Animate windows as they are opened and closed

Ever wandered how to animate a window that's opening or closing in the same way that Windows uses when minimizing applications to the task bar?

We do it using the following Windows API function, defined in `Windows.pas`:

```
function DrawAnimatedRects(hwnd: HWND; idAni: Integer;
  const lprcFrom, lprcTo: TRect): BOOL; stdcall;
```

This function animates the window whose handle is *hwnd*. It is animated from the rectangle specified in *lprcFrom* to the rectangle specified by *lprcTo*. These rectangles are usually set to the the bounds of the source and destination windows, in screen co-ordinates. If you want the window's caption to be animated as when minizing to the task bar we pass *IDANI_CAPTION* as the *idAni* parameter. Other values can be passed, but we don't deal with them here – experiment yourself!

Example

Create an application that has a button that, when clicked, toggles a non-modal form open and closed. We will animate the dialog form when opened to make it appear to "grow" from the button and reverse the process when the dialog is closed.

Create a new application, name the main form *MainForm* and save it as `FmMain.pas`. Add a *TButton* to *MainForm* with caption "Show / Hide Dialog". Now add a new form named *DialogForm* and save it as `FmDialog.pas`. Make sure *DialogForm* is auto-created in the project file.

The main form requires three event handlers: an *OnClick* handler for the button and *OnClose* and *OnShow* handlers for the form. Add `FmDialog` to the uses clause of the main form's implementation section then implement the event handlers as follows:

```
procedure TMainForm.Button1Click(Sender: TObject);
begin
  // Toggle visibility of DialogForm. DialogForm's OnShow / OnHide event
  // handlers take care of animation
  DialogForm.Visible := not DialogForm.Visible;
end;

procedure TMainForm.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  // Close dialog (if open) with animation before app closes
  DialogForm.Hide;
end;

procedure TMainForm.FormShow(Sender: TObject);
begin
  // Set SourceCtrl here rather than FormCreate to ensure DialogForm created
  DialogForm.SourceCtrl := Button1;
end;
```

In the *FormShow* event handler we set a custom property of the dialog box form that holds a reference to the button – this is so the dialog box knows where to animate the dialog to and from, as we'll see in a moment.

In the dialog box unit add handlers for the form's *OnShow* and *OnHide* events and then complete the dialog form's interface section as follows:

```
type
  // Ensure this form is auto-created in project file
  TDialogForm = class(TForm)
    procedure FormHide(Sender: TObject);
    procedure FormShow(Sender: TObject);
  private
    fSourceCtrl: TControl;
    function GetSourceCtrlBounds: TRect;
    procedure AnimateDialog(const Src, Dest: TRect);
  public
```

```
    property SourceCtrl: TControl read fSourceCtrl write fSourceCtrl;
  end;

var
  DialogForm: TDialogForm;
```

Implement the methods of *TDialogForm* as follows:

```
procedure TDialogForm.AnimateDialog(const Src, Dest: TRect);
begin
  DrawAnimatedRects(Handle, IDANI_CAPTION, Src, Dest);
end;

procedure TDialogForm.FormHide(Sender: TObject);
begin
  if WindowState <> wsMinimized then
    AnimateDialog(BoundsRect, GetSourceCtrlBounds);
end;

procedure TDialogForm.FormShow(Sender: TObject);
begin
  if WindowState <> wsMinimized then
    AnimateDialog(GetSourceCtrlBounds, BoundsRect);
end;

function TDialogForm.GetSourceCtrlBounds: TRect;
begin
  Assert(Assigned(fSourceCtrl));
  Result.TopLeft := fSourceCtrl.ClientToScreen(Point(0, 0));
  Result.BottomRight := fSourceCtrl.ClientToScreen(
    Point(fSourceCtrl.Width, fSourceCtrl.Height)
  );
end;
```

AnimateDialog calls *DrawAnimatedRects* to perform the animation. *FormShow* and *FormHide* first check the dialog isn't minimized then call *AnimataDialog*, passing the bounding rectangle of the dialog and the button as the appropriate start and end points of the animation. *GetSourceCtrlBounds* simply calculates the bounding rectangle of the main form's button in screen co-ordinates. It uses the *SourceCtrl* property to get a reference to the button.

Author:	Peter Johnson
Contributor:	Peter Johnson
Added:	2007-08-14
Last updated:	2007-08-14