Beating the System: CoolBars – The Route To Ultra-Cool Apps

by Dave Jewell

Yes, yes, I know. Last month, I mentioned the possibility of devoting some more time to the COM-based interfaces inside the Windows95 and NT 4.0 shell, but I've decided to defer that a bit longer. Having just spent the last couple of days getting Delphi to work with CoolBars, I thought it would be really "cool" if I showed how to add CoolBars to your own Delphi programs.

So What's A CoolBar?

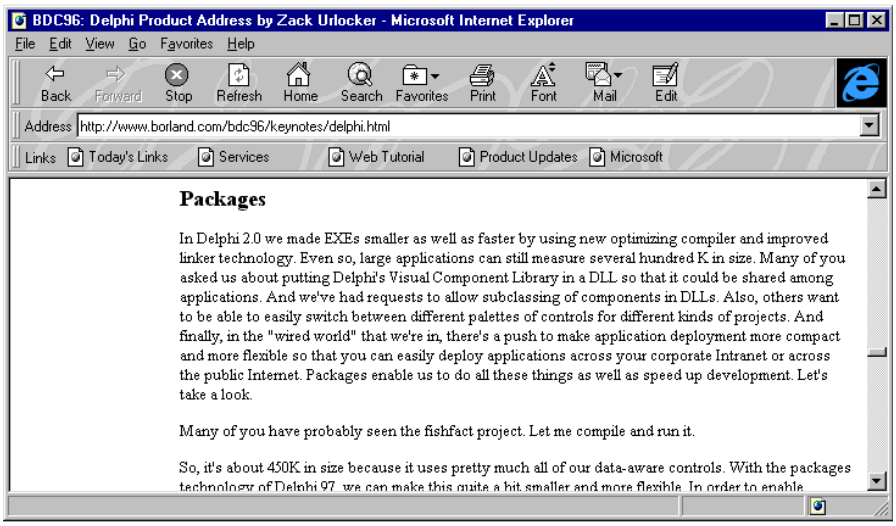
If you've got a copy of Microsoft's Internet Explorer 3.0 installed on your system, then you've already seen CoolBars in action. Take a look at Figure 1 where you can see Internet Explorer displaying the Web page at

```
http://www.borland.com/bdc96/
  keynotes/delphi.html
```

Incidentally, the reason I mention this Web page specifically is because it's an official Borland Web page that provides quite a bit of interesting information on the upcoming Delphi97 product. It's a keynote speech put together by Anders Hejlsberg (sadly, no longer with Borland) and Zack Urlocker. Enjoy!

Anyway, I digress. If you look at the screen shot, you'll see that there are three separate rows, or 'bands' of controls between the menu bar at the top of the window and the client area proper. This is what Microsoft have christened a CoolBar. Yes, it's a ghastly name, but don't let it put you off...

A CoolBar is basically a container: it contains one or more other controls, just like a property sheet or a group box. To be strictly accurate, what a CoolBar really contains is one or more bands and it's the bands, in turn, that contain the controls you place into them.



➤ *Figure 1: Here's Microsoft Explorer sporting a CoolBar complete with several 'bands' of components. This may look tricky, but actually it's very easy: especially when you do it the Delphi way!*

As you'll know if you've played around with Internet Explorer, you can place all the bands on one row, individually resize them and pull the CoolBar 'down' so as to increase the number of available rows.

The CoolBar control is implemented inside the COMCTL32.DLL: it's this Windows system component which implements most of the functionality of the Common Controls library. If you've used Visual Basic much, you'll know that it comes with a set of OCX controls which purport to be ActiveX controls; in actual fact, the ActiveX components are just wrappers around this library. In the code presented in this month's column, you must have the new COMCTL32.DLL library in order for things to work. If Internet Explorer 3.0 (or later) is installed, then you're in business. On my system, COMCTL32.DLL is dated 15th October 1996, and it's 379,152 bytes long.

