Comunicare con Turbo Vision

Ad aprile abbiamo iniziato un discorso sulla programmazione per eventi, cercando di mettere in evidenza come non solo l'interfaccia utente (l'apparenza), ma anche i meccanismi in azione dietro le quinte (la sostanza) richiedano cure particolari. Vi proposi anche un esempio di un'applicazione in cui, trascurando i principi della programmazione per eventi, si rischiava di imboccare un vicolo cieco. Per fortuna, per operare correttamente è spesso sufficiente cercare l'«aggancio» giusto...

di Sergio Polini

Scrivo queste righe in un caldo mese di agosto, in un momento in cui gli echi della cronaca sono più deboli. Posso assicurarvi, comunque, che gli *agganci* di cui parlo non hanno nulla a che vedere con mani poco pulite.

Come precisavo ad aprile, mi piace chiamare agganci quei punti, presenti in ogni ambiente ad eventi che si rispetti, che consentono di inserirsi nel meccanismo generale per ottenere l'effetto giusto nel momento giusto. Per poter trattare in modo concreto l'argomento, vi ho proposto la gestione della comunicazione tramite porta seriale; come già accaduto per le operazioni di stampa, anche in questa occasione abbiamo esaminato in primo luogo come operare sotto Windows. È stato così che abbiamo potuto apprezzare l'importanza della funzione *EnableCommNotification*, nonché discutere delle possibilità di intervento nel Message Loop di un'applicazione. Sotto Windows la programmazione per eventi si traduce in programmazione per messaggi; la funzione *EnableCommNotification* offre un «aggancio» in quanto trasforma in messaggio la ricezione di un carattere, il *Message Loop* rappresenta invece l'aggancio principe, in quanto è il luogo in cui vanno eseguite quelle azioni che non dipendono dalla ricezione di un messaggio.

Anche Turbo Vision è un ambiente ad eventi, ma richiede forse uno sforzo di concentrazione in più rispetto a Windows: è apparentemente possibile, infatti, limitarsi a quel minimo necessario per far funzionare l'interfaccia utente, mantenendo per il resto uno stile di programmazione tradizionale. Può però così capitare, ad esempio, che ci si trovi a lavorare più del necessario. Ricordo, in particolare, una discussione svoltasi qualche tempo fa su MC-link: un abbonato (non me ne chiedete il nome!), voleva associare un particolare effetto alla pressione di un pulsante in una dialog box; credeva di poter procedere ridefinendo il metodo *TButton.HandleEvent*. Purtroppo, così facendo raggiungeva comunque, sia pure faticosamente, lo scopo; sarebbe stato meglio, tuttavia, se avesse per prima cosa cercato l'aggancio giusto. Avrebbe così scoperto che esiste anche un metodo *TButton.Press*, che sta lì proprio per consentire di associare con semplicità un qualsiasi comportamento alla pressione di un pulsante. Più in generale, non vanno trascurati gli agganci offerti dalla funzione *Message* (che consente di tradurre in evento quasi tutto quello che si vuole), né dal metodo *TApplication.Idle* (la cui ridefinizione corrisponde, in buona sostanza, alla modifica del *Message Loop* di un'applicazione Windows).

Anche in Turbo Vision, la comunicazione seriale costituisce un ottimo esempio di programmazione per eventi non limitata alla sola interfaccia.

La unit ASYNC12

Avevo progettato, in un primo momento, di realizzare una unit per la comunicazione seriale, con caratteristiche analoghe a quelle delle funzioni della API di Windows. Il progetto, in verità, è stato quasi portato a termine, ma l'esposizione di una tale unit sulla rivista richiederebbe troppo spazio (762 righe di listato) e, quindi, troppo tempo.

Ci serviremo invece di ASYNC12, una unit di pubblico dominio disponibile anche su MC-link (il nome del file è ASYNC12.ZIP e comprende anche i sorgenti), che consente la gestione contemporanea di quattro porte seriali, con handshake hardware o software, con velocità di trasmissione da 50 a 115200 baud.

