Per qualche controllo in più

di Sergio Polini

Nel numero di febbraio, rispondendo ad un lettore di Bologna, ricordavo la mia unit TSR (illustrata nei numeri 92-97, dal gennaio al giugno '90), aggiungendo che è possibile chiedere in redazione il dischetto con i sorgenti. Evidentemente sono stato troppo vago; è successo, infatti, che alcuni lettori hanno scritto direttamente a me chiedendo quel dischetto. Preciso quindi che il dischetto va ordinato mediante il modulo utilizzabile per tutto il software di pubblico dominio distribuito da MC. Il codice del dischetto è VAR/10

La volta scorsa abbiamo esaminato i diversi aspetti dell'utilizzo di una stampante sotto Windows, per mettere a fuoco i momenti di una sistemazione che ci proponiamo di trasportare, per quanto possibile, in Turbo Vision. L'installazione e la configurazione di una stampante avviene, sotto Windows, mediante diverse dialog box, in cui si fa uso anche di controlli non immediatamente disponibili in Turbo Vision. Prima di proseguire, quindi, dovremo arricchire un po' l'interfaccia utente proposta dalla Borland per il mondo MS-DOS.

Windows dispone di un folto insieme di controlli, solo per una parte dei quali è possibile trovare un perfetto equivalente in Turbo Vision. La figura 1 vi propone una comparazione dei controlli disponibili nei due ambienti e, non a caso, la prima considerazione da fare è che la colonna «Windows» è tutt'altro che completa; ho incluso tutte le classi, ma solo alcuni degli stili possibili. Va anche chiarito il disallineamento tra gli stili BS_CHECKBOX e BS_RADIOBUTTON da una parte, e le classi *TCheckBoxes* e *TRadioButtons* dall'altra; la presenza di una «s» finale in queste ultime non è infatti casuale. In Windows, un BUTTON con stile BS_CHECKBOX definisce una sola check box; in Turbo Vision, la classe *TCheckBoxes* deriva da *TCluster*, in quanto con essa è possibile definire non solo una, ma un gruppo di check box. Per mettere insieme più check box, in Windows si devono utilizzare gli stili WS_GROUP e WS_TABSTOP, ed eventualmente definire un BUTTON con stile BS_GROUPBOX per dare un titolo e un bordo al tutto. Analogamente per i radio button.

Meno dubbi possono sorgere per gli spazi vuoti nella colonna «Turbo Vision»: per alcuni dei controlli di Windows non esiste un corrispondente controllo in Turbo Vision. Uno STATIC con stile BS_BLACKFRAME, ad esempio, appare come una cornice rettangolare con i lati neri; se lo stile è BS_GRAYRECT, invece, otteniamo un'area rettangolare grigia (o del colore

Windows		Turbo Vision	
Classi	**Stili**	**Classi**	**Opzioni**
BUTTON	BS_PUSHBUTTON	TButton	AmDefault=False
	BS_DEFPUSHBUTTON		AmDefault=True
	BS_CHECKBOX		
	BS_RADIOBUTTON		
	BS_GROUPBOX	TCluster	TCheckBoxes
			TRadioButtons
STATIC	SS_SIMPLE	TStaticText	
	SS_BLACKRECT		
	SS_GRAYRECT		
	SS_WHITERECT		
	SS_BLACKFRAME		
	SS_GRAYFRAME		
	SS_WHITEFRAME		
EDIT	ES_LEFT	TInputLine	
	ES_MULTILINE		
SCROLLBAR	SBS_VERT	TScrollBar	Size.X = 1
	SBS_HORZ		Size.Y = 1
LISTBOX	LBS_STANDARD	TListBox	NumCols = 1
	LBS_MULTICOLUMN		NumCols > 1
COMBOBOX	CBS_SIMPLE	(TInputLine + TListBox)	
	CBS_DROPDOWN		
	CBS_DROPDOWNLIST		

Figura 1 - Comparazione tra i controlli di Windows e quelli di Turbo Vision.

scelto come sfondo). Nulla di simile in Turbo Vision, nel quale manca anche qualcosa che assomigli alle combo box di Windows; si può solo emulare una combo box semplice associando una *TInputLine* e una *TListBox*, con meccanismi analoghi a quelli usati nella unit STDDLG. Purtroppo, a noi servono le *drop down list combo box*; ci farebbe comodo, inoltre, incorniciare alcune aree di una dialog box, tanto per rendere più chiara la sua suddivisione in diversi gruppi di scelte tra loro correlate.

