Tips & Tricks

*Edited by Mike Orriss. **Please send your Tips direct to Mike at mjo@compuserve.com***

Interfaced Classes

The new Interfaces support in Delphi 3 is excellent and is useful for 'ordinary' programming as well as for the obvious COM and DCOM support. Soon, however, I needed to base an interface on a `TList` class and the easy answer of adding and implementing the three required methods (`QueryInterface`, `_AddRef` & `_Release`) seemed a bit messy. Mike Scott (mikes@compuserve.com) came to the rescue by supplying me with an `IInterfaceList` and the code shown in Listing 1 is his, but with my formatting:

Contributed by Mike Scott, (mikes@compuserve.com)

Limiting A Form's Width And Height

On occasion, you'll want to limit the dimensions that a form can be resized. To do this, you must capture the `WM_GETMINMAXINFO` message. This message is sent to a window whenever its size or position is about to be modified. It is within this message handler that you can

➤ *Listing 1*

```
unit IntfList;
interface
uses Classes;
type
  IInterfaceList =
    interface['{BEC6B350-40CF-11D1-91D9-00C0DF119781}']
    // property implementation methods
    function  GetCount: integer;
    function  GetItem(Index: integer):IUnknown;
    procedure SetItem(Index: integer;
      const Value: IUnknown);
    // methods
    procedure Add(const Unknown: IUnknown);
    procedure Clear;
    procedure Delete(Index: integer);
    procedure Exchange(Index1,Index2: integer);
    function  IndexOf(const Unknown: IUnknown): integer;
    procedure Insert(Index: integer;
      const Unknown: IUnknown);
    procedure Move(CurIndex,NewIndex: integer);
    procedure Pack;
    procedure Remove(const Unknown: IUnknown);
    // properties
    property  Count: integer read GetCount;
    property  Items[Index: integer]: IUnknown
      read GetItem write SetItem; default;
  end;
  TInterfaceList = class(TInterfacedObject,IInterfaceList)
  private
    fItems: TList;
  protected
    function  GetCount: integer; virtual;
    function  GetItem(Index: integer): IUnknown; virtual;
    procedure SetItem(Index: integer;
      const Value: IUnknown); virtual;
  public
    constructor Create;
    destructor Destroy; override;
    procedure Add(const Unknown: IUnknown);
    procedure Clear;
    procedure Delete(Index: integer);
    procedure Exchange(Index1,Index2: integer);
    function  IndexOf(const Unknown: IUnknown): integer;
    procedure Insert(Index: integer;
      const Unknown: IUnknown);
    procedure Move(CurIndex,NewIndex: integer);
    procedure Pack;
    procedure Remove(const Unknown: IUnknown);
    property  Count: integer read GetCount;
    property  Items[Index: integer]: IUnknown
      read GetItem write SetItem; default;
  end;
implementation
constructor TInterfaceList.Create;
begin
  inherited Create;
  FItems:= TList.Create;
end;
destructor TInterfaceList.Destroy;
begin
  Clear;
  FItems.Free;
  inherited Destroy;
end;
function  TInterfaceList.GetCount: integer;
begin
  Result:= FItems.Count;
end;
function  TInterfaceList.GetItem(Index: integer): IUnknown;
begin
  Result:= IUnknown(FItems[Index]);
end;
procedure TInterfaceList.SetItem(Index: integer;
  const Value: IUnknown);
var
  Unknown: IUnknown;
begin
  Unknown:= IUnknown(FItems[Index]);
  if Assigned(Unknown) then Unknown._Release;
  FItems[Index]:= pointer(Value);
  if Assigned(Value) then Value._AddRef;
end;
procedure TInterfaceList.Add(const Unknown: IUnknown);
begin
  FItems.Add(pointer(Unknown));
  if Assigned(Unknown) then Unknown._AddRef;
end;
procedure TInterfaceList.Insert(Index: integer;
  const Unknown: IUnknown);
begin
  FItems.Insert(Index,pointer(Unknown));
  if Assigned(Unknown) then Unknown._AddRef;
end;
procedure TInterfaceList.Delete(Index: integer);
var Unknown: IUnknown;
begin
  Unknown:= IUnknown(FItems[Index]);
  if Assigned(Unknown) then Unknown._Release;
  FItems.Delete(Index);
end;
function  TInterfaceList.IndexOf(const Unknown: IUnknown):
  integer;
begin
  Result:= FItems.IndexOf(pointer(Unknown));
end;
procedure TInterfaceList.Move(CurIndex,NewIndex: integer);
begin
  FItems.Move(CurIndex,NewIndex);
end;
procedure TInterfaceList.Pack;
begin
  FItems.Pack;
end;
procedure TInterfaceList.Remove(const Unknown: IUnknown);
var Index: integer;
begin
  Index:= FItems.IndexOf(pointer(Unknown));
  if Index >= 0 then Delete(Index);
end;
procedure TInterfaceList.Clear;
var
  ix: integer;
  Item: pointer;
begin
  for ix := FItems.Count-1 downto 0 do begin
    Item:= FItems[ix];
    if Assigned(Item) then IUnknown(Item)._Release;
  end;
  FItems.Clear;
end;
procedure TInterfaceList.Exchange(Index1,Index2: integer);
begin
  FItems.Exchange(Index1,Index2);
end;
end.
```

