The Delphi CLINIC

Edited by Brian Long

Problems with your Delphi project?

Just email Brian Long, our Delphi Clinic Editor, on clinic@blong.com or write/fax us at The Delphi Magazine

Data Aware Control Request

Q Now that we have the nice date/time picker in Delphi 3, has anyone made a data aware version yet? I could do with one but don't know how to make it myself.

A I haven't encountered such a beast, but taking the challenge I read through the data aware controls chapter of *The Revolutionary Guide to Delphi 2* (Wrox Press, various authors) and set to work. The results are in DBPicker.Pas (Listing 1 for the class definition).

The general idea with making field-based data aware controls is to make a new class that contains a `TFieldDataLink` object (created and destroyed respectively by the constructor and destructor). The data link object is the real machine that drives a data aware control. The standard properties on offer from a data aware control (`ReadOnly`, `Field`, `DataField` and `DataSource`) are implemented as routines that return or set properties of the data link. Listing 2 shows the reader and writer for the `DataSource` property.

The `DataChange` and `UpdateData` methods (Listing 3) are event handlers for the field data link object. The `OnDataChange` event triggers when the value of the underlying field changes, either by the current record changing or the field itself being modified. If the data aware control hasn't been connected to a field in an open table, the control displays the current date or time.

`OnUpdateData` fires when the user changes the field value via the data aware control. The appropriate field object is given new values taken from the `Date` and `Time` properties of `TDBDateTimePicker`.

In common with any other component with published properties that can be connected to other components, the `TDBDateTimePicker` overrides the `Notification` method (Listing 4). To ensure notifications arrive of the data source is on another form or data module, its `FreeNotification` method is called when the datasource is assigned.

The `DataSource` property can be connected to the `TDataSource` component, but what if the component is deleted from the form designer? The `Notification` method is called when a component is added or deleted. It checks whether the data source component matches the one reflected by the `DataSource` property. If so, it sets the property to `nil` to avoid further problems.

➤ *Listing 1*

```
TDBDateTimePicker = class(TDateTimePicker)
private
  FDataLink: TFieldDataLink;
protected
  procedure SetDataSource(Value: TDataSource);
  function GetDataSource: TDataSource;
  procedure SetDataField(const Value: String);
  function GetDataField: String;
  function GetField: TField;
  procedure SetReadOnly(Value: Boolean);
  function GetReadOnly: Boolean;
  procedure DataChange(Sender: TObject);
  procedure UpdateData(Sender: TObject);
  procedure Notification(AComponent: TComponent;
    Operation: TOperation); override;
  procedure CMEnter(var Msg: TCMEnter); message cm_Enter;
  procedure CMExit(var Msg: TCMExit); message cm_Exit;
  procedure CNNotify(var Msg: TWMNotify); message cn_Notify;
public
  constructor Create(AOwner: TComponent); override;
  destructor Destroy; override;
  property Field: TField read GetField;
published
  property DataField: String read GetDataField write SetDataField;
  property DataSource: TDataSource read GetDataSource write SetDataSource;
  property ReadOnly: Boolean read GetReadOnly write SetReadOnly;
end;
```

```
procedure TDBDateTimePicker.SetDataSource(Value: TDataSource);
begin
  FDataLink.DataSource := Value
  if Value <> nil then Value.FreeNotification(Self);
end;
function TDBDateTimePicker.GetDataSource: TDataSource;
begin
  Result := FDataLink.DataSource
end;
```

➤ *Above: Listing 2*

➤ *Below: Listing 3*

```
procedure TDBDateTimePicker.DataChange(Sender: TObject);
begin
  if (Field = nil) or (FDataLink.DataSet.State = dsInsert) then
    case Kind of //If no data link is set up, show current date or time
      dtkDate: Date := SysUtils.Date;
      dtkTime: Time := SysUtils.Time
    end
  else
    case Kind of //Update control if field changes
      dtkDate: Date := FDataLink.Field.AsDateTime;
      dtkTime: Time := FDataLink.Field.AsDateTime
    end
end;
procedure TDBDateTimePicker.UpdateData(Sender: TObject);
begin
  if Kind = dtkDate then //Update field as necessary
    FDataLink.Field.AsDateTime := Date
  else
    FDataLink.Field.AsDateTime := Time
end;
```

Pretty much all that's left here are some Windows message handlers (shown in Listing 5), although the messages they handle are manufactured by Delphi. cm_Enter and cm_Exit are sent when a control gains and loses focus respectively. When the control gains focus it should ensure it has the current field value. When it loses focus it should make sure the underlying field object has the value shown in the control. If there is some validation problem, then focus is left on the control.

cn_Notify notifies the control when Windows sends its parent useful message information. In particular, when the calendar part of the control closes up the value is written to the field. Also when the control's value actually changes we ensure the data link knows the field value has been modified.

