Movimento per punti

di Sergio Polini

Nella nostra chiacchierata di aprile sul controllo delle stampanti sotto Windows avevamo ricordato che, fin dalla versione 3.0 di questo, era teoricamente possibile distinguere tra una impostazione permanente e una temporanea, la prima analoga ad un intervento mediante il Pannello di Controllo, la seconda valida per un solo documento, o per una sola sessione di lavoro. Nella realtà, tuttavia, anche a causa dell'adozione di driver risalenti alle versioni precedenti, era raro trovare applicazioni che rendessero effettive per l'utente tali potenzialità. Ora finalmente, grazie a Windows 3.1, le cose sono cambiate: i menù propongono, accanto a una Impostazione stampante (Print Setup), *anche una* Impostazione pagina (Page Setup), *utile per variare non solo i margini, ma anche il formato della carta, perfino nell'ambito di una stessa stampa. Prendiamone nota...*

Nelle puntate precedenti abbiamo illustrato gran parte della unit PRSETUP che, con le sue classi e alcune funzioni e procedure ausiliarie, consente di precisare le caratteristiche *temporanee* di una stampante. Ricordo che, nel numero di luglio, abbiamo distinto tra «definizione» delle caratteristiche permanenti di una stampante (quelle che possono essere fissate o variate solo dal programma di installazione) e «impostazione» di quelle temporanee (che possono essere modificate da qualsiasi applicazione).

La distinzione si riflette nella dichiarazione di due diverse strutture di dati: *TPrinterDef* e *TPrinterSetup*. Per prima cosa dovremo terminare l'esposizione della unit PRSETUP, utilizzabile sia nel programma di installazione che nelle normali applicazioni, in quanto dedicata alle caratteristiche temporanee di una stampante; metteremo quindi a punto il nostro programma di installazione, capace di gestire anche le caratteristiche permanenti.

Controllo del dialogo di impostazione

Abbiamo già illustrato l'implementazione di alcuni metodi della classe *TPrSetupDialog*; il constructor, in particolare, provvede all'attivazione dei controlli per i tipi carattere, il formato della carta, la porta di output. Ci manca solo l'esame dei meccanismi che sovraintendono al coordinamento del tutto, mediante i metodi *HandleEvent* e *Valid*.

Il metodo *HandleEvent* (figura 1) ripete in primo luogo il comportamento ereditato da *TDialog*, per poi badare a due casi particolari.

Il constructor associa i pulsanti *Aggiungi* e *Cancella* (vi rimando alla figura 5 della puntata di luglio) ai comandi *cmFontAdd* e *cmFontDelete;* con il primo si aggiunge un tipo carattere alla lista di quelli supportati da una data stampante, con il secondo lo si elimina. Quando l'utente preme il pulsante *Aggiungi,* si controlla che il nome, la stringa di attivazione e la dimensione dei caratteri siano stati correttamente immessi, e che il nome non sia quello di un tipo carattere già definito; superati tali controlli, il nuovo tipo viene aggiunto alla lista. Quando l'utente preme il pulsante Cancella, si elimina dalla lista il tipo carattere selezionato nella *TFontBox*. Infine, si azzerano i campi per l'immissione delle caratteristiche dei tipi carattere e si seleziona il campo dedicato al nome per proporre l'immissione di un nuovo tipo.

È evidente che si può aggiungere un tipo carattere solo dopo averlo definito e, per altro verso, che si possono eliminare tipi carattere solo da una lista che ne comprenda almeno uno. Ad evitare possibili errori, i due pulsanti vengono abilitati e disabilitati secondo necessità.

Si usa allo scopo il comando *cmReceivedFocus*, inviato mediante un evento di tipo *evBroadcast* da ogni *View* che riceva il *focus* (che diventi, cioè, quella pronta ad accettare l'interazione con l'utente). Quando l'utente si muove sui campi *FName, FCtrl* o *FCPI*, viene abilitato il comando *cmFontAdd* e disabilitato *cmFontDelete*; quando l'utente si porta sulla lista dei tipi carattere *(FBox)*, *cmFontAdd* viene disabilitato e *cmFontDelete* viene abilitato solo se la lista non è vuota.

Analoghe verifiche vengono condotte quando l'utente chiude la dialog box premendo il tasto di Invio o il pulsante *Ok*: il metodo *Valid* (figura 2) ritorna FALSE, infatti, se non è stato definito nessun tipo carattere; se un tipo risulta definito — magari solo parzialmente — ma non ancora aggiunto alla lista, l'utente viene avvertito mediante un apposito messaggio.