I first discovered that the CoolBar was implemented in the COMCTL32.DLL (and so accessible to other applications) when I read about it in the October 1996 issue of *Microsoft Systems Journal*. Since all magazines have a lead time of at least several weeks, it followed that the author of this article must have had 'inside information'. Such was indeed the case: the article was authored by a support engineer within Microsoft. Armed with this information, I thought that it would be fun to build a Delphi application that used the CoolBar control. However, when I examined the article in more detail (and downloaded the accompanying source code) I discovered that the all important header files were missing. It was obviously time to reach for my trusty Windows dis-assembler.

Unfortunately, this sort of situation isn't by any means unusual. As with the CTL3D library, Microsoft have a history of adding new user interface features to their applications some time before telling other developers how to access those same capabilities. Well, that's their prerogative, but having had the stick dangled in front of me, I was determined to get my hands on the carrot...

CoolBars Step By Step

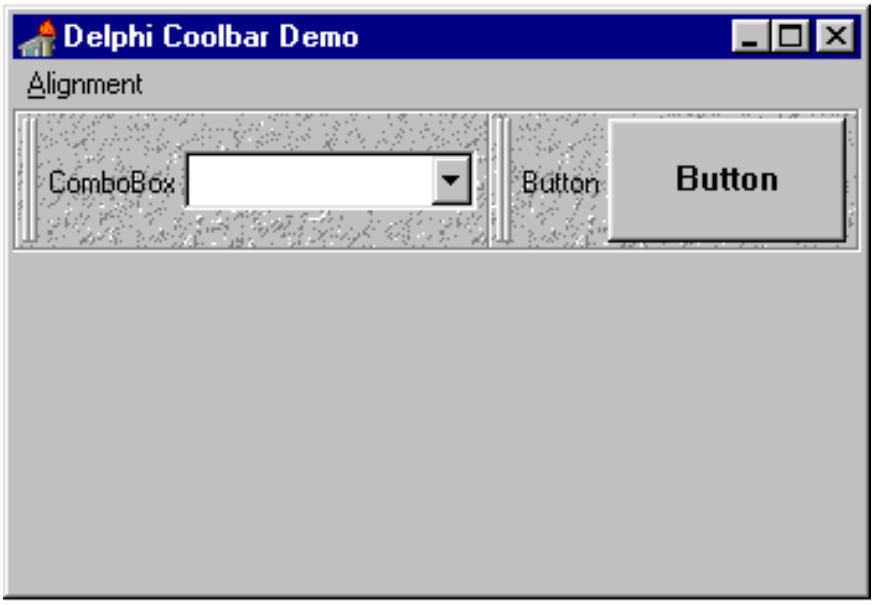
The first step in creating a CoolBar is to call the new `InitCommonControlsEx` API routine. Like most of the constants and structure definitions I'm using in this article, the declaration for this routine hasn't yet made it into Delphi's COMMCTRL unit. Consequently, you must use the declaration that I've provided below. This is a critically important routine because it initialises the COMCTL32 library ready for use by your application. The `InitCommonControlsEx` entry point doesn't exist in older versions of the library, which means that if you try and run my code on an older system, the Windows EXE loader will refuse to start the application, complaining that there's an undefined dynalink reference.

If you go ahead and build a Coolbar application based around the code I've included here, then you need to ask yourself whether the program might inadvertently be executed on a system that doesn't have the new COMCTL32 library. Just letting the system display an error message about undefined dynalinks isn't very user-friendly.

A better solution is to link to `InitCommonControlsEx` at run time rather than at link time. You could do that by using the `GetProcAddress` API routine to determine if the required routine is exported by the library. If you need more help with this, look at the way in which VCL interfaces with the CTL3D32 library (hint: open the FORMS.PAS file and search for `LoadLibrary`).

The `InitCommonControlsEx` routine is similar to the old `InitCommonControls` API call, except that it takes a pointer to a simple data structure. This structure contains a series of bit flags which tell the Common Controls library which window classes are going to be used by the application. Doing things like this is more efficient, because the library code only needs to initialise the window classes that are going to be used. With the old system, everything was initialised, whether used or not.