L'interfaccia della unit è molto ricca; comprende, infatti, 6 costanti, 5 tipi, 17 variabili e 28 funzioni e procedure. Ai nostri scopi, tuttavia, saranno sufficienti le funzioni e procedure indicate nella figura 1.

```
function OpenCom(ComPort: Byte;
                 InBufferSize, OutBufferSize: Word): Boolean;

procedure ComParams(ComPort: Byte; Baud: Longint; WordSize: Byte;
                    Parity: Char; StopBits: Byte);

procedure SetRTSMode(ComPort: Byte; Mode: Boolean; RTSOn, RTSOff: Word);

procedure SetCTSMode(ComPort: Byte; Mode: Boolean);

procedure SoftHandshake(ComPort: Byte; Mode: Boolean;
                        Start, Stop: Char);

procedure ComWriteChW(ComPort: Byte; Ch: Char);

function ComReadCh(ComPort: Byte): Char;

procedure CloseCom(ComPort: Byte);
```

Figura 1 - Le funzioni e procedure della unit ASYNC12 utilizzate nel programma TV-LINK.

```
unit CommWin;

interface

uses Objects, Drivers, Views, Async12;

const
  cmOpen = 101;
  cmHangUp = 102;
  cmCommSetup = 201;
  cmSaveSetup = 202;
  cmRestoreSetup = 203;
  cmEchoChar = 100;
  MAXROWS = 21;
  MAXCOLS = 78;

type
  PInterior = ^TInterior;
  TInterior = object(TView)
    constructor Init(var Bounds: TRect);
    procedure Draw; virtual;
    procedure EchoChar(Ch: Char);
  private
    xCursor,yCursor: Integer;
    Screen: array[0..MAXROWS-1, 0..MAXCOLS-1] of Char;
  end;
  PCommWindow = ^TCommWindow;
  TCommWindow = object(TWindow)
    Interior: PInterior;
    constructor Init(ATitle: String);
    destructor Done; virtual;
    procedure HandleEvent(var Event: TEvent); virtual;
    procedure Open;
    procedure HangUp;
    procedure CommSetup;
    procedure SaveSetup;
    procedure RestoreSetup;
  private
    ComPort: Integer;
    Profile: String[12];
    f: File;
    procedure WriteProfile;
    procedure ReadProfile;
    procedure SetCommParams;
  end;
```

Figura 2 - L'interfaccia della unit COMMWIN nella versione per Turbo Vision.

Tutte le funzioni e procedure che useremo richiedono un primo parametro *ComPort*, che deve essere un byte con valore da 1 a 4, per le quattro porte gestibili dalla unit.

Una porta si apre con la funzione *OpenCom*, che richiede, oltre all'indicazione del numero della porta, le dimensioni dei buffer in entrata e in uscita, corrispondenti alle code *InQueue* e *OutQueue* usate sotto Windows. La funzione ritorna TRUE in caso di successo, FALSE in caso contrario.

La procedura *ComParams* imposta i parametri di comunicazione; richiede la velocità di trasmissione (*Baud*, da 50 a 115200), i bit di dati e di stop (*WordSize*, da 5 a 8, e *StopBits*, 1 o 2) e la parità (*Parity*, un carattere tra N, E, O, M e S per nessuna, pari, dispari, mark e spazio). La funzione *SetCommState* di Windows permette anche di impostare l'handshake e ritorna un valore diverso da zero in caso di errore; *ComParams*, invece, non fa nulla se *ComPort* non ha un valore accettabile e adotta valori di default in caso di errori negli altri parametri; quanto all'handshake, occorre usare altre procedure: chiamando *SetRTSMode* con TRUE come secondo parametro, il segnale RTS verrà abbattuto quando nel buffer di input vi saranno *RTSOff* o più caratteri, per essere alzato di nuovo quando vi saranno non più di *RTSOn* caratteri; chiamando *SetCTSMode* con TRUE come secondo parametro, l'invio di caratteri verrà sospeso quando il computer remoto abbatterà il segnale CTS e ripreso quando il segnale risulterà nuovamente alzato; chiamando *SoftHandshake* con TRUE come secondo parametro, si attiva un handshake software regolato dai caratteri *Start* e *Stop* (normalmente Ctrl-Q e Ctrl-S).