specify the limits for the window's width and height. The message handler for the WM_GETMINMAXINFO message looks like this:

```
procedure WMGetMinMaxInfo(var Message:
  TWMGetMinMaxInfo); message WM_GetMinMaxInfo;
```

The TWMGetMinMaxInfo structure is defined in the unit Message as:

```
TWMGetMinMaxInfo = record
  Msg: Cardinal;
  Unused: Integer;
  MinMaxInfo: PMinMaxInfo;
  Result: Longint;
end;
```

The field of interest here is MinMaxInfo. The PMinMaxInfo type is defined in the Windows unit as:

```
PMinMaxInfo = ^TMinMaxInfo;
TMinMaxInfo = packed record
  ptReserved: TPoint;
  ptMaxSize: TPoint;
  ptMaxPosition: TPoint;
  ptMinTrackSize: TPoint;
  ptMaxTrackSize: TPoint;
end;
```

The ptReserved field can be ignored. ptMaxSize specifies the maximum width and height for the window whenever its borders are completely extended. ptMaxPosition specifies the left and top position of the window whenever it is maximised. ptMinTrackSize and ptMaxTrackSize specify the minimum and maximum widths and heights for the window when the window's borders are used to resize it.

You can modify the value for any of the above mentioned fields to specify a sizing limit for any window or form. The example code in Listing 2 illustrates this.

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls;
type
  TForm1 = class(TForm)
  public
    procedure WMGetMinMaxInfo(var Message:
      TWMGetMinMaxInfo); message WM_GetMinMaxInfo;
  end;
var
  Form1: TForm1;
implementation
procedure TForm1.WMGetMinMaxInfo(var Message:
  TWMGetMinMaxInfo);
begin
  with Message.MinMaxInfo^ do begin
    ptMaxPosition.x := 100;
    ptMaxPosition.y := 100;
    ptMaxSize.x := 640;
    ptMaxSize.y := 480;
    ptMaxTrackSize.x := 640;
    ptMaxTrackSize.y := 480;
    ptMinTrackSize.x := 300;
    ptMinTrackSize.y := 200;
  end;
  Message.Result := 0;
end;
{$R *.DFM}
end.
```

➤ *Listing 2*

Contributed by Xavier Pacheco (xpacheco@xap.cnchost.com)

Matching Strings

If you do not have the Delphi 3 Client Server version, read no further as this tip is not for you (unfortunately). There is a new TMask class defined in MASKS.PAS in SOURCE\INTERNET that provides an extremely useful Boolean function called MatchesMask. As an example, suppose you wanted to check if a string contained a particular filename, regardless of position or case:

