Una nuova unit TOOLBAR

Il mese scorso abbiamo esaminato la unit TOOLBAR, fornita insieme al Pascal 7.0, al fine prendere nota delle soluzioni suggeriteci dalla Borland per dotare applicazioni Windows MDI di una barra strumenti, e magari anche di una riga di stato. Abbiamo discusso sia delle assunzioni operate (ad esempio, circa la dimensione dei pulsanti), sia della flessibilità dell'impianto generale. Ne abbiamo concluso che alcune assunzioni sono accettabili e altre meno, ma, soprattutto, che la unit così come è non consente di dotare una applicazione di più di una barra (oppure di una barra strumenti e di una riga di stato); potremmo aggiungere che non è neppure possibile nascondere l'unica barra consentita. A tutto, comunque, c'è rimedio

di Sergio Polini

Uno dei punti di forza della programmazione orientata all'oggetto risiede nella possibilità di modificare il comportamento di una classe senza intervenire sulla sua implementazione (senza modificare i sorgenti), ma semplicemente derivando una nuova classe.

Per altro verso, Brad J. Cox, uno dei santoni della OOP, raccomanda di assegnare ad ogni classe un *indice di maturità*, per tenere conto del fatto che l'efficacia di una classe può essere verificata solo mediante il suo uso in diverse applicazioni «vere». Prima che una classe possa essere giudicata matura, quindi, le modifiche ai sorgenti sono non solo accettabili anche da un punto di vista teorico, ma addirittura necessarie.

È questo il motivo per cui, dopo aver utilizzato la unit TOOLBAR e, soprattutto, dopo averne riscontrato i limiti, ho scelto di modificarne i sorgenti piuttosto che di proporvi di appesantire le vostre applicazioni aggiungendo un'ulteriore classe. Ne è risultata una unit che, come vedremo, consente di utilizzare più barre strumenti in una stessa applicazione, ognuna con pulsanti di dimensioni anche diverse dal 20x20 assunto dalla unit originaria, di nasconderle se l'utente preferisce farne a meno, di cambiarne con libertà l'orientamento, di collocarle, se si vuole, anche lungo il bordo inferiore della *frame window*.

Qualche ritocco

Le barre realizzate mediante la unit originaria possono essere disposte solo lungo i bordi superiore, sinistro e destro della *frame window*; inoltre, se si prevede una barra strumenti, questa deve risultare sempre visibile.

Per superare tali limitazioni, occorre prevedere in primo luogo due ulteriori costanti; a *tbHorizontal*, *tbLeftVertical* e *tbRightVertical*, quindi, aggiungiamo le costanti *tbHidden* e *tbDownHorizontal* (figura 1).

Per semplificare l'implementazione del metodo *AMCalcParentClientRect*, discussa più avanti, aggiungiamo inoltre alla classe *TToolbar* una nuova variabile d'istanza *Rect* (figura 2).

Queste sono le uniche modifiche all'interfaccia della unit. Tutte le altre riguardano i metodi della classe *TToolbar*. Nel constructor *Init* modifichiamo le assegnazioni ai campi della variabile *Attr*, attribuendo alla finestra larghezza e altezza iniziali nulle (figura 3). Ciò si rende necessario in quanto, se si creasse la barra con *Orient* pari a *tbHidden*, e quindi come inizialmente

```
const
  am_CalcParentClientRect = wm_User + 120;
  tbHidden       = $00;                          (* nuova *)
  tbHorizontal   = $01;
  tbLeftVertical = $02;
  tbRightVertical= $04;
  tbDownHorizontal=$08;                          (* nuova *)
  DenyRepaint  = 0;
  AllowRepaint = 1;
```

Figura 1 - La sezione di dichiarazione delle costanti nella interfaccia della nuova versione della unit TOOLBAR, con le due nuove costanti tbHidden *e* tbDownHorizontal.

```
PToolbar = ^TToolbar;
TToolbar = object(TWindow)
  ResName: PChar;
  Tools: TCollection;
  Capture: PTool;
  Orientation: Word;
  Rect: TRect;                                   (* nuova *)
  constructor Init(AParent: PWindowsObject; AName: PChar; Orient: Word);
  ...
end;
```

