# Un po' di tipi per le stampanti

di Sergio Polini

*Stampanti: un bel problema. Si va dalla varietà dei tipi (a 9 o 24 aghi, a margherita o a getto d'inchiostro, laser PCL o PostScript), alla moltitudine delle diverse sequenze di controllo con le quali ogni stampante pretende che le venga indicato come portare a termine una data operazione. Mentre Windows ci aiuta notevolmente ad astrarre dalle specifiche caratteristiche delle varie periferiche, sotto MS-DOS si è condannati a tener conto di ogni particolarità, al punto che si va da programmi in grado di funzionare solo su un tipo di stampante, a programmi accompagnati da decine e decine di driver, la maggior parte dei quali praticamente inutili. Da questo mese, cominceremo a tentare concretamente di porre rimedio, per quanto possibile, ad una tale situazione*

Nel numero di aprile abbiamo passato in rassegna i diversi meccanismi mediante i quali Windows ci permette di prescindere, in buona misura, dalle particolarità di ogni stampante. Il nostro scopo era quello di trarre qualche ispirazione per tratteggiare una soluzione del problema della dipendenza dall'hardware anche sotto MS-DOS, pure se limitata alla stampa di soli testi (e, ovviamente, a programmi realizzati con il Turbo Pascal).

Astraendo dalle specificità di Windows, avevamo individuato i seguenti requisiti: possibilità di installare e iniziare a usare rapidamente qualsiasi stampante, senza bisogno di cimentarsi nella scrittura di un device driver; rapido accesso a informazioni circa le stampanti installate e l'eventuale stampante di default; possibilità di cambiare da qualsiasi applicazione, temporaneamente o permanentemente, le impostazioni variabili di una stampante; possibilità di scegliere facilmente i font disponibili e di tenere sotto controllo la posizione corrente della testina di stampa sul foglio; possibilità di interrompere in ogni momento una stampa e di mantenere pieno controllo in caso di errore critico.

È ora giunto il momento di metterci all'opera.

## Una directory ad hoc

Il nostro primo obiettivo sarà quello di realizzare programmi che possano tutti accedere a comuni definizioni di stampanti (e solo quelle che l'utente intende concretamente utilizzare), mantenute in una apposita directory, nonché di mettere in grado l'utente di aggiungere in ogni momento nuove definizioni, o modificare quelle già presenti, mediante un programma di utilità. Tale directory sarà, per default, C:\TVPRNS; rimarrà comunque possibile scegliere un'altra directory, a condizione di renderla accessibile mediante una variabile dell'environment di nome TVPRNDIR.

Nella interfaccia della unit PRSETUP (figura 1) compaiono quindi, tra l'altro, una funzione *ChangeToTVPRNDIR* e una procedura *RestoreDirectory*, che vanno ovviamente usate «in coppia»: la funzione cambia la directory corrente in quella dedicata alle definizioni delle stampanti, ritornando TRUE se l'operazione riesce; la procedura ripristina la directory che era corrente prima della chiamata della funzione. Da notare che, grazie all'uso della procedura *ChDir* del Turbo Pascal, i cambiamenti di directory hanno effetto anche su diversi drive.

La unit PRSETUP è troppo lunga per poter essere illustrata tutta in una volta; la figura 2 propone quindi un breve stralcio della sua implementazione, comprendente l'indicazione delle altre unit «usate», le dichiarazioni della funzione e della procedura di cui stiamo parlando e delle variabili di cui queste hanno bisogno.

È tutto piuttosto semplice; mi limito quindi a farvi osservare che la ricerca della directory viene effettuata in due tempi: prima si cerca un'eventuale variabile dell'environment di nome TVPRNDIR, quindi, se questa non risulta definita, si assume che la directory sia C:\VPRNS. Vedremo più avanti come il programma di installazione delle stampanti e i programmi applicativi facciano uso del tutto.

## Valori numerici

Abbiamo bisogno di una interfaccia utente mediante la quale accedere alle caratteristiche di una stampante. In concreto, abbiamo bisogno di dialog box con alcuni tipi di «campi» attraverso i quali visualizzare e modificare in modo semplice ma corretto alcuni valori.

