Classi per applicazioni MDI

Negli ultimi appuntamenti abbiamo visto come prepararsi a dotare un'applicazione Windows, in particolare un'applicazione MDI, di barra strumenti e riga di stato. Ci rimane da vedere come si inizializzano righe e barre, come si definiscono i campi di una riga di stato, come i cambiamenti nel numero e nel tipo delle finestre aperte abbiano immediato effetto sul menu, sulle barre strumenti, sulla riga di stato. Per far questo, approntereremo alcune unit che, derivate da quelle già disponibili in ObjectWindows, ne estendano la funzionalità

di Sergio Polini

Due anni fa, in quel 1992 che sembra ormai lontano un secolo, avevo illustrato una serie di unit per la realizzazione di applicazioni MDI dotate di una barra strumenti popolata di normali pulsanti e di una riga di stato piuttosto spartana. Nel frattempo, ci siamo abituati ad usare applicazioni con barre strumenti dotate di coloratissimi pulsanti e con righe di stato disegnate mediante bitmap. Più in generale, siamo diventati tutti più esigenti.

Tenendo conto di questo (nonché del fatto che quelle unit, perfettamente funzionanti sotto Windows 3.0, non lo erano altrettanto sotto Windows 3.1), è diventato necessario ritoccare profondamente il codice che vi avevo a suo tempo proposto.

La client window

Ricordiamo ancora che, in un'applicazione MDI, non si indirizza l'output alla *client area* della finestra principale; la *client area* di quest'ultima, infatti, detta *frame window*, è normalmente tutta occupata da una finestra detta *client window*; l'output viene indirizzato a finestre, dette *child window*, aperte nell'ambito della *client window* in quanto «figlie» di questa. In ObjectWindows, quindi, troviamo le classi *TMDIWindow* per la *frame window* e *TMDIClient* per la *client window*, mentre le *child window* potrebbero anche essere semplici istanze di *TWindow*.

Abbiamo già visto che, per rendere visibili una barra strumenti o una riga di stato, dobbiamo lasciare scoperta parte della *client area* della *frame window*, riducendo le dimensioni della *client window*. Ora vedremo come derivare classi per i tre tipi di finestre che ricorrono in un'applicazione MDI, in modo da rendere agevole la gestione di righe e barre.

Il comportamento di un'applicazione MDI fa capo, in buona parte, alla *client window*, sempre costruita sulla base della classe predefinita di Windows MDICLIENT (qui «classe» non va inteso nel senso della programmazione orientata all'oggetto, ma nel senso di quelle strutture che si passano come argomento alla funzione *RegisterClass*). Cominceremo, quindi, da una unit MDICLNT (figura 1).

In essa si dichiara una classe *TNewMDIClient* che, derivata dalla classe di ObjectWindows *TMDIClient*, aggiunge ad essa il solo metodo *UMMDIChildDestroy*.

Il metodo risponde al messaggio omonimo (dichiarato nella unit MDICONST) che viene inviato da una *child window* - istanza della classe *TNewMDIChild* - quando viene chiusa. In questo modo, la *client window* può verificare se vi sono altre finestre aperte e, in caso contrario, avvertire la *frame window* perché provveda di conseguenza.

L'assenza di finestre aperte viene riconosciuta mediante il messaggio WM_MDIGETACTIVE, che ritorna un *Longint* la cui parola «bassa» ha valore diverso da zero solo se vi è almeno una finestra aperta (quel valore è l'handle della finestra attiva); la *frame window* viene avvertita mediante il messaggio *um_SetActiveChild*, anch'esso dichiarato in MDICONST, con parametri entrambi nulli.

Preciso che il ricorso al messaggio WM_MDIGETACTIVE è necessario. Si potrebbe pensare, infatti, che, essendo le *child window* «figlie» della *client window*, sarebbe sufficiente esaminare la variabile *ChildList* che *TNewMDIClient* eredita da *TWindowsObject*. Per motivi che mi restano misteriosi, però, in ObjectWindows le finestre di volta in volta aperte vengono appese alla lista della *frame window* invece che della *client window*, nonostante dipendano da quest'ultima (il metodo *TileChildren* di *TMDIWindow*, ad esempio, non fa altro che chiamare l'omonimo metodo di *TMDIClient*, il quale a sua volta non fa che inviare alla *client window* il messaggio WM_MDITILE).

