Definizione e impostazione di una stampante

di Sergio Polini

Il lettore Roberto Pachi, dopo aver letto l'articolo in cui tratteggiavo l'utilizzo delle stampanti sotto Windows, mi chiede di indicargli qualche pubblicazione in cui trovare elencate le «funzioni interne» di Windows. La risposta è semplice, in quanto non c'è fonte più completa dei manuali dell'SDK, che la Microsoft vende anche separatamente. Ottimi, inoltre, i testi di Petzold e di Richter, più volte citati in passato. Il vero problema è un altro: tutti i testi del genere espongono la sintassi delle funzioni dell'API in C, ma Roberto usa il Visual Basic. Riguardo ai «se» e ai «come» di una chiamata di quelle funzioni da Visual Basic, i frequentatori dell'area WIN-BASIC di MC-link si sono trovati un po' in difficoltà (io sono fuori causa, dal momento che non programmo in Basic dal 1984). Se magari qualche altro lettore volesse dare una mano...

```
constructor TInputCtrlString.Init(Bounds: TRect; Mandatory: Boolean);
begin
  TInputLine.Init(Bounds, 30);
  Mand := Mandatory;
end;

procedure TInputCtrlString.GetData(var Rec);
begin
  TStr30(Rec) := CStr;
end;

procedure TInputCtrlString.SetData(var Rec);
var
  S: TStr30;
begin
  S := CtrlToStr(TStr30(Rec));
  TInputLine.SetData(S);
end;

function TInputCtrlString.Valid(Command: Word): Boolean;
begin
  if (Command <> cmCancel) and (Command <> cmValid) then begin
    if Data^[0] = #0 then begin
      if Mand then begin
        Select;
        MessageBox(#3'Stringa comando non definita.', nil,
          mfError or mfOkButton);
        Valid := False;
        Exit;
      end;
    end
    else if not StrToCtrl(PStr30(Data)^, CStr) then begin
      Select;
      MessageBox(#3'Stringa comando non valida.', nil,
        mfError or mfOkButton);
      Valid := False;
      Exit;
    end;
  end;
  Valid := True;
end;
```

Figura 1 - L'implementazione dei metodi della classe TInputCtrlString.

La volta scorsa abbiamo iniziato l'esame di una unit PRSETUP, che potrà essere utilizzata sia per l'installazione di una stampante su un qualsiasi sistema, mediante un programma di utilità, sia per l'impostazione delle caratteristiche variabili della stampante che si è scelto di usare (font, formato della carta, porta), nell'ambito di qualsiasi applicazione.

Abbiamo descritto le funzioni per la definizione e l'uso di una directory destinata a contenere i file descrittivi delle varie stampanti, nonché l'implementazione di alcune semplici classi per l'input di valori numerici. Abbiamo esaminato, infine, due funzioni nascoste nella sezione **implementation** della unit: *StrToCtrl* converte una stringa del tipo "\aaa\aaa\aaa..." (dove «aaa» sta per il codice ASCII di un qualsiasi carattere) nella corrispondente sequenza di caratteri (i comandi o le sequenze di *escape* di una data stampante); *CtrlToStr* esegue la conversione inversa. Riprenderemo ora con l'illustrazione della classe dedicata all'input di tali stringhe.

TInputCtrlString

La classe *TInputCtrlString* (figura 1) deriva da *TInputLine*. La principale differenza risiede nella presenza di una variabile d'istanza *CStr* che è destinata, come dicevamo il mese scorso, a contenere la traduzione della stringa digitata dall'utente in una sequenza di caratteri di controllo. Vi è comunque anche una variabile d'istanza *Mand*, con la quale si può distinguere tra stringhe obbligatorie e stringhe che possono essere omesse. Il constructor, prima di assegnare a *Mand* il valore del parametro *Mandatory*, chiama *TInputLine.Init* con un 30 come secondo parametro, in quanto la lunghezza delle stringhe immesse dall'utente non potrà superare tale limite.

