# TV-Link

*Nel numero di ottobre abbiamo iniziato l'esame di una versione per Turbo Vision di W-LINK, il semplice programma di comunicazione per Windows illustrato nei mesi scorsi. Abbiamo visto l'interfaccia utente e, in particolare, la dialog box per l'impostazione dei parametri di comunicazione. Ci siamo poi soffermati sulla manutenzione del file di profilo del programma, in quanto destinato a mantenere memoria dei parametri più graditi all'utente. Vedremo ora come il programma provvede al dialogo con l'utente da un lato e con il computer remoto dall'altro*

**di Sergio Polini**

Il menu principale di TV-LINK propone due soli sottomenu, *Collegamenti* e *Opzioni*. Scegliendo *Parametri di comunicazione* da quest'ultimo, si apre una dialog box che consente di impostare il nome della porta, la velocità di trasmissione, la parità, i bit di dati e di stop, il tipo di handshake. Chiudendo con il pulsante *Ok* (o con il tasto *Invio*) la dialog box, vengono aggiornati i valori dei campi di una variabile *CommTransBuf*. Non succede altro. Se si desidera salvare i valori nel file di profilo si deve selezionare l'opzione *Salva* del sottomenu *Opzioni*; per contro, i valori così salvati possono essere nuovamente selezionati semplicemente scegliendo l'opzione *Ripristina*.

Questo è quanto abbiamo visto ad ottobre, approfittando dell'argomento trattato per discutere brevemente delle possibilità di validazione dell'input offerte dalle ultime versioni di ObjectWindows e Turbo Vision.

Vedremo ora il file TV-LINK.PAS, per mostrare come dotare un'applicazione Turbo Vision di una *main window* analoga a quella delle applicazioni Windows.

## Il file TV-LINK.PAS

Vi sono due tipi di finestre in Windows, finestre-applicazione e finestre-documento. Queste ultime compaiono solo in applicazioni in grado di gestire contemporaneamente più finestre (la cosiddetta interfaccia MDI), aprendole, chiudendole, cambiandone posizione e dimensioni; le prime sono un tutt'uno con l'applicazione: il loro titolo è, di norma, il nome dell'applicazione, cambiarne la posizione o le dimensioni vuol dire cambiare l'area dello schermo riservata all'applicazione.

In Turbo Vision, invece, all'applicazione non è associata una finestra (istanza di *TWindow*), ma un *DeskTop* che occupa tutto lo schermo e su cui non si può scrivere, in quanto rappresenta solo l'ambiente in cui possono aprirsi finestre analoghe alle finestre-documento di Windows.

In TV-LINK, come dicevamo la volta scorsa, non ci occorrono finestre di cui sia possibile cambiare posizione e dimensioni, in quanto dovremo emulare un'interfaccia TTY. Dal momento che si lavora in modo testo, non è possibile, come in W-LINK, adattare il tipo di carattere alle dimensioni della finestra; riducendo le dimensioni di una finestra, quindi, perderemmo qualcosa: se ne diminuissimo la larghezza, ad esempio, non riusciremmo a vedere tutti i caratteri di una riga. TV-LINK deve avere una finestra che occupi costantemente il maggiore spazio possibile, al fine di non nascondere mai all'utente i momenti del dialogo con il computer remoto.

Ci occorre, quindi, una finestra analoga alle finestre-applicazione di Windows, che si sovrapponga al *DeskTop* fin dall'inizio e, condividendo l'area occupata da questo, non sia né mobile né ridimensionabile. Così si spiegano alcune «stranezze» del file TV-LINK.PAS (figura 1).

Notiamo, in primo luogo, che il constructor della classe *TTVLinkApp* provvede subito all'apertura di una finestra istanza di *TCommWindow*, che, quindi, compare sullo schermo non appena inizia l'esecuzione del programma. Conseguentemente, come avrete già notato, il sistema di menu creato dal metodo *InitMenuBar* non comprende alcuna opzione per l'apertura e la chiusura di finestre, né vi sono combinazioni di tasti suggerite, a tale scopo, dalla riga di stato creata da *InitStatusLine:* la *Window* di TV-LINK si apre automaticamente e si chiude solo al termine dell'esecuzione.

