSDN. The correct definition for the function prototype which I use in the example program is:

```
function DeviceCapabilitiesA(
  pDevice, pPort: Pchar;
  fwCapability: Word; pOutpu:
  Pchar; DevMode: PdeviceMode):
  Integer; stdcall;
  external 'winspool.drv';
```

The main form's `OnCreate` event handler simply populates the combo box with the list of available printers. The `OnChange` event handler for the `TComboBox` is the central point of the application where the methods to retrieve the printer information are called.

The first page on the form, `General Data`, contains general information about the printer device. You'll see that the printer's device name, driver and port location are obtained by calling the `TPrinter.GetPrinter` method. This method also retrieves a handle to a `TDevMode` structure for the currently selected printer. This information is then added to the `General Data` page.

In order to retrieve the printer driver version, you use the `DeviceCapabilities` function and pass the `DC_DRIVER` index. The rest of the `PrinterComboChange` event handler calls the various routines to populate the listboxes on the various pages of the main form.

The `GetBinNames` method illustrates how to use the `DeviceCapabilitiesA` function to retrieve the bin names for the selected printer. This method first gets the number of bin names available by calling `DeviceCapabilitiesA`, passing the `DC_BINNAMES` index and by passing `nil` as the `pOutput` and `DevMode` parameters. The result of this function call specifies how much memory must be allocated to hold the bin names.

According the documentation on `DeviceCapabilitiesA`, each bin name is defined as an array of 24 characters. Therefore, I defined this data type as:

```
TBinName = array[0..23] of char;
```

and I define an array of `TBinName` as:

```
TBinNames = array[0..0] of TBinName;
```

This type is used to typecast a pointer as an array of `TBinNames`. In order to access an element at some index into the array range checking must be disabled, since this array is defined to have a range of `0..0` as illustrated in the `GetBinNames` method. The bin names are added to the appropriate list box.

I use this same technique of determining the amount of memory required and allocating this memory dynamically in other places where the number of elements returned are not known at design time. Specifically, in `GetDevCapsPaperNames`, `GetCasePaperNames` and `GetResolutions`.

Note that I illustrate two ways of getting the paper names for a selected printer in the methods `GetCasePaperNames` and `GetDevCapsPaperNames`. `GetCasePaperNames` uses a `case` statement and the `DC_PAPERS` index with `DeviceCapabilitiesA` to retrieve an array of word values that map to the various `DMPAPER_XXXX` constants defined in WINDOWS.PAS. These constants refer to the various paper size names. `GetDevCapsPaperNames` uses the `DC_PAPERNAMES` index with the `DeviceCapabilitiesA` function to retrieve the list of paper names from the printer driver.

The methods `GetDuplexSupport`, `GetCopies` and `GetEMFStatus` all use the `DeviceCapabilitiesA` function to return a value of the requested information. For example:

```
DeviceCapabilitiesA(Device,
  Port, DC_DUPLEX, nil, nil);
```

determines if the printer supports duplex printing by returning a value of 1 if so, 0 if not. Also:

```
DeviceCapabilitiesA(Device,
  Port, DC_COPIES, nil, nil);
```

returns the maximum number of copies the device can print.

The remaining methods use the `GetDeviceCaps` function in order to determine the various capabilities of the selected device. In some cases, `GetDeviceCaps` returns the specific value requested. For example, the statement:

```
GetDeviceCaps(
  Printer.Handle, HORZSIZE);
```

returns the width in millimeters of the printer device. In other cases, `GetDeviceCaps` returns an integer value whose bits are masked to determine a particular capability. For example, the `GetRasterCaps` method first retrieves the integer value with the bit-masked fields:

```
RCaps := GetDeviceCaps(
  Printer.Handle, RASTERCAPS);
```

then, for example, to determine if the printer supports banding it masks out the `RC_BANDING` field by performing an `and` operation whose result should equal the value of `RC_BANDING`:

```
if (RCaps and RC_BANDING) =
  RC_BANDING then
  Add('Banding: Yes')
```

Other fields are masked in the same way. This same technique is used in other areas where bit-masked fields are returned from `GetDeviceCaps` as well as from the `DeviceCapabilitiesA` function, such as in the `GetTrueTypeInfo` method.

You'll find both functions, `DeviceCapabilties` and `GetDeviceCaps`, documented in the online API help. You can also refer to the code with this article for more examples.

Next Time

In the next article we'll look at creating print preview.

Finally, I'd like to thank Joe Hecht of Borland's Delphi Developer Support for his excellent technical reviewing of this article.