La procedura *ComWriteChW* invia un carattere al computer remoto; la «w» finale sta per *wait*, in quanto, prima di mettere il carattere nel buffer di uscita, la procedura attende che vi sia spazio disponibile in esso. La funzione *ComReadCh* legge dalla porta un carattere inviato dal computer remoto; se il buffer d'entrata è vuoto, il numero della porta non è valido, o la porta non è stata aperta con *OpenCom*, la funzione ritorna il carattere con codice ASCII zero.

CloseCom, infine, rilascia i buffer allocati da *OpenCom* e chiude la porta abbattendo i segnali DTR e RTS (questi e gli altri segnali sono stati discussi nel numero di aprile).

La unit non è stata progettata specificamente per un ambiente ad eventi; può essere usata anche in programmi realizzati in modo tradizionale. Il suo utilizzo in Turbo Vision richiede quindi qualche accorgimento: mancando una funzione corrispondente a *EnableCommNotification*, non si può contare su una automatica trasformazione in evento (o in messaggio) della ricezione di un carattere.

La unit COMMWIN

Prima di affrontare il problema della ricezione dei caratteri, sarà opportuno predisporre il contesto in cui ciò dovrà accadere. Costruiremo, a questo scopo, un programma TV-LINK dall'interfaccia e dalla funzionalità strettamente analoghe a quelle di W-LINK, il semplice programma di comunicazione sotto Windows illustrato nei mesi scorsi.

Avremo un menu con due soli sottomenù, *Collegamenti* e *Opzioni*; il primo consentirà, come in W-LINK, di aprire e chiudere un collegamento e di uscire dall'applicazione; il secondo permetterà di impostare i parametri di comunicazione, salvandoli in un file di profilo e richiamandoli da questo. Per ovvi motivi, mancherà la possibilità di scegliere il tipo di carattere (qui siamo in modo testo!).

Il dialogo con l'utente si svolgerà in una finestra che verrà aperta automaticamente all'inizio dell'esecuzione; sarà una finestra che l'utente non potrà chiu-

dere né muovere, di cui non potrà mutare le dimensioni (non potendo tali mutamenti essere accompagnati dal variare delle dimensioni dei caratteri) ed in cui, come accadeva in W-LINK, verrà simulata una tradizionale interfaccia TTY: i caratteri riempiono lo schermo da sinistra verso destra, dall'alto vero il basso. Vedremo che la tecnica per l'implementazione di tale interfaccia sarà analoga a quella già adottata in W-LINK, con una importante differenza: seguendo le raccomandazioni della Borland, non scriveremo direttamente nella finestra, in quanto le finestre del Turbo Vision non sono lavagne pronte per i nostri colpi di gesso, ma *gruppi* (derivano da *TGroup*); per scrivere in una finestra occorre inserire in essa una vista (l'istanza di una classe derivata da *TView*) che sappia come mostrarsi all'utente. Analogamente alle viste proposte nei manuali Borland, chiameremo la nostra *interior*; per altro verso, integreremo le esemplificazioni proposte da quei manuali (sarei tentato di dire che colmeremo una lacuna), mostrando cosa avviene del *focus* di una vista interna ad una finestra e come adattare i meccanismi di default a specifiche esigenze.

L'interfaccia della unit COMMWIN sarà quella riprodotta nella figura 2.