I file di definizione delle stampanti

Le caratteristiche delle stampanti, sia la definizione che l'impostazione di default, vengono mantenute in file con estensione «.PRN», collocati nella directory C:\TVPRNS (o in altra, il cui nome sia assegnato alla variabile TVPRNDIR dell'environment). Nella figura 3 sono riprodotte le funzioni *WritePrinterInfo* e *ReadPrinterInfo*, mediante le quali si possono scrivere su disco e poi rileggere tutte le informazioni.

Potete notare che le due funzioni esi-

```
procedure TPrSetupDialog.HandleEvent(var Event: TEvent);
var
  F: PFont;
function SameName(P: Pointer): Boolean; far;
  begin
    SameName := PFont(P)^.Name = F^.Name;
  end;
begin
  TDialog.HandleEvent(Event);
  if Event.What = evCommand then begin
    case Event.Command of
      cmFontAdd, cmFontDelete:
        case Event.Command of
          cmFontAdd: begin
            if FName^.Valid(cmFontAdd) and FCtrl^.Valid(cmFontAdd)
            and FCPI^.Valid(cmFontAdd) then begin
              F := (New(PFont,
                    Init(FName^.Data^, FCtrl^.CStr, FCPI^.Value)));
              if FBox^.List^.FirstThat(@SameName) = nil then
                FBox^.List^.Insert(F)
              else
                MessageBox(#3'Gia'' definito un tipo carattere "%s".',
                  @FName^.Data, mfError or mfOkButton);
            end
            else begin
              ClearEvent(Event);
              Exit;
            end;
          end;
          cmFontDelete: begin
            if FBox^.Range > 0 then
              FBox^.List^.AtFree(FBox^.Focused);
          end;
      end;
    end;
    with FBox^ do begin
      SetRange(List^.Count);
      if Range > 0 then
        FocusItem(Range-1);
      DrawView;
    end;
    FName^.SetData(NullString);
    FCtrl^.SetData(NullString);
    FCPI^.SetData(NullReal);
    FName^.Select;
    ClearEvent(Event);
  end
  else if Event.What = evBroadcast then begin
    case Event.Command of
      cmReceivedFocus:
        if (Event.InfoPtr = FName) or (Event.InfoPtr = FCtrl)
        or (Event.InfoPtr = FCPI) then begin
          EnableCommands([cmFontAdd]);
          DisableCommands([cmFontDelete]);
        end
        else if Event.InfoPtr = FBox then begin
          if FBox^.Range > 0 then EnableCommands([cmFontDelete]);
          DisableCommands([cmFontAdd]);
        end
        else begin
          DisableCommands([cmFontAdd]);
          DisableCommands([cmFontDelete]);
          if Event.InfoPtr = Del then U^.Select;
        end;
    end;
  end;
end;
```

▲
Figura 1 - Il metodo HandleEvent *della classe* TPrSetupDialog.

gono un argomento di tipo *TPrinterDef*, una struttura che comprende dati ulteriori rispetto a quelli che l'utente può immettere mediante la unit PRSETUP. Una qualsiasi applicazione, infatti, ha certamente necessità di conoscere anche le caratteristiche permanenti di una stampante, pur non potendo modificarle; è questo il motivo per cui nella unit compare anche la dichiarazione di un *record* e di funzioni estranee alla semplice «impostazione». La prima «definizione» di una stampante, ed eventuali successive modifiche, vengono realizzate mediante una classe *TPrDefDialog*, dichiarata in una distinta unit PRDEF destinata ad essere usata dal programma di installazione.

La figura 4 riproduce la prima metà di tale unit: l'interfaccia e il constructor della classe *TPrDefDialog*; la figura 5 vi propone un'immagine della relativa dialog box, con i diversi campi avvalorati per una stampante IBM Proprinter.

L'interfaccia della unit comprende solo la dichiarazione della classe e di un array di stringhe, sulle quali torneremo tra un attimo. Il constructor crea una dialog box comprendente i controlli corrispondenti ai campi del record *TPrinterDef*. Vi sono due normali *TInputLine* per i nomi della stampante e del file in cui le caratteristiche di questa vengono memorizzate, mentre due *TBlackFrame* racchiudono i comandi per attivare e disattivare grassetto e sottolineato, immessi mediante istanze di *TInputCtrlString*. Segue un analogo spazio per immettere il comando con cui reinizializzare (o resettare, se preferite) la stampante, affiancato da un pulsante *Imposta*, attraverso il quale si accede ad una seconda dialog box per l'impostazione, identica a quella riprodotta nella figura 5 del numero scorso.