Le child window

Per beneficiare di tutti i vantaggi offerti dalle unit sin qui viste, tutte le finestre «figlie» di un'applicazione MDI dovranno essere derivate dalla classe *TNewMDIChild*, dichiarata nella unit MDICHLD (figura 2).

La classe ha una variabile di istanza *Fields* di tipo *PCollection*, che viene inizializzata a **nil** dal constructor. Si tratta di un puntatore ai campi della riga di stato che sarà associata ad ogni singola finestra; ogni finestra, infatti, come abbiamo visto il mese scorso, avrà una propria riga di stato, in modo da poter costantemente proporre all'utente informazioni relative al proprio stato.

Ogni finestra, inoltre, avrà il proprio menu, che verrà passato come argomento al constructor e da questo assegnato al campo *Menu* della variabile *Attr*. La unit non svela come si ottenga che la variabile *Fields* punti a qualcosa, in quanto illustra solo le caratteristiche minime comuni a tutte le finestre di un'applicazione MDI. Possiamo solo notare che, se *Fields* non vale più **nil** quando la finestra viene chiusa, il destructor provvede a rilasciare la memoria allocata.

Vi sono due metodi pubblici e quattro metodi privati.

Il primo dei due metodi pubblici, *Get-*

SubMenuHelpCode, al momento ritorna uno zero; vedremo in seguito come potrà essere utilizzato per mostrare sulla riga di stato messaggi esplicativi delle opzioni del menu.

L'altro metodo pubblico, *WMMDIActivate*, viene eseguito in risposta al messaggio WM_MDIACTIVATE. Questo viene inviato da Windows alla *client window*, poi da questa sia alla *child window* che viene attivata (con *WParam* diverso da zero), sia a quella che viene disattivata (con *WParam* uguale a zero). Il metodo, quindi, non fa nulla se *Msg.WParam* è zero, ma, in caso contrario, aggiorna il menu dell'applicazione e comunica alla *frame window* sia il proprio stato di finestra attiva, sia la struttura della propria riga di stato.

Procediamo con ordine. È lecito aspettarsi che, in un'applicazione MDI, la struttura del menu possa cambiare, soprattutto se si propone all'utente di aprire tipi diversi di finestre (di testo, di grafica, ecc.). Quando viene attivata una finestra, può accadere sia che il menu corrispondente al tipo di questa sia già visualizzato (in quanto la precedente finestra attiva era dello stesso tipo), sia che non lo sia (in quanto, ad esempio, viene attivata una finestra grafica in sostituzione di una finestra di testo). In quest'ultimo caso, si comunica alla *client window* che il nuovo menu deve essere *Attr.Menu*, mediante il messaggio WM_MDISETMENU, e si provvede a renderlo visibile con la funzione *DrawMenuBar*. Prima ancora di ciò, tuttavia, occorre determinare quale opzione del nuovo menu conterrà l'elenco delle finestre aperte (si tratta, normalmente, dell'opzione che apre il menu pop-up *Finestre*). Si ottiene quindi il numero delle voci del nuovo menu principale, si sottrae uno nel caso la finestra sia massimizzata (l'icona di «Ripristino», quella che appare lungo il bordo destro del menu, viene inclusa nel numero reso dalla funzione *GetMenuItemCount*), si cerca il menu pop-up che contiene l'opzione *Affianca (Tile)*, opzione tipica del menu *Finestre*.

Dopo aver sistemato il menu, si inviano alla *frame window* i messaggi *um_SetActiveChild* e *um_PaintStatusLine*. Il primo comunica l'identità della finestra attiva, espressa da *Self*; il secondo segnala che occorre attivare una riga di stato di tipo *sl_Child*, i cui campi siano quelli contenuti nella *collection* puntata da *Fields*.