You can see my CoolBar demo application running in Figure 2.



➤ *Figure 2: Here's our first crack at a Delphi program with CoolBar support. This is essentially a Delphi re-write of Microsoft's sample code. Although it looks pretty good, the code is inflexible and you can't add arbitrary controls to the CoolBar.*

This has pretty much the same functionality as the original Microsoft sample application except that, because it's written in Delphi instead of barefoot C, there's a great deal less source code to worry about. Listing 1 shows the source code to the CoolBar unit which implements the real meat of the code. It needs to be said at the outset that this isn't exactly a general purpose unit: it's "hard-wired" to add specific controls to the CoolBar. I'll explain later how to make things more general purpose.

What might surprise you about Microsoft's CoolBar control is the fact that you can only place a single control into each band. This isn't at all clear from the aforementioned article but I came to this conclusion after a certain amount of experimentation. The reason that Internet Explorer appears to be able to place several buttons into a single band is because it's using another type of control, `ToolBarWindow32`, which is also implemented through the COMCTL32 library. Internet Explorer first creates a toolbar, fills it full of buttons and then places a single toolbar control into each band.

There are only two interface routines to the CoolBar unit: `AddCoolBar` which adds a CoolBar control to a designated window, and `AlignCoolBar` which forces the control to one of the four sides of the parent window. The `AddCoolBar` routine kicks off by initialising a windows style word according to the type of CoolBar that's required: horizontal or vertical. It then calls the `CreateWindowEx` routine which actually creates the child window, using the passed window handle as the parent. The choice of parent is important since, as with all Windows child windows, its the parent window which receives notification messages sent by the child. Finally, if the window creation call was successful, then the `PopulateCoolBar` and `AlignCoolBar` routines are called to populate the CoolBar with controls and align it to the desired position.