Il metodo *SetData* viene usato per convertire una sequenza di caratteri di controllo in una stringa conforme al formato "\aaa\aaa\aaa...", mediante la funzione *CtrlToStr*; la stringa risultante viene passata quindi al metodo ereditato da *TInputLine*. Il metodo *GetData*

esegue, per così dire, l'operazione inversa; il «dato» restituito, infatti, è la sequenza di caratteri corrispondente alla stringa già impostata con *SetData*, eventualmente modificata. I due metodi vengono usati per leggere le sequenze di controllo presenti in un file e quindi nuovamente salvare nello stesso o in un altro file le sequenze corrispondenti alle stringhe confermate o variate dall'utente.

La variabile d'istanza *CStr* viene avvalorata dal metodo *Valid*. In esso viene trattato separatamente il caso di stringa nulla (se *Mand* è TRUE, viene emesso un appropriato messaggio d'errore); se la stringa non è nulla, se ne tenta la conversione mediante la funzione *StrToCtrl*, cui viene passata come parametro la variabile *CStr*. Se la conversione non ha successo viene emesso un messaggio d'errore, altrimenti *CStr* conterrà la sequenza di controllo pronta per essere utilizzata mediante *GetData*.

TFont e TFontBox

La classe *TFont*, derivata da *TObject*, viene utilizzata per consentire l'indicazione dei diversi tipi di carattere ammessi da una data stampante (figura 2). Vi sono tre variabili d'istanza: *Name* sarà il nome del font, *Ctrl* sarà la sequenza di caratteri di controllo da inviare alla stampante per selezionarlo, *CPI* sarà la dimensione dei caratteri espressa in termini del numero di caratteri in un pollice.

Il constructor *Init* si limita ad assegnare alle tre variabili i valori dei suoi parametri, mentre il constructor *Load* trae i valori da uno *stream* in cui essi siano stati registrati mediante la procedura *Store*. Come vedremo, buona parte dei dati caratteristici di una stampante potrà essere gestita mediante un normale *record*, ma per i tipi dei caratteri è preferibile ricorrere ad una lista (in concreto, una *collection*), in quanto il loro numero massimo non è determinabile a priori. In considerazione della disponibilità di oggetti e metodi già pronti per la gestione di strutture di dati dinamiche, comprese la scrittura/lettura da disco, la maggiore flessibilità che così si ottiene (rispetto, ad esempio, ad un array) non comporta alcun particolare onere di programmazione. In altri termini, non facciamo altro che beneficiare della riusabilità del codice: la OOP mantiene le sue promesse!

Per consentire all'utente di definire gli attributi di ogni tipo di carattere, si deriva dalla classe *TListBox* una classe *TFontBox* (figura 3). *TListBox* richiede che venga avvalorata con il metodo *NewList* una sua variabile d'istanza *List*, in modo che questa punti alla *collection* che si intende consentire all'utente di esaminare; è presente, inoltre, un metodo *GetText* (che *TListBox* eredita da *TListViewer*), che è necessario ridefinire in quanto utilizzato per fornire una stringa che sia rappresentazione dei singoli elementi della *collection*. Ecco il motivo per cui la classe *TFont* dispone anche di un metodo *Str*, che compone una tale stringa mediante l'omonima procedura della unit *System* e la funzione *CtrlToString*.

Quanto a *TFontBox*, il constructor chiama *TListBox.Init* aggiungendo un parametro per indicare che vi sarà una sola colonna, mentre *GetText* restituisce la stringa ottenuta mediante il metodo *Str* del font con numero d'ordine *Item*.

Definizione e impostazione

Fino ad ora abbiamo usato genericamente i termini «definizione» e «impostazione», ma è giunto il momento di una maggiore precisione.

Nella interfaccia della unit PRSETUP, illustrata il mese scorso, compare la dichiarazione di due tipi *record*: *TPrinterSetup* e *TPrinterDef*, che riproduco per vostra comodità nella figura 4. Con *TPrinterDef* intendiamo rappresentare l'insieme delle caratteristiche perma-

```
constructor TFont.Init(AName: TStr30; ACtrl: TStr30; ACPI: Real);
begin
  TObject.Init;
  Name := AName;
  Ctrl := ACtrl;
  CPI  := ACPI;
end;

constructor TFont.Load(var S: TStream);
begin
  S.Read(Name, SizeOf(Name));
  S.Read(Ctrl, SizeOf(Ctrl));
  S.Read(CPI, SizeOf(CPI));
end;

function TFont.Str: String;
const
  Fmt = '%-30s   %-30s   %5s';
var
  S: String;
  St: TStr30;
  CPIStr: String[5];
  Params: array[0..2] of Longint;
begin
  St := CtrlToStr(Ctrl);
  System.Str(CPI:5:2, CPIStr);
  Params[0] := Longint(@Name);
  Params[1] := Longint(@St);
  Params[2] := Longint(@CPIStr);
  FormatStr(S, Fmt, Params);
  Str := S;
end;

procedure TFont.Store(var S: TStream);
begin
  S.Write(Name, SizeOf(Name));
  S.Write(Ctrl, SizeOf(Ctrl));
  S.Write(CPI, SizeOf(CPI));
end;
```