Si noti che la classe *TNewMDIChild* comprende un metodo *UMPaintStatusLine* che, come vedremo, viene eseguito se è la *frame window* che invia il messaggio *um_PaintStatusLine*. Ciò avviene quando, dopo aver utilizzato la riga di stato per chiarire il significato dell'opzione di un menu, occorre ripristinare il precedente contenuto. La *frame window*, in questi casi, invia alla finestra attiva quel messaggio con *sl_Child* come *WParam* e zero come *LParam*; la finestra lo rimanda indietro dopo aver avvalorato *LParam* con *Fields*.

Quando tutte le finestre vengono chiuse, si deve ripristinare il menu originario. Per fare questo, si intercetta con un apposito metodo il messaggio WM_DESTROY, che Windows manda ad una finestra prima di chiuderla: basta inviare alla *client window* il messaggio *um_MDIChildDestroy* e poi chiamare il metodo della classe base. Abbiamo visto sopra come risponde la classe *TNewMDIClient*. Da notare che, per ottenere che il messaggio giunga a destinazione solo dopo che la *child window* viene chiusa, si usa la funzione *PostMessage* invece di *SendMessage*; questa, infatti, aspetta che il messaggio venga processato dal destinatario prima di restituire il controllo all'istruzione successiva, mentre quella ritorna subito, limitandosi a porre il messaggio nella coda della finestra destinataria.

```
unit MDIClnt;

interface

uses
  WinTypes, WinProcs, Objects, OWindows,
  MDIConst;

type
  PNewMDIClient = ^TNewMDIClient;
  TNewMDIClient = object(TMDIClient)
  private
    procedure UMMDIChildDestroy(var Msg: TMessage);
      virtual wm_First + um_MDIChildDestroy;
  end;

implementation

procedure TNewMDIClient.UMMDIChildDestroy(var Msg: TMessage);
begin
  if LoWord(SendMEssage(HWindow, wm_MDIGetActive, 0, 0)) = 0 then
    SendMessage(Parent^.HWindow, um_SetActiveChild, 0, Longint(nil));
end;

end.
```

Figura 1 - La unit MDICLNT, in cui si dichiara una classe per la client window *di un'applicazione MDI.*

Descrizione delle opzioni dei menu

Quando l'utente si posiziona su una opzione di un menu, prima ancora del click con mouse o della pressione del tasto di Invio, Windows manda il mes-

MF_BITMAP	$0004	l'opzione è una Bitmap
MF_CHECKED	$0008	l'opzione ha un *checkmark*
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.*

```
unit MDIChld;

interface

uses
  WinTypes, WinProcs, Objects, OWindows,
  MDIConst;

type
  PNewMDIChild = ^TNewMDIChild;
  TNewMDIChild = object(TWindow)
    Fields: PCollection;
    constructor Init(AParent: PWindowsObject; ATitle: PChar;
                     AMenu: HMenu);
    destructor Done; virtual;
    function GetSubMenuHelpCode(ASubMenu: HMenu): Word; virtual;
    procedure WMMDIActivate(var Msg: TMessage);
      virtual wm_First + wm_MDIActivate;
  private
    procedure WMDestroy(var Msg: TMessage);
      virtual wm_First + wm_Destroy;
    procedure WMMenuSelect(var Msg: TMessage);
      virtual wm_First + wm_MenuSelect;
    procedure WMEnterIdle(var Msg: TMessage);
      virtual wm_First + wm_EnterIdle;
    procedure UMPaintStatusLine(var Msg: TMessage);
      virtual wm_First + um_PaintStatusLine;
  end;

implementation

constructor TNewMDIChild.Init(AParent: PWindowsObject; ATitle: PChar;
                              AMenu: HMenu);
var
  i: Integer;
begin
  inherited Init(AParent, ATitle);
  Attr.Menu := AMenu;
  Fields := nil;
end;

destructor TNewMDIChild.Done;
begin
  inherited Done;
  if Fields <> nil then
    Dispose(Fields, Done);
end;

function TNewMDIChild.GetSubMenuHelpCode(ASubMenu: HMenu): Word;
begin
  GetSubMenuHelpCode := 0;
end;
procedure TNewMDIChild.WMMDIActivate(var Msg: TMessage);
var
  HWinMenu: HMenu;
  n       : Word;
  Found   : Boolean;
begin
  if Msg.WParam <> 0 then begin
    if Attr.Menu <> GetMenu(Parent^.HWindow) then begin
      n := GetMenuItemCount(Attr.Menu);
      if IsZoomed(HWindow) then
        Dec(n);                          {escludi l'icona di "Ripristino"}
      Found := FALSE;
      while (n > 0) and (not Found) do begin
        HWinMenu := GetSubMenu(Attr.Menu, n - 1);
        if GetMenuState(HWinMenu,
                        cm_TileChildren, mf_ByCommand) <> $FFFF
        then
          Found := TRUE;
        Dec(n);
      end;
      SendMessage(PMDIWindow(Parent)^.ClientWnd^.HWindow, wm MDISetMenu,
        0, MakeLong(Attr.Menu, HWinMenu));
      DrawMenuBar(Parent^.HWindow);
    end;
```