The `AddCoolBar` routine receives three parameters which provide IDs for the CoolBar itself, and for the combo box and control button that are created. Why are these IDs important? The reason is that, with the largely API-level implementation of a CoolBar given here, you need the child ID of a window in order to figure out which child window sent the notification. For example, if you've got five push buttons in a window, they will each send `WM_COMMAND` messages to the

```
unit CoolBar;
interface
{$R *.RES }
uses Messages, Windows, Forms, SysUtils, CommCtrl;
type
  TCoolBarPosition = (cbp_Left, cbp_Top, cbp_Right,
                      cbp_Bottom);
function AddCoolBar(Wnd: hWnd; Pos: TCoolBarPosition; id1,
                    id2, id3: Integer): hWnd;
procedure AlignCoolBar(CoolBarWnd: hWnd;
                       Pos: TCoolBarPosition);
implementation
const
  { Flags for dwICC bitmask in TICCEx record }
  ICC_ListView_Classes =     $00000001;
  ICC_TreeView_Classes =     $00000002;
  ICC_Bar_Classes      =     $00000004;
  ICC_TAB_Classes      =     $00000008;
  ICC_UpDown_Class     =     $00000010;
  ICC_Progress_Class   =     $00000020;
  ICC_HotKey_CLASS     =     $00000040;
  ICC_Animate_CLASS    =     $00000080;
  ICC_Win95_Classes    =     $000000FF;
  ICC_Date_Classes     =     $00000100;
  ICC_UserEx_Classes   =     $00000200;
  ICC_Cool_Classes     =     $00000400;
  // Common Control Styles
  CCS_Vert          =     $00000080;
  CCS_Left          =     (CCS_VERT or CCS_TOP);
  CCS_Right         =     (CCS_VERT or CCS_BOTTOM);
  CCS_NoMoveEx      =     (CCS_VERT or CCS_NOMOVEY);
  RBIM_Style        =     $00000001;
  RBIM_ImageList    =     $00000002;
  RBIM_Background   =     $00000004;
  RBS_ToolTips      =     $00000100;
  RBS_VarHeight     =     $00000200;
  RBS_BandBorder    =     $00000400;
  RBS_FixedOrder    =     $00000800;
  RBBS_Break        =     $00000001;
  RBBS_FixedSize    =     $00000002;
  RBBS_KeepHeight   =     $00000004;
  RBBS_ChildEdge    =     $00000004;
  RBBS_Hidden       =     $00000008;
  RBBS_NoVert       =     $00000010;
  RBBS_FixedBmp     =     $00000020;
  RBBIM_Style       =     $00000001;
  RBBIM_Colors      =     $00000002;
  RBBIM_Text        =     $00000004;
  RBBIM_Image       =     $00000008;
  RBBIM_Child       =     $00000010;
  RBBIM_ChildSize   =     $00000020;
  RBBIM_Size        =     $00000040;
  RBBIM_Background  =     $00000080;
  RBBIM_Id          =     $00000100;
  RB_InsertBandA    =     wm_User +1;
  RB_DeleteBand     =     wm_User +2;
  RB_GetBarInfo     =     wm_User +3;
  RB_SetBarInfo     =     wm_User +4;
  RB_GetBandInfo    =     wm_User +5;
  RB_SetBandInfoA   =     wm_User +6;
  RB_SetParent      =     wm_User +7;
  RB_EraseDark      =     wm_User +8;
  RB_Animate        =     wm_User +9;
  RB_InsertBandW    =     wm_User +10;
  RB_SetBandInfoW   =     wm_User +11;
  RB_GetBandCount   =     wm_User +12;
  RB_GetRowCount    =     wm_User +13;
  RB_GetRowHeight   =     wm_User +14;
  RB_InsertBand     =     RB_InsertBandA;
  RB_SetBandInfo    =     RB_SetBandInfoA;
  RBN_HeightChange  =     -831;
  ReBarClassName    =     'ReBarWindow32';
type
  TICCEx = record
    dwSize: DWord;
    dwFlags: DWord;
  end;
  TRebarInfo = record
    cbSize: UInt;
    fMask: UInt;
    fStyle: UInt;
    himl: HImageList;
    hbmBack: HBitmap;
  end;
  TRebarBandInfo = record
    cbSize: UInt;
    fMask: UInt;
    fStyle: UInt;
