Visualizzazione di informazioni su una riga di stato

Il mese scorso abbiamo iniziato l'esame di una serie di unit per applicazioni MDI. L'obiettivo è quello di rendere più semplice la realizzazione di applicazioni che siano dotate di elementi ormai essenziali in un'interfaccia utente, quali una riga di stato ed una barra strumenti, ma anche di intervenire sulla classe TEDIT in modo da consentire la creazione di finestre di editing dotate di un proprio data segment e capaci di fornire all'utente informazioni quali la posizione del cursore e la lunghezza del testo. Completeremo, ora, l'esame della unit dedicata alle righe di stato

di Sergio Polini

Abbiamo già esaminato una unit MDICONST, contenente l'insieme minimo di costanti richiesto da un'applicazione MDI conforme allo standard quanto alla struttura del menu, ma anche dotata di una riga di stato e di una barra strumenti. Abbiamo anche iniziato l'esame di una unit STATLINE contenente la dichiarazione di due classi: *TStatusLine* per la riga di stato, *TStatusField* per ognuno dei diversi «campi» di questa.

Ricordo che, completata l'illustrazione della classe *TStatusField*, abbiamo già visto i metodi che presiedono alla visualizzazione di un'istanza di *TStatusLine*, al fine di rendere chiara la tecnica da utilizzare per evitare il tremolio dello schermo in occasione di aggiornamenti delle informazioni fornite.

L'aspetto forse più interessante delle istanze della classe *TStatusLine*, tuttavia, è la loro flessibilità.

In molte applicazioni risulta utile suddividere la riga di stato in «campi»; in questo modo, infatti, si possono fornire contemporaneamente all'utente diversi tipi di informazioni, quali (leggo a caso dal word processor che sto usando) il numero della pagina, la posizione del cursore, la situazione di tasti quali BLOC NUM o INS, ecc.; si può disegnare la riga di stato mediante bitmap ed utilizzare accorgimenti grafici per raggruppare elementi di informazioni tra loro correlate, distinguendole, al tempo stesso, da informazioni di altra natura; si può evitare di riscrivere tutta la riga quando va modificato il contenuto di uno solo dei suoi campi. In applicazioni MDI, tuttavia, è possibile che all'utente sia consentito aprire finestre di tipo diverso (ad esempio: testo e grafica) e che ad ogni tipo di finestra siano associate informazioni di natura diversa; è possibile, quindi, che ad ogni tipo di finestra sia opportuno far corrispondere una specifica suddivisione in campi della riga di stato. Potrebbe non avere senso, inoltre, mostrare i diversi campi quando l'utente non abbia ancora aperto alcuna finestra, o quando le abbia chiuse tutte; in particolare, nel caso di applicazioni MDI con finestre di diverso tipo, ognuna con una propria riga di stato, non si saprebbe quale privilegiare. La suddivisione in campi, infine, può essere d'impaccio quando si tratta di mostrare messaggi esplicativi delle opzioni dei menu, in quanto potrebbe risultare impossibile prevedere un campo sufficientemente lungo per la loro visualizzazione.

Per tenere conto di tutte queste esigenze, la unit MDICONST contiene la dichiarazione delle tre costanti *sl_Frame, sl_Child* e *sl_Plain* per tre diversi tipi di riga di stato: la prima può essere utilizzata quando non vi è alcuna finestra aperta, la seconda quando è aperta almeno una finestra, la terza quando si vuole mostrare un messaggio «a tutto campo», facendo temporaneamente sparire la suddivisione in campi.

Le costanti non vengono usate dalle classi dichiarate in STATLINE, ma la classe *TStatusLine* è disegnata in modo da poter cambiare aspetto mediante pochi semplici metodi.

```
constructor TStatusLine.Init(AParent: PWindowsObject;
                             FrameFields: PCollection;
                             Orient: Word);
begin
  inherited Init(AParent, nil);
  Attr.Style := ws_Child or ws_Visible or ws_Border;
  SetFlags(wb_MDIChild, False);
  DefaultProc := @DefWindowProc;
  Attr.X := -1;
  Attr.Y := -1;
  Attr.W := 20;
  Attr.H := 20;
  Orientation := Orient;
  FillChar(LF, SizeOf(LF), 0);
  with LF do begin
    lfHeight := -11;
    lfPitchAndFamily := Variable_Pitch;
    StrCopy(lfFaceName, 'Helv');
  end;
  WhitePen := GetStockObject(White_Pen);
  Left := LoadBitmap(HInstance, 'LEFT');
  Middle := LoadBitmap(HInstance, 'MIDDLE');
  Right := LoadBitmap(HInstance, 'RIGHT');
  Txt := nil;
  Fields := FrameFields;
end;

destructor TStatusLine.Done;
begin
  DeleteObject(Left);
  DeleteObject(Middle);
  DeleteObject(Right);
  DeleteObject(MemBM);
  inherited Done;
end;
```

Figura 1 - Il constructor ed il destructor della classe TStatusLine.