Si può notare una forte analogia con l'omonima unit predisposta per l'applicazione Windows (l'interfaccia della unit usata da W-LINK è stata illustrata nell'articolo di luglio/agosto). I metodi della classe *TCommWindow* sono ora ripartiti tra una classe denominata anch'essa *TCommWindow*, derivata da

```
implementation

uses Dialogs, App, MsgBox, MoreCtls;

const
  PortName: array[0..3] of String[4] = ('COM1', 'COM2', 'COM3', 'COM4');
  BaudRate: array[0..9] of String[5] =
                              ('110','300','600','1200','2400','4800',
                               '9600','14400','19200','38400');
  Parity  : array[0..4] of String[7] =
    ('Nessuna', 'Dispari', 'Pari', 'Mark', 'Spazio');
  DataBits: array[0..3] of String[1] = ('5', '6', '7', '8');
  StopBits: array[0..1] of String[1] = ('1', '2');
  Handshake: array[0..1] of String[8] = ('XON/XOFF', 'Hardware');

type
  PCommSetupDlg = ^TCommSetupDlg;
  TCommSetupDlg = object(TDialog)
    constructor Init;
  end;
  TCommTransBuf = record
    PortSel  : String[4];
    BaudSel  : String[5];
    ParitySel: String[7];
    DataSel  : String[1];
    StopSel  : String[1];
    HShakeSel: String[8];
  end;

var
  CommTransBuf: TCommTransBuf;

constructor TCommSetupDlg.Init;
var
  R: TRect;
  CB: PDDLCombo;
  i: Integer;
begin
  R.Assign(0,0,53,14);
  inherited Init(R, 'Parametri di comunicazione');
  Options := Options or ofCentered;

  R.Assign(14,2,24,3);
  CB := New(PDDLCombo, Init(@Self, R, 4, 1));
  for i := 3 downto 0 do
    CB^.Add(PortName[i]);
  Insert(CB);
  R.Assign(2,2,9,3);
  Insert(New(PLabel, Init(R, '~P~orta:', CB)));

  R.Assign(13,4,24,5);
  CB := New(PDDLCombo, Init(@Self, R, 5, 2));
  for i := 9 downto 0 do
    CB^.Add(BaudRate[i]);
```

Validazione dell'input in ObjectWindows e in Turbo Vision

In una dialog box possono essere inseriti diversi tipi di controlli. Potremmo distinguere tra quelli che richiedono e quelli che non richiedono la verifica della correttezza delle scelte operate dall'utente. Nel caso di *check box* o di *radio button*, ad esempio, l'utente non può che scegliere una o più tra le possibilità che gli vengono offerte; nel caso di una *input line*, al contrario, l'utente è libero di scrivere quello che crede, anche «pippo» quando gli si chiede un numero. Le input line, quindi, pongono spesso il problema della validazione dell'input.

Nella prima versione di Turbo Vision, si potevano percorrere due strade: verificare la correttezza dell'input nel metodo *Valid* della dialog box, o derivare classi specifiche da *TInputLine*. Nel primo caso, l'utente veniva avvertito dell'eventuale errore solo al momento della chiusura della dialog box invece che subito dopo l'immissione del dato; nel secondo caso, il programmatore era costretto a creare molteplici classi, magari appesantendo ulteriormente una gerarchia che già la Borland proponeva comprensiva di classi come *TKeyInputLine* o *TNumInputLine*.

Nel Pascal 7.0, tuttavia, la classe *TInputLine* comprende una variabile di istanza *Validator* e un metodo *SetValidator*. Diventa così possibile verificare la coerenza tra il valore immesso dall'utente e i valori richiesti dall'applicazione, senza la necessità di creare nuove classi.

Alla variabile *Validator* possono essere assegnate istanze di classi tutte derivate da una classe astratta *TValidator*; sono già disponibili, in una unit VALIDATE, le classi *TFilterValidator, TRangeValidator, TStringLookupValidator, TPXPictureValidator*.

