La frame window

Il mese scorso abbiamo preso atto della necessità di disporre di nuove classi per le applicazioni MDI, in modo da poter ottenere una gestione automatica della visualizzazione di barre strumenti e righe di stato e, nel caso di quest'ultima, di informazioni relative allo stato della finestra attiva, oppure di descrizioni sintetiche delle opzioni dei menu, o anche di messaggi per l'utente (del tipo «Salvataggio in corso...»). Abbiamo anche esaminato le classi per la client window *e per le* child window*; vedremo ora come attrezzarci per la* frame window

di Sergio Polini

Nell'esaminare la classe *TNewMDIChild*, ci siamo soffermati in particolare sul meccanismo un po' complicato attraverso il quale è possibile individuare il menu, o l'opzione di un menu, su cui sia posizionato l'utente, al fine di visualizzare una sua descrizione sintetica nella riga di stato. Abbiamo visto che, prima che l'utente scelga un menu o un'opzione con un click del mouse o con la pressione del tasto di Invio, Windows manda all'applicazione il messaggio WM_MENUSELECT che, intercettato dalla *frame window*, viene da questa girato (nel nostro schema) alla *child window* eventualmente attiva, per riceverne indietro il messaggio *um_SetHelpCode*, oppure *um_PaintStatusLine* quando l'utente abbandona il menu.

Ci siamo imbattuti, inoltre, in altre situazioni in cui si deve avere uno scambio di messaggi tra *frame window*, *client window* e *child window:* quando una *child window* diventa finestra attiva, invia alla *frame window* i messaggi *um_SetActiveChild* e *um_PaintStatusLine;* quando una *child window* viene chiusa, prima di sparire manda il messaggio *um_ChildDestroy* alla *client window*, che risponde inviando alla *frame window* il messaggio *um_SetActiveChild* per informarla circa l'eventuale assenza di altre *child window*; quando l'utente abbandona il *system menu*, la *frame window* invia alla *child window* eventualmente attiva il messaggio *um_PaintStatusLine*, per riceverne indietro lo stesso messaggio con l'indirizzo della variabile *Fields* e poter quindi ripristinare il contenuto che la riga di stato mostrava prima che la si usasse per illustrare le opzioni dei menu.

Abbiamo quindi bisogno di una classe *TNewMDIFrame*. La sua interfaccia (figura 1) non è delle più semplici e, quindi, conviene distribuire variabili e metodi in quattro gruppi: creazione della *frame window* e determinazione del suo assetto (figura 2), gestione delle *child window* (figura 3), visualizzazione nella riga di stato di messaggi, compresi quelli esplicativi delle opzioni dei menu (figura 4), gestione di barre strumenti e riga di stato (che dobbiamo rinviare al mese prossimo).

Creazione della frame window *e suo assetto*

Il constructor *Init* aggiunge WS_CLIPCHILDREN agli stili della finestra, per evitare che venga ridisegnata la parte occupata da finestre «figlie» come barre strumenti o riga di stato; provvede quindi ad inizializzare le variabili d'istanza: *ActiveChild* terrà memoria della *child window* attiva, *Toolbar* e *StatusLine* potranno essere utilizzate per dotare l'applicazione di una barra strumenti e di una riga di stato, *Fields* per definire i campi della riga di stato; *HelpCode* e *DispHelp* verranno utilizzate per mostrare messaggi sulla riga di stato.

Il destructor, prima di chiamare quello ereditato da *TMDIWindow*, ripristina il menu della *frame window*. Per comprendere appieno il motivo di ciò, sarà necessario vedere come utilizzare la classe in un'applicazione. Per il momento, anticipiamo che il problema è quello della distruzione dei menu utilizzati dalle *child window*, che possono essere più d'uno e diversi dal menu della *frame window* (quello che appare quando non vi sono *child window* aperte); ad esempio, il menu della *frame window* potrebbe contenere i soli menu pop-up *File* e *Guida*, il menu di una finestra di editing potrebbe contenere opzioni come *Trova, Sostituisci, Carattere*, ecc., il menu di una finestra grafica potrebbe contenere opzioni per delimitare le coordinate o per scegliere il *mapmode*, e così via.