Inizializzazione

Nella figura 1 potete trovare il constructor ed il destructor della classe *TStatusLine*.

Il constructor vuole tre parametri: la finestra madre (*AParent*), il puntatore ad una collezione di campi (*FrameFields*), una delle costanti di orientamento già viste nella unit TOOLBAR (*Orient*).

Le prime istruzioni del constructor sono molto simili a quelle già viste nel constructor della classe *TToolbar*. In particolare, viene inizializzata la variabile d'istanza *Orientation*, nonostante che una riga di stato possa apparire solo lungo il bordo inferiore della *frame window*. Ricordiamo, infatti, che avevamo aggiunto una costante *tbHidden* alla unit TOOLBAR, per consentire all'utente di nascondere la barra strumenti; la riga di stato, come vedremo tra breve, potrà essere nascosta in modo analogo.

Le istruzioni successive avvalorano la variabile d'istanza *LF*, di tipo *TLogFont*, con costanti corrispondenti al carattere Helvetica 8 e caricano le bitmap contenute nel file di risorse STATLINE.RES; abbiamo visto il mese scorso come la variabile *LF* e le bitmap vengano utilizzate per l'output.

Le ultime due istruzioni assegnano **nil** e *FrameFields* alle variabili d'istanza *Txt* e *Fields*. La prima di queste è destinata ad essere avvalorata con un puntatore ad una stringa ASCIIZ contenente un messaggio da visualizzare prescindendo dalla suddivisione in campi (la descrizione dell'opzione di un menu, «salvataggio in corso», ecc.) e, quindi, assume inizialmente il valore **nil**; la seconda, invece, viene inizializzata con una descrizione dei campi in cui si desidera suddividere la riga di stato quando nessuna finestra è stata ancora aperta.

Il destructor, dal canto suo, rilascia la memoria occupata dalle bitmap caricate dal constructor e da quella (*MemBM*) creata dal metodo *SetupWindow* (che abbiamo visto il mese scorso).

Visualizzazione sì/no

La figura 2 raccoglie i metodi mediante i quali si può intervenire per cambiare quello che, per una riga di stato, abbiamo definito «orientamento».

La unit TOOLBAR da cui siamo partiti (fornita insieme al compilatore) prevedeva tre orientamenti diversi: orizzontale superiore (subito sotto la barra dei menu), verticale sinistro e verticale destro. A questi abbiamo aggiunto un orientamento orizzontale inferiore (lungo il bordo inferiore della *frame window*) ed un orientamento «nascosto», cioè la possibilità di scegliere se mostrare o no la barra di stato.

Nella unit STATLINE si cerca di mantenere la maggiore coerenza possibile con la struttura della unit TOOLBAR, in quanto è certo preferibile trattare in modo analogo oggetti analoghi (come già detto il mese scorso, sarebbe possibile un'organizzazione migliore del tutto, ma ho voluto contenere al minimo gli interventi su una unit già disponibile).

È questo il motivo per cui, forse con qualche forzatura, la scelta tra la visualizzazione o meno della riga di stato è proposta in termini di scelta di un orientamento. Quando viene creata una riga di stato, per il terzo parametro del constructor si possono utilizzare le costanti *tbHidden* o *tbDownHorizontal* (già usate per le barre strumenti); in pratica, poiché la riga di stato, se visibile, ha senso solo se collocata lungo il bordo inferiore della *frame window*, il metodo *AMCalcParentClientRect* della classe *TStatusLine* assume che qualsiasi valore diverso da *tbHidden* vada trattato come se fosse *tbDownHorizontal*.