Figura 2 - Per semplificare l'implementazione del metodo AMCalcParentClientRect, *aggiungiamo una variabile d'istanza* Rect *alla classe* TToolbar.

```
constructor TToolbar.Init(AParent: PWindowsObject; AName: PChar;
  Orient: Word);
begin
  ...
  Attr.W := 0;                                   (* invece di Attr.W := 5 *)
  Attr.H := 0;                                   (* invece di Attr.H := 5 *)
  SetRect(Rect, 0, 0, 0, 0);                     (* nuova *)
  ...
end;
```

Figura 3 - Attribuiamo, nel costructor, larghezza e altezza iniziali nulle alla finestra TToolbar; *in caso contrario, vedremmo un rettangolino grigio in alto a sinistra nella* frame window *se la barra venisse creata con* Orient *pari a* tbHidden *(e quindi inizialmente invisibile).*

invisibile, vedremmo un rettangolino grigio, una vera e propria «macchia», in alto a sinistra nella *frame window*.

Per lo stesso motivo, è necessario intervenire sui metodi *NextToolOrigin* e *ReadResource*. Quest'ultimo cerca nell'eseguibile una risorsa di tipo TOOLBARDATA e, per ognuno dei Tool in essa definiti (nel modo descritto il mese scorso), dopo averli creati col metodo *CreateTool*, chiama il metodo *NextToolOrigin* che, tra l'altro, incrementa larghezza e altezza della barra. Le dimensioni della barra vengono poi incrementate per lasciare un po' di spazio tra i suoi bordi e quelli dei pulsanti. Tutto ciò va benissimo se la barra è visibile; se, tuttavia, la si vuole inizialmente nascondere, l'incremento delle sue dimensioni provoca quella «macchia» che dicevamo e che vogliamo assolutamente evitare. A questo scopo, modifichiamo il metodo *NextToolOrigin* aggiungendo il caso *tbDownHorizontal*, ma lasciando per il resto le cose come stanno (figura 4); ne seguirà che, nel caso la variabile d'istanza *Orientation* valga *tbHidden*, il metodo non farà nulla (in particolare, non incrementerà le dimensioni della barra).

Quanto a *ReadResource*, sostituiamo l'incremento incondizionato delle dimensioni con un incremento sottoposto alla condizione che *Orientation* non valga *tbHidden* (figura 5). Ho sperimentato, inoltre, che un incremento di 8 produce risultati accettabili con pulsanti creati come bitmap 20x20, meno con pulsanti più piccoli; un incremento di 6, invece, appare adatto quale che sia la dimensione dei pulsanti.

Gli stessi problemi (barra inizialmente invisibile e dimensioni dei pulsanti) impongono di modificare anche il metodo *SetOrientation* (figura 6): la larghezza e l'altezza iniziali della barra diventano pari a zero, ogni loro incremento viene escluso se la barra è invisibile, mentre, se la barra è visibile, si effettua un incremento di 6 invece che di 8 dopo l'esecuzione, per ogni *Tool*, della procedura *ResetOrigins*.

L'ultima tra le modifiche minori concerne il metodo *Paint*: se la barra è invisibile, non vi è nulla da disegnare sullo schermo; se, quindi, *Orientation* vale *tbHidden*, si esce subito (figura 7).

Un metodo tutto nuovo

Il mese scorso avevamo visto come riscrivere il metodo *AMCalcParentClientRect* in modo da gestire più di una bar-

```
procedure TToolbar.NextToolOrigin(Num: Integer; var Origin: TPoint;
  P: PTool);
begin
  case Orientation of
    tbHorizontal,
    tbDownHorizontal:                                  (* nuovo *)
      begin
        Inc(Origin.X, P^.GetWidth);
        Attr.H := Max(Attr.H, P^.GetHeight);
      end;
   tbLeftVertical,
   tbRightVertical:
     begin
       Inc(Origin.Y, P^.GetHeight);
       Attr.W := Max(Attr.W, P^.GetWidth);
     end;
  end;
end;
```