TBlackFrame

Cominciamo proprio dalle cornici. La figura 2 contiene il sorgente di una unit MORECTLS, in cui vengono definiti due nuovi controlli; il primo di questi è *TBlackFrame*, un controllo derivato da *TLabel* che costruisce una cornice rettangolare nella zona della dialog box in cui è inserito.

Viene ridefinito il solo constructor, che vuole quattro parametri. Nel primo, di tipo *PDialog*, va passato un puntatore

```
unit MoreCtls;

interface

uses Objects, Views, Dialogs, Drivers;

type

  PBlackFrame = ^TBlackFrame;
  TBlackFrame = object(TLabel)
    constructor Init(AnOwner: PDialog;
      var Bounds: TRect; AText: String; ALink: PView);
  end;

  PDDLCombo = ^TDDLCombo;
  TDDLCombo = object(TInputLine)
    constructor Init(AnOwner: PDialog;
      var Bounds: TRect; AMaxLen: Integer; HistoryId: Byte);
    procedure Add(S: String);
    function GetPalette: PPalette; virtual;
    procedure HandleEvent(var Event: TEvent); virtual;
  private
    Id: Byte;
  end;

implementation

uses HistList;

type
  PDDLHistory = ^TDDLHistory;
  TDDLHistory = object(THistory)
    procedure HandleEvent(var Event: TEvent); virtual;
  end;

procedure TDDLHistory.HandleEvent(var Event: TEvent);
const
  NullString: String[1] = '';
var
  S: String;
begin
  if (Event.What = evMouseDown)
  or ((Event.What = evKeyDown) and (CtrlToArrow(Event.KeyCode) = kbDown)
  and (Link^.State and sfFocused <> 0)) then begin
    Link^.GetData(S);
    Link^.SetData(NullString);
    THistory.HandleEvent(Event);
    if (Link^.Data^[0] = #0) and (S[0] <> #0) then
      Link^.SetData(S);
  end
  else
    THistory.HandleEvent(Event);
end;

constructor TBlackFrame.Init(AnOwner: PDialog;
  var Bounds: TRect; AText: String; ALink: PView);
var
  R    : TRect;
  T    : String;
  Len,i: Integer;
begin
  Len := Length(AText);
  if Pos('~', AText) <> 0 then
    Dec(Len,2);
  R.Assign(Bounds.A.X+1, Bounds.A.Y, Bounds.A.X+Len+2, Bounds.A.Y+1);
  TLabel.Init(R, AText, ALink);
  R.Assign(Bounds.A.X, Bounds.A.Y, Bounds.A.X+1, Bounds.A.Y+1);
  AnOwner^.Insert(New(PStaticText, Init(R, #218)));
  R.Assign(Bounds.A.X+Len+3, Bounds.A.Y, Bounds.B.X-2, Bounds.A.Y+1);
  T[0] := Char(R.B.X - R.A.X + 1);
  FillChar(T[1], Byte(T[0]), #196);
  AnOwner^.Insert(New(PStaticText, Init(R, T)));
  R.Assign(Bounds.B.X-2, Bounds.A.Y, Bounds.B.X-1, Bounds.A.Y+1);
  AnOwner^.Insert(New(PStaticText, Init(R, #191)));
  for i := Bounds.A.Y+1 to Bounds.B.Y-1 do begin
    R.Assign(Bounds.A.X, i, Bounds.A.X+1, i+1);
    AnOwner^.Insert(New(PStaticText, Init(R, #179)));
    R.Assign(Bounds.B.X-2, i, Bounds.B.X-1, i+1);
    AnOwner^.Insert(New(PStaticText, Init(R, #179)));
  end;
  R.Assign(Bounds.A.X, Bounds.B.Y-1, Bounds.A.X+1, Bounds.B.Y);
  AnOwner^.Insert(New(PStaticText, Init(R, #192)));
  R.Assign(Bounds.A.X+1, Bounds.B.Y-1, Bounds.B.X-2, Bounds.B.Y);
  T[0] := Char(R.B.X - R.A.X + 1);
  FillChar(T[1], Byte(T[0]), #196);
  AnOwner^.Insert(New(PstaticText, Init(R, T)));
  R.Assign(Bounds.B.X-2, Bounds.B.Y-1, Bounds.B.X-1, Bounds.B.Y);
  AnOwner^.Insert(New(PStaticText, Init(R, #217)));
end;

constructor TDDLCombo.Init(AnOwner: PDialog;
  var Bounds:TRect; AMaxLen:Integer; HistoryId:Byte);
var
  H: PHistory;
begin
  Dec(Bounds.B.X, 4);
  TInputLine.Init(Bounds, AMaxLen);
  Id := HistoryId;
  Bounds.A.X := Bounds.B.X;
  Bounds.B.X := Bounds.A.X + 3;
  H := New(PDDLHistory, Init(Bounds, @Self, HistoryId));
  AnOwner^.Insert(H);
end;

procedure TDDLCombo.Add(S: String);
begin
  HistoryAdd(Id, S);
end;

function TDDLCombo.GetPalette: PPalette;
const
  DDLPalette: String[4] = #20#20#20#22;
begin
  GetPalette := @DDLPalette;
end;

procedure TDDLCombo.HandleEvent(var Event: TEvent);
var
  CtrlKey: Word;
begin
  if Event.What = evKeyDown then begin
    CtrlKey := CtrlToArrow(Event.KeyCode);
    if (CtrlKey = kbBack) or (CtrlKey = kbDel) or (CtrlKey = kbIns)
    or (Event.CharCode in [' '..#255]) or (Event.CharCode=^Y) then begin
      ClearEvent(Event);
      Exit;
    end;
  end;
  TInputLine.HandleEvent(Event);
end;

end.
```