I menu delle *child window* non possono essere agevolmente gestiti dalla classe *TNewMDIChild*; almeno, dopo aver tentato numerose soluzioni, ho preferito lasciar gestire quei menu dalla *frame window* o, meglio, dalla classe derivata da *TNewMDIFrame* che comparirà nell'applicazione: questa inizializzerà i vari menu nel suo constructor e li distruggerà nel suo destructor. Perché questo avvenga senza problemi, però, è necessario assicurarsi che nessuno di quei menu sia attivo, in quanto, in caso contrario, si cercherebbe di distruggerlo due volte (una ad opera del destructor ereditato, una seconda ad opera del destructor ridefinito). Ecco il motivo per cui si riattiva il menu della *frame window* prima di chiamare il destructor ereditato da *TMDIWindow*.

Il metodo *InitClientWindow* sostituisce quello ereditato, in modo da garantire che la *client window* sia un'istanza di *TNewMDIClient*.

Il metodo *WMSize*, dopo aver provveduto ad aggiornare i campi della variabile *Attr*, chiama il metodo *RedoClientRect*.

Scopo di questo è la riduzione della *client area* al fine di contenere le dimensioni della *client window* e consentire la visualizzazione di barre strumenti e riga di stato. A questo scopo, invia il messaggio *am_CalcParentClientRect* a tutte

le finestre «figlie» della *frame window* inviando le dimensioni della *client area* perché ognuna le modifichi in modo da crearsi un proprio spazio, secondo un meccanismo che abbiamo già discusso trattando delle unit TOOLBAR e STATUSLINE.

Aggiungo al primo gruppo di metodi anche *CMTileVertically*. A mio parere, il comando *Tile (Affianca)* che si trova in ogni applicazione MDI è poco utile, in quanto, se vi sono due finestre attive, le dispone l'una a destra e l'altra a sinistra, riducendo l'ampiezza di ognuna. Mi trovo più a mio agio con una disposizione delle finestre l'una sopra l'altra, con riduzione dell'altezza invece che della larghezza.

L'affiancamento in verticale è ottenibile semplicemente usando un *WParam* pari a uno, invece che a zero, nel messaggio WM_MDITILE.

Gestione delle child window

Come abbiamo già visto, quando una *child window* viene chiusa la *client window* invia il messaggio *um_SetActiveChild* con parametri nulli. Quando una *Child window* diventa attiva, invia lo stesso messaggio usando il proprio indirizzo (nella forma *Longint(@Self)*) ed il messaggio *um_PaintStatusLine* con parametri *sl_Child* e l'indirizzo della propria variabile *Fields*.

Il metodo *UMSetActiveChild* assegna il valore di *Msg.Lparam* alla variabile d'istanza *ActiveChild*; se quel valore contiene l'indirizzo della finestra attiva,