```
If (MatchesMask(Edit1.Text, '*abc.Pas*') then ...
```

Or suppose that you wanted to check against ABC1.PAS thru ABC5.PAS:

```
If (MatchesMask(Edit1.Text, '*abc[1-5].Pas*') then ...
```

Or you just wanted to check for filenames starting with 'AB' and ending with 3 or 5:

```
If (MatchesMask(Edit1.Text,'*ab*[35].Pas*') then ......
```

Or not ending in 3 or 5:

```
If (MatchesMask(Edit1.Text,'*ab*[!35].Pas*') then ......
```

As well as * as a multi-character wildcard, you can also use ? for a single character. Thus to check a fully qualified filename for a name of four characters and a standard three character extension you would supply a mask of *\????.???*. The documentation for this function is quite sparse: it does not appear in the VCL reference but I did find an entry under MatchesMask in the online help.

Contributed by Mike Orriss (mjo@compuserve.com)

FDBGrid Current Cell Position

How can you figure out the co-ordinates (specifically, left and top) of the currently selected Cell in a DBGrid?

A neat trick here is to make use of the DBGrid's DrawDataCell event handler. The example below displays the X and Y co-ordinates of the selected grid cell (relative to the grid's Left and Top properties:

```
procedure TMainForm.DBGrid1DrawDataCell(
  Sender: TObject; const Rect: TRect;
  Field: TField; State: TGridDrawState);
begin
   if gdFocused in State then
      Panel1. Caption :=
         Format('%d,%d',[Rect.Left,Rect.Top]);
end;
```

Contributed by Mike Orriss (mjo@compuserve.com)

```
procedure SetupFieldsAndOpenDataset(DataSet: TDataSet);
var
  FieldNum,DefNum: Integer;
begin
  with DataSet do begin
    if Active then
      Close ;
    FieldDefs.Update; {dataset must be closed}
    { look up each pre-defined TField in
      DataSet.FieldDefs: }
    for FieldNum:= FieldCount-1 downto 0 do
      with Fields[FieldNum] do begin
        DefNum := FieldDefs.IndexOf(FieldName);
        if DefNum < 0 then
          raise Exception.CreateFmt(
            'Field "%s" not in dataset "%s"',
            [FieldName,Dataset.Name]);
        {adjust size property:}
        Size := FieldDefs[DefNum].Size;
      end;
    Open;
  end;
end;
```

➤ *Listing 3*

Horizontal Scrolls For ListBox

When working In Delphi 1.0, it can be frustrating that there is no horizontal scroll in the TListBox. One technique is to set the TListBox hint to the item under the cursor in the OnMouseMove event and show that hint on a status bar.

A better alternative is to add scrollbars. The SetListHorzScrolls routine below checks the contents of a TListBox to see if any of the items are wider than the box, if so, a horizontal scroll is added.

```
procedure SetListHorzScrolls(lst: TListBox);
var
   i: integer;
begin
   { check if horizontal scroll bars
     are necessary }
   for i := 0 to lst.Items.Count - 1 do
      if (lst.Canvas.TextWidth(
        lst.Items.Strings[i]) > lst.Width)
        then begin
        SendMessage(lst.Handle,
          LB_SetHorizontalExtent, 1000, longint(0));
        Break;
      end;
end;
```

Contributed by Tom Corcoran (tomc@unitime.com)

Oldest And Newest File

Recently I had to write a routine that assigned a unique file name. When the maximum number of unique file names were used it had to display the oldest file matching the given mask.

I came up with the routine shown in Listing 4, GetFileInDir. It uses a function of mine, FindFile, that was listed in Issue #17, page 62 (for convenience, FINDFILE.PAS is included on this month's disk as well). By passing different dates as the first parameter one can find the oldest, newest or closest file to a given date for any given file mask in any given directory.

Contributed by Tom Corcoran (tomc@unitime.com)

