

## Una frame window per applicazioni MDI

di Sergio Polini

*Carlo Fanciotti di Bologna, che ci legge da un anno, mi chiede come realizzare un programma residente capace di memorizzare l'immagine grafica del video. Ricordo che ho trattato dei programmi residenti nei numeri 92-97, dal gennaio al giugno del '90. Ci sono esempi di memorizzazione dell'immagine del video in modo testo, facilmente adattabili ad un video grafico. Si può chiedere in redazione il dischetto con i sorgenti. Cristiano Tonezzer di Caldonazzo (TN) mi espone problemi nell'uso della unit EXECSWAP (discussa nei numeri 89-91) con il Turbo Pascal 6.0. Ricordo che nella versione 6.0 cambia la gestione dell'heap e, quindi, i problemi sono inevitabili, a meno di non modificare la unit. Ritengo tuttavia che l'implementazione di una shell DOS come realizzabile in Turbo Vision renda superfluo, salvo casi particolari, il ricorso a EXECSWAP*

La volta scorsa abbiamo cominciato a disegnare le prime tessere del nostro mosaico: un insieme di unit che consentano di realizzare facilmente, con ObjectWindows, applicazioni MDI con un ribbon e una riga di stato. La unit STRIPES contiene la definizione delle caratteristiche più generali di un ribbon, inteso come dialog box non modale, e di una riga di stato, intesa come sede di campi di tipo STATIC, uno dei quali dedicato alla visualizzazione di brevi descrizioni delle opzioni dei menu. La unit MDICLNT propone una classe *TNewMDIClient*, da usare in luogo della classe *TMDIClient*, in quanto provvede a cambiare il menu e ad aggiornare la riga di stato secondo che sia attiva o no una child window. Nella unit MDICHL

compare una classe *TNewMDIChild*, che può essere usata sia come classe base, in luogo di *TWindow*, sia come schema delle modifiche da apportare ad una classe non direttamente derivata da *TWindow* per renderla compatibile con il resto. Abbiamo peraltro rimandato l'esposizione dei metodi *WMMMenuSelect* e *WMEnterIdle* di *TNewMDIChild*, in quanto la loro comprensione richiede il preliminare esame dell'ultima tessera, cioè di una unit *TNewMDIFrame* da usare in luogo di *TMDIFrame*.

### La unit TNewMDIFrame

La definizione della classe *TNewMDIFrame* (figura 1) è piuttosto ricca, con sei variabili d'istanza, un constructor, due metodi virtuali e nove metodi dinamici; molti dei metodi, tuttavia, sono abbastanza semplici.

La variabile *ActiveChild* contiene un puntatore all'eventuale child window attiva; inizializzato a **nil** nel constructor, viene mantenuto aggiornato dal metodo *UMSetActiveChild*, che risponde al messaggio *um\_SetActiveChild* (inviato dalla client window o da una child window appena attivata) e provvede anche ad abilitare o disabilitare il ribbon secondo che vi sia o no una finestra attiva.

Le variabili *Ribbon* e *StatusBar*, anch'esse inizializzate a **nil**, sono destinate a contenere l'indirizzo degli eventuali ribbon e riga di stato di cui sarà dotata la frame window del-

```

unit MDIFrame;

interface

uses WinTypes, WinProcs, WObjects, Stripes, MDIClnt;

type
  PNewMDIFrame = ^TNewMDIFrame;
  TNewMDIFrame = object(TMDIWindow)
    ActiveChild: PWindow;
    Ribbon: PMDIRibbon;
    StatusBar: PMDIStatusBar;
    HelpCode: Word;
    DispHelp: Boolean;
    SBText: PChar;
    constructor Init(ATitle: PChar; AMenu: HMenu);
    procedure InitClientWindow; virtual;
    procedure SetupWindow; virtual;
    procedure WMSize(var Msg: TMessage);
      virtual wm_First + wm_Size;
    procedure UMSetActiveChild(var Msg: TMessage);
      virtual wm_First + um_SetActiveChild;
    procedure WMMMenuSelect(var Msg: TMessage);
      virtual wm_First + wm_MenuSelect;
    procedure WMEnterIdle(var Msg: TMessage);
      virtual wm_First + wm_EnterIdle;
    procedure UMSetHelpCode(var Msg: TMessage);
      virtual wm_First + um_SetHelpCode;
    procedure UMPaintStatusBar(var Msg: TMessage);
      virtual wm_First + um_PaintStatusBar;
    procedure CMToggleRibbon(var Msg: TMessage);
      virtual cm_First + cm_ToggleRibbon;
    procedure CMToggleStatusBar(var Msg: TMessage);
      virtual cm_First + cm_ToggleStatusBar;
    procedure WMInitMenu(var Msg: TMessage);
      virtual wm_First + wm_InitMenu;
  end;