Figura 2. I metodi che presiedono alla creazione della frame window *ed alla determinazione del suo assetto.* ▶

```
implementation

uses
  Strings;

constructor TNewMDIFrame.Init(ATitle: PChar; AMenu: HMenu);
begin
  inherited Init(ATitle, AMenu);
  Attr.Style := Attr.Style or ws_ClipChildren;
  ActiveChild := nil;
  Toolbar := nil;
  StatusLine := nil;
  Fields := nil;
  HelpCode := 0;
  DispHelp := False;
end;

destructor TNewMDIFrame.Done;
begin
  SendMessage(ClientWnd^.HWindow, wm_MDISetMenu, 0,
              MakeLong(Attr.Menu, GetSubMenu(Attr.Menu, 0)));
  inherited Done;
end;

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

procedure TNewMDIFrame.RedoClientRect;
var
  R, Temp: TRect;
procedure NotifyChildren( P: PWindow ); far;
  begin
     if P^.HWindow <> 0 then
      SendMessage(P^.HWindow, am_CalcParentClientRect,
                  AllowRepaint, Longint(@R));
  end;
begin
  GetClientRect(HWindow, R);
  Temp := R;
  ForEach(@NotifyChildren);
  if R.Bottom < R.Top then begin
    Inc(Temp.Bottom, R.Top - R.Bottom);
    R := Temp;
    ForEach(@NotifyChildren);
  end;
  SetWindowPos(ClientWnd^.HWindow, 0, R.Left, R.Top,
                                      R.Right - R.Left,
                                      R.Bottom - R.Top, swp_NoZOrder);
end;
procedure TNewMDIFrame.WMSize(var Msg: TMessage);
var
  R: TRect;
begin
  if (Scroller <> nil) and (Msg.WParam <> sizeIconic) then
    Scroller^.SetPageSize;
  if Msg.wParam = sizeNormal then
  begin
    GetWindowRect(HWindow, R);
    Attr.H := R.bottom - R.top;
    Attr.W := R.right - R.left;
  end;
  RedoClientRect;
end;

procedure TNewMDIFrame.CMTileVertically(var Msg: TMessage);
begin
  SendMessage(ClientWnd^.HWindow, wm_MDITile, 1, 0);
end;
```

Figura 1. L'interfaccia della unit MDIFRAME. ▼

```
unit MDIFrame;

interface

uses
  WinTypes, WinProcs, Objects, OWindows,
  MDIConst, MDIClnt, Toolbar, StatLine;

type
  PNewMDIFrame = ^TNewMDIFrame;
  TNewMDIFrame = object(TMDIWindow)
    ActiveChild: PWindow;
    Toolbar: PToolbar;
    StatusLine: PStatusLine;
    Fields: PCollection;
    constructor Init(ATitle: PChar; AMenu: HMenu);
    destructor Done; virtual;
    procedure WMSize(var Msg: TMessage);
      virtual wm_First + wm_Size;
    procedure SwitchToolbar(NewName: PChar);
    procedure UMSetActiveChild(var Msg: TMessage);
      virtual wm_First + um_SetActiveChild;
  private
    HelpCode: Word;
    DispHelp: Boolean;
    procedure InitClientWindow; virtual;
    procedure RedoClientRect;
    procedure UMPaintStatusLine(var Msg: TMessage);
      virtual wm_First + um_PaintStatusLine;
    procedure WMMenuSelect(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 CMToolbar(var Msg: TMessage);
      virtual cm_First + cm_Toolbar;
    procedure CMStatusLine(var Msg: TMessage);
      virtual cm_First + cm_StatusLine;
    procedure OrientToolbar(Command: Word);
    procedure CMHorizontalToolbar(var Msg: TMessage);
      virtual cm_First + cm_HorizontalToolbar;
    procedure CMDownHorizontalToolbar(var Msg: TMessage);
      virtual cm_First + cm_DownHorizontalToolbar;
    procedure CMLeftVerticalToolbar(var Msg: TMessage);
      virtual cm_First + cm_LeftVerticalToolbar;
    procedure CMRightVerticalToolbar(var Msg: TMessage);
      virtual cm_First + cm_RightVerticalToolbar;
    procedure CMTileVertically(var Msg: TMessage);
      virtual cm_First + cm_TileVertically;
  end;
```

```
procedure TNewMDIFrame.SwitchToolbar(NewName: PChar);
begin
  if (Toolbar <> nil)
  and (StrComp(Toolbar^.GetResName, NewName) <> 0) then begin
    Toolbar^.Show(sw_Hide);
    Toolbar^.SwitchTo(NewName);
    RedoClientRect;
    Toolbar^.Show(sw_Show);
  end;
end;

procedure TNewMDIFrame.UMSetActiveChild(var Msg: TMessage);
var
  HFrameMenu, HWinMenu: HMenu;
begin
  ActiveChild := Pointer(Msg.LParam);
  if ActiveChild = nil then begin
    HFRameMenu := Attr.Menu;
    HWinMenu := GetSubMenu(HFrameMenu, 0);
    SendMessage(ClientWnd^.HWindow, wm_MDISetMenu, 0,
                MakeLong(HFrameMenu, HWinMenu));
    DrawMenuBar(HWindow);
    SendMessage(HWindow, um_PaintStatusLine, sl_Frame, 0);
  end;
end;
procedure TNewMDIFrame.UMPaintStatusLine(var Msg: TMessage);
const
  Buffer: array[0..100] of Char = ''#0;
begin
  if StatusLine <> nil then begin
    case Msg.WParam of
      sl_Frame: StatusLine^.SwitchTo(Fields);
      sl_Child: StatusLine^.SwitchTo(PCollection(Msg.LParam));
      sl_Plain : begin
                   if Msg.LParam = 0 then begin
                     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*)
                     StatusLine^.SetText(Buffer);
                     DispHelp := False;
                   end
                   else
                     StatusLine^.SetText(PChar(Msg.LParam));
                 end
    end;
  end;
end;
```

