Using Form Components With Delphi 1

by Rohit Gupta

Did Borland lead you to believe that you couldn't have form components in Delphi 1? Well, you can, with a minute bit of trickery. I have been using this technique for months now without any problems at all. I discovered it when I got sick of duplicating code again and again...

Creating And Using The Form Component

The method is to create a unit (not a form) which consists of one or more form definitions, see Listing 1 (over the page) for one that I use as my default form. We will come back to what it does a little later. This is your form component.

To use the form component, first create a form normally (an example is shown in Listing 2). Then change the class of the form from `TForm` to `TNewForm`. Finally add `NewForms` to the uses clause. Listing 3 shows the finished product. That is all the jiggery pokery required!

Drawbacks

Well there had to be a catch somewhere! Firstly, there is no way to have access to a property editor, so the best way to set properties is in `CreateParams` (see Listing 4). Secondly, it's a pain if you want the form component to own components that are inherited, but it's not impossible.

Using NewForms

You are free to do what you wish with `NewForms`. It is my idea of what the default form should have been and is still evolving day by day. Currently it supports the following:

- A `Launch` procedure which creates a new form if one is not present, otherwise it displays, un-minimises and brings to the front the existing form.
- A conversion of `Enter` key-presses to `Tab` key-presses if required (ideal for data entry applications).
- An easy method of setting the minimum and maximum sizes and the position of the form when maximised.
- Automatic setting of `Action` to `caFree` in `FormClose` if the form is `MdiChild`.

➤ *Listing 2*

```
unit Unit1;
interface
uses Forms;
type
  TForm1 = class(TForm)
  private
  public
  end;
var Form1: TForm1;
implementation
{$R *.DFM}
end.
```

So how does it work? The `Create` constructor calls the inherited `Create` first. Next if `Set_Enter` has been set to `True` then it saves the current event handler for `OnKeyPress` and sets a new `OnKeyPress`. It does the same with

➤ *Listing 3: Sample Form1 using the NewForms component*

```
unit Unit1;
interface
uses Forms, NewForms;
type
  TForm1 = class(TNewForm)
  private
  public
  end;
var Form1: TForm1;
implementation
{$R *.DFM}
end.
```

➤ *Listing 4: Using NewForms fully*

```
unit Cln_Form;
interface
uses Forms, NewForms;
type
  TClnForm = class(TNewForm)
  procedure CreateParams(var Params : TCreateParams); override;
  private
  public
  end;
var ClnForm: TClnForm;
procedure Launch_Cln_Form;

implementation
{$R *.DFM}

procedure Launch_Cln_Form;
begin
  Launch(TClnForm,ClnForm);
end;

procedure TClnForm.CreateParams(var Params : TCreateParams);
begin
  if Cfg.MDI_App then              { I have it as a user customised option }
    { if Mdichild the form will close correctly }
    FormStyle := fsMdiChild;
  Set_Enter := True;               { Enable Cr to Tab conversion }
  Set_Max    := True;              { Set maximum size }
  Max_Width  := 600;
  Max_Height := 400;
  Set_Min    := True;              { Set minimum size }
  Min_Width  := 300;
  Min_Height := 200;
  Set_Pos    := True;              { Set Maximise position }
  Max_Left   := 50;
  Max_Top    := 50;
  inherited CreateParams(Params);
end;

end.
```