Figura 4 - Prevediamo il caso che Orientation *valga* tbDownHorizontal; *nel caso valga* tbHidden *non si fa nulla; in particolare, non si incrementano le dimensioni della barra.*

```
procedure TToolbar.ReadResource;
type
  ...
var
  ...
begin
  ResIDHandle := FindResource(HInstance, ResName, 'ToolBarData');
  ...
  if Orientation <> tbHidden then begin    (* nuovo *)
    Inc(Attr.H, 6);                        (* invece di Inc(Attr.H, 6) *)
    Inc(Attr.W, 6);                        (* invece di Inc(Attr.W, 6) *)
  end;                                     (* nuovo *)
  UnlockResource(ResDataHandle);
  FreeResource(ResDataHandle);
end;
```

Figura 5 - Modifichiamo il metodo ReadResource *in modo che l'incremento delle dimensioni della barra abbia luogo solo se la barra è visibile.*

```
procedure TToolbar.SetOrientation(NewOrient: Word);
var
  ...
procedure ResetOrigins(P : PTool); far;
  begin
  ...
  end;
begin
  Orientation := NewOrient;
  Attr.H := 0;                             (* invece di Attr.H := 5 *)
  Attr.W := 0;                             (* invece di Attr.W := 5 *)
  if Orientation <> tbHidden then begin    (* nuovo *)
    X := 0;
    Origin.X := 2;
    Origin.Y := 2;
    Tools.ForEach(@ResetOrigins);
    Inc(Attr.W, 6);                        (* invece di Inc(Attr.H, 8) *)
    Inc(Attr.H, 6);                        (* invece di Inc(Attr.W, 8) *)
  end;                                     (* nuovo *)
  SetWindowPos(HWindow, 0, -1, -1, Attr.W, Attr.H,  swp_NoZOrder or
    swp_NoRedraw);
end;
```

Figura 6 - Le modifiche da apportare al metodo SetOrientation.

Figura 7 - Se la barra è invisibile, non c'è nulla da ridisegnare sullo schermo. Se Orientation *vale* tbHidden, *quindi, il metodo* Paint *non fa nulla.*

```
procedure TToolbar.Paint(DC: HDC; var PS: TPaintStruct);
var
  ...
procedure PaintIt(Item: PTool); far;
  ...
begin
  if Orientation = tbHidden then Exit;          (* nuova *)
  ...
  DeleteDC(MemDC);
end;
```

ra. L'implementazione era però incompleta (non avevamo ancora previsto barre inferiori o invisibili) e dichiaratamente inefficiente (non si faceva nessuno sforzo per impedire il tremolio dell'immagine sullo schermo).

È giunto il momento di fare sul serio.

Il nuovo metodo è riprodotto nella figura 8; le differenze rispetto a quello originario sono tante e tali, che conviene evitare il confronto col vecchio e procedere direttamente alla illustrazione del nuovo.

Ricordiamo che il metodo può essere chiamato con due scopi: se si desidera solo conoscere le coordinate della *client area* come ridotta a causa della presenza di barre, si usa la costante *DenyRepaint* per *Msg.WParam*; se si vuole ridisegnare tutte le finestre «figlie» della *frame window*, barre e *client area*, si usa invece la costante *AllowRepaint* (questo è quanto avviene quando la *frame window* deve rispondere ad un messaggio *wm_Size*). In entrambi i casi, in *Msg.LParam* viene passato un puntatore ad una variabile di tipo *TRect* contenente le coordinate iniziali della *client area*; ad ogni esecuzione del metodo, le coordinate di quel rettangolo vengono modificate, in modo da costringere la *client area* a lasciare spazio nella *frame window* per barre o altri oggetti che rispondano al messaggio *am_CalcParentClientRect*.