La «centralità» della finestra di TV-LINK è confermata anche dall'assenza di un metodo *HandleEvent*: poiché la classe *TTVLinkApp* non deve occuparsi di altro che di comandi del tutto standard, come *cmQuit*, sono più che sufficienti i comportamenti ereditati dal metodo come definito per la classe *TApplication*. La gestione degli eventi viene quindi delegata, come vedremo, a *TCommWindow*; si ottiene così di semplificare alcuni controlli che si renderanno necessari sulle variabili d'istanza di questa classe.

Oltre a quelli già menzionati, rimane il metodo *TTVLinkApp.Idle*, su cui torneremo tra breve.

## La finestra principale

Quando si apre un'istanza di *TWindow*, il constructor assegna un valore di default alla variabile *Flags*: vengono attivati i flag *wfMove, wfGrow, wfClose* e *wfZoom*; i primi due consentono che la finestra possa essere mossa e ridimensionata, gli altri fanno apparire le icone di chiusura e di zoom che possono essere usate col mouse.

Cambiare la posizione o le dimensioni, chiudere o zoommare, sono tutte operazioni che vogliamo escludere. Il constructor di *TCommWindow* (figura 2), quindi, dopo aver chiamato quello ereditato da *TWindow* indicando come dimensioni della finestra quelle del *DeskTop*, azzera la variabile *Flags*.

Come ricordavamo la volta scorsa, non è consigliabile scrivere direttamente su una finestra in quanto questa, derivando da *TGroup*, è soprattutto un ambiente in cui viene coordinata la presenza di altri oggetti (sottoviste). Si prosegue, quindi, «inserendo» nella finestra un'istanza della classe *TInterior*, sulla quale verranno resi visibili sia i caratteri digitati dall'utente che quelli ricevuti dal computer remoto. Come accade per

ogni vista, il cursore è invisibile per default e, quindi, all'inizio non compare.

La variabile *ComPort* viene inizializzata a –1, valore che viene usato per indicare lo stato di «porta non aperta». La variabile *Profile* viene inizializzata con il titolo della finestra (solo i primi otto caratteri se ve ne sono di più), cui viene aggiunta l'estensione INI; la variabile viene subito usata dal metodo *ReadProfile* per assegnare i parametri di comunicazione salvati nel file ai campi della variabile *CommTransBuf*, o per assegnare loro valori di default in caso di errore di accesso al file (tipicamente: se il file non esiste, in quanto si sta eseguendo il programma per la prima volta).

Viene disabilitato, infine, il comando *cmHangUp*, in quanto non ha senso chiudere un collegamento che non è stato aperto.

### Apertura e chiusura del collegamento

Il metodo *Open* (come gli altri nella figura 2) viene eseguito quando si sceglie l'opzione *Apri* dal menu *Collegamenti*. Alla variabile *ComPort* viene assegnato il numero della porta, ricavato dal quarto carattere del suo nome (1 per COM1, 2 per COM2, ecc.). Si chiama quindi la funzione *OpenCom* della unit ASYNC12 e, se l'apertura ha successo, si procede ad impostare i parametri di comunicazione, ad abilitare il comando *cmHangUp*, a disabilitare i comandi *cmOpen* (la porta è già stata aperta), *cmCommSetup*, *cmSaveSetup* e *CmRestoreSetup* (non viene consentito il cambiamento dei parametri durante la comunicazione) e, infine, a rendere visibile il cursore. Come nel caso di W-LINK, usiamo la visibilità del cursore per indicare all'utente l'avvenuta apertura della porta. Se l'apertura non ha successo, tutto rimane come prima e si mostra un messaggio d'errore.