Figura 2 - L'implementazione dei metodi della classe TFont.

```
constructor TFontBox.Init(var Bounds: TRect; AScrollBar: PScrollBar);
begin
  TListBox.Init(Bounds, 1, AScrollBar);
end;

function TFontBox.GetText(Item: Integer; MaxLen: Integer): String;
begin
  GetText := PFont(List^.At(Item))^.Str;
end;
```

Figura 3 - L'implementazione dei metodi della classe TFontBox.

```
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;
```

◀ *Figura 4 - Le strutture di dati per la definizione (caratteristiche permanenti) e l'impostazione (caratteristiche variabili) di una stampante.*

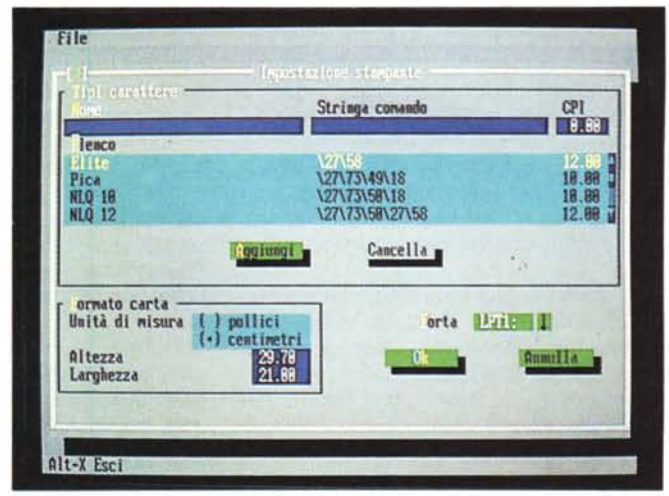

Figura 5 - La dialog box per l'impostazione delle caratteristiche variabili di una stampante (qui una IBM Proprinter). ▶

nenti di una stampante, quelle che, per così dire, la «definiscono»; *TPrinterSetup*, invece, comprende le caratteristiche variabili, quelle mediante le quali una stampante viene «impostata» per un uso invece che per un altro.

Sono caratteristiche variabili il formato della carta, definito mediante altezza e larghezza di una pagina (*PaperHeight* e *PaperWidth*) e affiancato da *MUnit*, cioè dalla unità di misura (pollici o centimetri) mediante la quale il formato stesso viene descritto; è ovviamente variabile la porta alla quale la stampante è connessa (*OutPort*, che può assumere uno dei valori dell'array *Ports*, anch'esso dichiarato nella interfaccia della unit), ma anche la lista dei font, in quanto, ad esempio, alcune stampanti consentono di aggiungere o sostituire una cartuccia di font.