TFilterValidator può essere usata ogni volta che si desidera che i caratteri digitati dall'utente siano compresi in un dato insieme (indicato come argomento del constructor), mentre *TRangeValidator* controlla che l'utente immetta un intero compreso tra un minimo e un massimo. *TStringLookupValidator* permette di verificare che la stringa immessa dall'utente sia compresa in una *TStringCollection* passata come argomento al constructor, mentre *TPXPictureValidator* controlla che la stringa immessa sia coerente con uno dei formati usati in Paradox.

Supponiamo, ad esempio, che il nostro programma di comunicazione chieda all'utente l'indirizzo in esadecimale di una data porta seriale. Saranno accettabili solo i caratteri da '0' a '9' e da 'A' a 'F'. Potremmo procedere così:

```
  Insert(CB);
  R.Assign(2,4,13,5);
  Insert(New(PLabel, Init(R, '~V~elocità:', CB)));
  R.Assign(11,6,24,7);
  CB := New(PDDLCombo, Init(@Self, R, 7, 3));
  for i := 4 downto 0 do
    CB^.Add(Parity[i]);
  Insert(CB);
  R.Assign(2,6,10,7);
  Insert(New(PLabel, Init(R, 'P~a~rità:', CB)));

  R.Assign(44,2,51,3);
  CB := New(PDDLCombo, Init(@Self, R, 1, 4));
  for i := 3 downto 0 do
    CB^.Add(DataBits[i]);
  Insert(CB);
  R.Assign(25,2,38,3);
  Insert(New(PLabel, Init(R, 'Bit di ~d~ati:', CB)));

  R.Assign(44,4,51,5);
  CB := New(PDDLCombo, Init(@Self, R, 1, 5));
  for i := 1 downto 0 do
    CB^.Add(StopBits[i]);
  Insert(CB);
  R.Assign(25,4,38,5);
  Insert(New(PLabel, Init(R, 'Bit di ~s~top:', CB)));

  R.Assign(37,6,51,7);
  CB := New(PDDLCombo, Init(@Self, R, 8, 6));
  for i := 1 downto 0 do
    CB^.Add(Handshake[i]);
  Insert(CB);
  R.Assign(25,6,37,7);
  Insert(New(PLabel, Init(R, '~H~andshake:', CB)));

  R.Assign(11,10,23,12);
  Insert(New(PButton, Init(R, 'O~k~', cmOk, bfDefault)));
  R.Assign(31,10,43,12);
  Insert(New(PButton, Init(R, ' Annulla', cmCancel, 0)));

  SelectNext(False);
end;
```

Figura 3 - La prima parte della implementazione della unit COMMWIN, con le dichiarazioni relative alla dialog box usata per l'impostazione dei parametri di comunicazione.

TWindow, e una classe *TInterior*, derivata da *TView*. Come si indovina agevolmente dai suoi metodi, e ancor più dalle sue variabili d'istanza, *TInterior* è dedicata esclusivamente alla simulazione dell'interfaccia TTY: si propone all'utente uno schermo di 21 righe per 78 colonne (tante quante ne rimangono tolte le righe per il menu e la riga di stato e la cornice della finestra), in cui la posizione corrente è mantenuta nelle variabili *xCursor* e *yCursor*.

La classe *TCommWindow*, per parte sua, si incarica di aprire e chiudere i collegamenti, di impostare i parametri di comunicazione, di gestire il file di profilo, di inviare e ricevere caratteri.

I parametri di comunicazione

L'impostazione dei parametri di comunicazione avviene mediante una dialog box istanza di una classe *TCommSetupDlg*, derivata da *TDialog* e simile a quella usata per W-LINK (illustrata a giugno), dichiarata nella implementazione della unit (figura 3).

Le costanti utilizzate sono praticamente le stesse; va rilevato, magari, che manca l'opzione della misura di un bit e mezzo per i bit di stop, in quanto la unit ASYNC12 ne ammette solo uno o due.