La unit PRSETUP definisce quindi in primo luogo alcuni tipi e alcune classi derivate da *TInputLine*. *TInputWord* e *TInputReal*, ad esempio, consentono l'immissione di numeri interi non negativi e numeri reali positivi. Vengono ridefiniti i metodi *DataSize, SetData, GetData* e *Valid*, come illustrato nella figura 3. Anche qui è tutto molto semplice; sottolineo solo alcuni punti.

Ambedue le classi hanno una variabile d'istanza *Value*, di tipo *Word* in un caso e *Real* nell'altro, cui viene assegnato il valore numerico della stringa

immessa dall'utente; nel metodo *TInputWord.Valid*, tuttavia, la conversione da stringa a numero mediante la procedura *Val* non viene effettuata direttamente su *Value*, ma su una variabile locale *V* di tipo *Integer*. Preciso subito che, nonostante sia in generale più corretto usare una variabile di tipo *Longint* (ricordate come mi colse in fallo Salvatore Besso su MC-link? Chi non ricordi, può rileggersi la chiacchierata di marzo), in questo caso non è necessario, in quanto l'utente non potrà immettere più di quattro cifre e questo esclude a priori problemi di overflow. La variabile *V* serve solo per verificare che non venga immesso un numero negativo. La classe *TInputWord*, in altre parole, non ha portata generale, ma è disegnata in modo da rispondere alle specifiche esigenze di alcune unit; analogo il destino della classe *TInputReal*, in cui, ad esempio, il metodo *SetData* è implementato in modo da rappresentare sempre il numero con due cifre decimali.

Qualcuno potrebbe ora osservare chè non è questo il modo migliore per defi-

```
unit PrSetup;
{*$F+,O+,X+,S-*}
interface

uses Objects, Drivers, Views, Dialogs, MoreCtls;

const
  Ports: array[1..8] of String[5] =
    ('LPT1:','LPT2:','LPT3:','COM1:','COM2:','COM3:','COM4:','FILE');

type

  PStr30 = ^TStr30;
  TStr30 = String[30];

  PInputCtrlString = ^TInputCtrlString;
  TInputCtrlString = object(TInputLine)
    CStr: TStr30;
    Mand: Boolean;
    constructor Init(Bounds: TRect; Mandatory:Boolean);
    procedure GetData(var Rec); virtual;
    procedure SetData(var Rec); virtual;
    function Valid(Command: Word): Boolean; virtual;
  end;

  PInputWord = ^TinputWord;
  TInputWord = object(TInputLine)
    Value: Word;
    function DataSize: Word; virtual;
    procedure GetData(var Rec); virtual;
    procedure SetData(var Rec); virtual;
    function Valid(Command: Word): Boolean; virtual;
  end;

  PInputReal = ^TInputReal;
  TInputReal = object(TInputLine)
    Value: Real;
    function DataSize: Word; virtual;
    procedure GetData(var Rec); virtual;
    procedure SetData(var Rec); virtual;
    function Valid(Command: Word): Boolean; virtual;
  end;

  PFont = ^TFont;
  TFont = object(TObject)
    Name: TStr30;
    Ctrl: TStr30;
    CPI : Real;
    constructor Init(AName: TStr30; ACtrl: TStr30; ACPI: Real);
    constructor Load(var S: TStream);
    function Str: String;
    procedure Store(var S: TStream); virtual;
  end;

  PFontBox = ^TFontBox;
  TFontBox = object(TListBox)
    constructor Init(var Bounds: TRect; AScrollBar: PScrollBar);
    function GetText(Item: Integer; MaxLen: Integer): String; virtual;
  end;

  PPrinterSetup = ^TPrinterSetup;
  TPrinterSetup = record
    MUnit: Word;
    PaperHeight, PaperWidth: Real;
    OutPort: String[5];
    FontList: PCollection;
  end;

  PPrinterDef = ^TPrinterDef;
  TPrinterDef = record
    Name: String[30];
    FName: String[8];
    BoldOn, BoldOff, ULOn, ULOff: TStr30;
    Reset: TStr30;
    HeadOn, HeadOff: TStr30;
    Res: Word;
    ResetFont: Word;
    Cmd: String[37];
    Offset: Word;
    Setup: TPrinterSetup;
  end;

  PPrSetupDialog = ^TPrSetupDialog;
  TPrSetupDialog = object(TDialog)
    FName: PInputLine;
    FCtrl: PInputCtrlString;
    FCPI : PInputReal;
    FBox : PFontBox;
    Del: PButton;
    U: PRadioButtons;
    H,W: PInputReal;
    P: PDDLCombo;
    constructor Init(var Bounds: TRect);
    procedure SetData(var Rec); virtual;
    procedure GetData(var Rec); virtual;
    procedure HandleEvent(var Event: TEvent); virtual;
    function Valid(Command: Word): Boolean; virtual;
  end;

function ChangeToTVPRNDIR: Boolean;
function ReadPrinterInfo(F: FNameStr; var PD: TPrinterDef): Boolean;
function WritePrinterInfo(F: FNameStr; var PD: TPrinterDef): Boolean;
procedure RestoreDirectory;

const
RFont: TStreamRec = (
  ObjType: 20000;
  VmtLink: Ofs(TypeOf(TFont)^);
  Load   : @TFont.Load;
  Store  : @TFont.Store);
```