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 1 CONTINUED... }

  ListBox1.Items.Clear;
  { First, determine how many paper types are available. }
  Rslt := DeviceCapabilitiesA(
    Device, Port, DC_PAPERS, nil, nil);
  if Rslt > 0 then begin
    GetMem(P, Rslt*SizeOf(Word)); // Allocate array of words
    try
      { Retrieve array of words that refer to paper types.
        Then show that type in the appropriate list box. }
      DeviceCapabilitiesA(Device, Port, DC_PAPERS, P, nil);
      for i := 0 to Rslt - 1 do begin
        pSize := TPageSizeArray(P^)[i];
        with ListBox1.Items do
          case pSize of
            DMPAPER_LETTER: Add('DMPAPER_LETTER');
            //  DMPAPER_FIRST: Add('DMPAPER_LETTER') Same as above
            DMPAPER_LETTERSMALL: Add('DMPAPER_LETTERSMALL');
            DMPAPER_TABLOID: Add('DMPAPER_TABLOID');
            DMPAPER_LEDGER: Add('DMPAPER_LEDGER');
            DMPAPER_LEGAL: Add('DMPAPER_LEGAL');
            DMPAPER_STATEMENT: Add('DMPAPER_STATEMENT');
            DMPAPER_EXECUTIVE: Add('DMPAPER_EXECUTIVE');
            DMPAPER_A3: Add('DMPAPER_A3');
            DMPAPER_A4: Add('DMPAPER_A4');
            DMPAPER_A4SMALL: Add('DMPAPER_A4SMALL');
            DMPAPER_A5: Add('DMPAPER_A5');
            DMPAPER_B4: Add('DMPAPER_B4');
            DMPAPER_B5: Add('DMPAPER_B5');

            ... SEE DISK FILES FOR LINES OMMITTED HERE ...

            DMPAPER_A3_EXTRA_TRANSVERSE:
              Add('DMPAPER_A3_EXTRA_TRANSVERSE');
            //   DMPAPER_LAST: Add('DMPAPER_LAST'); SAME AS ABOVE
            DMPAPER_USER: Add('DMPAPER_USER');
        else
          ListBox1.Items.Add(IntToStr(pSize));
        end;
      end;
    finally
      FreeMem(P, Rslt*SizeOf(Word));
    end;
  end;
  {$R+}  // Turn range checking back on.
end;

procedure TForm1.GetDevCapsPaperNames;
{ gets the paper types available on a selected printer from
  the DeviceCapabilitiesA function. }
var PaperNames: Pointer;
    i: integer;
begin
{$R-}  // Range checking off.
  ListBox2.Items.Clear;
  { First get the number of paper names available. }
  Rslt := DeviceCapabilitiesA(
    Device, Port, DC_PAPERNAMES, nil, nil);
  if Rslt > 0 then begin
    { Now allocate the array of paper names. Each paper name
      is 64 bytes. Therefore, allocate Rslt*64 of memory. }
    GetMem(PaperNames, Rslt*64);
    try
      { Retrieve List of names into allocated memory block }
      if DeviceCapabilitiesA(Device, Port, DC_PAPERNAMES,
      PaperNames, nil) = - 1 then
        raise Exception.Create('DevCap Error');
      { Add the paper names to the appropriate list box. }
      for i := 0 to Rslt - 1 do
       ListBox2.Items.Add(StrPas(TPNames(PaperNames^)[i]));
    finally
      FreeMem(PaperNames, Rslt*64);
    end;
  end;
  {$R+} // Range checking back on
end;