One final thing to note is that the destructor destroys the field data link object and then proceeds to set its object reference to nil (see Listing 6). This is a common practice to avoid access violations, but in a destructor one might think it overkill. However it is very important. During the inherited destruction the Notification method gets called again. If the object reference is left with a non-nil value after the data link has gone we will get an access violation.

For additional information on making data aware components in Delphi 3, choose Help | Developing Applications and scroll down until you see the section of links entitled Making a control data-aware.

Listing 4

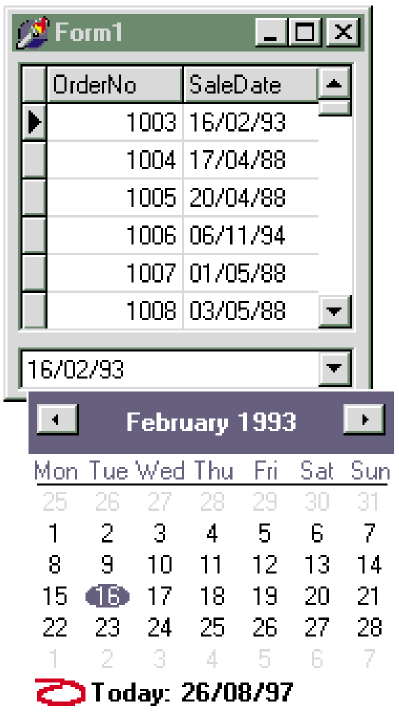
```
procedure TDBDateTimePicker.Notification(
  AComponent: TComponent; Operation: TOperation);
begin
  inherited;
  if (Operation=opRemove) and Assigned(FDataLink)
    and (AComponent=DataSource) then
      //Blank out datasource value if datasource is deleted from form
      DataSource := nil
end;
```

```
procedure TDBDateTimePicker.CMEnter(var Msg: TCMEnter);
begin
  Invalidate; //Make sure calendar is up to date upon entry
  inherited
end;
procedure TDBDateTimePicker.CMExit(var Msg: TCMExit);
begin
  try //Update other data aware controls when user tabs away
    FDataLink.UpdateRecord
  except
    SetFocus;
    raise
  end;
  inherited
end;
procedure TDBDateTimePicker.CNNotify(var Msg: TWMNotify);
begin
  case Msg.NMHdr^.Code of
    dtn_CloseUp: //Update all other data aware controls
      Perform(cm_Exit, 0, 0); //when calendar pops up
    dtn_DateTimeChange: //Update field value when calendar changes
      begin
        FDataLink.Edit;
        FDataLink.Modified;
      end;
  end;
  inherited;
end;
```

Above: Listing 5

Below: Listing 6

```
constructor TDBDateTimePicker.Create(AOwner: TComponent);
begin
  inherited;
  FDataLink := TFieldDataLink.Create;
  FDataLink.OnDataChange := DataChange;
  FDataLink.OnUpdateData := UpdateData
end;
destructor TDBDateTimePicker.Destroy;
begin
  FDataLink.Free;
  FDataLink := nil;
  inherited
end;
```



Form1

OrderNo	SaleDate
1003	16/02/93
1004	17/04/88
1005	20/04/88
1006	06/11/94
1007	01/05/88
1008	03/05/88

16/02/93

February 1993

Mon	Tue	Wed	Thu	Fri	Sat	Sun
25	26	27	28	29	30	31
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
1	2	3	4	5	6	7

Today: 26/08/97

Figure 1

I am not suggesting that this is necessarily a foolproof bug free implementation of a data-aware date-time picker, but it will hopefully give you something to work from. Figure 1 shows it in use against the SaleDate field in the grid.

Clipboard

Q I want to store some data in the clipboard in a proprietary format. However, I still want to be able to paste a human readable version into applications like Notepad and Word. How do I store multiple versions of my data in the clipboard?

A It all stems from using the TClipBoard object represented by the ClipBrd unit's ClipBoard variable in an appropriate way. Delphi 2 actually changed Clipboard from being a global variable to being a function in order to help the smart linker strip out code that used the ClipBrd unit but never referred to the Clipboard symbol.

Generally speaking, you write information into the clipboard in one format. To write a string, you simply write (and then read) Clipboard.AsText. To read or write a

TPicture or TBitmap object you call the Clipboard.Assign, passing the appropriate object as a parameter, or you call that object's Assign method, passing the Clipboard object as a parameter. So the following assignments copy text and a bitmap or metafile to and from the clipboard:

```
Clipboard.AsText := Edit1.Text;
Edit2.Text := Clipboard.AsText;
...
Clipboard.Assign(Image1.Picture);
Image1.Picture.Bitmap.Assign(
  Clipboard);
```

In order to write information to the clipboard in more than one format, you have to open up the clipboard, do all the storing and then close it (remembering to use a try/finally statement to ensure the clipboard definitely gets closed).