Questo metodo, infatti, tratta in mo-

```
procedure TStatusLine.AMCalcParentClientRect(var Msg: TMessage);
var
  R,              (* client area *)
  SL,             (* status line esistente *)
  NewSL,          (* nuova status line *)
  InvSL: TRect;   (* parte della status line da ridisegnare *)
  CX,
  CY: Integer;    (* larghezza e altezza di NewSL *)
  DC: HDC;
begin
  if Orientation = tbHidden then begin
    SetRect(Rect, 0, 0, 0, 0);
    Exit;
  end;
  R := PRect(Msg.LParam)^;
  GetWindowRect(HWindow, SL);
  CX := R.Right - R.Left;    (* larghezza quella della client area *)
  CY := SL.Bottom - SL.Top; (* altezza della status line *)
  if Msg.WParam = AllowRepaint then begin
    NewSL.Top := R.Bottom - CY + 1;
    SetRect(NewSL, R.Left - 1, NewSL.Top, CX + 2, CY);
    SetWindowPos(HWindow, 0,
      NewSL.Left, NewSL.Top, NewSL.Right, NewSL.Bottom,
      swp_NoZOrder or swp_NoRedraw);
    Attr.W := CX;
    if (NewSL.Left <> Rect.Left) or (NewSL.Top <> Rect.Top) then
      InvalidateRect(HWindow, nil, True)
    else if NewSL.Right > Rect.Right then begin
      SetRect(InvSL, Rect.Right - 2, -1, NewSL.Right, CY);
      InvalidateRect(HWindow, @InvSL, True);
    end;
    Rect := NewSL;
  end;
  Dec(R.Bottom, CY - 1);
  PRect(Msg.LParam)^ := R;
  DeleteObject(MemBM);
  SetRect(R, 0, 0, Rect.Right-2, Rect.Bottom-2);
  DC := GetDC(HWIndow);
  MemBM := CreateCompatibleBitmap(DC, R.Right, R.Bottom);
  ReleaseDC(HWindow, DC);
end;

procedure TStatusLine.SetOrientation(NewOrient: Word);
begin
  Orientation := NewOrient;
  if Orientation = tbHidden then begin
    Attr.W := 0;
    Attr.H := 0;
  end
  else begin
    Attr.W := 20;
    Attr.H := 20;
  end;
  SetWindowPos(HWindow, 0, -1, -1, Attr.W, Attr.H, swp_NoZOrder or
               swp_NoRedraw);
end;
function TStatusLine.GetOrientation: Word;
begin
  GetOrientation := Orientation;
end;
```

Figura 2 - La scelta se visualizzare o no la riga di stato viene operata con modalità analoghe a quelle già viste per la barra di stato.

```
procedure TStatusLine.SetText(T: PChar);
begin
  Fields := nil;
  Txt := T;
  InvalidateRect(HWindow, nil, True);
  UpdateWindow(HWindow);
end;

procedure TStatusLine.SwitchTo(NewFields: PCollection);
begin
  Fields := NewFields;
  Txt := nil;
  InvalidateRect(HWindow, nil, True);
  UpdateWindow(HWindow);
end;
```

Figura 3 - I metodi utilizzati per cambiare la suddivisione in campi della riga di stato.

do specifico solo il caso *tbHidden*; in ogni altro caso, si determinano la larghezza della *client area* e l'altezza della riga di stato, si ridisegna il rettangolo di questa in modo che sia collocato lungo il bordo inferiore della *frame window* e si modificano le dimensioni e la posizione della *client area* (passate in *Msg.LParam*) in modo da lasciare spazio perché la riga di stato sia visibile. Notiamo anche che è in questo metodo che viene «ricreata» *MemBM*, quella bitmap compatibile con il *device context* della finestra che, come abbiamo visto il mese scorso, viene usata dai metodi *Paint* per ottenere un output senza tremolio.

Per rendere visibile una riga di stato nascosta, o per nascondere una riga di stato visibile, si userà il metodo *SetOrientation*, analogo all'omonimo metodo della classe *TToolbar*.

Il metodo *GetOrientation* rende lo stato corrente (visibile o no) della riga di stato, attraverso il valore della variabile d'istanza *Orientation*.

Variazione del tipo e del contenuto della riga di stato

Nella figura 3 sono riprodotti i metodi utilizzati per cambiare tipo di riga di stato.

Il metodo *SetText* inverte l'inizializzazione delle variabili *Txt* e *Fields* operata dal constructor, assegnando il parametro *T* alla prima e **nil** alla seconda. Come abbiamo visto il mese scorso, il metodo *Paint* della classe *TStatusLine* seleziona la bitmap *MemBM* in un *memory device context* compatibile con quello «fisico», *MemDC*, e lo riempie usando un *brush* di colore grigio per disegnare lo sfondo della riga di stato; quindi, se la variabile *Fields* vale **nil**, scrive direttamente su *MemDC* il testo puntato da *Txt*; i campi della riga di stato vengono disegnati mediante le bitmap *Left*, *Middle* e *Right* solo se *Fields* non vale **nil**.

Il metodo *SwitchTo* opera in senso inverso (assegna **nil** a *Txt* ed il parametro *NewFields* a *Fields*) e, quindi, permette di ripristinare il normale aspetto della riga di stato e dei suoi campi. I nomi *SwitchTo* (ispirato dall'omonimo metodo della classe *TToolbar*) e *NewFields*, tuttavia, sono stati scelti in quanto il metodo può essere utilizzato anche per cambiare la suddivisione in campi o il contenuto di questi. Come vedremo, le classi dedicate alla creazione delle diverse finestre di un'applicazione MDI chiameranno il metodo *SwitchTo* per ottenere che, ogni volta che si apra o chiuda una finestra, come anche ogni volta che si passi dall'una all'altra delle diverse finestre aperte, venga mostrata una riga di stato coerente con la finestra di volta in volta attiva.