Il record *TCommTransBuf* viene dichiarato al solo scopo di disporre di una variabile *CommTransBuf*, utilizzata sia per inizializzare i controlli della dialog box e per leggere i nuovi valori scelti dall'utente (mediante i metodi di *TDialog SetData* e *GetData*), sia per le ope-

```
const
  HexDigits: Set of Char = ['0'..'9', 'A'..'F', 'a'..'f'];
var
  R: TRect;
  IL: PInputLine;
begin
  ...
  R.Assign(x1, y1, x2, y2);
  IL := New(PInputLine, Init(R, 4));
  IL^.Validator := New(PFilterValidator, Init(HexDigits));
  Insert(IL);
  R.Assign(x1-12, y1, x2-7, y2);
  Insert(New(PLabel, Init(R, '~I~ndirizzo:', IL)));
  ...
end;
```

Agendo sul campo Options della *input line*, settando o no il *flag ofValidate*, si potrà scegliere quando sarà chiamato il metodo *Valid* del validatore, se non appena la *input line* perde il *focus* o solo quando viene chiusa la dialog box. In ogni caso, quel metodo controlla il dato imesso e, se questo risulta non conforme alle aspettative, chiama un metodo *Error*, grazie al quale si ottiene l'automatica produzione di un messaggio d'errore.

Va segnalato che un meccanismo analogo è disponibile anche in ObjectWindows; anche qui, infatti, alla classe *TEdit* è stata aggiunta una variabile di istanza *Validator*.

Avremmo potuto costruire la dialog box per l'impostazione dei parametri di comunicazione usando i *validator*. In verità, ritengo che, in questo caso, le *drop down list combo box* (sia quelle di Windows, sia quelle che abbiamo aggiunto a Turbo Vision con la unit MORECTLS) servano meglio allo scopo. Rimane evidente, tuttavia, che in molte altre occasioni i *validator* possono semplificare notevolmente il lavoro di programmazione.

razioni di lettura e scrittura sul file di profilo.

La classe *TCommSetupDlg* non richiede particolari commenti: viene ridefinito il solo constructor, per inserire nella dialog box i vari controlli. Questi, con la sola eccezione dei due pulsanti *Ok* e *Annulla*, sono tutti di tipo *TDDLCombo*, sono cioè *drop down list combo box*, quel tipo presente in Windows e che abbiamo aggiunto nel maggio dello scorso anno anche a Turbo Vision (con una unit MORECTLS) e usato poi più volte.

```
procedure TCommWindow.SaveSetup;
begin
  WriteProfile;
end;

procedure TCommWindow.RestoreSetup;
begin
  ReadProfile;
end;

procedure TCommWindow.WriteProfile;
begin
  {$I-}
  Assign(f, Profile);
  Rewrite(f, SizeOf(TCommTransBuf));
  BlockWrite(f, CommTransBuf, 1);
  System.Close(f);
  {$I+}
  if IOResult <> 0 then
    MessageBox('Errore scrittura file di profilo.', nil,
      mfError or mfOkButton);
end;

procedure TCommWindow.ReadProfile;
var
  Errore: Integer;
procedure Default;
  begin
    with CommTransBuf do begin
      PortSel   := PortName[0];
      BaudSel   := BaudRate[6];
      ParitySel:= Parity[0];
      DataSel   := DataBits[3];
      StopSel   := StopBits[0];
      HShakeSel:= Handshake[1];
    end;
  end;
begin
  {$I-}
  Assign(f, Profile);
  Reset(f, SizeOf(TCommTransBuf));
  {$I+}
  Errore := IOResult;
  if Errore = 2 then begin
    Default;
    MessageBox('File di profilo non trovato.'+
               #13'Usati parametri di default.',
               nil, mfInformation or mfOkButton);
  end
  else begin
    if Errore = 0 then begin
      {$I-}
      BlockRead(f, CommTransBuf, 1);
      System.Close(f);
      {$I+}
      Errore := IOResult;
    end;
    if Errore <> 0 then begin
      Default;
      MessageBox('Errore lettura file di profilo.'+
                 #13'Usati parametri di default.',
                 nil, mfError or mfOkButton);
    end;
  end;
end;
```