procedure TForm1.GetDevCaps;
{ Retrieves various capabilities of printer by using
  GetDeviceCaps. Refer to Online API help for details }
begin
  with DevCapsLB.Items do begin
    Clear;
    Add('Width in millimeters: '+
      IntToStr(GetDeviceCaps(Printer.Handle, HORZSIZE)));
    Add('Height in millimeter:'+
      IntToStr(GetDeviceCaps(Printer.Handle, VERTSIZE)));
    Add('Width in pixels: '+
      IntToStr(GetDeviceCaps(Printer.Handle, HORZRES)));
    Add('Height in pixels: '+
      IntToStr(GetDeviceCaps(Printer.Handle, VERTRES)));
    Add('Pixels per horizontal inch: '+
      IntToStr(GetDeviceCaps(Printer.Handle, LOGPIXELSX)));
    Add('Pixels per vertical inch: '+
      IntToStr(GetDeviceCaps(Printer.Handle, LOGPIXELSY)));
    Add('Color bits per pixel: '+
      IntToStr(GetDeviceCaps(Printer.Handle, BITSPIXEL)));
    Add('Number of color planes: '+
      IntToStr(GetDeviceCaps(Printer.Handle, PLANES)));
    Add('Number of brushes: '+
      IntToStr(GetDeviceCaps(Printer.Handle, NUMBRUSHES)));
    Add('Number of pens: '+
      IntToStr(GetDeviceCaps(Printer.Handle, NUMPENS)));
    Add('Number of fonts: '+
      IntToStr(GetDeviceCaps(Printer.Handle, NUMFONTS)));
    Rslt := GetDeviceCaps(Printer.Handle, NUMCOLORS);
    if Rslt = -1 then
      Add('Number of entries in color table: > 8')
    else Add('Number of entries in color table: '+
      IntToStr(Rslt));
    Add('Relative pixel drawing width: '+
      IntToStr(GetDeviceCaps(Printer.Handle, ASPECTX)));
    Add('Relative pixel drawing height: '+
      IntToStr(GetDeviceCaps(Printer.Handle, ASPECTY)));
    Add('Diagnal pixel drawing width: '+
      IntToStr(GetDeviceCaps(Printer.Handle, ASPECTXY)));
    if GetDeviceCaps(Printer.Handle, CLIPCAPS) = 1 then
      Add('Clip to rectangle: Yes')
    else
      Add('Clip to rectangle: No');
  end;
end;

procedure TForm1.GetRasterCaps;
{ Gets the various raster capabilities of printer by using
  GetDeviceCaps with the RASTERCAPS index. Refer to the
  online help for information on each capability. }
var RCaps: Integer;
begin
  RasterCapLB.Clear;
  with RasterCapLB   .Items do begin
    RCaps := GetDeviceCaps(Printer.Handle, RASTERCAPS);
    if (RCaps and RC_BANDING) = RC_BANDING then
      Add('Banding: Yes')
    else Add('Banding: No');
    if (RCaps and RC_BITBLT) = RC_BITBLT then
      Add('Bitblt capable: Yes')
    else Add('BitBlt capable: No');
    if (RCaps and RC_BITMAP64) = RC_BITMAP64 then
      Add('Supports bitmaps > 64K: Yes')
    else Add('Supports bitmaps > 64K: No');
    if (RCaps and RC_DI_BITMAP) = RC_DI_BITMAP then
      Add('DIB support: Yes')
    else Add('DIB support: No');
    if (RCaps and RC_FLOODFILL) = RC_FLOODFILL then
      Add('Floodfill support: Yes')
    else Add('Floodfill support: No');
    if (RCaps and RC_GDI20_OUTPUT) = RC_GDI20_OUTPUT then
      Add('Windows 2.0 support: Yes')
    else Add('Windows 2.0 support: No');
    if (RCaps and RC_PALETTE) = RC_PALETTE then
      Add('Palette based device: Yes')
    else Add('Palette based device: No');
    if (RCaps and RC_SCALING) = RC_SCALING then
      Add('Scaling support: Yes')
    else Add('Scaling support: No');
    if (RCaps and RC_STRETCHBLT) = RC_STRETCHBLT then
      Add('StretchBlt support: Yes')
    else Add('StretchBlt support: No');
    if (RCaps and RC_STRETCHDIB) = RC_STRETCHDIB then
      Add('StretchDIBits support: Yes')
    else Add('StretchDIBits support: No');
  end;
end;

procedure TForm1.GetCurveCaps;
{ Gets curve capabilities of printer using GetDeviceCaps
funtion with CURVECAPS index. Details in online help }
var CCaps: Integer;
begin
  CurveCapLB.Clear;
  with CurveCapLB.Items do begin
    CCaps := GetDeviceCaps(Printer.Handle, CURVECAPS);
    if (CCaps and CC_NONE) = CC_NONE then
      Add('Curve support: No')
    else Add('Curve support: Yes');
    if (CCaps and CC_CIRCLES) = CC_CIRCLES then
      Add('Circle support: Yes')
    else Add('Circle support: No');
    if (CCaps and CC_PIE) = CC_PIE then
      Add('Pie support: Yes')
    else Add('Pie support: No');
    if (CCaps and CC_CHORD) = CC_CHORD then
      Add('Chord arc support: Yes')
    else Add('Chord arc support: No');
    if (CCaps and CC_ELLIPSES) = CC_ELLIPSES then
      Add('Ellipse support: Yes')
    else Add('Ellipse support: No');
    if (CCaps and CC_WIDE) = CC_WIDE then
      Add('Wide border support: Yes')
    else Add('Wide border support: No');
    if (CCaps and CC_STYLED) = CC_STYLED then
      Add('Styled border support: Yes')
    else Add('Styled border support: No');
    if (CCaps and CC_WIDESTYLED) = CC_WIDESTYLED then
      Add('Wide and styled border support: Yes')
    else Add('Wide and styled border support: No');
    if (CCaps and CC_INTERIORS) = CC_INTERIORS then
      Add('Interior support: Yes')
    else Add('Interior support: No');
    if (CCaps and CC_ROUNDRECT) = CC_ROUNDRECT then
      Add('Round rectangle support: Yes')
    else Add('Round rectangle support: No');
  end;
end;