```

(continua a p. 340)

l'applicazione reale, come vedremo nel demo. Il metodo *WMInitMenu* risponde al messaggio *WM\_INITMENU*, mandato da Windows prima di visualizzare il menu principale, per aggiungere un *checkmark* (una specie di «V») accanto alle opzioni del menu preposte all'attivazione o meno del ribbon e della riga di stato, se l'uno o l'altra è presente e visibile. I metodi *CMToggleRibbon* e *CMToggleStatusBar* rispondono alla scelta di quelle opzioni: scegliendole una volta si rende invisibile il ribbon o la riga di stato e si fa scomparire il relativo *checkmark*, scegliendole ancora si ottiene l'effetto contrario, e così via.

La variabile *SbText*, come quella omonima presente in *TNewMDIChild*, è destinata a tenere memoria dell'eventuale testo da visualizzare nel campo *MainText* della riga di stato, quando non vi vengono mostrate le descrizioni delle opzioni dei menu. Del pari analogo all'omonimo di quella classe è il metodo *UMPaintStatusBar*. Il metodo *SetupWindow* si incarica di mostrare *SbText* in *MainText* quando il programma inizia l'esecuzione.

Vedremo tra breve l'uso delle variabili *HelpCode* e *DispHelp*, alla cui assegnazione provvedono i metodi *WMMenuSelect* sia della frame window che delle child window, queste ultime tramite il messaggio *um\_SetHelpCode* e il corrispondente metodo *UMSetHelpCode* della frame window.

Sofferamoci intanto sui metodi *InitClientWindow* e *WMSize*. Il primo viene ridefinito per assegnare alla frame window una client window che sia istanza di *TNewMDIClient*, il secondo risponde al messaggio *WM\_SIZE*, inviato da Windows ogni volta che viene cambiata la dimensione della frame window. Si tratta, in questo caso, di cambiare coerentemente le dimensioni della client window, tenendo conto dell'eventuale presenza di un ribbon o di una riga di stato. Le nuove dimensioni della frame window, che Windows manda in *Msg.LParam*, vengono memorizzate nei campi *h* e *w* della variabile *Attr* (ereditata da *TWindow*), per poter poi attivare il metodo con un messaggio *WM\_SIZE* inviato dai metodi *CMToggleRibbon* e *CMToggleStatusBar*.

## Il messaggio WM\_MENUSELECT

Ci rimane da vedere come visualizzare sulla riga di stato sintetiche descrizioni delle opzioni dei menu, mentre l'utente le percorre con la tastiera o con il mouse.