```
function GetFileInDir(date: TDateTime;
  path, fileMask: string): string;
{ to find newest file pass Now as date
  to find oldest file pass 0 as date
  to find newest file on 19 Sep 97 pass 20 Sep 97 as date}
var
  slstFiles: TStringList;
  dateTime, savedDateTime: TDateTime;
  index: integer;
  i: integer;
  fileDate: longInt;
begin
  slstFiles := TStringList.Create;
  try
    (* issue #17, page 62
    procedure FindFile(initialPath : string;
      fileMask : string;      { mask to look for   } }
      recursive: boolean;     { search subdirectories?  }
      stopOnFirstMatch: boolean;
      showFile: boolean;      { add file or dir to list } }
      files: TStringList);    { addmatch(es) to list  }
    *)
    FindFile(path, fileMask, False, False, True,
      slstFiles);
    if slstFiles.Count > 0 then begin
      dateTime := 0;
      savedDateTime := 0;
      if date = 0 then begin
        { find oldest file }
        savedDateTime := now;
        for i := 0 to slstFiles.Count - 1 do begin
          fileDate := FileAge(slstFiles.Strings[i]);
          dateTime := FileDateToDateTime(fileDate);
          if dateTime < savedDateTime then begin
            savedDateTime := dateTime;
            index := i;
          end;
        end;
      end else
        { find file nearest (<= date) }
        for i := 0 to slstFiles.Count - 1 do begin
          fileDate := FileAge(slstFiles.Strings[i]);
          dateTime := FileDateToDateTime(fileDate);
          if ((dateTime > savedDateTime) and
            (dateTime < date)) then begin
            savedDateTime := dateTime;
            index := i;
          end;
        end;
        Result :=
          ExtractFileName(slstFiles.Strings[index]);
      end;
    else
     Result := '';
  finally
    slstFiles.Free;
  end;
end;
```

➤ *Listing 4*

Paint On A GroupBox

Ever wanted to paint on the Canvas of a Group Box? Canvas is a Protected property of TGroupBox and is thus inaccessible via normal methods.

However, you can declare a TGroupBox descendant that makes the Canvas property public, as shown in the code fragment below:

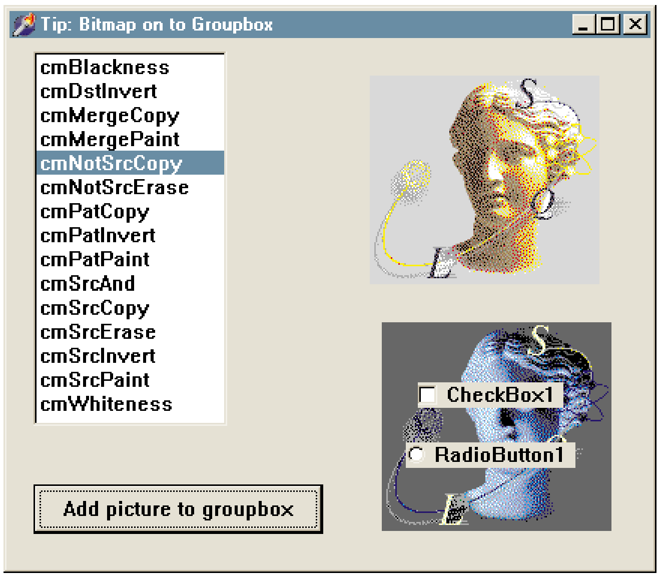
```
type
  TMyGroupBox = class(TGroupBox)
  public
     property Canvas;
  end;
procedure SomeProcedure;
begin
  ...
  with TMyGroupBox(GroupBox1).Canvas do
    CopyRect(ClipRect, Image1.Canvas, ClipRect);
  ...
end;
```

Listing 5 and Listing 6 are the form and unit files respectively for an example that shows the effects of