*Figura 1. - L'interfaccia delle unit PRSETUP.*

nire classi per l'input di dati numerici, e avrebbe ragione. Sarebbe preferibile prevedere un valore minimo e uno massimo, da inizializzare mediante appositi parametri del constructor; si dovrebbe poi verificare che il valore immesso non sia né inferiore al minimo né superiore al massimo; si dovrebbe, in breve, seguire l'esempio della classe *TNumInputLine* proposta dalla Borland nella unit FIELDS (che trovate nella directory TVDEMOS). Non ho ritenuto, tuttavia, di usare quella classe, in quanto progettata per l'input di valori di tipo *LongInt* mentre mi servivano *Word* e *Real*; ho cercato, per altro verso, di contenere quanto più possibile il numero e la complessità dei metodi ridefiniti, allo scopo di non rendere ancora più lunga una unit già non breve.

## Sequenze di escape

Ogni stampante ha le sue. Per sottolineare una frase, per scriverne un'altra in grassetto, occorre inviare alla stampante sequenze di caratteri, dette anche sequenze di controllo, introdotte spesso da un escape (il carattere ASCII 27). Per definire il comportamento di una stampante, occorre prevedere numerose di tali sequenze; a ciò si provvede con la classe *TInputCtrlString*.

Osservando la figura 1, si nota che si tratta di una classe derivata anch'essa da *TInputLine*, con due variabili d'istanza *CStr* e *Mand*. Quest'ultima, che verrà

```
implementation

uses Dos, App, MsgBox;

(* dichiarazione delle costanti           *)

var
  Path, OldPath: PathStr;

(* dichiarazione dei metodi delle classi *)
(* e di funzioni e procedure ausiliarie  *)

function ChangeToTVPRNDIR: Boolean;
var
  S: SearchRec;
begin
  GetDir(0, OldPath);
  Path := GetEnv('TVPRNDIR');
  if Path = '' then
    Path := 'C:\TVPRNS';
  {$I-} ChDir(Path); {$I+}
  ChangeToTVPRNDIR := IOResult = 0;
end;

procedure RestoreDirectory;
begin
  {$I-} ChDir(OldPath); {$I+}
end;

end.
```

*Figura 2 - Lo «scheletro» della implementazione della unit PRSETUP, con la dichiarazione, per ora, unicamente della funzione* ChangeToTVPRNDIR *e della procedura* RestoreDirectory.

avvalorata mediante il parametro *Mandatory* del constructor, serve a distinguere tra le «stringhe di controllo» che l'utente deve necessariamente indicare e quelle facoltative (magari perché fuori della portata della sua stampante). La variabile *CStr* appartiene ad un tipo *TStr30*; è cioè una stringa di 30 caratteri, come è anche la stringa immessa dall'utente. Vedremo, infatti, che il constructor chiamerà *TInputLine.Init* con un parametro *AMaxLen* pari proprio a 30. Si usa cioè una doppia stringa per ogni sequenza di controllo.