L'impostazione dei parametri di comunicazione avviene mediante il metodo *SetCommParams*, che traduce i valori dei campi di *CommTransBuf* in valori adatti ad essere passati alle procedure *ComParams*, *SoftHandshake*, *SetRTSMode* e *SetCTSMode* della unit ASYNC12.

Il collegamento si chiude mediante la procedura *CloseCom* della stessa unit, che può essere chiamata sia dal destructor di *TCommWindow* che dal metodo *HangUp* (eseguito quando si sceglie l'opzione *Chiudi* del menu *Collegamenti*). La procedura viene chiamata solo se il valore di *ComPort*, in quanto diverso da –1, indica che una porta è stata effettivamente aperta. Il metodo *HangUp* provvede anche a ripristinare le abilitazioni delle opzioni dei menu e a nascondere il cursore.

### Trasmissione e ricezione di caratteri

Oltre al metodo *CommSetup*, che apre la dialog box per l'impostazione dei parametri di comunicazione, ci rimane da esaminare il metodo *HandleEvent*.

In esso vengono trattati tre tipi di eventi. Nel caso di *evCommand*, non si fa altro che eseguire i metodi corrispondenti alle opzioni del sistema dei menu; nel caso di *evKeyDown*, se qualche porta è aperta (*ComPort* diverso da –1) le si invia il carattere digitato mediante la procedura *ComWriteChW* della unit ASYNC12. Si può notare che il carattere inviato non viene visualizzato, in quanto,

```
program TVLink;
(*$X+*)

uses Objects, Drivers, Views, Menus, App, CommWin;

type
 TTVLinkApp = object(TApplication)
   Window: PCommWindow;
   constructor Init;
   procedure Idle; virtual;
   procedure InitStatusLine; virtual;
   procedure InitMenuBar; virtual;
 end;

constructor TTVLinkApp.Init;
begin
 inherited Init;
 New(Window, Init('TV-Link'));
 if ValidView(Window) <> nil then
   InsertWindow(Window);
end;

procedure TTVLinkApp.Idle;
begin
 inherited Idle;
 Message(Window, evBroadcast, cmEchoChar, nil);
end;

procedure TTVLinkApp.InitMenuBar;
var R: TRect;
begin
 GetExtent(R);
 R.B.Y := R.A.Y + 1;
 MenuBar := New(PMenuBar, Init(R, NewMenu(
  NewSubMenu('~C~ollegamenti', hcNoContext, NewMenu(
   NewItem('~A~pri', '', 0, cmOpen, hcNoContext,
   NewItem('~C~hiudi', '', 0, cmHangUp, hcNoContext,
   NewLine(
   NewItem('~E~sci', 'Alt-X', kbAltX, cmQuit, hcNoContext,
   nil))))),
  NewSubMenu('~O~pzioni', hcNoContext, NewMenu(
   NewItem('~P~arametri comunicazione...', '', 0,
       cmCommSetup, hcNoContext,
   NewLine(
   NewItem('~S~alva', '', 0, cmSaveSetup, hcNoContext,
   NewItem('~R~ipristina', '', 0, cmRestoreSetup, hcNoContext,
   nil))))),
  nil))
 ));
end;
procedure TTVLinkApp.InitStatusLine;
var R: TRect;
begin
 GetExtent(R);
 R.A.Y := R.B.Y - 1;
 StatusLine := New(PStatusLine, Init(R,
  NewStatusDef(0, $FFFF,
   NewStatusKey('', kbF10, cmMenu,
   NewStatusKey('~Alt-X~ Exit', kbAltX, cmQuit,
   nil)),
  nil)
 ));
end;

var
 TVLinkApp: TTVLinkApp;

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

*Figura 1 - Il programma TV-LINK.PAS.*

grazie al full duplex, tornerà indietro dal computer remoto e potrà essere trattato come i caratteri ricevuti.

Il terzo tipo di evento è *evBroadCast*; si tratta, in questo caso, di messaggi inviati dal metodo *Idle* di *TTVLinkApp*. Tale metodo, ereditato da *TProgram*, viene eseguito ogni volta che la coda degli eventi è vuota e, quindi, permette l'esecuzione di compiti non connessi a eventi «normali» senza interferire con la gestione di questi. Quando viene rilevato un evento di tipo *evBroadcast*, si verifica che si tratti effettivamente di un comando *cmEchoChar* (quello azionato da *TTVLinkApp.Idle*) e che la porta sia aperta (*ComPort* diverso da –1); in caso affermativo, si leggono i caratteri in arrivo mediante la funzione *ComReadCh* di ASYNC12, fino a che questa ritorna zero, e si visualizzano con il metodo *EchoChar*.

## Output sul video

Per poter mostrare all'utente i momenti del suo dialogo con il computer remoto, si usa un'istanza della classe *TInterior* (figura 3).

Si tratta di una classe ispirata alle omonime classi usate dalla Borland nei suoi «demo»; come in essi, in particolare, si ridefinisce il metodo *Draw*, usando la procedura *WriteStr* per visualizzare il contenuto di un array bidimensionale di caratteri.

L'array (la variabile d'istanza *Screen*) viene avvalorato dal metodo *EchoChar*, che tratta a parte i caratteri Carriage Return, Line Feed, Form Feed e Backspace, mentre mostra subito sul video gli altri. Nel contempo, vengono aggiornate le variabili *xCursor* e *yCursor*; se si va