Clickando con il mouse o premendo il tasto Enter su un'opzione, si provoca l'invio da parte di Windows di un messaggio *WM\_COMMAND*, normalmente intercettato dai metodi dinamici la cui numerazione abbia come base la costante *cm\_First*. Prima ancora, tuttavia, quando l'utente si posiziona su un'opzione, Windows manda il messaggio *WM\_MENUSELECT* alla finestra cui il menu appartiene. In *Msg.LParam* viene posta una combinazione dei valori riprodotti

```
implementation

constructor TNewMDIFrame.Init(ATitle: PChar; AMenu: HMenu);
begin
  TMDIWindow.Init(ATitle, AMenu);
  ActiveChild := nil;
  Ribbon := nil;
  StatusBar := nil;
  HelpCode := 0;
  DispHelp := False;
  SbText := '';
end;

procedure TNewMDIFrame.InitClientWindow;
begin
  ClientWnd := New(PNewMDIClient, Init(@Self));
end;

procedure TNewMDIFrame.SetupWindow;
begin
  TMDIWindow.SetupWindow;
  if (StatusBar <> nil) and StatusBar^.Visible then
    StatusBar^.PaintField(idf_MainText, SbText);
end;

procedure TNewMDIFrame.WMSize(var Msg: TMessage);
var R: TRect;
begin
  Attr.w := Msg.LParamLo;
  Attr.h := Msg.LParamHi;
  R.Left := 0;
  R.Top := 0;
  if (Ribbon <> nil) and Ribbon^.Visible then
    Inc(R.Top, Ribbon^.Height);
  R.Right := Msg.LParamLo;
  R.Bottom := Msg.LParamHi - R.Top;
  if (StatusBar <> nil) and StatusBar^.Visible then
    Dec(R.Bottom, StatusBar^.Height);
  MoveWindow(ClientWnd^.HWindow, 0, R.Top, R.Right, R.Bottom, True);
  if (Ribbon <> nil) and Ribbon^.Visible then
    MoveWindow(Ribbon^.HWindow, 0, 0, R.Right, Ribbon^.Height, True);
  if (StatusBar <> nil) and StatusBar^.Visible then begin
    MoveWindow(StatusBar^.HWindow, 0, Msg.LParamHi - StatusBar^.Height,
      R.Right, Msg.LParamHi, True);
    InvalidateRect(StatusBar^.HWindow, nil, True);
  end;
end;

procedure TNewMDIFrame.UMSetActiveChild(var Msg: TMessage);
var
  Enabled: Boolean;
begin
  ActiveChild := Pointer(Msg.LParam);
  if Ribbon <> nil then begin
    Enabled := ActiveChild <> nil;
    EnableWindow(Ribbon^.HWindow, Enabled);
  end;
end;

procedure TNewMDIFrame.WMMenuSelect(var Msg: TMessage);
var
  Menu: HMenu;
begin
  HelpCode := 0;
  DispHelp := True;
  if Msg.LParam = $0000FFFF then begin (* abbandonato il menu' *)
    if ActiveChild <> nil then with ActiveChild^ do
      SendMessage(HWindow, um_PaintStatusBar, idf_MainText, 0)
    else
      SendMessage(HWindow, um_PaintStatusBar, idf_MainText, 0);
    Exit;
  end;
  if (Msg.LParamLo and mf_Disabled) <> 0 then (* spazio "vuoto" *)
    Exit;
  case Msg.LParamLo and (mf_Popup or mf_SystemMenu) of
    0: (* Opzione di un menu', ma non del system menu *)
      if ActiveChild <> nil then begin
        (* se e' aperta una child window *)
        if IsZoomed(ActiveChild^.HWindow) then
          (* se e' massimizzata, siamo sul suo system menu? *)
            if GetMenuState(GetSubMenu(GetMenu(HWindow), 0),
              Msg.WParam, mf_ByCommand) <> $FFFF then
              Msg.LParam := Msg.LParam or mf_SystemMenu;
          with Msg do (* che se ne occupi comunque la child window *)
            SendMessage(ActiveChild^.HWindow, Message, WParam, LParam);
          end
        else
          HelpCode := ids_MenuItem + Msg.WParam;
      mf_Popup: (* un menu' pop-up (un'opzione del menu' principale) *)
      if ActiveChild <> nil then begin