Si usano sei variabili locali, quattro di tipo *TRect* e due di tipo *Integer*. Quanto alle prime, *R* viene avvalorata con le coordinate della *client area*, *TB* con quelle della barra; *NewTB* conterrà le coordinate della barra ridisegnata secondo le dimensioni assunte dalla *client area* (a causa di un cambiamento nelle dimensioni della *frame window*, oppure della visualizzazione di un'altra barra); *InvTB* conterrà le coordinate della porzione della barra che va ridisegnata, nel caso non sia necessario ridisegnarla tutta. Le variabili *CX* e *CY* vengono usate per la larghezza e l'altezza della barra.

Il metodo si avvale anche della variabile d'istanza *Rect* (aggiunta alla dichiarazione di *TToolbar*); come vedremo, in essa si tiene memoria delle coordinate che la barra assume dopo l'esecuzione del metodo, in modo da poter confrontare, in occasione di una successiva esecuzione, coordinate vecchie e nuove.

Se la barra non è invisibile (nel qual caso il metodo si limita ad azzerare la variabile *Rect*) vengono in primo luogo avvalorate le variabili *R* e *TB*.

Se la barra è orizzontale, si assegna a *CX* la larghezza della *client area*, mantenendo in *CY* l'altezza. Ciò è sufficiente a rettificare le coordinate della *client area* (aggiungendo *CY-1* a *R.Top* o sottraendo la medesima quantità da *R.Bottom*; si toglie 1 da *CY* in quanto, come visto il mese scorso, la barra ha un bordo di cui vogliamo vedere solo il lato adiacente alla *client area*; è questo il motivo dei ricorrenti «-1» o «+2»). Se *Msg.WParam* vale *AllowRepaint*, si assegnano a *NewTB* nuove coordinate, calcolate in relazione a quelle della *client area*, e si riposiziona conseguentemente la barra con la funzione *SetWindowPos*. Questa

```
procedure TToolbar.AMCalcParentClientRect(var Msg: TMessage);

var
  R,                                (* client area *)
  TB,                               (* toolbar esistente *)
  NewTB,                            (* nuovo toolbar *)
  InvTB: TRect;                     (* parte del toolbar da ridisegnare *)
  CX, CY: Integer;                  (* larghezza e altezza di NewTB *)

begin

  if Orientation = tbHidden then begin
    SetRect(Rect, 0, 0, 0, 0);
    Exit;
  end;

  R := PRect(Msg.LParam)^;
  GetWindowRect(HWindow, TB);

  if (Orientation and (tbHorizontal or tbDownHorizontal) <>0) then begin
    CX := R.Right - R.Left;
    CY := TB.Bottom - TB.Top;
    if Msg.WParam = AllowRepaint then begin
      if Orientation = tbHorizontal then
        NewTB.Top := R.Top - 1
      else
        NewTB.Top := R.Bottom - CY + 1;
      SetRect(NewTB, R.Left - 1, NewTB.Top, CX + 2, CY);
      SetWindowPos(HWindow, 0,
        NewTB.Left, NewTB.Top, NewTB.Right, NewTB.Bottom,
        swp_NoZOrder or swp_NoRedraw);
      Attr.W := CX;
      if (NewTB.Left <> Rect.Left) or (NewTB.Top <> Rect.Top) then
        InvalidateRect(HWindow, nil, True)
      else if NewTB.Right > Rect.Right then begin
        SetRect(InvTB, Rect.Right - 2, -1, NewTB.Right, CY);
        InvalidateRect(HWindow, @InvTB, True);
      end;
      Rect := NewTB;
    end;
    if Orientation = tbHorizontal then
      Inc(R.Top, CY - 1)
    else
      Dec(R.Bottom, CY - 1);
  end
    else begin (* tbLeftVertical o tbRightVertical *)
      CX := TB.Right - TB.Left;
      if Msg.WParam = AllowRepaint then begin
        if Orientation = tbLeftVertical then
          NewTB.Left := R.Left - 1
        else
          NewTB.Left := R.Right - CX + 1;
        SetRect(NewTB, NewTB.Left, R.Top - 1, CX, R.Bottom - R.Top + 2);
        SetWindowPos(HWindow, 0,
          NewTB.Left, NewTB.Top, NewTB.Right, NewTB.Bottom,
          swp_NoZOrder or swp_NoRedraw);
        if (NewTB.Left <> Rect.Left) or (NewTB.Top <> Rect.Top) then
          InvalidateRect(HWindow, nil, True)
        else if NewTB.Bottom > Rect.Bottom then begin
          SetRect(InvTB, -1, Rect.Bottom - 2, CX, NewTB.Bottom);
          InvalidateRect(HWindow, @InvTB, True);
        end;
        Rect := NewTB;
      end;
      if Orientation = tbLeftVertical then
        Inc(R.Left, CX - 1)
      else
        Dec(R.Right, CX - 1);
    end;

    PRect(Msg.LParam)^ := R;
  end;
```