{ CONTINUED OVERLEAF... }
```

```
{ LISTING 1 CONTINUED... }

procedure TForm1.GetLineCaps;
{ Gets line drawing capabilities of printer using
  GetDeviceCaps funtion with LINECAPS index. Refer to the
  online help for information on each capability. }
var LCaps: Integer;
begin
  LineCapLB.Clear;
  with LineCapLB.Items do begin
    LCaps := GetDeviceCaps(Printer.Handle, LINECAPS);
    if (LCaps and LC_NONE) = LC_NONE then
      Add('Line support: No')
    else Add('Line support: Yes');
    if (LCaps and LC_POLYLINE) = LC_POLYLINE then
       Add('Polyline support: Yes')
    else Add('Polyline support: No');
    if (LCaps and LC_MARKER) = LC_MARKER then
       Add('Marker support: Yes')
    else Add('Marker support: No');
    if (LCaps and LC_POLYMARKER) = LC_POLYMARKER then
      Add('Multiple marker support: Yes')
    else Add('Multiple marker support: No');
    if (LCaps and LC_WIDE) = LC_WIDE then
      Add('Wide line support: Yes')
    else Add('Wide line support: No');
    if (LCaps and LC_STYLED) = LC_STYLED then
      Add('Styled line support: Yes')
    else Add('Styled line support: No');
    if (LCaps and LC_WIDESTYLED) = LC_WIDESTYLED then
      Add('Wide and styled line support: Yes')
    else Add('Wide and styled line support: No');
    if (LCaps and LC_INTERIORS) = LC_INTERIORS then
      Add('Interior support: Yes')
    else Add('Interior support: No');
  end;
end;

procedure TForm1.GetPolyCaps;
{ gets polygonal capabilities of printer using GetDeviceCaps
with POLYGONALCAPS index. Refer to online help for details }
var PCaps: Integer;
begin
  PolyCapLB.Clear;
  with PolyCapLB.Items do begin
    PCaps := GetDeviceCaps(Printer.Handle, POLYGONALCAPS);
    if (PCaps and PC_NONE) = PC_NONE then
      Add('Polygon support: No')
    else Add('Polygon support: Yes');
    if (PCaps and PC_POLYGON) = PC_POLYGON then
      Add('Alternate fill polygon support: Yes')
    else Add('Alternate fill polygon support: No');
    if (PCaps and PC_RECTANGLE) = PC_RECTANGLE then
      Add('Rectangle support: Yes')
    else Add('Rectangle support: No');
    if (PCaps and PC_WINDPOLYGON) = PC_WINDPOLYGON then
      Add('Winding-fill polygon support: Yes')
    else Add('Window-fill polygon support: No');
    if (PCaps and PC_SCANLINE) = PC_SCANLINE then
      Add('Single scanline support: Yes')
    else Add('Single scanline support: No');
    if (PCaps and PC_WIDE) = PC_WIDE then
      Add('Wide border support: Yes')
    else Add('Wide border support: No');
    if (PCaps and PC_STYLED) = PC_STYLED then
      Add('Styled border support: Yes')
    else Add('Styled border support: No');
    if (PCaps and PC_WIDESTYLED) = PC_WIDESTYLED then
      Add('Wide and styled border support: Yes')
    else Add('Wide and styled border support: No');
    if (PCaps and PC_INTERIORS) = PC_INTERIORS then
      Add('Interior support: Yes')
    else Add('Interior support: No');
  end;
end;