Il motivo dovrebbe essere evidente: si tratta di trovare un modo convenzionale di permettere all'utente di immettere stringhe di caratteri diversi da lettere, numeri e segni di punteggiatura (in particolare quelli con i codici da zero a 31), ma anche di convertire tali stringhe in sequenze comprensibili per una stampante. Quanto al primo aspetto, appare utile la convenzione adottata nel Lotus 1-2-3: per ogni carattere, un *backslash* («\») seguito dal suo codice ASCII in decimale. La variabile d'istanza Data, ereditata da *TInputLine*, punterà appunto a stringhe così composte. I file di definizione delle stampanti, peraltro, dovranno contenere le corrispondenti sequenze di controllo; le nostre dialog box, quindi, dovranno essere in grado sia di leggere tali sequenze e di convertirle nel formato convenzionale, sia di effettuare l'operazione inversa. La variabile d'istanza *CStr* conterrà la traduzione delle stringhe digitate dall'utente nel formato adatto per le stampanti. Per eliminare possibili ambiguità, chiameremo «stringa» quanto digitato dall'utente nel formato convenzionale e «sequenza» il corrispondente insieme di caratteri da inviare alla stampante.

Nella figura 4 è riprodotta la definizio-

```
function TInputWord.DataSize: Word;
begin
  DataSize := SizeOf(Word);
end;

procedure TInputWord.GetData(var Rec);
begin
  Word(Rec) := Value;
end;

procedure TInputWord.SetData(var Rec);
begin
  Str(Word(Rec), Data^);
  SelectAll(True);
end;

function TInputWord.Valid(Command: Word): Boolean;
var
  Code: Integer;
  V: Integer;
begin
  if (Command <> cmCancel) and (Command <> cmValid) then begin
    Val(Data^, V, Code);
    if (Code <> 0) or (V < 0) then begin
      Select;
      MessageBox(#3'Numero non valido', nil, mfError + mfOkButton);
      Valid := False;
    end
    else begin
      Value := V;
      Valid := True;
    end;
  end
  else
    Valid := True;
end;
```

```
function TInputReal.DataSize: Word;
begin
  DataSize := SizeOf(Real);
end;


procedure TInputReal.GetData(var Rec);
begin
  Real(Rec) := Value;
end;


procedure TInputReal.SetData(var Rec);
begin
  Str(Real(Rec):MaxLen:2, Data^);
  SelectAll(True);
end;

function TInputReal.Valid(Command: Word): Boolean;
var
  Code: Integer;
begin
  if (Command <> cmCancel) and (Command <> cmValid) then begin
    Val(Data^, Value, Code);
    if (Code <> 0) or (Value <= 0.001) then begin
      Select;
      MessageBox(#3'Numero non valido', nil, mfError + mfOkButton);
      Valid := False;
    end
    else
      Valid := True;
  end
  else
    Valid := True;
end;
```

*Figura 3 - I metodi ridefiniti per le classi* TInputWord *e* TInputReal.

```
function StrToCtrl(S: TStr30; var CS: TStr30): Boolean;
const
  Digits: set of Char = ['0'..'9'];
var
  i,j: Integer;
  Stat: Integer;
  V: Integer;
begin
  StrToCtrl := False;
  CS := '';
  if (Length(S) < 2) or (S[1] <> '\') then
    Exit;
  Stat := 1;
  i := 2;
  j := 1;
  V := 0;
  while i <= Length(S) do begin
    case Stat of
      1:    if S[i] in Digits then begin
              V := Ord(S[i])-Ord('0');
              Stat := 2;
            end
            else
              Stat := 5;
      2..3: if S[i] in Digits then begin
              V := V * 10 + Ord(S[i])-Ord('0');
              Inc(Stat);
            end
            else if S[i] = '\' then begin
              Dec(i);
              Stat := 4;
            end
            else
              Stat := 5;
      4:    if (S[i] = '\') and (V <= 255) then begin
              CS[j] := Char(V);
              Inc(j);
              Stat := 1;
            end
            else
              Stat := 5;
      5:    begin
              CS := '';
              Exit;
            end;
    end;
    Inc(i);
  end;
  if (S[i-1] in Digits) and (V <= 255) then begin
    CS[j] := Char(V);
    CS[0] := Char(j);
    StrToCtrl := True;
  end

    else
      CS := '';
  end;


function CtrlToStr(CS: TStr30): TStr30;
label
  Out;
var
  i,j,d,m,n: Integer;
  S30: TStr30;
begin
  if Length(CS) > 15 then
    CS[0] := Char(15);
  j := 0;
  for i := 1 to Length(CS) do begin
    Inc(j);
    S30[j] := '\';
    if j = 30 then goto Out;
    m := Byte(CS[i]);
    d := 100;
    while d > 0 do begin
      n := m div d;
      if (n > 0) or (d = 1) then begin
        Inc(j);
        S30[j] := Chr(Ord('0')+n);
        if j = 30 then goto Out;
        m := m mod d;
      end;
      d := d div 10;
    end;
  end;
Out:
  S30[0] := Chr(j);
  CtrlToStr := S30;
end;
```