➤ *Listing 1*

```
unit NewForms;
{$R-,S-,I-,O-,F-,A+,U+,K+,W-,V+,B-,X+,T-,P+,L+,Y+,D-}
interface
uses
  WinTypes, WinProcs, Messages, Controls,
  Classes, Forms;
type
  TNewForm = class(TForm)
    constructor Create(AOwner : TComponent); override;
    procedure CreateParams(var Params : TCreateParams);
      override;
    procedure FormKeyPress(Sender: TObject; var Key:
      Char); virtual;
    procedure FormKeyDown(Sender: TObject; var Key:
      Word; Shift: TShiftState); virtual;
    procedure FormClose(Sender: TObject; var Action:
      TCloseAction); virtual;
  private
    procedure WMGetMinMaxInfo(var Message:
      TWMGetMinMaxInfo ); message WM_GETMINMAXINFO;
  public
    Set_Enter  : Boolean; { Set to convert CR to TAB }
    Set_Max    : Boolean;
    { Set and init following to fix max size }
    Max_Width,
    Max_Height : Word;
    Set_Min    : Boolean;
    { Set and init following to fix min size }
    Min_Width,
    Min_Height : Word;
    Set_Pos    : Boolean;
    { Set and init following to fix posn }
    Max_Left,
    Max_Top    : Word;
    OldOnkeyPress : TKeyPressEvent;
    OldOnkeyDown  : TKeyEvent;
    OldOnClose    : TCloseEvent;
    FormVar       : ^TForm;  { set by Launch }
  end;
type
  TFormClass = class of TNewForm;
  TFormPtr   = ^TNewForm;
{ Launch a form if not already open, else un-minimise
  and bring to front }
procedure Launch(LClass : TFormClass; var LForm);

implementation

constructor TNewForm.Create(AOwner : TComponent);
begin
  inherited Create(AOwner);
  if Set_Enter then begin
    OldOnKeyPress := OnKeyPress;
    OnKeyPress    := FormKeyPress;
    OldOnKeyDown  := OnKeyDown;
    OnKeyDown     := FormKeyDown;
    KeyPreview    := TRUE;
  end;
  OldOnClose := OnClose;
  OnClose    := FormClose;
end;

procedure TNewForm.CreateParams(
  var Params : TCreateParams);
begin
  if Set_Max then begin
    Params.Width   := Max_Width;
    Params.Height  := Max_Height;
  end;
  inherited CreateParams(Params);
end;

procedure TNewForm.FormKeyPress(
  Sender: TObject; var Key: Char);
begin
  case Key of
    #13 :
      begin
        PostMessage(Handle, WM_NEXTDLGCTL, 0, 0);
        Key := #0;
      end;
    else begin
      if assigned(OldOnKeyPress) then
        OldOnKeyPress(Sender, Key);
    end;
  end;
end;

procedure TNewForm.FormKeyDown(
  Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  case Key of
    Vk_Return :
      Key := Vk_Tab;
    else begin
      if assigned(OldOnKeyDown) then
        OldOnKeyDown(Sender, Key, Shift);
    end;
  end;
end;

procedure TNewForm.FormClose(
  Sender: TObject; var Action: TCloseAction);
begin
  Application.OnHint := nil;
  if FormStyle = fsMdiChild then
    Action := caFree;
  if assigned(OldOnClose) then
    OldOnClose(Sender,Action);
  if assigned(FormVar) then
    FormVar^ := nil;
end;

procedure TNewForm.WMGetMinMaxInfo(
  var Message :TWMGetMinMaxInfo);
begin
  with Message.MinMaxInfo^ do begin
    if Set_Max then begin
      if Max_Width <> 0 then
        ptMaxSize.X := Max_Width;  {Width when maximized}
      if Max_Height <> 0 then
        {Height when maximized}
        ptMaxSize.Y := Max_Height;
      {Tell windows you have changed minmaxinfo}
      Message.Result := 0;
    end;
    if Set_Pos then begin
      if Max_Left <> 0 then
        {Left position when maximized}
        ptMaxPosition.X := Max_Left;
      if Max_Top <> 0 then
       {Top position when maximized}
        ptMaxPosition.Y := Max_Top;
      {Tell windows you have changed minmaxinfo}
      Message.Result := 0;
    end;
    if Set_Min then begin
      if Min_Width <> 0 then
        ptMinTrackSize.X := Min_Width;  {Minimum width}
      if Min_Height <> 0 then
        ptMinTrackSize.Y := Min_Height; {Minimum height}
      {Tell windows you have changed minmaxinfo}
      Message.Result := 0;
    end;
    if Set_Max then begin
      if Max_Width <> 0 then
        ptMaxTrackSize.X := Max_Width;  {Maximum width.}
      if Max_Height <> 0 then
        ptMaxTrackSize.Y := Max_Height; {Maximum height.}
    end;
  end;
  inherited;
end;

procedure Launch(LClass : TFormClass; var LForm);
var
  LNewForm : TNewForm absolute LForm;
begin
  if LNewForm = nil then begin
    Application.CreateForm(LClass,LNewForm);
    LNewForm.FormVar := @LNewForm;
  end else with LNewForm do begin
    BringToFront;
    If WindowState = wsMinimized then
      WindowState := wsNormal;
  end;
end;

end.
```

OnKeyDown. Then it sets KeyPreview to True: without this the new form will not see the keystrokes first. Finally, it saves the OnClose handler and installs a new one.

CreateParams sets the Width and Height if Set_Max is True and then calls the inherited one.

FormKeyPress (the replacement for OnKeyPress) converts the Enter key-press to Tab and passes other keys to the original OnKeyPress. Recall that OnKeyPress was trapped only if the conversion was requested. FormKeyDown (the new OnKeyDown) does the same.

FormClose sets Application.OnHint to nil. I set the Application.OnHint in each form to a local ShowHint. This cleans it up for me. If FormStyle is fsMdiChild then it sets Action to caFree, thus closing the form. If OldOnClose is assigned then it calls it. Finally, if FormVar is assigned it sets it to nil. This allows Launch to determine if the form is already running.

WMGetMinMaxInfo is a Windows message method. It is called by Windows prior to sizing, minimising or maximising the form.

The Launch procedure checks to see if the LForm variable passed is assigned. If it's nil then the form is created and the address of this variable is stored in the local variable FormVar. This may seem convoluted but it is required. FormClose checks this variable and sets the LForm to nil via this pointer. If LForm is already assigned then the Launch procedure brings the form to the front and if it is minimised un-minimises it.

Conclusion

I hope you find that this technique eliminates repetitive chores but, more importantly, ensures that you do not forget to do the chore. I have many other form components inherited from this basic one in my projects, thus allowing me to have common code in one place.

Rohit Gupta lives and works in New Zealand. He specialises in Veterinary and Hairdressing software and can be contacted by email as rohit@cfl.co.nz