procedure TForm1.GetTextCaps;
{ Gets text drawing capabilities of printer using
  GetDeviceCaps withTEXTCAPS index. Details in help }
var  TCaps: Integer;
begin
  TextCapLB.Clear;
  with TextCapLB.Items do begin
    TCaps := GetDeviceCaps(Printer.Handle, TEXTCAPS);
    if (TCaps and TC_OP_CHARACTER) = TC_OP_CHARACTER then
      Add('Character output precision: Yes')
    else
      Add('Character output precision: No');
    if (TCaps and TC_OP_STROKE) = TC_OP_STROKE then
      Add('Stroke output precision: Yes')
    else
      Add('Stroke output precision: No');
    if (TCaps and TC_CP_STROKE) = TC_CP_STROKE then
      Add('Stroke clip precision: Yes')
    else
      Add('Stroke clip precision: No');
    if (TCaps and TC_CR_90) = TC_CR_90 then
      Add('90 degree character rotation: Yes')
    else
      Add('90 degree character rotation: No');
    if (TCaps and TC_CR_ANY) = TC_CR_ANY then
      Add('Any degree character rotation: Yes')
    else
      Add('Any degree character rotation: No');
    if (TCaps and TC_SF_X_YINDEP) = TC_SF_X_YINDEP then
      Add('Independent scale in X and Y direction: Yes')
    else
      Add('Independent scale in X and Y direction: No');
    if (TCaps and TC_SA_DOUBLE) = TC_SA_DOUBLE then
      Add('Doubled character for scaling: Yes')
    else
      Add('Doubled character for scaling: No');
    if (TCaps and TC_SA_INTEGER) = TC_SA_INTEGER then
      Add(
        'Integer multiples only for character scaling: Yes')
    else
      Add(
        'Integer multiples only for character scaling: No');
    if (TCaps and TC_SA_CONTIN) = TC_SA_CONTIN then
      Add('Any multiples for exact character scaling: Yes')
    else
      Add('Any multiples for exact character scaling: No');
    if (TCaps and TC_EA_DOUBLE) = TC_EA_DOUBLE then
      Add('Double weight characters: Yes')
    else
      Add('Double weight characters: No');
    if (TCaps and TC_IA_ABLE) = TC_IA_ABLE then
      Add('Italicized characters: Yes')
    else
      Add('Italicized characters: No');
    if (TCaps and TC_UA_ABLE) = TC_UA_ABLE then
      Add('Underlined characters: Yes')
    else
      Add('Underlined characters: No');
    if (TCaps and TC_SO_ABLE) = TC_SO_ABLE then
      Add('Strikeout characters: Yes')
    else
      Add('Strikeout characters: No');
    if (TCaps and TC_RA_ABLE) = TC_RA_ABLE then
      Add('Raster fonts: Yes')
    else
      Add('Raster fonts: No');
    if (TCaps and TC_VA_ABLE) = TC_VA_ABLE then
      Add('Vector fonts: Yes')
    else
      Add('Vector fonts: No');
    if (TCaps and TC_SCROLLBLT) = TC_SCROLLBLT then
      Add('Scrolling using bit-block transfer: No')
    else
      Add('Scrolling using bit-block transfer: Yes');
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  { Store the printer names in the combo box. }
  PrinterCombo.Items.Assign(Printer.Printers);
  { Display the default printer in the combo box. }
  PrinterCombo.ItemIndex := Printer.PrinterIndex;
  PrinterComboChange(nil);  // Invoke combo's OnChange event
end;

procedure TForm1.PrinterComboChange(Sender: TObject);
begin
  Screen.Cursor := crHourGlass;
  try
    { Populate combo with available printers }
    Printer.PrinterIndex := PrinterCombo.ItemIndex;
    with Printer do
      GetPrinter(Device, Driver, Port, ADevMode);
    { Fill the general page with printer information }
    with GenDataLB.Items do begin
      Clear;
      Add('Port: '+Port);
      Add('Device: '+Device);
      Rslt := DeviceCapabilitiesA(
        Device, Port, DC_DRIVER, nil, nil);
      Add('Driver Version: '+IntToStr(Rslt));
    end;
    { Functions below use GetDeviceCapabilitiesA: }
    GetBinNames;
    GetDuplexSupport;
    GetCopies;
    GetEMFStatus;
    GetResolutions;
    GetTrueTypeInfo;
    { Functions below use GetDeviceCaps: }
    GetCasePaperNames;    // Fill the Paper Types page
    GetDevCapsPaperNames;
    GetDevCaps;    // Fill Device Caps page
    GetRasterCaps; // Fill Raster Caps page
    GetCurveCaps;  // Fill Curve Caps page
    GetLineCaps;   // Fill Line Caps page
    GetPolyCaps;   // Fill Polygonal Caps page
    GetTextCaps;   // Fill Text Caps page
  finally
    Screen.Cursor := crDefault;
  end;
end;

end.
```