Prima dei due pulsanti *Ok* e *Annulla*, viene creata un'ulteriore *TBlackFrame* dedicata al movimento per punti, cioè all'aspetto forse più interessante di tutto l'impianto: per poter tener conto della diversa ampiezza dei caratteri usati in una stampa, occorre che un programma

Figura 2 - Il metodo Valid *della classe* TPrSetupDialog.
▼

```
function TPrSetupDialog.Valid(Command: Word): Boolean;
var
  Ok: Boolean;
  R: TRect;
begin
  if (Command <> cmCancel) and (Command <> cmValid) then begin
    if (FName^.Data^[0] = #0) and (FCtrl^.Data^[0] = #0)
    and (FCPI^.Data^ = ' 0.00') and (FBox^.Range > 0) then begin
      Valid := H^.Valid(Command) and W^.Valid(Command);
    end
    else if FBox^.Range = 0 then begin
      MessageBox(#3'Non definito alcun tipo di carattere.', nil,
        mfError or mfOkButton);
      FName^.Select;
      Valid := False;
    end
    else begin
      R.Assign(0,0,45,9);
      R.Move((Desktop^.Size.X - R.B.X) div 2,
             (Desktop^.Size.Y - R.B.Y) div 2);
      if MessageBoxRect(R, #3'Tipo carattere parzialmente definito'+
      #13#3'e/o non aggiunto all''elenco.'#13#3' Lo abbandoni?',
      nil, mfWarning or mfYesButton or mfNoButton) = cmYes then
        Valid := True
      else begin
        FName^.Select;
        Valid := False;
      end;
    end;
  end
  else
    Valid := True;
end;
```

sappia costantemente dove è posizionata la testina di stampa e che, per correttamente allineare i margini o incolonnare parole o frasi scritte con diversi tipi di carattere, possa muovere la testina con la massima risoluzione possibile. La posizione corrente della testina può essere agevolmente calcolata sulla base dell'ampiezza dei caratteri, che va indicata, come abbiamo visto, per ogni tipo di carattere che si voglia aggiungere alla lista di quelli disponibili; per muovere la testina occorrono appositi comandi, spesso molto diversi da una stampante all'altra. Per definire tali comandi, occorrono una sequenza di «inizio», una codifica dell'ampiezza del movimento espressa in numero di punti e che tenga conto della «risoluzione» della stampante, una eventuale sequenza di «fine comando»; occorre inoltre annotare se, una volta spostata la testina, va ripetuto il comando di selezione del tipo di carattere in uso. A tutto ciò provvedono due *TInputCtrlString* per le sequenze di inizio e fine comando, una *TInputWord* per la risoluzione, due *TRadioButtons* per l'eventuale reimpostazione del tipo carattere, accompagnate da una *TDDLCombo* e un'altra *TInputWord* per un offset; mediante la combo box si può scegliere uno dei comandi corrispondenti alle quattro stringhe dell'array *HMove*.

Movimento per punti

Ogni stampante ha un suo proprio comando per muovere la testina. In alcuni casi, è sufficiente scorrere il manuale utente, cercare qualcosa come «posizionamento del cursore», e il gioco è fatto. Altre volte, tuttavia, occorre arrangiarsi: si tratta, in concreto, di ricorrere al «modo grafico».

Devo precisare che ho tratto ispirazione da LP, un prodotto della Softfocus. Si tratta di una piccola software house canadese, che ha in catalogo librerie di funzioni in C per diversi tipi di applicazioni; LP, in particolare, è una serie di funzioni utilizzabili in programmi che vogliano produrre stampe formattate con qualsiasi tipo di stampante. Vi sono routine per impostare margini, per centrare e giustificare una stringa, ecc. A mio parere, è preferibile trattare separatamente i meccanismi fondamentali del governo di una stampante, lasciando alle singole applicazioni il compito di valersi di questi per dare al proprio output il formato desiderato. Si deve anche dire che il C non è il C++; in altri termini, non è orientato all'oggetto. Ricorrendo alla OOP è facile creare e poi estendere una gerarchia di classi; sarà così possibile derivare altre classi da quelle che

```
function WritePrinterInfo(F: FNameStr; var PD: TPrinterDef): Boolean;
var
  S: TBufStream;
begin
  S.Init(PD.FName+'.PRN', stCreate, 512);
  S.Write(PD, SizeOf(PD) - SizeOf(Pointer));
  S.Put(PD.Setup.FontList);
  S.Done;
  Dispose(PD.Setup.FontList, Done);
  WritePrinterInfo := S.Status = stOk;
end;

function ReadPrinterInfo(F: FNameStr; var PD: TPrinterDef): Boolean;
var
  S: TBufStream;
begin
  S.Init(F, stOpen, 512);
  S.Read(PD, SizeOf(PD) - SizeOf(Pointer));
  PD.Setup.FontList := PCollection(S.Get);
  S.Done;
  ReadPrinterInfo := S.Status = stOk;
end;
```