Figura 8 - Il metodo AMCalcParentClientRect *completamente riscritto.*

viene chiamata con un parametro dato dalla combinazione delle costanti *swp_NoZOrder* e *swp_NoRedraw*; la prima evita che venga alterato l'ordine delle finestre, la seconda indica che non vogliamo ridisegnare la barra.

Prima di disegnarla, infatti, vogliamo verificare che ciò sia effettivamente necessario. Se sono cambiate le coordinate del vertice superiore sinistro (i campi *Left* e *Top* di *NewTB* sono diversi da quelli memorizzati in *Rect*), occorre ridisegnare tutto mediante una chiamata di *InvalidateRect* con **nil** come secondo parametro. In caso contrario, può essere necessario intervenire solo se la barra si è allargata (il campo *Right* di *NewTB* è maggiore di quello di *Rect*), disegnando il rettangolo mancante; se la barra è divenuta meno larga, infatti, il rettangolo in eccesso verrà semplicemente ignorato (in quanto cadrà fuori della *frame window* o verrà coperto da altro, ad esempio da una barra verticale).

Al termine, si tiene memoria delle nuove coordinate della barra assegnando *NewTB* a *Rect*.

Si procede analogamente se la barra è verticale; naturalmente, se non occorre ridisegnare tutta la barra, si procede a disegnare il rettangolo mancante solo quando la barra si è allungata verso il basso.

L'applicazione

È necessario intervenire anche sul codice della finestra che ospita le nostre barre. Parlo genericamente di «finestra», invece che di *frame window*, perché è possibile aggiungere una o più barre anche ad applicazioni non MDI.

Per cominciare, infatti, vi propongo una unit EDITWIN che modifica la classe *TEditWindow*, compresa nella unit standard OSTDWNDS, per predisporla all'uso di barre strumenti (figura 9).

La nuova classe *TNewEditWindow* ridefinisce tre metodi della classe madre.

Il metodo *WMSize* è il risultato di una combinazione dei metodi *WMSize* e *RedoClientRect* proposti in MFILEAPP.PAS, il demo che accompagna la unit TOOLBAR.

Con qualcosa in più.

Si assesta in primo luogo lo *Scroller* e si aggiusta la variabile *Attr*, come in *TWindow.WMSize*; ottenute poi le coordinate della *client area*, queste vengono aggiustate inviando a tutte le finestre «figlie» il messaggio *am_CalcParentClientRect* (al quale rispondono, ovviamente, solo le due barre). La vera e propria finestra di editing (la variabile *Editor*, istanza di *TEdit*) viene ridisegnata in modo da coprire tutto lo spazio della *client area* lasciato libero dalle barre.

Fin qui le parti di *WMSize* tratte da MFILEAPP.PAS. Occorre prevedere espressamente, tuttavia, il caso che la *client area* assuma un'altezza negativa; ciò può verificarsi se si ridimensiona la finestra principale (con il mouse o con l'opzione *Ridimensiona* del *system menu*), riducendone l'altezza fino a renderla insufficiente a contenere entrambe le barre. Ne risultano effetti estetici piuttosto sgradevoli, in quanto le due barre tendono a disegnarsi l'una sull'altra. Si prevede, quindi, una variabile *OldR*, cui assegnare le coordinate della *client area* prima che queste vengano modificate dalle barre; se l'altezza risultante è negativa, si aumenta quella originaria e si ripete. Grazie a questo artificio, la barra inferiore sparirà dalla *client area* se questa si riduce troppo.