```
constructor TCommWindow.Init(ATitle: String);
var
  Bounds: TRect;
begin
  DeskTop^.GetExtent(Bounds);
  inherited Init(Bounds, ATitle, wnNoNumber);
  Flags := 0;
  Bounds.Grow(-1, -1);
  Interior := New(PInterior, Init(Bounds));
  Insert(Interior);
  ComPort := -1;
  Profile := Copy(ATitle, 1, 8) + '.INI';
  ReadProfile;
  DisableCommands([cmHangUp]);
end;

destructor TCommWindow.Done;
begin
  if Comport >= 1 then
    CloseCom(ComPort);
  inherited Done;
end;

procedure TCommWindow.HandleEvent(var Event: TEvent);
var
  Ch: Char;
begin
  inherited HandleEvent(Event);
  case Event.What of
    evKeyDown: if ComPort >= 1 then
                 ComWriteChW(ComPort, Event.CharCode);
    evBroadcast: if (Event.Command = cmEchoChar)
                 and (ComPort >= 1) then begin
                   Ch := ComReadCh(ComPort);
                   while Ch <> #0 do begin
                     Interior^.EchoChar(Ch);
                     Ch := ComReadCh(ComPort);
                   end;
                 end;
    evCommand: case Event.Command of
                 cmOpen: Open;
                 cmHangUp: HangUp;
                 cmCommSetup: CommSetup;
                 cmSaveSetup: SaveSetup;
                 cmRestoreSetup: RestoreSetup;
               end;
  end;
end;

procedure TCommWindow.Open;
var
  Code: Integer;
begin
  Val(CommTransBuf.PortSel[4], ComPort, Code);
  if OpenCom(ComPort, 512, 512) then begin
    SetCommParams;
    EnableCommands([cmHangUp]);
    DisableCommands([cmOpen, cmCommSetup,
                     cmSaveSetup, cmRestoreSetup]);
    Interior^.ShowCursor;
  end
  else begin
    ComPort := -1;
    MessageBox('Errore apertura porta.', nil,
               mfError or mfOkButton);
  end;
end;

procedure TCommWindow.HangUp;
begin
  if ComPort >= 1 then begin
    CloseCom(ComPort);
    DisableCommands([cmHangUp]);
    EnableCommands([cmOpen, cmCommSetup,
                    cmSaveSetup, cmRestoreSetup]);
    Interior^.HideCursor;
  end;
end;