    clrFore: TColorRef;
    clrBack: TColorRef;
    lpText: PChar;
    cch: UInt;
    iImage: Integer;
    hWndChild: hWnd;
    cxMinChild: UInt;
    cyMinChild: UInt;
    cx: UInt;
    hbmBack: hBitmap;
    wID: UInt;
  end;
function InitCommonControlsEx(var ICCRec: TICCEx): Boolean;
  stdcall; external 'COMCTL32.DLL';
```

```
procedure CoolBarInit;
var ICCRec: TICCEx;
begin
  ICCRec.dwSize := sizeof(ICCRec);
  ICCRec.dwFlags := ICC_Cool_Classes;
  InitCommonControlsEx(ICCRec);
end;
procedure AlignCoolBar(CoolBarWnd: hWnd;
  Pos: TCoolBarPosition);
var rcForm, rcCoolBar: TRect;
    x, y, width, height: Integer;
begin
  GetClientRect(CoolBarWnd, rcCoolBar);
  GetClientRect(GetParent(CoolBarWnd), rcForm);
  if Pos = cbp_Right then
    x := rcForm.right - rcCoolBar.right
  else
    x := 0;
  if Pos = cbp_Bottom then
    y := rcForm.bottom - rcCoolBar.bottom
  else
    y := 0;
  if Pos in [cbp_Left, cbp_Right] then
    width := rcCoolBar.right
  else
    width := rcForm.right;
  if Pos = cbp_Bottom then
    height := rcCoolBar.bottom
  else
    height := rcForm.Bottom;
  MoveWindow(CoolBarWnd, x, y, width, height, True);
end;
procedure AddBand1(CoolBarWnd: hWnd; idCombo: Integer);
var
  rc: TRect;
  i: Integer;
  style: DWord;
  hWndCombo: hWnd;
  rbbi: TRebarBandInfo;
  szBuff: array [0..100] of Char;
begin
  { Create a combo box and fill it with junk }
  style := ws_Visible or ws_Child or ws_Border or
    ws_ClipChildren or ws_ClipSiblings or ws_TabStop or
    ws_VScroll;
  style := style or cbs_DropDown or cbs_AutoHScroll;
  hWndCombo := CreateWindow('combobox', Nil, style,
    0, 0, 100, 200, CoolBarWnd, idCombo, hInstance, Nil);
  for i := 0 to 24 do
    SendMessage(hWndCombo, cb_AddString, 0,
      LongInt(StrPCopy(szBuff, Format('Item %d', [i+1]))));
  FillChar(rbbi, sizeof(rbbi), 0);
  GetWindowRect(hWndCombo, rc);
  with rbbi do begin
    cbSize := sizeof(rbbi);
    fMask := RBBIM_Child or RBBIM_ChildSize or RBBIM_Id or
      RBBIM_Style or RBBIM_Colors or RBBIM_Text or
      RBBIM_Background;
    cxMinChild := rc.right - rc.left;
    cyMinChild := rc.bottom - rc.top;
    clrFore := GetSysColor(Color_BtnText);
    clrBack := GetSysColor(Color_BtnFace);
    fStyle := RBBS_ChildEdge or RBBS_FixedBmp;
    wID := idCombo;
    hwndChild := hWndCombo;
    lpText := 'ComboBox';
    hbmBack := LoadBitmap(hInstance, PChar(1));
    iImage := 0;
  end;
  SendMessage(CoolBarWnd, RB_InsertBand, -1, LongInt(@rbbi));
end;
procedure AddBand2(CoolBarWnd: hWnd; idButton: Integer);
var
  rc: TRect;
  hWndButton: hWnd;
  rbbi: TRebarBandInfo;
begin
  { Create a button control }
  hWndButton := CreateWindow('Button', 'Button', WS_Child or
    BS_PushButton, 0, 0, 100, 50, CoolBarWnd, idButton,
    hInstance, Nil);
  FillChar(rbbi, sizeof(rbbi), 0);
  GetWindowRect(hWndButton, rc);
  with rbbi do begin
    cbSize := sizeof(rbbi);
    fMask := RBBIM_Child or RBBIM_ChildSize or RBBIM_Id or
      RBBIM_Style or RBBIM_Colors or RBBIM_Text or
      RBBIM_BackGround;
    cxMinChild := rc.right - rc.left;
    cyMinChild := rc.bottom - rc.top;
    clrFore := GetSysColor(COLOR_BtnText);
    clrBack := GetSysColor(COLOR_BtnFace);
    fStyle := RBBS_ChildEdge or RBBS_FixedBmp;
    wID := idButton;
    hwndChild := hWndButton;
    lpText := 'Button';
    hbmBack := LoadBitmap(hInstance, PChar(1));
  end;
  SendMessage(CoolBarWnd, RB_InsertBand, -1, LongInt(@rbbi));
end;
{ *** CONTINUED ON FACING PAGE *** ⟶ }
```