Figura 4 - I metodi di TCommWindow *per la gestione del file di profilo.*

Analogamente a quanto avveniva in W-LINK, i parametri di comunicazione scelti dall'utente possono essere salvati in un file di profilo e da questo ripristinati. Chiudendo la dialog box con il pulsante *Ok*, vengono modificati i valori dei campi della variabile *CommTransBuf*; tali valori possono quindi essere usati in una sessione di comunicazione e anche salvati nel file di profilo (opzione *Salva* del menu *Opzioni*). Se i nuovi valori non vengono salvati, si possono ripristinare altri valori precedentemente salvati nel file di profilo, mediante l'opzione *Ripristina* del menu *Opzioni*.

La gestione del file di profilo non è flessibile come sotto Windows: manca (ovviamente) la possibilità di scegliere tra WIN.INI e un file specifico dell'applicazione, o di collocare il file in una directory di sistema; manca, soprattutto, l'impostazione automatica di valori di default nel caso il file non venga trovato (ad esempio, quando l'applicazione viene eseguita per la prima volta).

Quanto al primo punto, si adotta la soluzione più semplice, assumendo che il file si trovi, o vada creato, nella directory corrente; per il resto, si provvede «manualmente».

Alle opzioni *Salva* e *Ripristina* del menu *Opzioni* corrispondono i metodi *SaveSetup* e *RestoreSetup* di *TCommWindow*, i quali si limitano a chiamare, rispettivamente, i metodi *WriteProfile* e *ReadProfile* (*ReadProfile* è chiamato anche dal constructor di *TCommWindow*; *WriteProfile* è distinto da *SaveSetup* unicamente per simmetria...). Il primo scrive i valori contenuti in *CommTransBuf* in un file il cui nome è assegnato alla variabile *Profile* dal constructor di *TCommWindow* (è il nome della finestra - TV-LINK nel nostro caso - con il suffisso .INI), emettendo un messaggio in caso di errore; il secondo legge i valori dal file, assegnandoli a *CommTransBuf*; se il file non viene trovato, o se si verifica un errore durante la sua lettura, si chiama una procedura *Default* che assegna a *CommTransBuf* valori di default (COM1, 9600 baud, nessuna parità, otto bit di dati, un bit di stop, handshake hardware).

Per ora ci fermiamo qui; mi preme, infatti, sottolineare a lato alcune interessanti caratteristiche della versione di Turbo Vision disponibile con il Pascal 7.0, non ancora discusse sulla rivista. Vi aspetto tra un mese per proseguire nell'esame del nostro semplice programma di comunicazione. MC

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

COME DIRIGERE UN'ORCHESTRA DI 400.000 ELEMENTI?

Avete bisogno solo di un Personal Computer ed un lettore CD ROM per scoprire le fantastiche possibilità del COMPACT DISC MONACI, la banca dati della Guida Monaci su disco a lettura ottica.

Pensate. 200.000 Aziende ed Enti del settore privato e pubblico, 210.000 persone con cariche e qualifiche. Un mondo di informazioni costantemente aggiornate. E tutto in un piccolo disco di 12 centimetri!

Ma il bello è che non c'è nessun limite a quello che può fare il COMPACT DISC MONACI.

Con 24 parametri di ricerca fra loro combinabili, potete segmentare la clientela per aree geografiche e per settore produttivo, per classi di fatturato e dipendenti, stampare indirizzi per azioni di Direct Marketing, individuare nuovi mercati oppure... scegliete Voi cosa.

COMPACT DISC MONACI. Musica Nuova in Ufficio.

GUIDA MONACI

ROMA 00189 Via Vitorchiano 107-109 Tel. 06-3331333 Telefax 06-3335555 - MILANO 20145 Via V. Monti 86 Tel. 02-33610567 Telefax 02-33103204