Non basta: nonostante le cure con cui si è riscritto il metodo *AMCalcParentClientRect* di *TToolbar*, se ci si limitasse a quanto fin qui visto le barre ver-

```
unit EditWin;

interface

uses WinTypes, WinProcs, OWindows, OStdWnds, Toolbar;

type
  PNewEditWindow = ^TNewEditWindow;
  TNewEditWindow = object(TEditWindow)
    function GetClassName: PChar; virtual;
    procedure GetWindowClass(var AWndClass: TWndClass); virtual;
    procedure WMSize(var Msg: TMessage); virtual wm_First + wm_Size;
  end;

implementation

function TNewEditWindow.GetClassName: PChar;
begin
  GetClassName := 'NewEditWindow';
end;

procedure TNewEditWindow.GetWindowClass(var AWndClass: TWndClass);
begin
  inherited GetWindowClass(AWndClass);
  AWndClass.Style := AWndClass.Style and not (cs_VRedraw or cs_HRedraw);
end;

procedure TNewEditWindow.WMSize(var Msg: TMessage);
var
  OldR, R: TRect;
procedure NotifyChildren( P: PWindow ); far;
  begin
    if P^.HWindow <> 0 then
      SendMessage(P^.HWindow, am_CalcParentClientRect,
                  AllowRepaint, Longint(@R));
  end;
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;
  GetClientRect(HWindow, R);
  OldR := R;
  ForEach(@NotifyChildren);
  if R.Bottom < R.Top then begin
    Inc(OldR.Bottom, R.Top - R.Bottom);
    R := OldR;
    ForEach(@NotifyChildren);
  end;
  SetWindowPos(Editor^.HWindow, 0, R.Left - 1, R.Top - 1,
               R.Right - R.Left + 2, R.Bottom - R.Top + 2,
               swp_NoZOrder);
end;

end.
```

Figura 9 - Una unit che ridefinisce la classe TEditWindow, *compresa nella unit OSTDWNDS, per predisporla all'uso di barre strumenti.*

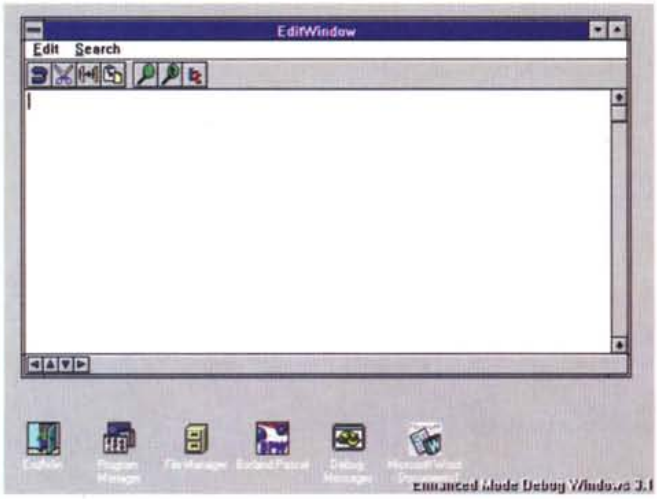
rebbero ridisegnate con un sensibile tremolio ogni volta che si cambiassero le dimensioni della finestra. Questo avviene perché le finestre di ObjectWindows appartengono a «classi» (attenzione: qui si tratta delle classi di Windows, non di quelle della OOP!) che vengono registrate con gli stili *cs_HRedraw* e *cs_VRedraw*, che comportano il ridisegno completo di tutta la *client area* quando cambiano le dimensioni della finestra principale.

È necessario, quindi, prevedere un'apposita classe (di Windows) per *TNewEditWindow*, escludendo quegli stili. A ciò provvedono i metodi *GetClassName* e *GetWindowClass*.