Abbiamo già fatto cenno all'opportunità di prevedere diverse suddivisioni in campi per diversi tipi di finestre; va notato, peraltro, che il metodo *SwitchTo* sarebbe necessario anche nel caso di applicazioni che consentissero di aprire un solo tipo di finestra.

Immaginiamo, ad esempio, un editor MDI capace di gestire contemporaneamente più file di testo, con una riga di stato contenente i campi «riga» e «colonna» per la posizione del cursore. L'utente potrebbe aver aperto due file ed essersi posizionato sul decimo carattere della quinta riga del primo, sul ventesimo carattere della prima riga del secondo. Quando è attiva la finestra aperta sul primo file, la riga di stato deve mostrare i numeri 5 e 10; quando è attiva la finestra sul secondo, deve mostrare 1 e 20. L'utente può passare come e quando vuole dall'una all'altra finestra; da ciò segue che il contenuto della riga di stato non va aggiornato solo quando cambia la posizione del cursore nella finestra attiva, ma anche quando cambia la finestra attiva. In quest'ultimo caso, sarebbe troppo laborioso ricalcolare i valori da mostrare nella riga di stato (preciso: anche un ormai modesto 80386 ci metterebbe un attimo; sarebbe laborioso per il programmatore predisporre un tale ricalcolo, in quanto da un lato le informazioni da visualizzare possono avere le fonti più diverse, dall'altro occorrerebbe prevedere per ogni informazione due diverse modalità di acquisizione, l'una quando cambia l'informazione in un dato contesto - ad esempio quando si muove il cursore in una finestra - l'altra quando cambia il contesto dell'informazione - quando, cioè, cambia la finestra attiva).

Ho preferito prevedere, quindi, che ogni finestra gestisse i propri campi attraverso una collezione di istanze di *TStatusField*. La riga di stato di ogni finestra di editing, ad esempio, avrà un campo destinato a mostrare la posizione del cursore; in tale campo risulterà costantemente «scritta» una stringa del tipo «Ri yyyy Col xxxx» con la posizione corrente del cursore (il numero di riga al posto di «yyyy» ed il numero di colonna al posto di «xxxx»). Quando l'utente muove il cursore, il contenuto del campo viene aggiornato mediante il metodo *SetText* di *TStatusField*, il quale, come abbiamo visto il mese scorso, provoca un *repaint* della zona della riga di stato occupata dal campo. Quando l'utente passa ad un'altra finestra, il metodo *SwitchTo* di *TStatusLine* provoca il *repaint* di tutta la riga di stato; in questo caso, viene semplicemente visualizzato il contenuto dei campi gestiti dalla nuova finestra attiva, cui la riga di stato accede mediante il parametro *NewFields*.

Nuove classi per finestre frame, client e child

Termina così l'illustrazione della unit STATLINE.

Nonostante i miei sforzi, dubito che possiate aver tratto un'idea pienamente chiara del funzionamento di una istanza della classe *TStatusLine*. I metodi di questa, infatti, costituiscono solo le «risposte» ai «messaggi» che le diverse finestre di un'applicazione MDI possono inviare ad una riga di stato.

Per completare il quadro, dovremo vedere come una *frame window* inizializza una riga di stato e, in particolare, come ne definisce i campi; dovremo vedere come le singole finestre possono preparare eventuali distinte suddivisioni in campi della riga di stato; dovremo vedere come far sì che l'apertura o la chiusura di una finestra, o il passaggio dall'una all'altra delle finestre aperte, comporti automaticamente l'aggiornamento della riga di stato. Dovremo, in sintesi, approntare unit specifiche per la *frame window* e la *client window* di applicazioni MDI, e vedere come dichiarare classi di finestre che, derivate in ultima istanza da *TWindow*, siano adatte ad operare come *child window* di tali applicazioni.

Vi do quindi appuntamento tra trenta giorni. MC

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

CHIEDI LA LUNA.

©RcpAssociati

Oggi è possibile per tutti vivere l'emozione di avere la luna, la stessa emozione di chi 25 anni fa riuscì a conquistarla. Basta avere la videocassetta della storia delle missioni Apollo che, oltre alle immagini dello storico sbarco trasmesse sulla terra, contiene i filmati originali a colori girati dagli stessi astronauti sul suolo lunare.

Per ordinare la videocassetta (lire 29.000) compilate il modulo pubblicato nell'ultima pagina della rivista indicando il numero di codice A&E ØØ1.