Figura 3. I metodi che intervengono quando si apre o si chiude una child window, *o quando cambia la* child window *attiva.*

la variabile potrà essere utilizzata dai metodi che si occupano della visualizzazione, sulla riga di stato, di descrizioni delle opzioni dei menu; se il valore è invece **nil** non c'è più alcuna finestra attiva e, quindi, la *frame window* ripristina i propri menu e invia a se stessa il messaggio *um_PaintStatusLine* con un *WParam* pari a *sl_Frame*.

Il metodo è dichiarato nella sezione «pubblica» dell'interfaccia della classe *TNewMDIFrame*, in quanto va ridefinito nella classe che verrà derivata da questa in un'applicazione concreta, al fine di accompagnare il passaggio da una finestra attiva ad un'altra, o ad una situazione in cui non vi siano finestre aperte, con l'attivazione della barra strumenti associata alla finestra attiva o alla *frame window*. Ciò potrà essere fatto mediante il metodo *SwitchToolbar*, che confronta il nome della nuova barra col nome di quella visualizzata e, se sono diversi, provvede a chiamare il metodo *SwitchTo* della barra con il nuovo nome.

Il metodo *UMPaintStatusLine* provvede ad aggiornare la riga di stato. Come abbiamo visto quando abbiamo esaminato la unit STATLINE, si usano tre costanti per tre possibili condizioni: *sl_Frame* denota la condizione in cui va mostrata l'articolazione in campi associata alla *frame window* e descritta nella sua variabile *Fields*; *sl_Child* indica che vi è una finestra attiva e, quindi, si adotta la variabile *Fields* di questa, il cui indirizzo viene trasmesso al metodo mediante il parametro *Msg.LParam*; *sl_Plain* viene usata quando si intende usare una riga di stato senza «campi» per comunicare qualcosa all'utente.

In quest'ultimo caso, il parametro *Msg.LParam* viene usato per passare l'indirizzo della stringa da visualizzare (ad esempio, durante operazioni complesse, la spiegazione del motivo per cui il cursore del mouse ha assunto la forma di una clessidra); il metodo *WMEnterIdle* usa invece un parametro nullo per indicare che si vuole mostrare la spiegazione di un'opzione di un menu, a cui corrisponde, come vedremo subito, un codice *HelpCode* attraverso il quale la stringa da visualizzare potrà essere letta dalla risorsa STRINGTABLE. Se *HelpCode* vale zero, peraltro, si visualizza una stringa nulla; si mostra, cioè, una riga di stato «bianca».