```
object frmMain: TfrmMain
  Left = 535
  Top = 143
  Width = 435
  Height = 378
  Caption = 'Tip: Bitmap on to Groupbox'
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -13
  Font.Name = 'System'
  Font.Style = []
  Position = poScreenCenter
  Scaled = False
  PixelsPerInch = 96
  TextHeight = 16
  object Image1: TImage
    Left = 240
    Top = 24
    Width = 153
    Height = 139
  end
  object GroupBox1: TGroupBox
    Left = 248
    Top = 188
    Width = 153
    Height = 139
    Caption = 'GroupBox1'
    TabOrder = 0
    object CheckBox1: TCheckBox
      Left = 24
      Top = 40
      Width = 97
      Height = 17
      Caption = 'CheckBox1'
      TabOrder = 0
    end
    object RadioButton1: TRadioButton
      Left = 16
      Top = 80
      Width = 113
      Height = 17
      Caption = 'RadioButton1'
      TabOrder = 1
    end
  end
  object Button1: TButton
    Left = 16
    Top = 296
    Width = 193
    Height = 33
    Caption = 'Add picture to groupbox'
    TabOrder = 1
    OnClick = Button1Click
  end
  object ListBox1: TListBox
    Left = 16
    Top = 8
    Width = 129
    Height = 249
    ItemHeight = 16
    Items.Strings = (
      'cmBlackness'
      'cmDstInvert'
      'cmMergeCopy'
      'cmMergePaint'
      'cmNotSrcCopy'
      'cmNotSrcErase'
      'cmPatCopy'
      'cmPatInvert'
      'cmPatPaint'
      'cmSrcAnd'
      'cmSrcCopy'
      'cmSrcErase'
      'cmSrcInvert'
      'cmSrcPaint'
      'cmWhiteness')
    TabOrder = 2
    OnClick = ListBox1Click
  end
end
```

➤ *Listing 5 (above), Listing 6 (below)*

```
unit Main;
interface
uses
  SysUtils,WinTypes,WinProcs,Messages,Classes,Graphics,
  Controls,Forms,Dialogs,StdCtrls,ExtCtrls;
type
  TMyGroupBox = class(TGroupBox)
  public
    property Canvas;
  end;
  TfrmMain = class(TForm)
    Image1: TImage;
    GroupBox1: TGroupBox;
    Button1: TButton;
    CheckBox1: TCheckBox;
    RadioButton1: TRadioButton;
    ListBox1: TListBox;
    procedure Button1Click(Sender: TObject);
    procedure ListBox1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  frmMain: TfrmMain;
  copMode: TCopyMode;
implementation
{$R *.DFM}
procedure TfrmMain.Button1Click(Sender: TObject);
begin
  with TMyGroupBox(GroupBox1).Canvas do begin
    CopyMode := copMode;
    CopyRect(ClipRect,Image1.Canvas,ClipRect);
  end;
end;
procedure TfrmMain.ListBox1Click(Sender: TObject);
begin
  case (Sender as TListBox).ItemIndex of
    0: copMode := cmBlackness;
    1: copMode := cmDstInvert;
    2: copMode := cmMergeCopy;
    3: copMode := cmMergePaint;
    4: copMode := cmNotSrcCopy;
    5: copMode := cmNotSrcErase;
    6: copMode := cmPatCopy;
    7: copMode := cmPatInvert;
    8: copMode := cmPatPaint;
    9: copMode := cmSrcAnd;
    10: copMode := cmSrcCopy;
    11: copMode := cmSrcErase;
    12: copMode := cmSrcInvert;
  end;
end;
end.
```



➤ *Figure 1*

the various Copy Modes when executing a `CopyRect` procedure. Figure 1 shows the technique in action.

Contributed by Mike Orriss (mjo@compuserve.com)

Radio Group And ActiveControl

A developer I spoke to recently had a radio group on a form and wanted to call context sensitive help when a user pressed `F1`. He set the `HelpContext`, but found that `ActiveControl.HelpContext` always returned zero. All the other controls worked just fine.

After some investigation I found the problem was that the `ActiveControl` is the `RadioButton` not the `RadioButtonGroup`. To get the desired behaviour you