parent window whenever they're pressed. The parent window uses the child ID (sent as part of the WM_COMMAND message) to discriminate between the different push buttons, between any other child controls and between different menu selections. This sort of stuff is usually behind the scenes as far as the Delphi VCL programmer is concerned, but we need to deal with it here.

PopulateCoolBar just calls a couple of other routines: AddBand1 and AddBand2. You add a control to the CoolBar by first filling in a data structure which describes a band, placing the control into the band and then inserting the band into the CoolBar. To see how this works, take a look at the code for AddBand1. This creates a combo box using the CoolBar as the parent window and using the window ID specified by idCombo. The combo box is filled with some junk string items and a data structure of type TRebarBandInfo is then initialised. I'll explain the meaning of the various fields in this data structure as we work through the code.

As with most recently introduced data structures, cbSize specifies the size of the whole data structure. This value is checked internally by the control library to ensure that both parties agree regarding the version of data structure in use! The fMask field contains a large number of bit flags, each of which is set to True to indicate that an associated field is valid. Thus, you set RBBIM_Child if you're specifying a window handle in hWndChild, RBBIM_ChildSize if you're setting cxMinChild or cyMinChild, and so on. These last two fields allow you to set minimum horizontal and vertical values for a child window. This is important where you don't want a child window to be shrunk less than a certain amount. The clrBack and clrFore fields allow you to specify a background colour for the band and a foreground colour for any text that will be drawn on the band. The lpText field is used to specify display text to be included in the band.

The bit flags to the fStyle field have a significant effect on the appearance of the CoolBar. I've used two flags here: RBBS_ChildEdge tells the control library to draw a border around each control. Without this flag, the top and bottom edges of the button control would be right next to the top and bottom of the bands, which would look unpleasant.

The RBBS_FixedBmp flag is used in conjunction with the hbmBack bitmap which is used to specify a background bitmap for the band. You can see this used in Internet Explorer where a 'squiggly-line' bitmap is drawn behind all four bars *[Oh, so that's how Microsoft **intended** it? Yeukh! Editor]*. The RBBS_FixedBmp flag tells the control library to draw the bitmap as a single, unbroken image running through all the bands. If the bitmap doesn't fill the area occupied by all the bands it is tiled. As a corollary to this, resizing an individual band doesn't move the underlying bitmap. When the RBBS_FixedBmp flag isn't set, the bitmap moves along with the band. Finally, the RB_InsertBand message is sent to the CoolBar, passing it the address of the TRebarBandInfo data structure. This adds the band to the CoolBar.

The AlignCoolBar routine is straightforward enough. It simply calculates the X, Y, Width and Height parameters for the CoolBar window for each of the four possible alignment positions. The Windows API MoveWindow routine is then called to move the CoolBar to its new position.

Coolness In Action

Listing 2 shows the form unit from which the CoolBar code is called. The window handle of the CoolBar is stored as a private variable in the form's class declaration, as is the current alignment position. When the form is created, the FormCreate handler gets triggered. This calls the CoolBarInit routine, passing it an initial alignment value of cbp_Top. This value is stored and then the AddCoolBar routine is called, passing it the three ID values for the different controls that are required.

It's important that the CoolBar is notified each time the parent window size changes. If this wasn't done, the CoolBar would (for example) remain the same length as the form was widened. It's easy to do this with Delphi: just define a FormResize routine and call AlignCoolBar from inside the event handler.

The original Microsoft code included a small menu which allows the user to set the CoolBar position by making one of four choices. I've included this functionality into the sample program but, once again, the equivalent Delphi code is a fraction of the size of the original (ok, ok, that's enough gratuitous pro-Delphi propaganda!) A simple and convenient technique for discriminating between multiple menu items is to use the Tag field. From an efficiency point of view, this is much, more efficient than writing your menu handler like this:

➤ *Facing page and below: Listing 1*

```
{ *** CONTINUED FROM FACING PAGE *** }
procedure PopulateCoolBar(
  CoolBarWnd: hWnd; id2, id3: Integer);
begin
  AddBand1(CoolBarWnd, id2);
  AddBand2(CoolBarWnd, id3);
end;
function AddCoolBar(Wnd: hWnd; Pos: TCoolBarPosition; id1,
  id2, id3: Integer): hWnd;
var style: DWord;
begin
  style := ws_Visible or ws_Child or ws_Border or
    ws_ClipChildren or ws_ClipSiblings;
  style := style or RBS_ToolTips or RBS_VarHeight or
    rbs_BandBorder;
  style := style or CCS_NoDivider or CCS_NoParentAlign;
  if Pos in [cbp_Left, cbp_Right] then
    style := style or CCS_Vert;
  Result := CreateWindowEx(ws_ex_ToolWindow, ReBarClassName,
    Nil, style, 0, 0, 200, 100, Wnd, id1, hInstance, Nil);
  if Result <> 0 then begin
    PopulateCoolBar(Result, id2, id3);
    AlignCoolBar(Result, Pos);
  end;
end;
{ Initialisation code for the unit }
begin
  CoolBarInit;
end.
```

```
if Sender = MenuItem1 then
  ...
else if Sender = MenuItem2 then
  ...
else if Sender = MenuItem3 then
  ... [etc]
```