        if IsZoomed(ActiveChild^.HWindow) then
          (* il system menu di una child window? *)
            if Msg.WParam = GetSubMenu(GetMenu(HWindow), 0) then
              Msg.WParam := Msg.WParam or mf_SystemMenu;
          with Msg do
```

```

        SendMessage(ActiveChild^.HWindow, Message, WParam, LParam);
    end
    else begin (* cerca il numero d'ordine del menu' *)
        Menu := GetMenu(HWindow);
        HelpCode := GetMenuItemCount(Menu);
        while (HelpCode > 0)
            and (GetSubMenu(Menu, HelpCode) <> Msg.WParam) do
            Dec(HelpCode);
            HelpCode := HelpCode + ids_PopupMenu + 1;
        end;
        mf_SysMenu: (* una opzione del system menu dell'applicazione *)
        HelpCode := ids_MenuItem + ((Msg.WParam and $0FFF) shr 4);
        mf_Popup or mf_SysMenu: (* il system menu dell'applicazione *)
        HelpCode := ids_PopupMenu;
    end;
end;
procedure TNewMDIFrame.WMEnterIdle(var Msg: TMessage);
const
    Buffer: array[0..100] of Char = '';
begin
    if Msg.WParam <> msgf_Menu then Exit;
    if not DispHelp then Exit;
    if HelpCode = 0 then
        Buffer[0] := #0
    else
        (*$IFDEF DEMO*)
        begin
            WVSprintf(Buffer, '%5d: ', HelpCode);
            LoadString(HInstance, HelpCode, Buffer+7, SizeOf(Buffer)-7);
        end;
        (*$ELSE*)
        LoadString(HInstance, HelpCode, Buffer, SizeOf(Buffer));
        (*$ENDIF*)
        SendMessage(HWindow, um_PaintStatusBar, idf_MainText,
            Longint(@Buffer));
        DispHelp := False;
    end;
end;
procedure TNewMDIFrame.UMSetHelpCode(var Msg: TMessage);
begin
    HelpCode := Msg.WParam;
    DispHelp := True;
end;
procedure TNewMDIFrame.UMPaintStatusBar(var Msg: TMessage);
begin
    if (StatusBar <> nil) and StatusBar^.Visible then begin
        if (Msg.LParam = 0) and (Msg.WParam = idf_MainText) then
            Msg.LParam := Longint(SBText);
        StatusBar^.PaintField(Msg.WParam, PChar(Msg.LParam));
    end;
end;
procedure TNewMDIFrame.CMToggleRibbon(var Msg: TMessage);
begin
    if Ribbon = nil then Exit;
    Ribbon^.Visible := not Ribbon^.Visible;
    if Ribbon^.Visible then
        CheckMenuItem(GetMenu(HWindow), cm_ToggleRibbon, mf_Checked)
    else
        CheckMenuItem(GetMenu(HWindow), cm_ToggleRibbon, mf_UnChecked);
    SendMessage(HWindow, wm_Size, SizeNormal,
        Longint(Attr.h) shl 16 + Attr.w);
end;
procedure TNewMDIFrame.CMToggleStatusBar(var Msg: TMessage);
begin
    if StatusBar = nil then Exit;
    StatusBar^.Visible := not StatusBar^.Visible;
    if StatusBar^.Visible then
        CheckMenuItem(GetMenu(HWindow), cm_ToggleStatusBar, mf_Checked)
    else
        CheckMenuItem(GetMenu(HWindow), cm_ToggleStatusBar, mf_UnChecked);
    SendMessage(HWindow, wm_Size, SizeNormal,
        Longint(Attr.h) shl 16 + Attr.w);
end;
procedure TNewMDIFrame.WMInitMenu(var Msg: TMessage);
begin
    if (Ribbon <> nil) and Ribbon^.Visible then
        CheckMenuItem(GetMenu(HWindow), cm_ToggleRibbon, mf_Checked)
    else
        CheckMenuItem(GetMenu(HWindow), cm_ToggleRibbon, mf_UnChecked);
    if (StatusBar <> nil) and StatusBar^.Visible then
        CheckMenuItem(GetMenu(HWindow), cm_ToggleStatusBar, mf_Checked)
    else
        CheckMenuItem(GetMenu(HWindow), cm_ToggleStatusBar, mf_UnChecked);
end;
end.