saggio WM_MENUSELECT all'applicazione cui il menu appartiene.

In *LParam* viene posta una combinazione dei flag illustrati nella figura 3, in *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 *WParam* l'handle del menu, se si sta su un'opzione che provoca l'invio di un comando, vi troviamo la costante che identifica quest'ultimo.

Il messaggio viene intercettato dal metodo *WMMenuSelect* della *frame window*, il quale, tuttavia, se vi è almeno una finestra aperta, lo gira a quest'ultima. Ecco perché anche la classe *TNewMDIClient* ha un metodo *WMMenuSelect*.

Questo verifica in primo luogo se *LParam* contiene $0000FFFF, costante usata da Windows quando si abbandona un menu con Esc o clickando altrove col mouse; in tal caso, si provoca il ripristino della riga di stato mediante invio alla *frame window* del messaggio *um_PaintStatusLine*.

Se, invece, *LParam* contiene il flag MF_DISABLED, si deve intendere che l'utente si è posizionato su una linea di separazione di un menu, oppure si sta spostando col mouse, mantenendo premuto il pulsante, fuori del menu. In questi casi, il metodo invia un messaggio *um_SetHelpCode* con due parametri entrambi nulli; come vedremo, la *frame window* reagirà mostrando una riga di stato «vuota» (senza descrizioni né campi).

Negli altri casi che interessano, occorre verificare i flag *mf_Popup* e *mf_SysMenu*; il primo indica che l'utente è posizionato su un menu pop-up, il secondo che è sopra un'opzione del *system menu* della finestra (quello che si apre premendo i tasti «Alt» e «-»); se sono settati entrambi o nessuno in *LParam*, l'utente si trova, rispettivamente, sull'icona del *system menu* della finestra o su un'opzione di un menu (ma non del *system menu*).

In quest'ultimo caso, si tratta di capire se l'utente si è posizionato su un'opzione normale, oppure sui titoli delle *child window* aperte, riportati in calce al menu *Finestre*. Possiamo ricordare che i codici dei comandi corrispondenti ai titoli vanno da *id_FirstMDIChild* (costante predefinita) a *id_FirstMDIChild+8*, mentre il valore *id_FirstMDIChild+9* corrisponde all'opzione *Altre finestre*, che compare, se vi sono più di nove finestre aperte, dopo il titolo della nona. Si verifica quindi il valore di *WParam* e, se è maggiore o uguale a *id_FirstMDIChild*, lo si cambia in *cm_ActivateChild* o in *cm_ChildList*. Il valore di *WParam* viene quindi assegnato alla variabile locale *HelpCode*, per essere poi inviato alla *frame window*.