Figura 2 - La unit MORECTLS, che definisce i controlli TBlackFrame *e* TDDLCombo.

```
program Controls;

uses
  Objects, Views, Drivers, Dialogs, Menus, App, MoreCtls;

const
  cmDemoDialog = 100;

type
  TCombo = object(TApplication)
    procedure InitMenuBar; virtual;
    procedure InitStatusLine; virtual;
    procedure HandleEvent(var Event: TEvent); virtual;
    procedure DemoDialog;
  end;

procedure TCombo.InitMenuBar;
var
  R: TRect;
begin
  GetExtent(R);
  R.B.Y := R.A.Y + 1;
  MenuBar := New(PMenuBar, Init(R, NewMenu(
    NewSubMenu('~F~ile', hcNoContext, NewMenu(
      NewItem('~E~sci', 'Alt-X', kbAltX, cmQuit, hcNoContext,
      nil)),
    NewSubMenu('~D~emo', hcNoContext, NewMenu(
      NewItem('~D~ialog', 'F2', kbF2, cmDemoDialog, hcNoContext,
      nil)),
    nil))))));
end;

procedure TCombo.InitStatusLine;
var
  R: TRect;
begin
  GetExtent(R);
  R.A.Y := R.B.Y - 1;
  StatusLine := New(PStatusLine, Init(R,
    NewStatusDef(0, $FFFF,
      NewStatusKey('~Alt-X~ Esci', kbAltX, cmQuit,
      NewStatusKey('~F2~ Demo', kbF2, cmDemoDialog,
      nil)),
  nil)));
end;

procedure TCombo.HandleEvent(var Event: TEvent);
begin
  TApplication.HandleEvent(Event);
  if (Event.What = evCommand) then begin
    case Event.Command of
      cmDemoDialog: DemoDialog;
    else
      Exit;
    end;
    ClearEvent(Event);
  end;
end;

procedure TCombo.DemoDialog;
const
  PaperSize: array[1..7] of String[30] = (
    'Letter 8 1/2 x 11 in',
    'Legal 8 1/2 x 14 in',
    'A3 297 x 420 mm',
    'A4 210 x 297 mm',
    'A5 148 x 210 mm',
    'B4 250 x 354 mm',
    'B5 182 x 257 mm');
var
  R     : TRect;
  D     : PDialog;
  DDLCB : PDDLCombo;
  i     : Integer;
  Scelta: String[30];
begin
  R.Assign(5,4,64,18);
  D := New(PDialog, Init(R, 'Demo Dialog'));
  (* TDDLCombo *)
  R.Assign(21,4,57,5);  (* R.B.X - R.A.X = 30 + 6 *)
  DDLCB := New(PDDLCombo, Init(D, R, 30, 1));
  R.Assign(3,4,20,5);
  D^.Insert(New(PLabel, Init(R, '~F~ormato carta:', DDLCB)));
  for i := 7 downto 1 do
    DDLCB^.Add(PaperSize[i]);
  (* TBlackFrame *)
  R.Assign(2,2,58,7);
  D^.Insert(New(PBlackFrame,
    Init(D, R, '~D~rop Down List Combo Box', DDLCB)));
  (* TButton *)
  R.Assign(11,9,23,11);
  D^.Insert(New(PButton, Init(R, 'O~k~', cmOk, bfDefault)));
  R.Assign(33,9,45,11);
  D^.Insert(New(PButton, Init(R, 'Annulla', cmCancel, bfNormal)));
  (* TBlackFrame *)
  R.Assign(2,7,58,12);
  D^.Insert(New(PBlackFrame, Init(D, R, 'Pulsanti', nil)));
  D^.Insert(DDLCB);
  if DeskTop^.ExecView(D) = cmOk then begin
    D^.GetData(Scelta);
    MessageBox('Formato scelto: '#13+Scelta, nil,
      mfInformation or mfOkButton);
  end;
  Dispose(D, Done);
end;

var
  Combo: TCombo;

begin
  Combo.Init;
  Combo.Run;
  Combo.Done;
end.
```