Per mettere la unit EDITWIN all'opera, sono sufficienti poche modifiche al file EDITAPP.PAS: basta aggiungere le

```
program TextEditor;
(*$R EDITAPP.RES*)

uses WinTypes, WinProcs, OWindows, OStdWnds, EditWin, Toolbar;

type

  ...

  PMyEditWindow = ^TMyEditWindow;
  TMyEditWindow = object(TNewEditWindow)
    Toolbar1: PToolbar;
    Toolbar2: PToolbar;
    constructor Init(AParent: PWindowsObject; ATitle: PChar);
  end;

constructor TMyEditWindow.Init(AParent: PWindowsObject; ATitle: PChar);
begin
  inherited Init(AParent, ATitle);
  Attr.Menu := LoadMenu(HInstance, 'EditCommands');
  Toolbar1 := New(PToolbar, Init(@Self, 'TOOLBAR_1', tbHorizontal));
  Toolbar2 := New(PToolbar, Init(@Self, 'TOOLBAR_2', tbDownHorizontal));
end;

...
```

Figura 10 - Le modifiche da apportare ad EDITAPP.PAS per dotare la finestra principale di due barre strumenti.



Il text editor del demo EDITAPP integrato con due barre strumenti: una lungo il bordo superiore della client area, *l'altra lungo il bordo inferiore.*

Figura 11 - Il file EDITAPP.RC da cui conviene iniziare per definire le risorse per il programma EDITAPP modificato come indicato nella figura 10. Includendo OWINDOWS.INC, è semplice definire le due barre. Si può poi utilizzare il Resource WorkShop per trasformare il file in EDITAPP.RES ed aggiungere le bitmap.

```
#include "owindows.inc"

TOOLBAR_1 TOOLBARDATA LOADONCALL MOVEABLE DISCARDABLE
BEGIN
  8
  750
  cm_EditUndo
  751
  cm_EditCut
  752
  cm_EditCopy
  753
  cm_EditPaste
  0
  8
  754
  cm_EditFind
  755
  cm_EditFindNext
  756
  cm_EditReplace
END


TOOLBAR_2 TOOLBARDATA LOADONCALL MOVEABLE DISCARDABLE
BEGIN
  4
  1001
  0
  1002
  0
  1003
  0
  1004
  0
END
```

unit TOOLBAR e EDITWIN alla clausola **uses**, modificare il constructor della classe *TMyEditWindow* come indicato nella figura 10, predisporre un file di risorse EDITAPP.RES.

Quanto a quest'ultimo, vi propongo nella figura 11 il corrispondente file EDITAPP.RC: conviene prendere le mosse da questo per scrivere la definizione delle due barre, poi convertirlo in RES ed aggiungere le bitmap numerate da 750 a 756 e da 1001 a 1004. Come si vede nella figura, alle bitmap del primo gruppo sono associati i comandi dei menu *Edit* e *Search*, a quelle del secondo gruppo, più piccole (11x11), non sono associati comandi: stanno lì solo per mostrare come sia possibile sia visualizzare una barra lungo il bordo inferiore della finestra, sia usare pulsanti di dimensioni diverse da 20x20.

Per ora ci fermiamo qui. Vedremo, nei prossimi appuntamenti, come realizzare una riga di stato in grado di coesistere con le barre della unit TOOLBAR, come inserire il tutto in applicazioni MDI, come associare messaggi esplicativi nella riga di stato alla pressione dei pulsanti della barra strumenti. MC

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

Per entrare nel mondo delle tecnologie e dei prodotti dedicati all'ascolto in automobile c'è una strada sicura: le pagine di Audiocarstereo. Recensioni dagli alti contenuti tecnici, prove di installazione, un vasto panorama di aggiornamenti mensili - anche sui prezzi - sono una lettura obbligata per i professionisti del settore come per i semplici appassionati, e costituiscono il migliore osservatorio per ascoltare al meglio. Infine le sezioni dedicate alla telefonia cellulare, ai test sugli antifurto, alle recensioni musicali completano Audiocarstereo, accompagnando chiunque voglia percorrere in auto la strada dell'alta fedeltà.

La strada migliore per l'alta fedeltà in auto.