Se l'utente è su una opzione che comporta l'apertura di un menu pop-up, si tratta di vedere se si tratta di un'opzione del menu principale o di un altro menu pop-up. La funzione *GetSubMe-*

```
    SendMessage(Parent^.HWindow, um_SetActiveChild, 0, Longint(@Self));
    SendMessage(Parent^.HWindow,
                um_PaintStatusLine, sl_Child, Longint(Fields));
  end;
end;

procedure TNewMDIChild.WMDestroy(var Msg: TMessage);
begin
  PostMessage(PMDIWindow(Parent)^.ClientWnd^.HWindow,
              um_MDIChildDestroy, 0, 0);
  TWindow.WMDestroy(Msg);
end;

procedure TNewMDIChild.WMEnterIdle(var Msg: TMessage);
begin
  with Msg do
    SendMessage(Parent^.HWindow, Message, WParam, LParam);
end;

procedure TNewMDIChild.UMPaintStatusLine(var Msg: TMessage);
begin
  with Msg do begin
    if (WParam = sl_Child) and (LParam = 0) then
      LParam := Longint(Fields);
    SendMessage(Parent^.HWindow, Message, WParam, LParam);
  end;
end;
procedure TNewMDIChild.WMMenuSelect(var Msg: TMessage);
var
  HelpCode: Word;
  Menu: HMenu;
begin
  if Msg.LParam = $0000FFFF then begin              {abbandonato il menu'}
    SendMessage(Parent^.HWindow,
                um_PaintStatusLine, sl_Child, Longint(Fields));
    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 window}
      begin
        if (Msg.WParam >= id_FirstMDIChild)
        and (Msg.WParam <= id_FirstMDIChild + 8) then
          Msg.WParam := cm_ActivateChild
        else if Msg.WParam = id_FirstMDIChild + 9 then
          Msg.WParam := cm_ChildList;
        HelpCode := ids_ChildMenuItem + Msg.WParam;
      end;
    mf_Popup:          {un menu' pop-up}
      begin
        HelpCode := GetSubMenuHelpCode(Msg.WParam);
        if HelpCode = 0 then 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;
      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;

end.
```

Figura 2 - La unit MDICHILD, da cui devono essere derivate le classi per le finestre di un'applicazione MDI.

nuHelpCode serve proprio a questo, in quanto è destinata ad essere ridefinita in classi derivate in modo da ritornare un codice adeguato.

Rimandiamo per il momento questa complicazione, per vedere come si agisce nel caso di opzione del menu principale: si assegna a *HelpCode* il numero delle opzioni nel menu e si percorrono tutte queste fino a trovarne una il cui handle sia uguale a *WParam*; si aggiunge quindi a *HelpCode* la costante *ids_ChildPopupMenu*; infine, se la finestra non è massimizzata si aggiunge 1. Nel caso l'utene si trovi sull'icona che apre il *system menu*, si assegna direttamente a *HelpCode* il valore *ids_ChildPopupMenu*.

Si ottiene così un numero pari a *ids_ChildPopupMenu* più il numero d'ordine dell'opzione; ad esempio, se l'utente è sull'opzione File del menu principale, *HelpCode* vale *ids_ChildPopupMenu+1*. Le opzioni del menu sono contate da Windows partendo da zero (ad esempio, se le prime opzioni del menu principale sono *File, Modifica, Visualizza*, Windows assegna loro, rispettivamente, i numeri zero, uno e due). Se la finestra è massimizzata, però, Windows comincia a contare dall'icona del *system menu* che appare sul menu alla sinistra della prima opzione (tornando all'esempio, le opzioni del menu principale saranno: icona del *system menu, File, Modifica, Visualizza*, che Windows identificherà con i numeri zero, uno, due e tre). È questo il motivo per cui un *HelpCode* con valore *ids_ChildPopupMenu* può essere assunto come caratterizzante l'icona del *system menu*, ma, se la finestra non è massimizzata, si aggiunge uno al numero calcolato per HelpCode, in modo da poter sempre considerare identificata dal numero uno la prima opzione, dal numero due la seconda, e così via.

Rimane il caso delle opzioni del *system menu*. I codici dei comandi corrispondenti ad esse sono numeri che, in esadecimale, rispettano un curioso schema: la prima cifra è F, la quarta è sempre zero, la seconda e la terza variano da $00 a $13. Conviene, quindi, estrarre il numero composto dalla seconda e dalla terza cifra, per ottenere un valore «piccolo» da sommare a *ids_ChildMenuItem*.

La *frame window* utilizzerà il valore di *HelpCode* per individuare le stringhe da visualizzare nella riga di stato. Lo farà dopo aver ricevuto da Windows il messaggio WM_ENTERIDLE, prodotto quando l'utente, dopo essere entrato in un menu o in una dialog box, si ferma.

Il metodo *WMEnterIdle* di una *child window*, peraltro, si limita a girare il messaggio alla *frame window*. Il mese prossimo avremo occasione di vedere come la finestra principale coopera con le altre. MC

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