Figura 3 - Un programma CONTROLS, per mettere alla prova le classi TBlackFrame *e* TDDLCombo.

alla dialog che ospita il controllo; gli altri tre sono quelli del constructor di *TLabel*. Va tuttavia notato che le coordinate del rettangolo *Bounds* non sono quelle della zona il cui bisogna scrivere la stringa *AText* (che vengono calcolate dal constructor), ma quelle della più ampia zona che si vuole incorniciare.

La presenza di un parametro di tipo *PDialog* si rende necessaria in quanto, per dirla in breve, il constructor non fa altro che creare una *TLabel* e, insieme ad essa, una serie di *TStaticText* per disegnare i bordi. Il programma che usa la unit può «inserire» la *TLabel* nella dialog box, ma linee e trattini aggiuntivi vengono inseriti direttamente dal constructor di questa, che deve quindi «conoscere» la dialog box.

Sottolineo infine che ho derivato *TBlackFrame* da *TLabel* invece che da *TStaticText*, per consentire di raggiungere un gruppo di controlli mediante un solo tasto, quello corrispondente alla lettera evidenziata nella stringa posta sui bordi del rettangolo. Se la qualità di stampa ci è stata questa volta favorevole, potete verificare nelle figure 4 e 5 che la stringa «Drop Down List Combo Box» si comporta esattamente come la label «Formato carta».

TDDLCombo

Un po' più complessa la classe *TDDLCombo*. Con essa ho voluto aggiungere le drop down list combo box al Turbo Vision. Ricapitoliamo per un attimo: una combo box è una combinazione di una list box e di un controllo di edit (una *TInputLine*, in Turbo Vision). La combinazione può assumere tre forme diverse; in una combo box semplice, la list box e la riga di input sono sempre visibili e l'utente può sia scegliere un elemento dalla list box, sia immettere qualcosa nella riga di input; questo «qualcosa» può anche non essere un elemento della list box. Una drop down combo box è molto simile ad una combo box semplice; l'unica differenza risiede nel fatto che la list box è inizialmente invisibile e compare solo se si clicka col mouse sulla freccia rappresentata alla destra della zona di input. Una drop down list combo box è simile ad una drop down combo box, con una importante differenza: è possibile scegliere solo uno degli elementi della list box.