Sono caratteristiche permanenti il nome della stampante (*Name*), nonché il nome del file nel quale le sue caratteristiche sono memorizzate (*FName*), nonostante questo venga scelto dall'utente e possa essere modificato mediante il programma di installazione (è permanente nel senso che non può essere alterato da altri programmi, cioè da quelli che *usano* le stampanti). Sono caratteristiche permanenti anche le sequenze di controllo utilizzate per attivare e disattivare grassetto (*BoldOn* e *BoldOff*) e sottolineato (*ULOn* e *ULOff*), per resettare la stampante (*Reset*), nonché, infine, le sequenze di controllo e le informazioni necessarie per muovere la testina di stampa per «punti» invece che per caratteri (*HeadOn, HeadOff, Res, ResetFont, Cmd, Offset*). Per comodità, infine, *TPrinterDef* comprende anche un campo *Setup* di tipo *TPrinterSetup*; potremmo dire che caratteristica permanente di una stampante è anche la necessaria definizione di una impo-

```
const

  cmFontAdd     = 251;
  cmFontDelete  = 252;

  NullString: String[1] = '';
  NullReal    : Real    = 0.0;

constructor TPrSetupDialog.Init(var Bounds: TRect);
var
  R: TRect;
  V: PView;
  i: Integer;
begin
  TDialog.Init(Bounds, 'Impostazione stampante');

  R.Assign(2,3,34,4);
  FName := New(PInputLine, Init(R, 30));
  Insert(FName);
  R.Assign(1,1,78,13);
  Insert(New(PBlackFrame, Init(@Self, R, '~T~ipi carattere', FName)));
  R.Assign(2,2,7,3);
  Insert(New(PLabel, Init(R, '~N~ome', FName)));
  R.Assign(35,3,67,4);
  FCtrl := New(PInputCtrlString, Init(R, True));
  Insert(FCtrl);
  R.Assign(35,2,51,3);
  Insert(New(PLabel, Init(R, 'Stringa comando', FCtrl)));
  R.Assign(68,3,75,4);
  FCPI := New(PInputReal, Init(R, 5));
  Insert(FCPI);
  R.Assign(68,2,72,3);
  Insert(New(PLabel, Init(R, 'CPI', FCPI)));

  R.Assign(75,5,76,9);
  V := New(PScrollBar, Init(R));
  Insert(V);
  R.Assign(2,5,75,9);
  FBox := New(PFontBox, Init(R, PScrollBar(V)));
  Insert(FBox);
  R.Assign(2,4,9,5);
  Insert(New(PLabel, Init(R, '~E~lenco', FBox)));

  R.Assign(23,10,35,12);
  Insert(New(PButton, Init(R, '~A~ggiungi', cmFontAdd, bfNormal)));
  R.Assign(41,10,53,12);
  Del := (New(PButton, Init(R, '~C~ancella', cmFontDelete, bfNormal)));
  Insert(Del);


  R.Assign(19,14,35,16);
  U := New(PRadioButtons, Init(R,
  NewSItem('pollici',
  NewSItem('centimetri',
  nil))));
Insert(U);
R.Assign(2,14,18,15);
Insert(New(PLabel, Init(R, 'Unità di misura', U)));
```

stazione di default.

Devo a questo punto chiarire che l'impianto che sto descrivendo non è il più completo possibile. Si possono impostare, ad esempio, solo font non proporzionali; quel movimento per punti, inoltre, è previsto solo in senso orizzontale e, quindi, non è possibile variare il valore di interlinea.

Non si tratta, tuttavia, di limiti insormontabili. Tutt'altro: non è affatto difficile immaginare di poter specificare l'ampiezza in punti di ogni singolo carattere in luogo di un'ampiezza costante in termini di CPI, in modo da gestire font proporzionali. Come vedremo, sarebbero sufficienti piccoli ritocchi alle routine di gestione delle stampe. Analogamente, un'interlinea variabile potrebbe essere implementata con pochissimo sforzo.

I veri problemi sono lo spazio e i tempi della rivista: non me la sono sentita di proporvi una soluzione più completa, ma tale da comportare qualche centinaio di righe di codice in più (pensate all'interfaccia utente per l'immissione dell'ampiezza di ogni singolo carattere!). Date le dimensioni già ragguardevoli di MC, non sarebbe praticabile un aumento del numero di pagine dedicato alla rubrica; conseguentemente, ed essendo la rivista mensile, l'illustrazione di una soluzione «completa» rischierebbe di trascinarsi fino alla prossima Pasqua!