Descrizioni delle opzioni dei menu

Quando l'utente si posiziona su un menu, Windows invia il messaggio WM_MENUSELECT, che abbiamo già visto il mese scorso quando abbiamo esaminato come ad esso deve rispondere una *child window*. Il messaggio, in realtà, viene inviato alla *frame window* e da questa girato all'eventuale finestra attiva. Procediamo con ordine.

La *frame window* risponde al messaggio inizializzando a zero la variabile *HelpCode* e a *True* la variabile *DispHelp*.

Se il messaggio indica, con *LParam* pari a $0000FFFF, che l'utente ha abbandonato il menu (premendo Esc o clickando altrove col mouse), si esce dopo aver provveduto a ripristinare la riga di stato mediante il messaggio *um_PaintStatusLine*. In caso contrario, occorre esaminare se in *LParam* sono settati i bit corrispondenti alle costanti illustrate nella figura 3 del mese scorso.

Se è settato il bit MF_DISABLED, ciò indica che l'utente è posizionato sopra una zona «vuota» di un menu, ad esempio su una linea di separazione tra diversi gruppi di opzioni. In questo caso si esce immediatamente; il valore zero di *HelpCode* farà sì che venga mostrata una riga di stato «bianca».

Se non è settato nessuno dei bit MF_POPUP e MF_SYSMENU, l'utente si è posizionato sull'opzione di un menu, non compresa tra quelle del *system menu* dell'applicazione. Se non è aperta alcuna *child window*, si assegna ad *HelpCode* la somma di *ids_MenuItem* (dichiarata in MDICONST.PAS) e di *Msg.WParam* (che ha un valore pari alla costante che identifica il comando corrispondente all'opzione); altrimenti, si gira il messaggio alla finestra attiva. Pri-

Figura 4. I metodi che permettono la visualizzazione, sulla riga di stato, di sintetiche spiegazioni sulle opzioni dei menu.

```
procedure TNewMDIFrame.WMMenuSelect(var Msg: TMessage);
var
  M: HMenu;
begin
  HelpCode := 0;
  DispHelp := True;
  if Msg.LParam = $0000FFFF then begin  (* abbandonato il menu' *)
    if ActiveChild <> nil then
      SendMessage(ActiveChild^.HWindow, um_PaintStatusLine,
                  sl_Child, 0)
    else
      SendMessage(HWindow, um_PaintStatusLine, sl_Frame, 0);
    Exit;
  end;
  if (Msg.LParamLo and mf_Disabled) <> 0 then (* spazio "vuoto" *)
    Exit;
  case Msg.LParamLo and (mf_Popup or mf_SysMenu) of
    0: (* Opzione di un menu', ma non del system menu
          dell'applicazione *)
      if ActiveChild <> nil then begin
        if IsZoomed(ActiveChild^.HWindow) then
          (* opzione del system menu di una child window? *)
          if GetMenuState(GetSubMenu(GetMenu(HWindow), 0),
                          Msg.WParam, mf_ByCommand) <> $FFFF then
            Msg.LParam := Msg.LParam or mf_SysMenu;
        with Msg do
          (* lascia che se ne occupi la child window attiva *)
          SendMessage(ActiveChild^.HWindow,Message,WParam,LParam);
      end
      else
        HelpCode := ids_MenuItem + Msg.WParam;
    mf_Popup: (* un menu' pop-up, 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_SysMenu;
        with Msg do
          SendMessage(ActiveChild^.HWindow,Message,WParam,LParam);
      end
      else begin (* cerca il numero d'ordine del menu' *)
        M := GetMenu(HWindow);
        HelpCode := GetMenuItemCount(M);
        while (HelpCode > 0)
        and (GetSubMenu(M, HelpCode) <> Msg.WParam) do
          Dec(HelpCode);
        HelpCode := HelpCode + ids_PopupMenu + 1;
      end;
    mf_SysMenu: (* 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);
begin
  if Msg.WParam <> msgf_Menu then
    Exit;
  if not DispHelp then
    Exit;
  SendMessage(HWindow, um_PaintStatusLine, sl_Plain, 0);
end;

procedure TNewMDIFrame.UMSetHelpCode(var Msg: TMessage);
begin
  HelpCode := Msg.WParam;
  DispHelp := True;
end;
```