The ClipEg.Dpr project shows a simple example of this by copying text and a bitmap to the clipboard, and then pasting both back again (see Listing 7). To make it more interesting, we can make our own proprietary format to match the question more closely. Just making up an arbitrary data structure, we can see some code from ClipEg2.Dpr in Listing 8 that defines the custom record and registers a clipboard format.

Listing 9 also shows code that copies a version of the record (filled in from various user interface controls) and also a textual representation of the record into the clipboard. This involves using Windows routines to allocate memory (but not de-allocate it unless there is a problem) and locking and unlocking the memory block whilst writing into it.

The other half of the listing takes the information back from the clipboard, both the data record and the text version. Note that the memory block is owned by the clipboard so there is no need to free it.

Note also that you can check if a given information format is in the clipboard using Clipboard.HasFormat. Pre-defined formats include CF_TEXT for textual information, CF_COMPONENT for a Delphi component, CF_BITMAP for a bitmap and CF_PICTURE for any form of image supported by a TPicture object.

```
procedure TForm1.btnCopyClick(Sender: TObject);
begin
  Clipboard.Open;
  try
    Clipboard.AsText := Edit1.Text;
    Clipboard.Assign(Image1.Picture)
    { or Clipboard.Assign(Image1.Picture.Bitmap) }
  finally
    Clipboard.Close
  end
end;
procedure TForm1.btnPasteClick(Sender: TObject);
begin
  Clipboard.Open;
  try
    Edit2.Text := Clipboard.AsText;
    Image2.Picture.Assign(Clipboard) {or Image2.Picture.Bitmap.Assign(Clipboard)}
  finally
    Clipboard.Close
  end
end;
```

➤ *Above: Listing 7*

➤ *Below: Listing 8*

```
type
  TDataRecord = packed record
    Number1, Number2: Longint;
    AString: String[255];
  end;
var
  CF_CLINICRECORD: Word;
procedure TForm1.FormCreate(Sender: TObject);
begin
  CF_CLINICRECORD := RegisterClipboardFormat('Clinic Data Record');
end;
```

➤ *Listing 9*

```
procedure TForm1.btnCopyClick(Sender: TObject);
var
  Data: THandle;
  DataPtr: Pointer;
  DataRec: TDataRecord;
begin
  { Open clipboard for several formats to be added }
  Clipboard.Open;
  try
    { Allocate appropriate memory block }
    Data := GlobalAlloc(GMEM_MOVEABLE, SizeOf(DataRec));
    try
      DataPtr := GlobalLock(Data);  { Lock memory block }
      try
        { Set the record up }
        DataRec.Number1 := UpDown1.Position;
        DataRec.Number2 := UpDown2.Position;
        DataRec.AString := Edit3.Text;
        { Copy record to locked memory block }
        Move(DataRec, DataPtr^, SizeOf(DataRec));
        { Copy block into clipboard }
        Clipboard.SetAsHandle(CF_CLINICRECORD, Data);
        { Add a nice textual version into clipboard }
        Clipboard.AsText := Format('Number 1: %d'#13#10',
          'Number 2: %d'#13#10'AString: %s',
          [DataRec.Number1, DataRec.Number2,
          DataRec.AString])
      finally
        GlobalUnlock(Data)  { Unlock memory block }
      end
    except
      { Normally don't free this memory, it belongs to the
        clipboard. But if there's an error, then do the
        rightful thing }
      GlobalFree(Data);
      raise
    end
  finally
    Clipboard.Close     { Close clipboard }
  end
end;
procedure TForm1.btnPasteClick(Sender: TObject);
var
  Data: THandle;
  DataPtr: Pointer;
  DataRec: TDataRecord;
begin
  Clipboard.Open;
  try
    { Get memory block handle }
    Data := Clipboard.GetAsHandle(CF_CLINICRECORD);
    { Turn it into a pointer by locking it }
    DataPtr := GlobalLock(Data);
    try
      { Get data record and update UI }
      Move(DataPtr^, DataRec, SizeOf(DataRec));
      UpDown1.Position := DataRec.Number1;
      UpDown2.Position := DataRec.Number2;
      Edit3.Text := DataRec.AString;
      { Don't forget the textual version }
      Memo1.Text := Clipboard.AsText;
    finally
      GlobalUnlock(Data)    { Unlock memory block }
    end
  finally
    Clipboard.Close
  end;
end;
```