```

Figura 1 - La unit MDIFRAME, che definisce e implementa la classe base per la frame window di un'applicazione MDI.

nella figura 3, in *Msg.WParam* un numero che identifica l'opzione del menu; più precisamente, se si sta su un'opzione che attiva un menu pop-up, troviamo in *Msg.WParam* l'handle del menu, se si sta sull'opzione di un menu pop-up, vi troviamo la costante che identifica l'opzione.

Il metodo *WMMMenuSelect* della frame window inizializza per prima cosa *HelpCode* con uno zero e *DispHelp* con *True*. Verifica quindi se *Msg.LParam* contiene \$0000FFFF, costante usata da Windows nel caso si abbandonano un menu con *Esc* o cliccando altrove col mouse; in questo caso si provoca il ripristino della riga di stato da parte della frame window o della eventuale child attiva, mediante invio del messaggio *um\_PaintStatusBar* con uno zero in *LParam*.

Si vede poi se nel campo *LParam* del messaggio *WM\_MENUSELECT* non sia setato il bit corrispondente alla costante *MF\_DISABLED*. Stando alla documentazione (mi riferisco in primo luogo ai manuali dell'SDK di Windows), questa viene usata per elementi disabilitati di un menu; non tanto per opzioni non attivabili, le quali vengono di norma visualizzate con una diversa colorazione (si usa la costante *MF\_GRAYED*), quanto piuttosto per cose come il titolo di un gruppo di opzioni. Ho potuto verificare, smanettando un po', che quel bit viene setato sia quando ci si posiziona su una linea di separazione, sia quando ci si sposta con il mouse, mantenendo premuto il pulsante, fuori del menu. In questi casi si esce subito: avendo assegnato uno zero a *HelpCode*, ne risulterà un *MainText* vuoto nella riga di stato (per inciso: le unit che vi sto proponendo derivano in buona parte da un lavoro di conversione alla OOP di tecniche proposte da Jeffrey M. Richter nel suo *Windows 3: A Developer's Guide*, edito dalla M&T Books; ho dovuto riorganizzare in misura non trascurabile i suoi spunti, per ottenere quella flessibilità di cui vi dicevo il mese scorso e che solo la OOP può consentire; la mancata verifica di quel bit da parte di Richter rappresenta comunque l'unico neo in un lavoro di ottima fattura... nonostante si usino il C e la programmazione tradizionale).

## Capire dove siamo

Una volta completati i preliminari, si tratta di capire qual è l'opzione su cui si trova l'utente, esaminando se sono settati o no i bit corrispondenti alle costanti *MF\_POPUP* e *MF\_SYSMENU*. Si potrà quindi pervenire all'assegnazione alla variabile *HelpCode* del valore opportuno, assumendo come base una delle quattro costanti definite nella unit *STRIPES*: *ids\_PopupMenu* per i menu pop-up del menu della frame window, *ids\_MenuItem* per le loro opzioni, *ids\_ChildPopupMenu* per i menu pop-up di un menu di una child window, *ids\_ChildMenuItem* per le opzioni di tali menu.

Se nessuno dei due bit è settato, se ne può dedurre che si tratta dell'opzione di un menu, ma non del *system menu* dell'applicazione.

```

procedure TNewMDIChild.WMMenuSelect(var Msg: TMessage);
var
  HelpCode: Word;
  Menu: HMENU;
begin
  if Msg.LParam = $0000FFFF then begin (* abbandonato il menu' *)
    SendMessage(HWindow, um_PaintStatusBar, idf_MainText, 0);
    Exit;
  end;
  if (Msg.LParamLo and mf_Disabled) <> 0 then begin
    (* spazio "vuoto" *)
    SendMessage(Parent^.HWindow, um_SetHelpCode, 0, 0);
    Exit;
  end;
  case Msg.LParamLo and (mf_Popup or mf_SysMenu) of
    0: (* Opzione di un menu', ma non del system menu della child *)
      begin
        if (Msg.WParam >= id_FirstMDIChild)
          and (Msg.WParam <= id_FirstMDIChild + 8) then
          (* se siamo sul titolo di una child window *)
            Msg.WParam := cm_ActivateChild
          else if Msg.WParam = id_FirstMDIChild + 9 then
          (* se siamo sull'opzione "More Windows..." *)
            Msg.WParam := cm_ChildList;
            HelpCode := ids_ChildMenuItem + Msg.WParam;
          end;
        mf_Popup: (* un menu' pop-up (un'opzione del menu' principale) *)
          begin
            Menu := GetMenu(Parent^.HWindow);
            HelpCode := GetMenuItemCount(Menu);
            while (HelpCode > 0)
              and (GetSubMenu(Menu, HelpCode) <> Msg.WParam) do
                Dec(HelpCode);
            HelpCode := HelpCode + ids_ChildPopupMenu;
            if not IsZoomed(HWindow) then
              Inc(HelpCode);
            end;
            mf_SysMenu: (* una opzione del system menu della child window *)
              HelpCode := ids_ChildMenuItem + ((Msg.WParam and $0FFF) shr 4);
            mf_Popup or mf_SysMenu: (* il system menu della child window *)
              HelpCode := ids_ChildPopupMenu;
            end;
            SendMessage(Parent^.HWindow, um_SetHelpCode, HelpCode, 0);
          end;
        procedure TNewMDIChild.WMEnterIdle(var Msg: TMessage);
        begin
          with Msg do
            SendMessage(Parent^.HWindow, Message, WParam, LParam);
          end;
      end;
  end;
end;