Da un certo punto vista (certamente non quello visivo), una drop down list combo box equivale ad un gruppo di radio button: vengono proposte diverse opzioni, l'utente può scegliere solo una delle opzioni proposte. Evidente la differenza, o meglio la diversa utilità dei due controlli: un gruppo di radio button può contenere solo un numero limitato di opzioni, mentre una list box può contenerne, al limite, quanti se ne vuole, pur occupando lo stesso spazio che occuperebbe una riga di input. Per tornare alle nostre stampanti, sarebbe impensabile un gruppo di radio button per proporre all'utente la scelta tra i 26 formati di carta riconosciuti da Windows.

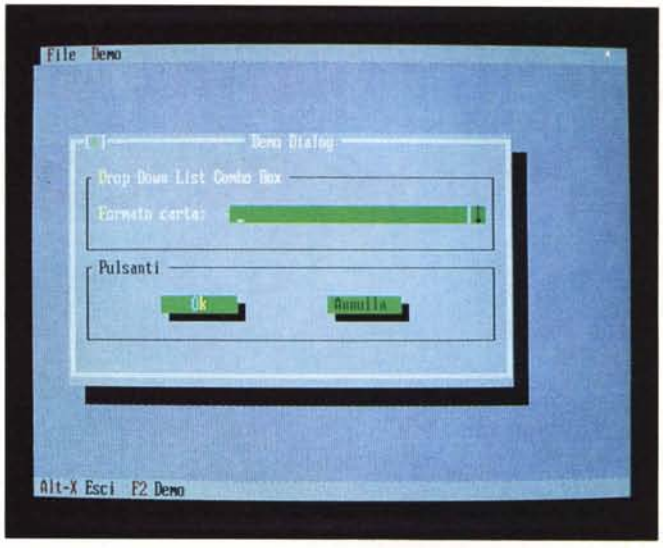
Una combo box semplice può essere realizzata associando una *TListBox* ad una *TInputLine*, ma Turbo Vision non ci offre nulla per gli altri due tipi. Si può

tuttavia notare che ad una *TInputLine* può essere associata una *history list*: tutti i valori immessi in una riga di input possono essere conservati in una lista, che può essere richiamata premendo il tasto «freccia in basso» o clickando col mouse sulla sua rappresentazione visiva; selezionando un elemento della lista si avvalora la riga di input, risparmiando tempo e tasti. Una *history list* assomiglia un po' alla list box di una drop down list combo box; possiamo quindi realizzare una drop down list combo box derivandola da *TInputLine* e associando ad essa una *THistory*, se riusciamo: a) ad aggiungere elementi alla lista prima della sua visualizzazione; b) ad escludere qualsiasi immissione di dati nella riga di input (in quanto si debbono poter scegliere solo gli elementi della list box).

Per aggiungere elementi alla lista, è sufficiente utilizzare la procedura *HistoryAdd*, definita nella unit HISTLIST, che vuole due argomenti: un byte che identifichi la lista e una stringa. Va magari ricordato che quel byte è necessario in quanto tutte le liste gestite da quella unit sono memorizzate in un unico buffer, in cui ogni stringa è preceduta dal byte che distingue una lista dall'altra. Il buffer viene allocato nel metodo *TApplication.Init*, che chiama la procedura *InitHistory*. La dimensione del buffer è di 1024 byte; se occorresse un buffer più grande, basterebbe assegnare un altro valore alla costante tipizzata *HistorySize* prima della chiamata di *InitHistory* (sia questa che *HistorySize* sono definiti in HISTLIST).