Figura 3 - Le funzioni ReadPrinterInfo *e* WritePrinterInfo, *che completano l'implementazione della unit PRSETUP.*

```
unit PrDef;
(*$F+,O+,X+,S-*)
interface

uses Objects, Drivers, Dialogs, PrSetup;

const

  HMove: array[1..4] of String[37] = (
    'Epson: Inizio, n%256, n/256, n nulli',
    'ANSI: Inizio, ASCII(n + offset)',
    'Inizio, CHR(n + offset)',
    '(n + offset) volte il comando Inizio');

type

  PPrDefDialog = ^TPrDefDialog;
  TPrDefDialog = object(TDialog)
    Name, FName: PInputLine;
    PS: TPrinterSetup;
    SetupDone: Boolean;
    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;

implementation

uses Views, MsgBox, MoreCtls, App;

const
  cmSetup = 253;

constructor TPrDefDialog.Init(var Bounds: TRect);
var
  R: TRect;
  P: PView;
  i: Integer;
begin
  SetupDone := False;
  Bounds.B.X := Bounds.A.X + 76;
  Bounds.B.Y := Bounds.A.Y + 21;
  TDialog.Init(Bounds, 'Definizione stampante');
  R.Assign(18,2,35,3);
  Name := New(PInputLine, Init(R, 30));
  Insert(Name);
  R.Assign(3,2,8,3);
  Insert(New(PLabel, Init(R, '~N~ome', Name)));
  R.Assign(55,2,72,3);
  FName := New(PInputLine, Init(R, 8));
  Insert(FName);
  R.Assign(40,2,46,3);
  Insert(New(PLabel, Init(R, '~F~ile', FName)));
  R.Assign(18,5,35,6);
  P := New(PInputCtrlString, Init(R, True));
  Insert(P);
  R.Assign(2,4,38,8);
  Insert(New(PBlackFrame, Init(@Self, R, '~G~rassetto', P)));
  R.Assign(18,6,35,7);
  Insert(New(PInputCtrlString, Init(R, True)));
  R.Assign(4,5,10,6);
```

Figura 4 - La prima metà della unit PRDEF: l'interfaccia e il constructor della classe TPrDefDialog.