The `Left1Click` handler is shared by all four menu items. It simply examines the tag field to see if it agrees with the current CoolBar alignment position. If not, the existing CoolBar is destroyed and then recreated in the new position. The `AlignmentClick` routine is called whenever the top-level alignment menu item is clicked. This happens immediately before the alignment menu is displayed. This is a good place to add the code which checks or un-checks menu items according to the current alignment setting.

Finally, the `WMCommand` routine receives `WM_COMMAND` messages for the main form. This includes notification messages sent from the combo box and the button in the CoolBar. The routine checks for messages for these controls by looking at the `ItemId` field of the `TWMCommand` data structure and, if appropriate, displays a dialog box to indicate that a notification was received. The `WMCommand` handler also receives messages from the menu items, but these are handled by calling the inherited handler at the end of the routine.

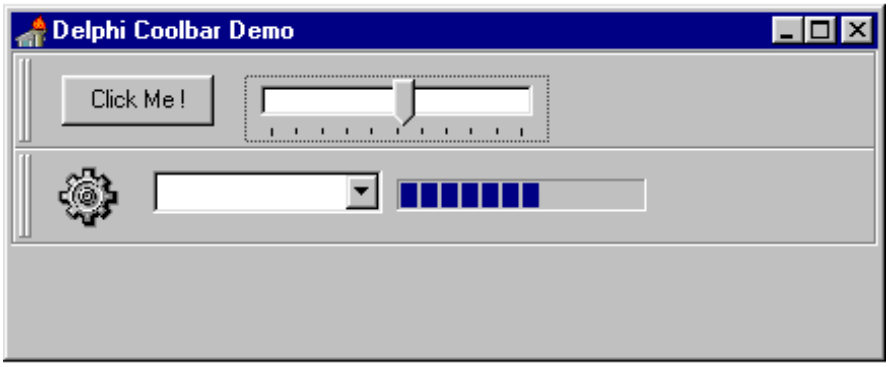
Doing It The Delphi Way

At this point, you're maybe thinking 'Ok, but so what?'. After all, I've shown you how to create controls on the fly using the API and how to hard-wire them into a CoolBar. The API way of doing things means that you have to muck around with control IDs and event notifications, rather than simply writing event handlers in the normal way. Is there are a better, more Delphi-compatible way of doing things?

Well, yes, there is. It turns out that it's very easy to modify the `AddBand` code I've presented so that, instead of creating an on the fly API control, you pass a `TWinControl` parameter instead. Within the `AddBand` code, you can obtain the API window handle of the passed control by using the `Handle` property in the usual way. The beauty of this scheme is that it immediately eliminates the restriction of having only one control per band. If you want more than one control in a band, you can just create a `TPanel` component in the normal way at design time, fill it with other Delphi components and then, at run time, plug the window handle of the `TPanel` into the `AddBand` routine. Things look best if you make your panels borderless. That way, it really does look as if there are multiple controls in each band. Another advantage of this approach is that because we're now using Delphi VCL components, we can just implement ordinary event handlers instead of keeping track of arbitrary numbers of control IDs.