```

Figura 2 - I metodi WMMenuSelect e WMEnterIdle della unit MDIChild.

cazione. Se non è attiva alcuna child window, si assegna ad *HelpCode* la somma di *ids\_MenuItem* e di *Msg.WParam*, altrimenti si gira il messaggio WM\_MENUSELECT alla child window attiva. In questo caso, si deve però prima verificare se la finestra attiva è massimizzata, in quanto ciò comporta che il suo *system menu* è diventato il primo menu pop-up del menu principale, circostanza che viene resa nota alla child window settando convenzionalmente in *Msg.LParam* il bit corrispondente alla costante WM\_SYSMENU.

Se è settato il bit di MF\_POPUP, si tratta di un menu pop-up. Si procede in modo analogo: se è attiva una child window le si

gira il messaggio, dopo aver verificato, se è massimizzata, se *Msg.WParam* è uguale all'handle del suo *system menu*; in caso contrario, si scandisce il menu principale per trovare il numero d'ordine del menu pop-up che ha un handle uguale a *Msg.WParam*, assegnando a *HelpCode* questo numero più *ids\_PopupMenu*, più 1 (per distinguerlo dal *system menu* dell'applicazione).

Se è settato il bit di MF\_SYSMENU, si tratta di un'opzione del *system menu* della frame window. I codici dei comandi corrispondenti alle opzioni di tale menu sono numeri che, in esadecimale, rispettano un curioso schema: la prima cifra è F, la quarta

MF_BITMAP	\$0004	l'opzione è una Bitmap
MF_CHECKED	\$0008	l'opzione ha un <i>checkmark</i>
MF_DISABLED	\$0002	l'opzione è disabilitata
MF_GRAYED	\$0001	l'opzione è "in grigio"
MF_MOUSESELECT	\$8000	si sta usando il mouse
MF_OWNERDRAW	\$0100	l'opzione è owner-draw
MF_POPUP	\$0010	si tratta di un menù pop-up
MF_SYSMENU	\$2000	l'opzione appartiene al system menu

Figura 3 - Le costanti usate nella word bassa del campo LParam del messaggio WM\_MENUSELECT. Si può notare che, trattandosi di potenze di due, in ogni costante è settato un solo bit.

è sempre zero, la seconda e la terza variano da \$00 a \$13. Può convenire quindi estrarre il numero composto dalla seconda e dalla terza cifra, per ottenere un numero «piccolo» da sommare a *ids\_MenuItem*.

Se infine sono settati ambedue i bit, l'utente si è posizionato sul *system menu* della frame window e non l'ha ancora aperto. Si assegna quindi a *HelpCode* la sola costante *ids\_PopupMenu*.