procedure TCommWindow.CommSetup;
var
  D: PCommSetupDlg;
begin
  New(D, Init);
  if Application^.ValidView(D) <> nil then begin
    D^.SetData(CommTransBuf);
    if DeskTop^.ExecView(D) = cmOk then
      D^.GetData(CommTransBuf);
  end;
end;
procedure TCommWindow.SetCommParams;
var
  Baud: Longint;
  Data: Byte;
  Par : Char;
  Stop: Byte;
  Code: Integer;
begin
  with CommTransBuf do begin
    Val(BaudSel, Baud, Code);
    Val(DataSel, Data, Code);
    case ParitySel[1] of
      'N': Par := 'N';
      'D': Par := 'O';
      'P': Par := 'E';
      'M': Par := 'M';
      'S': Par := 'S';
    end;
    Val(StopSel, Stop, Code);
  end;
  ComParams(ComPort, Baud, Data, Par, Stop);
  if CommTransBuf.HShakeSel[1] = 'X' then
    SoftHandshake(ComPort, TRUE, ^Q, ^S)
  else begin
    SetRTSMode(ComPort, TRUE, 256, 256);
    SetCTSMode(ComPort, TRUE);
  end;
end;
```

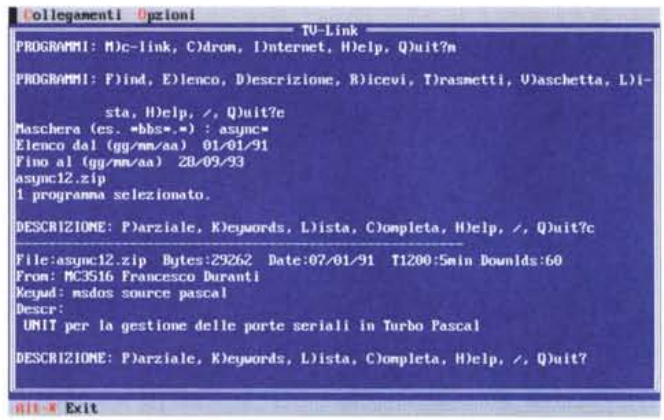
*Figura 2 - I metodi della classe TCommWindow.*

```
constructor TInterior.Init(var Bounds: TRect);
var
  i: Integer;
begin
  TView.Init(Bounds);
  Options := Options or ofSelectable;
  FillChar(Screen, SizeOf(Screen), ' ');
  xCursor := 0;
  yCursor := 0;
end;

procedure TInterior.Draw;
var
  Color: Byte;
  Y: Integer;
  B: TDrawBuffer;
begin
  TView.Draw;
  Color := GetColor(1);
  for Y := 0 to Size.Y - 1 do
    WriteStr(0, Y, Screen[Y], $02);
  SetCursor(xCursor, yCursor);
end;