*Figura 4 - Le funzioni* CtrlToStr *e* StrToCtrl.

ne di due funzioni che, pur non comparendo nell'interfaccia della unit PRSETUP, vengono utilizzate da diversi metodi.

La funzione *StrToCtrl* tenta di convertire una stringa digitata dall'utente in una sequenza di controllo, ritornando TRUE se il tentativo ha successo. Per prima cosa si verifica che la stringa comprenda almeno un carattere e che il primo carattere sia un *backslash*. Da notare che la funzione ritorna FALSE in caso di stringa nulla; vedremo, infatti, che il metodo *Valid* tratterà a parte il caso di stringa nulla, considerandolo accettabile se *Mand* sarà FALSE ed emettendo un messaggio d'errore con Mand TRUE; lo scopo di una tale sistemazione è quello di distinguere due possibili tipi di errore: una stringa nulla per una sequenza di controllo che sia obbligatorio definire, o una stringa non nulla, ma non ben formata.

Dopo il controllo preliminare, la funzione provvede all'analisi della stringa mediante una macchina a stati finiti. Per chi non abbia familiarità con tali tecniche di programmazione, vorrei ricordare che se ne era parlato diffusamente nel numero di novembre 1988. Dato che è passato tanto tempo, ripeto sinteticamente che si tratta di mettersi in uno «stato» diverso (in un diverso blocco di istruzioni) secondo il carattere che via via si esamina; da ogni stato si può poi passare ad un altro sulla base del carattere successivo, compreso uno stato di errore (il numero 5, nel nostro caso) se ci si imbatte in un carattere non ammesso. Vi sono esattamente tre stati per i caratteri numerici, in quanto vi possono essere al massimo tre cifre consecutive per codici ASCII compresi tra 0 e 255; oltre a questi e allo stato d'errore, vi è poi un ultimo stato dedicato, ovviamente, al *backslash*. Dopo aver esaminato tutta la stringa, si compie un'ultima verifica sul carattere terminale, che deve essere numerico.

La funzione *CtrlToStr* esegue l'operazione inversa: data una sequenza di caratteri, converte il codice ASCII di ognuno in un *backslash* seguito da tante cifre quante ne servono per descrivere quel codice. Le sequenze convertite da *CtrlToStr* sono sempre quelle generate da *StrToCtrl*; soprattutto, si tratta pur sempre di semplici sequenze di caratteri, e, quindi, non possono verificarsi errori durante la conversione. Potrebbe tuttavia risultare alterata una sequenza letta da un file, nel senso che potrebbe risultare troppo lunga per essere convertita in una stringa che deve contenere al massimo 30 caratteri; se ciò accadesse, ne risulterebbero errori difficili da diagnosticare durante lo sviluppo del programma. Una tale prospettiva è tanto sgradevole, che appare preferibile ricorrere al criticato GOTO per troncare drasticamente al trentesimo carattere una stringa che arrivasse eventualmente ad allungarsi tanto.

Nel prossimo numero completeremo l'esame della classe *TInputCtrlString*, per passare subito dopo ai font, ultima componente necessaria per descrivere una stampante.

MC

*Sergio Polini è raggiungibile tramite MC-link alla casella MC1166.*