Analoga la struttura del metodo *WMSelectMenu* di una child window (figura 2), in cui peraltro si assegnano valori alle variabili *HelpCode* e *DispHelp* solo indirettamente, mediante il messaggio *um\_SetHelpCode*. Sottolineo per il resto solo la particolarità del caso non sia settato nessuno di quei due bit. Si tratta qui di capire se l'utente si è posizionato su un'opzione normale di un menu pop-up, oppure sui titoli delle child window aperte, riportati in calce a quello che altre volte abbiamo chiamato «menu Finestre». Ci viene in aiuto la circostanza che i codici dei comandi corrispondenti ai titoli vanno da *id\_FirstMDIChild* a *id\_FirstMDIChild + 8*, essendo *id\_FirstMDIChild* una costante pari a \$F01 definita nella unit WOBJECTS. Il valore *id\_FirstMDIChild + 9* corrisponde invece all'opzione «More Windows» che compare, se vi sono più di nove finestre aperte, dopo il titolo della nona. In tali situazioni, vengono usate due costanti di comodo (*cm\_ActivateChild* e *cm\_ChildList*) per l'assegnazione a *HelpCode*.

## Il messaggio WM\_ENTERIDLE

Windows manda il messaggio WM\_ENTERIDLE quando, entrato in un menu o in una dialog box modale, l'utente si ferma, non provocando la generazione di altri messaggi oltre quelli già processati. La ricezione di quel messaggio rappresenta quindi il momento migliore per agire sulla riga di stato.

Il metodo *WMEnterIdle* della frame window termina subito se *Msg.WParam* non è uguale a MSGF\_MENU (vorrebbe dire che siamo su una dialog box) o se *DispHelp* vale False. Azzerava poi la stringa *Buffer* se *HelpCode* è uguale a zero, altrimenti vi carica dal file di risorse la stringa corrispondente al valore di *HelpCode*. Manda infine il messaggio *um\_PaintStatusBar* e assegna False a *DispHelp* (per evitare la continua esecuzione del metodo fino a quando l'utente non si posiziona su un'altra opzione, o ne sceglie una, o esca dal menu).

Il metodo *WMEnterIdle* delle child window non fa altro che girare il messaggio alla frame window.

Con ciò abbiamo terminato l'esposizione delle unit. Le vedremo all'opera tra trenta giorni. MS

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

# FLOPPERIA

Viale Monte Nero, 15 • 20135 Milano • ☎ (02) 55.18.04.84 r.a. • Fax (02) 55.18.81.05 (24 ore)  
 Negozio aperto al pubblico dalle 10 alle 13 e dalle 15 alle 19. Vendita anche per corrispondenza.

## Personal Computer EuroSys



Venduti in configurazione base (senza monitor), i nostri PC sono CONFIGURABILI SU MISURA, ovvero in base alle vostre preferenze, e sono coperti da garanzia totale per 12 mesi. Scegliete VOI il tipo di monitor, a colori o monocromatico, la scheda grafica che preferite; la capacità dell'hard disk (tutti velocissimi, con tempo d'accesso inferiore a 24 ms e velocità di trasferimento dati superiore ai 700KB/sec.); la quantità di memoria Ram, e così via.

Ecco alcuni esempi di configurazione:

Modello base 286-20	670.000	Modello base 386-25	1.140.000
Mouse	30.000	Mouse	30.000
Hard disk 40 MB	340.000	Hard disk 40 MB	340.000
Monitor VGA mono	220.000	Monitor colori 1024x768	595.000
<b>Totale IVA compresa</b>	<b>1.260.000</b>	<b>Totale IVA compresa</b>	<b>2.105.000</b>

### I modelli base EuroSys

I modelli base EuroSys sono composti dalla scheda CPU di vostra scelta e dalle seguenti parti: cabinet desktop "baby-size" con alimentatore switching da 200 Watt; 1 MB Ram espandibile su piastra, opzione ShadowRam per velocizzare Bios e grafica, 1 disk drive 3"1/2 da 1,44 MB, controller IDE AT-bus per 2 floppy più 2 hard disk; scheda video VGA 800x600; tastiera estesa 101 tasti (a scelta italiana o USA); scheda multi I/O con 2 porte seriali, 1 porta parallela, 1 interfaccia game.