procedure TInterior.EchoChar(Ch:Char);
begin
  case Ch of
    #13: xCursor := 0;
    #10: Inc(yCursor);
    #12: begin
           FillChar(Screen, MAXROWS*MAXCOLS, ' ');
           xCursor := 0;
           yCursor := 0;
           DrawView;
         end;
    #8 : if xCursor > 0 then Dec(xCursor);
    else begin
      Screen[yCursor,xCursor] := Ch;
      WriteChar(xCursor,yCursor,Ch,$02,1);
      Inc(xCursor);
    end;
  end;
  if xCursor >= MAXCOLS then begin
    xCursor := 0;
    Inc(yCursor);
  end;
  if yCursor >= MAXROWS then begin
    yCursor := MAXROWS - 1;
    Move(Screen[1,0], Screen[0,0],(MAXROWS - 1) * MAXCOLS);
    FillChar(Screen[MAXROWS - 1, 0], MAXCOLS, ' ');
    DrawView;
  end;
  SetCursor(xCursor, yCursor);
end;
```

◀ *Figura 3 - I metodi della classe TInterior.*




▲ *Figura 4 - Collegamento con MC-link mediante il programma TV-LINK. Si stanno chiedendo informazioni sulla unit ASYNC12.*

oltre l'ultima colonna di una riga si passa alla riga successiva; se si va oltre l'ultima riga, si fanno scorrere verso l'alto le righe di *Screen* e si ridisegna tutto lo schermo chiamando *DrawView*.

La posizione del cursore viene aggiornata sia da *Draw* che da *EchoChar*; questo, ovviamente, se il cursore è visibile. Ricordiamo che in ogni *View* il cursore è invisibile per default: compare solo dopo *ShowCursor*, scompare nuovamente dopo un *HideCursor*. Così dicono i manuali; in essi, tuttavia, gli esempi di uso di una vista come *Interior* sono limitati alla visualizzazione in sola lettura di un file di testo, senza le funzioni di editing offerte dalla unit EDITORS e, quindi, senza alcun bisogno di un cursore. Ne segue che, se semplicemente si seguono le indicazioni dei manuali (mi riferisco in particolare a quelli della versione 6.0), dopo *ShowCursor* il cursore non si vede!

Vediamo perché. Ogni «gruppo», cioè ogni istanza di una classe derivata da *TGroup*, quindi anche ogni istanza di *TWindow*, gestisce una lista di subviste, delle quali solo una può risultare, in un dato momento, quella correntemente selezionata; questa avrà il flag *sfSelected* settato e ad essa punterà la variabile d'istanza *Current* del «gruppo». In ogni «gruppo», a partite da *Application*, la subvista selezionata può a sua volta essere un «gruppo» ed avere, quindi, una subvista selezionata, e così via. Si stabilisce così una catena di «gruppi» collegati mediante la variabile *Current* di ognuno. La subvista selezionata dell'ultimo «gruppo» della catena viene detta *focused*; è quella «a fuoco», quella su cui si concentra l'attenzione, quella a cui vengono indirizzati gli eventi intercettati dal suo «gruppo» e dagli altri che lo precedono nella catena; è quella, quindi, in cui si svolge l'azione e si distingue da tutte le altre in quanto è l'unica ad avere il flag *sfFocused* settato.

Perché il cursore sia visibile, spiega il manuale, è necessario che siano settati i flag *sfFocused* e *sfCursorVis*. Il metodo *ShowCursor* provvede a settare *sfCursorVis*; quanto a *sfFocused*, però, occorre contare sul meccanismo attraverso il quale un «gruppo» passa il «fuoco» ad una delle sue viste. Affinché tutto proceda come ci aspettiamo, si deve ricordare che il constructor di *TView* azzera la variabile *Options*; in particolare, ciò comporta che non risulta settato il flag *ofSelectable* e, quindi, che una subvista risulta per default non selezionabile. Ciò consente di inserire in un «gruppo» elementi, per così dire, inerti, come le istanze di *TFrame* o di *TStaticText*; per inserire elementi che inerti non siano, occorre agire sulla loro variabile *Options* per settare il flag *ofSelectable*.

Scorrendo i sorgenti di Turbo Vision, infatti, si può notare che i constructor di *TFrame* o di *TStaticText* non modificano la variabile *Options*, mentre quelli di *TButton*, di *TCluster* o di *TInputLine* provvedono a settare alcuni flag, tra i quali, appunto, *ofSelectable*; così anche il constructor di *TEdit* nella unit EDITORS.

Per poter vedere il cursore nella nostra vista *TInterior*, quindi, non sarà sufficiente contare sul metodo *ShowCursor*; occorrerà anche settare quel flag nel constructor.

Abbiamo così terminato l'esame del programma TV-LINK in tutti i suoi aspetti. Vi do appuntamento al mese prossimo per parlare di modo reale e modo protetto, DPMI, uso della API di Windows sotto DOS. MC

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