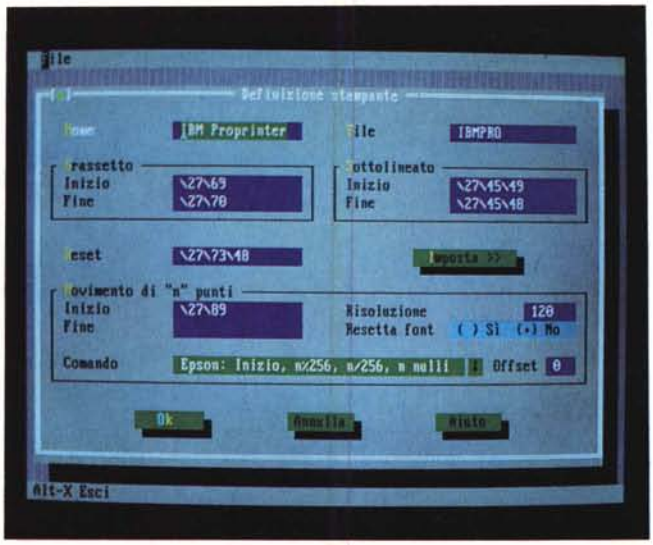

Figura 5 - La dialog box per la definizione di una stampante.

```
    Insert(New(PStaticText, Init(R, 'Inizio')));
    R.Assign(4,6,8,7);
    Insert(New(PStaticText, Init(R, 'Fine')));
    R.Assign(55,5,72,6);
    P := New(PInputCtrlString, Init(R, True));
    Insert(P);
    R.Assign(39,4,75,8);
    Insert(New(PBlackFrame, Init(@Self, R, '~S~ottolineato', P)));
    R.Assign(55,6,72,7);
    Insert(New(PInputCtrlString, Init(R, True)));
    R.Assign(41,5,47,6);
    Insert(New(PStaticText, Init(R, 'Inizio')));
    R.Assign(41,6,45,7);
    Insert(New(PStaticText, Init(R, 'Fine')));
    R.Assign(18,9,35,10);
    P := New(PInputCtrlString, Init(R, True));
    Insert(P);
    R.Assign(3,9,10,10);
    Insert(New(PLabel, Init(R, '~R~eset', P)));
    R.Assign(49,9,65,11);
    Insert(New(PButton, Init(R, '~I~mposta >>', cmSetup, bfNormal)));
    R.Assign(18,12,35,13);
    P := New(PInputCtrlString, Init(R, False));
    Insert(P);
    R.Assign(2,11,75,17);
    Insert(New(PBlackFrame,
           Init(@Self, R, '~M~ovimento di "n" punti', P)));
    R.Assign(4,12,10,13);
    Insert(New(PStaticText, Init(R, 'Inizio')));
    R.Assign(18,13,35,14);
    Insert(New(PInputCtrlString, Init(R, False)));
    R.Assign(4,13,8,14);
    Insert(New(PStaticText, Init(R, 'Fine')));
    R.Assign(65,12,72,13);
    P := New(PInputWord, Init(R, 4));
    Insert(P);
    R.Assign(40,12,52,13);
    Insert(New(PLabel, Init(R, 'Risoluzione', P)));
    R.Assign(55,13,72,14);
    Insert(New(PRadioButtons, Init(R,
      NewSItem('Sì',
      NewSItem('No',
      nil)))));
    R.Assign(41,13,53,14);
    Insert(New(PStaticText, Init(R, 'Resetta font')));
    R.Assign(18,15,61,16);
    P := New(PDDLCombo, Init(@Self, R, 37, 2));
    for i := 4 downto 1 do
      PDDLCombo(P)^.Add(HMove[i]);
    Insert(P);
    R.Assign(4,15,13,16);
    Insert(New(PStaticText, Init(R, 'Comando')));
    R.Assign(68,15,72,16);
    Insert(New(PInputWord, Init(R, 2)));
    R.Assign(61,15,67,16);
    Insert(New(PStaticText, Init(R, 'Offset')));
    R.Assign(22,18,33,20);
    Insert(New(PButton, Init(R, '0~k~', cmOk, bfDefault)));
    R.Assign(42,18,53,20);
    Insert(New(PButton, Init(R, 'Annulla', cmCancel, bfNormal)));
    SelectNext(False);
  end;
```

abbiamo visto e vedremo, a beneficio di applicazioni che abbiano comuni esigenze di formato. Al momento, quindi, è preferibile concentrarsi sul semplice movimento della testina di stampa.

In LP si trova un'utile classificazione dei diversi possibili comandi mediante i quali una stampante può muovere la sua testina. Si distinguono, infatti, quattro «strategie», la cui descrizione viene riportata nelle quattro stringhe dell'array *HMove*.

Nel modo Epson si dispone di una grafica a doppia densità, con la quale la stampante è in grado di stampare 60, 120 o 240 punti per pollice. Se la stampante adotta questo metodo, la risoluzione sarà appunto 60, 120 o 240, e il comando sarà costruito inviando una sequenza di controllo seguita da alcuni byte. Detto *n* il numero dei «punti» che misurano l'ampiezza del movimento desiderato, i primi due byte saranno, rispettivamente, il resto e il quoziente della divisione di *n* per 256; gli altri byte sono destinati a descrivere tratti grafici mediante sequenze di bit (normalmente, un byte per ogni «colonna» di aghi della testina, essendo ogni colonna larga un «punto»); dato che desideriamo solo muovere la testina senza tracciare segni grafici, possiamo inviare tanti zeri quanti sono i «punti». Di qui la sintetica descrizione del modo Epson assegnata a *HMove[1]*:, dove «Inizio» sta per la sequenza iniziale di controllo (i caratteri con codice ASCII 27 e 89 per stampanti come la IBM Proprinter o la Epson FX-85). Non è necessario definire una sequenza terminale di controllo.

Le altre costanti dell'array *HMove* descrivono situazioni più semplici. Una laser HP, ad esempio, segue la strategia ANSI: una sequenza iniziale, i codici ASCII dei caratteri corrispondenti al numero di punti desiderato (ad esempio «RF e «RF per 20), un carattere con codice ASCII 72 come «fine comando», per una risoluzione di 720 punti per pollice. Con una Brother HR, invece, occorre inviare la sequenza iniziale seguita dal carattere il cui codice sia pari al numero dei punti, incrementato di 1; il campo *Offset* serve proprio per indicare il valore che deve a volte essere aggiunto, come nel caso della Brother, al numero dei punti. In altre situazioni, può occorrere ripetere la sequenza iniziale tante volte quanti sono i punti.

Il mese prossimo completeremo l'illustrazione della unit, e vedremo come usarla nel programma di installazione.

MC

Sergio Polini è raggiungibile tramite MC-link alla casella MC1166 e tramite Internet all'indirizzo MC1166@mclink.it.