Ripeto, comunque, che quando avremo completato sia l'interfaccia utente che le routine di stampa non vi sarà difficile apportare estensioni come quelle qui accennate. Non mancherò di proporvi qualche spunto al momento opportuno.

```
  R.Assign(1,13,38,19);
  Insert(New(PBlackFrame, Init(@Self, R, '~F~ormato carta', U)));
  R.Assign(27,16,35,17);
  H := New(PInputReal, Init(R, 5));
  Insert(H);
  R.Assign(2,16,10,17);
  Insert(New(PLabel, Init(R, 'Altezza', H)));
  R.Assign(27,17,35,18);
  W := New(PInputReal, Init(R, 5));
  Insert(W);
  R.Assign(2,17,12,18);
  Insert(New(PLabel, Init(R, 'Larghezza', W)));

  R.Assign(57,14,69,15);
  P := New(PDDLCombo, Init(@Self, R, 5, 1));
  for i := 8 downto 1 do
    P^.Add(Ports[i]);
  Insert(P);
  R.Assign(49,14,55,15);
  Insert(New(PLabel, Init(R, '~P~orta', P)));

  R.Assign(44,16,56,18);
  Insert(New(PButton, Init(R, 'O~k~', cmOk, bfDefault)));
  R.Assign(62,16,74,18);
  Insert(New(PButton, Init(R, 'Annulla', cmCancel, bfNormal)));

  SelectNext(False);
end;

procedure TPrSetupDialog.SetData(var Rec);
var
  TB: TPrinterSetup absolute Rec;
begin
  FName^.SetData(NullString);
  FCtrl^.SetData(NullString);
  FCPI^.SetData(NullReal);
  if TB.FontList = nil then New(TB.FontList, Init(10,2));
  FBox^.NewList(TB.FontList);
  U^.SetData(TB.MUnit);
  H^.SetData(TB.PaperHeight);
  W^.SetData(TB.PaperWidth);
  P^.SetData(TB.OutPort);
end;


procedure TPrSetupDialog.GetData(var Rec);
var
  TB: TPrinterSetup absolute Rec;
begin
  TB.FontList := FBox^.List;
  U^.GetData(TB.MUnit);
  H^.GetData(TB.PaperHeight);
  W^.GetData(TB.PaperWidth);
  P^.GetData(TB.OutPort);
end;
```

Figura 6 - Il constructor e i metodi SetData e GetData della classe TPrSetupDialog.

TPrSetupDialog

Vediamo quindi come impostare le caratteristiche variabili di una stampante. Si usa naturalmente una dialog box, riprodotta nella figura 5, dietro la quale opera la classe *TPrSetupDialog*.

Il constructor di questa (figura 6) costruisce in primo luogo gli elementi necessari per l'immissione di un tipo carattere (la *TInputLine FName*, la *TInputCtrlString FCtrl* e la *TInputReal FCPI*); una *TBlackFrame* delimita un'area in cui, accanto ad essi, compaiono una *TFontBox (FBox)*, che propone costantemente i font via via immessi, e due *TButton*: mediante il primo si può aggiungere un tipo carattere alla lista, mediante l'altro si può eventualmente togliere dalla lista un font già immesso.

Una seconda *TBlackFrame* delimita i *TRadioButton* per la scelta dell'unità di misura per il formato della carta e le *TInputReal* per le due dimensioni (*H* per l'altezza e *W* per la larghezza).

Una *TDDLCombo* consente infine di precisare la porta cui la stampante è collegata (ricordo che le classi *TBlackFrame* e *TDDLCombo* sono state illustrate nel numero di maggio).

La figura 6 comprende anche la dichiarazione di alcune costanti: *cmFontAdd* e *cmFontDelete* per i comandi associati ai due bottoni per l'aggiunta o la cancellazione di un tipo carattere, *NullString* e *NullReal* per l'inizializzazione di alcuni campi.

Nella stessa figura compare, infine, il codice dei metodi *SetData* e *GetData*. Il primo provvede innanzi tutto ad inizializzare i campi dedicati all'immissione delle caratteristiche di un tipo carattere; subito dopo si esamina il record *TPrinterSetup* passato come parametro: se la lista dei font è assente, se ne crea una vuota che viene associata a *FBox* mediante il metodo *NewList* di questa. Gli altri campi di *TPrinterSetup* vengono usati così come sono per l'inizializzazione dei corrispondenti elementi della dialog box.

Il mese prossimo termineremo l'illustrazione della unit. Vedremo in particolare i metodi *HandleEvent* e *Valid* di *TPrSetupDialog*, nonché le funzioni *ReadPrinterInfo* e *WritePrinterInfo*, mediante le quali registrare su disco, e poi rileggere, tutte le informazioni immesse dall'utente. MC

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