➤ *Listing 2*

```
unit coolform;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ExtCtrls, Menus, Coolbar, Tabs;
type
  TForm1 = class(TForm)
  MainMenu1: TMainMenu;
  Alignment: TMenuItem;
  Left1: TMenuItem;
  Top1: TMenuItem;
  Right1: TMenuItem;
  Bottom1: TMenuItem;
  procedure FormCreate(Sender: TObject);
  procedure FormResize(Sender: TObject);
  procedure AlignmentClick(Sender: TObject);
  procedure Left1Click(Sender: TObject);
  private
    hWndCoolbar: hWnd;
    CoolBarPos: TCoolBarPosition;
    procedure CoolBarInit(Pos: TCoolBarPosition);
    protected
    procedure WMCommand(var Msg: TWMCommand); message wm_Command;
  public
  end;
var Form1: TForm1;
implementation
const
  id_CoolBar = 1111;
  id_Combo   = 2222;
  id_Button  = 3333;
{$R *.DFM}
procedure TForm1.CoolBarInit(Pos: TCoolBarPosition);
begin
  CoolBarPos := Pos;
  hWndCoolbar := AddCoolBar(Handle, Pos, id_CoolBar, id_Combo, id_Button);
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
  CoolBarInit(cbp_Top);
end;
procedure TForm1.FormResize(Sender: TObject);
begin
  AlignCoolBar(hWndCoolbar, CoolBarPos);
end;
procedure TForm1.AlignmentClick(Sender: TObject);
begin
  Left1.Checked   := CoolBarPos = cbp_Left;
  Top1.Checked    := CoolBarPos = cbp_Top;
  Right1.Checked  := CoolBarPos = cbp_Right;
  Bottom1.Checked := CoolBarPos = cbp_Bottom;
end;
procedure TForm1.Left1Click(Sender: TObject);
begin
  with Sender as TMenuItem do
    if Ord(CoolBarPos) <> Tag then begin
      DestroyWindow(hWndCoolbar);
      CoolBarInit(TCoolBarPosition(Tag));
      FormResize(Self);
    end;
end;
procedure TForm1.WMCommand(var Msg: TWMCommand);
var Str: String;
begin
  Str := '';
  case Msg.ItemID of
    id_Button :  Str := 'You clicked the button';
    id_Combo  :  if Msg.NotifyCode = cbn_SelChange then
                   Str := 'You changed the combo selection';
  end;
  if Str <> '' then
    MessageDlg(Str , mtInformation, [mbOK], 0);
  Inherited;
end;
end.
```



➤ *Figure 3: Now we're motoring! With just a few simple changes, it's possible to incorporate an entire Delphi Panel component into the CoolBar, complete with any subsidiary components it might contain.*

The proof of the pudding is in the eating, and you can see the pudding being consumed in Figure 3. This is a rather whimsical user interface which demonstrates two bands, each populated with Delphi components. I've removed the CoolBar alignment functionality from this second sample program because `TPanel` controls don't look particularly impressive when squashed into a vertical CoolBar. This means that if anything the code is shorter than it was before.

This month's companion disk includes two ZIP files: COOL1.ZIP and COOL2.ZIP. The first ZIP file contains the API-level code presented in Listings 1 and 2. The second ZIP file contains the simplified, Delphi-compatible code that I've just been discussing. Both ZIP files include complete Delphi projects and executables. Don't expand both ZIP files into the same directory because they use the same filenames: the one will overwrite the other.

I suspect that Borland will provide support for CoolBars in Delphi97, but if you can't wait for Delphi97, or you're happy to stay with Delphi 2, then the code presented here will help you easily incorporate CoolBars into your own applications. Do remember my opening caveat though: if there's any doubt about the version of COMCTL32.DLL installed on the host system, then you should be sure to include this redistributable file in your setup script.

Dave Jewell is a freelance consultant/programmer and technical journalist specialising in system-level Windows and DOS work. He is the author of *Instant Delphi Programming* published by Wrox Press. You can contact Dave as DaveJewell@msn.com, DSJewell@aol.com or 102354,1572 on CompuServe