Per inibire la riga di input, occorre ridefinire il suo metodo *HandleEvent*, in modo che non faccia nulla se l'utente digita caratteri o usa tasti di inserimento o cancellazione. È anche opportuno ridefinire la funzione *GetPalette*, al fine di dare alla riga di input un colore diverso da quello di una normale *TInputLine*.

Figura 4 - Ecco come si presenta, subito dopo l'apertura, una dialog box con due black frame *e una* drop down list combo box.



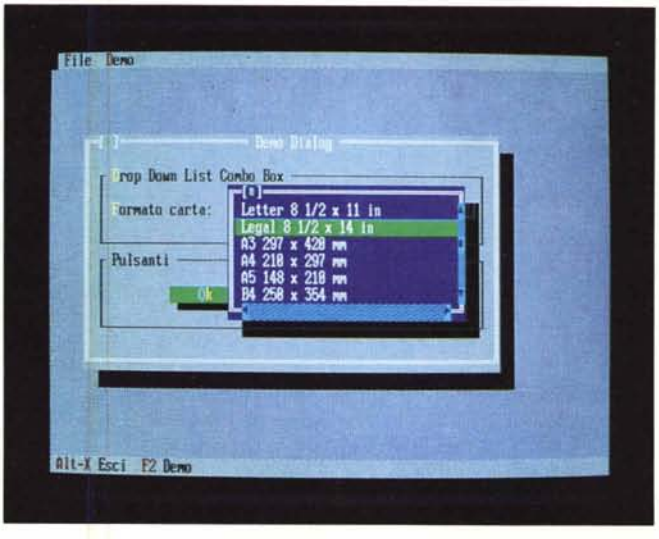


Figura 5 - La dialog box della figura precedente dopo l'apertura della drop down list, *ottenuta mediante la pressione del tasto «freccia in basso» o un click del mouse sulla rappresentazione visiva di questo.*

Una classe privata: TDDLHistory

In realtà tali accorgimenti non sono sufficienti. Succede infatti che, una volta selezionato un elemento della lista, questo viene copiato nella riga di input; accedendo poi ancora alla lista, l'elemento viene copiato in questa dalla riga di input e viene posto al primo posto, alterando l'ordine originario. Un po' come se le opzioni di un gruppo di radio button venissero rimescolate dopo ogni selezione. Una history list, infatti, non ha nulla in comune con una *TCollection* o con una *TSortedCollection*. Ne troviamo conferma nel programma proposto nella figura 3; in *TCombo.DemoDialog* dobbiamo aggiungere stringhe alla combo box nell'ordine inverso rispetto a quello desiderato, in quanto la prima verrà mostrata all'ultimo posto, l'ultima comparirà al primo posto.

Per rendere stabile l'ordine degli elementi della lista, occorre dichiarare una nuova classe da *THistory* per ridefinire anche qui il metodo *HandleEvent*. Quando l'utente apre la lista, il contenuto della riga di input viene salvato in una variabile *S* e poi azzerato, in modo da lasciare «senza materia prima» il meccanismo di aggiornamento del contenuto della lista. Viene poi chiamato il metodo *HandleEvent* di *THistory*, che apre la finestra che funge da list box; questa può venire chiusa in due modi: con la selezione di un elemento (che viene subito passato alla riga di input) o con una rinuncia alla selezione (un Esc o un colpo di mouse sull'icona di chiusura); in quest'ultimo caso la riga di input sarà vuota così come l'avevamo resa e quindi, se il suo precedente contenuto non era nullo, viene ripristinato.

Da notare che tale meccanica è funzionale esclusivamente alla implementazione della classe *TDDLCombo*; la classe *TDDLHistory*, quindi, viene definita nella implementazione della unit MORECTLS e rimane nascosta ad ogni programma o altra unit che usi MORECTLS. Si tratta, in altri termini, di una *classe privata*.

Concludiamo notando che il constructor di *TDDLCombo* ha una struttura analoga a quella di *TBlackFrame*: vuole un parametro di tipo *PDialog* per poter inserire nella finestra di dialogo la history list da associare alla riga di input. MS

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