**286/20 EuroSys 20 MHz** Landmark: 26 MHz **670.000**  
 CPU Intel 80286 16 bit • Memoria espandibile a 2 o 4 MB  
 0 wait states • Zoccolo per coprocessore opzionale 80287

**386/20-SX EuroSys 20 MHz** Landmark: 26 MHz **950.000**  
 CPU Intel 80386/SX 16/32 bit • Memoria espandibile a 2, 4 o 8 MB  
 0 wait states • Zoccolo per coprocessore opzionale 80387-SX

**386/25 EuroSys 25 MHz** Landmark: 33 MHz **1.140.000**  
**386/33-C EuroSys 33 MHz cache** Landmark: 56 MHz **1.390.000**  
 CPU Intel 80386 32 bit • Memoria espandibile a 4 o 8 MB  
 0 wait states • Zoccolo per coprocessore opzionale 80387

**486/33-C EuroSys 33 MHz cache** Landmark: 167 MHz **2.250.000**  
 CPU Intel 80486 32 bit • Memoria espandibile a 4, 8, 12 o 16 MB  
 0 wait states • Coprocessore 80487 presente

### Configurazioni su misura

#### Parte aggiuntiva

Parte aggiuntiva	Costo aggiuntivo
• Secondo disk drive 5"1/4 da 1,2 MB	<b>125.000</b>
• Mouse Genius	<b>30.000</b>
• SuperVGA 1024x768 256 colori • 1 MB Ram video Chip Tseng ET-4000	<b>160.000</b>
• SuperVGA 1024x768 256 colori • 1 MB Ram video Chip Trident T-8900-C	<b>100.000</b>
• H-VGA 1024x768 256 colori, 800x600 32.768 colori Chip Tseng	<b>285.000</b>
• Hard disk 40 MB Seagate 24 ms	<b>340.000</b>
• Hard disk 85 MB Seagate 18 ms	<b>570.000</b>
• Hard disk 125 MB Seagate 18 ms	<b>790.000</b>
• Hard disk 210 MB Western Digital 14 ms	<b>1.170.000</b>
• Backup tape streamer 250 MB	<b>1.100.000</b>
• Controller IDE High Speed cache 4 MB Ram (0,5 ms)	<b>990.000</b>
• Cabinet deluxe	<b>50.000</b>
• Cabinet minitower 4 posizioni	<b>90.000</b>
• Cabinet tower 6 posizioni	<b>185.000</b>
• Per ogni 1 MB di Ram aggiuntivo	<b>95.000</b>

Sistema operativo MS-DOS versione 5.0 in italiano	<b>160.000</b>
Crystal 14" Color VGA 800x600	<b>495.000</b>
Crystal 14" Color MultiScan 1024x768	<b>595.000</b>
Monitor a colori MultiScan • Pitch 0.28 • Con base basculante e cavo.	
Crystal 14" VGA 640x480	<b>220.000</b>
Crystal 14" MultiScan 1024x768	<b>255.000</b>
Monitor monocromatico f/bianchi • Schermo piatto • In versione duale, VGA o MultiScan.	



## Computer Commodore Amiga

**Amiga 500 Plus • 1 MB Ram • Kickstart/Workbench 2.0 • Chip ECS** **790.000**

**Amiga 500 Plus • 2 MB Ram • Kickstart/Workbench 2.0 • Chip ECS** **890.000**

Con giochi omaggio, joystick, garanzia Commodore.

**Amiga 2000** **1.350.000**

**Amiga 2000 • HD Supra 45 MB** **1.950.000**

**Amiga 2000 • HD Supra 105 MB** **2.350.000**

Con giochi omaggio, joystick, garanzia Commodore.

**Amiga 3000 25 MHz • HD 50 MB • 2 MB Chip • 2 MB Fast** **4.660.000**

**Amiga 3000 25 MHz • HD 100 MB • 2 MB Chip • 2 MB Fast** **5.290.000**

**Amiga 3000 Tower 25 MHz • HD 100 MB • 2 MB Chip • 4 MB Fast** **5.750.000**

**Amiga 3000 Tower 25 MHz • HD 200 MB • 2 MB Chip • 4 MB Fast** **6.750.000**

## CDTV

**Commodore CDTV** **1.195.000**

**Tastiera CDTV** **135.000**

**DISPONIBILI MIGLIAIA  
 DI PRODOTTI.**

**RICHIEDETE IL  
 NOSTRO CATALOGO  
 GRATUITO!**