ma di ciò, tuttavia, occorre verificare se la finestra attiva è massimizzata, in quanto, in tal caso, il suo *system menu* è diventato il primo menu pop-up del menu principale, circostanza che viene resa nota alla *child window* settando convenzionalmente il bit corrispondente alla costante WM_SYSMENU. Abbiamo visto il mese scorso come il metodo *WMMenuSelect* di *TNewMDIChild* utilizza l'informazione.

Se è settato il bit 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* e poi assegnare a *HelpCode* questo numero più la costante *ids_PopupMenu*, più uno (per distinguerlo dal *system menu* dell'applicazione; il motivo dell'incremento di uno è stato già esposto con maggiore dettaglio il mese scorso).

Se è settato il solo bit MF_SYSMENU, si tratta di un'opzione del *system menu* dell'applicazione; si assegna quindi a *HelpCode* la somma di *ids_MenuItem* e di un valore estratto da *Msg.WParam* (anche questo aspetto è stato già illustrato nell'appuntamento di novembre).

Se, infine, sono settati entrambi i bit MF_POPUP e MF_SYSMENU, l'utente si è posizionato sul *system menu* dell'applicazione ma non l'ha ancora aperto; si assegna quindi a *HelpCode* la sola costante *ids_PopupMenu*.

Alla fine delle operazioni, *HelpCode* ha un valore utile per il metodo *UMPaintStatusLine*. Se il messaggio WM_MENUSELECT è stato girato ad una finestra attiva, questa, al termine del suo metodo *WMMenuSelect*, invia un valore per *HelpCode* mediante il messaggio *um_SetHelpCode*; il corrispondente metodo della *frame window* assegna *Msg.WParam* a *HelpCode* e *True* a *DispHelp*.

L'aggiornamento della riga di stato avviene quando l'applicazione riceve il messaggio WM_ENTERIDLE, inviato da Windows allorché l'utente, entrato in un menu o in una *dialog box* modale, si ferma, non provocando la generazione di altri messaggi oltre quelli già processati. Le due situazioni vengono distinte mediante *WParam*, che può infatti assumere i valori MSGF_MENU o MSGF_DIALOGBOX.

Il metodo *WMEnterIdle* esce subito se *Msg.WParam* non è uguale a MSGF_MENU o se *DispHelp* vale *False*, altrimenti invia il messaggio *um_PaintStatusLine* con parametri *sl_Plain* e zero.

Gestione di barre strumenti e riga di stato

Dobbiamo rimandare al mese prossimo l'illustrazione dell'ultimo gruppo di metodi, quelli mediante i quali si può rendere visibile o invisibile una barra strumenti o la riga di stato, oppure modificare l'orientamento di una barra strumenti.

Il rinvio è dovuto non solo ad evidenti ragioni di spazio, ma anche alla necessità di apportare gli ultimi ritocchi alla unit TOOLBAR. A suo tempo, infatti, avevo tralasciato quelle modifiche atte a rendere possibile la visualizzazione di messaggi esplicativi sulla riga di stato quando l'utente si posiziona su un pulsante della barra strumenti, in quanto la loro comprensione richiedeva la preliminare illustrazione delle classi *TNewMDIChild* e *TNewMDIFrame*.

Quando avremo completato il quadro, potremo finalmente approntare un demo semplice ma «completo», che sarà anche l'occasione per vedere come preparare i file di help utilizzando strumenti che rendano il processo più agile di quanto avviene se si usa *WinWord*.

Sergio Polini è raggiungibile tramite MC-link alla casella MC1166 e tramite Internet all'indirizzo mc1166@mclink.it.