```
unit OvrEdit;
interface
uses
  WinProcs, WinTypes, Messages, SysUtils, Classes,
  Graphics, Controls, Forms, Dialogs, StdCtrls, Mask,
  DBCtrls;
type
  TInsertModeChangeEvent = procedure(
    Sender:TDBEdit; isInsert:Boolean) of object;
  TOvrDBEdit = class(TDBEdit)
  private
    FInsertMode: boolean;
    FOnInsertModeChange: TInsertModeChangeEvent;
    procedure SetInsertMode(value: boolean);
  protected
    procedure KeyDown(var Key: Word; Shift: TShiftState);
      override;
    procedure KeyPress(var Key: Char); override;
  public
    { Public declarations }
  published
    property InsertMode: boolean
      read FInsertMode write SetInsertMode;
    property OnInsertModeChange: TInsertModeChangeEvent
      read FOnInsertModeChange write FOnInsertModeChange;
  end;
procedure Register;
implementation
procedure TOvrDBEdit.SetInsertMode(value: boolean);
begin
  if FInsertMode <> value then begin
    FInsertMode := value;
    if Assigned(FOnInsertModeChange) then
      FOnInsertModeChange(Self,FInsertMode);
  end;
end;
procedure TOvrDBEdit.KeyDown(var Key: Word;
  Shift: TShiftState);
begin
  inherited KeyDown(Key,Shift);
  if (Key = VK_INSERT) then
    InsertMode := not InsertMode;
end;
procedure TOvrDBEdit.KeyPress(var Key: Char);
begin
  inherited KeyPress(Key);
  if (not InsertMode) and not (Key in [#8,#0]) then
    SelLength := 1
  else
    SelLength := 0;
end;
procedure Register;
begin
  RegisterComponents('Samples',[TOvrDBEdit]);
end;
end.
```

➤ *Listing 7*

need to set the HelpContext for each of the contained buttons, as shown in this Form.Show event handler:

```
procedure TForm1.FormShow(Sender: TObject);
var
  c: integer;
begin
  with RadioGroup1 do begin
    for c := 0 to ControlCount - 1 do
      TRadioButton(Controls[c]).HelpContext :=
        HelpContext;
  end;
end;
```

Contributed by Mike Orriss (mjo@compuserve.com)

DBEdit with INS/OVR

Users are familiar with pressing the INS (insert) key to toggled insert/override mode. It would be handy if the DBEdit control could respond in this way.

The simple TOvrDBEdit component shown in Listing 7 handles the INS key in this manner. It publishes an InsertMode property and also provides an InsertModeChange event that makes it easy for you to indicate its status on a panel.

Contributed by Mike Orriss (mjo@compuserve.com)

Update Again: Navigation With Cursor Keys (Issues 26/27)

Hey, what's with all this API calling in order to navigate around controls on a form, is *nobody* aware of SelectNext?

Contributed by Gurbhajan Singh Bagh (gurb.bagh@mailexcite.com)

Fields Editor At Runtime

How can you carry out Field Editor functions during run time? If you specify the fields you want at design time, you can adjust their properties (eg Size) at run time. For example, the procedure shown in Listing 3 will adjust each TField.Size to match the actual field size of a dataset about to be opened.

Contribued by Mike Orriss (mjo@compuserve.com)

Push A Button

If you've tried to get a TButton to "press itself" without user intervention, you will have found that calling the OnClick handler won't do it.

You *can* cause a button to be pressed or unpressed by sending it a BM_SETSTATE message. To press a button:

```
Button1.Perform(BM_SETSTATE,1,0);
```

and to unpress a button:

```
Button1.Perform(BM_SETSTATE,0,0);
```

Similarly, to find out whether the button is pressed:

```
ButtonPressed :=
  (Button1.Perform(BM_GETSTATE, 0,0) = 1);
```

Contributed by Mike Orriss (mjo@compuserve.com)

Got some useful routines you'd like to share? Found any nasty "gotchas" recently?

Email Mike Orris, our Tips & Tricks Editor, with your goodies at mjo@compuserve.com