Hints With Attitude

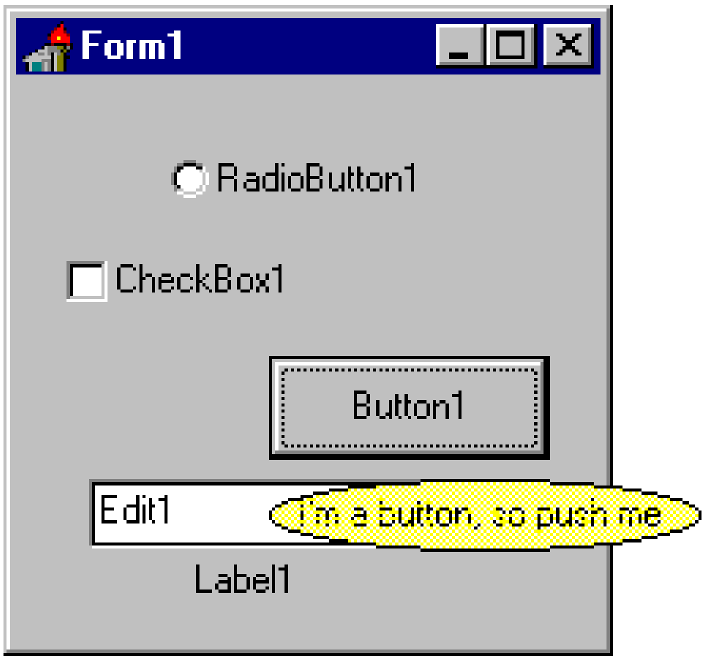
by Brian Long

A recent question sent to *The Delphi Clinic* seemed to necessitate a response longer than most so I thought a short article would be better. Here's the question:

"I know there are some properties that allow me to change the colour and delay of the tooltip/hint window, but how do I change it more drastically? For example, how would I create an elliptical hint window?"

In responding I'd firstly like to recap on the facilities available to us for simple modifications of the standard rectangular hint window. Then we'll look into changing the shape of it.

➤ *Hints, any shape you like...*



Customised Hints

Normally, you get hints (also known as tooltips) by setting a control's Hint property and then ensuring its own ShowHint property is True, either by setting it to True, or by ensuring its ParentShowHint is True and then making the parent control's ShowHint property True.

The Application object has a number of properties and events that affect hints. HintColor can be used to change the colour of all hints. This defaults to a value of $80FFFF in Delphi 1 and the system tooltip background colour (that is, clInfoBk) in Delphi 2.

The HintPause property specifies how long it takes for the first tooltip to be displayed. Delphi 1 defaults to 0.8 seconds and Delphi 2 reduces this to 0.5 seconds. Delphi 2 also has a HintHidePause property, used to hide the tooltip after a certain delay if the mouse remains stationary (2.5 seconds) and a HintShortPause property that is used when the mouse is moved from one control showing a tooltip to another one. This is set to 50 milliseconds and is used to reduce flicker if the mouse is quickly moved over a set of controls. Lastly, there is an Application.ShowHint property that can be used to disable tooltips for the entire application.

Additionally, the Application object has two events for use in the context of hints. OnShowHint is triggered just before a tooltip is about to be drawn and allows you to customise its colour, position and textual content and whether it will draw or not. It also informs you of the control the tooltip is for, so you can do extra customisation. OnHint is triggered whenever a tooltip could be drawn (even if it won't be) and also when you move around your menu structure. This event allows you do other types of hint generation such as hints on a status bar (run Delphi's Application Expert to see one of these set up). In the OnHint handler, the hint text that needs to be drawn is found in Application.Hint.

The project HINTDEMO.DPR on this month's disk shows an event handler for each of these. They are set up programmatically since the Application object does not appear on the Object Inspector.

If your hint has a pipe sign (|) in it, the text to the left of it will appear in the tooltip and the text to the right will be accessible by the OnHint handler. This allows you to have a tooltip and, say, a status bar hint with different text in. Some of the controls in HINTDEMO.DPR demonstrate this. The first part of the hint text is returned by the GetShortHint function and the second part by GetLongHint. if there is no pipe, they both return the whole hint text.

Get Shapely

In order to change the general shape of the tooltip, you need to understand a little about their implementation. The window that the tooltip is drawn in is a class of type THintWindow (a descendent of TCustomControl) as defined in the Controls unit. There is a class reference variable declared in the interface section of the Forms unit, HintWindowClass, that is initialised with a value of THintWindow. This variable can be given other values so long as they represent classes inherited from type THintWindow.

So the starting point is to derive a class from THintWindow, customised as required, and to assign the class type to HintWindowClass. However, doing this does not change the tooltip, which still looks just the same. This is because an instance of the class stored in

HintWindowClass is constructed in the initialization section of the Controls unit. This class is used throughout the duration of the program.

The way around the problem is to set the Application's ShowHint property to False whereupon the THintWindow instance gets destroyed (no hint window object means no hints). If the ShowHint property is set back to True another instance of the HintWindowClass class is created (this is how the Controls unit did it to start with). Of course, this time, it will be *your* class that gets created and so the tooltip will look suitably different.

So what goes to make up the THintWindow class? The class is shown in Listing 1. There are a few useful methods which are worth exploring. The cm_TextChanged message handler is designed to react to an internal component message that gets sent when the hint window's Caption is changed. This method simply changes the width and height of the hint window accordingly. CreateParams is used to modify the window styles and other API-level set-up of the window used by the hint control. ActivateHint is called when the hint window needs to be displayed. It ensures that the hint will be entirely drawn on-screen, and then shows the hint window, making it a 'stay on top' window at the same time.

In order to achieve an effect such as a more rounded hint window, one approach would be to get the normal rectangle to be drawn transparently (and borderless) and then draw our own ellipse in the window.

To get the normal window to be clear and borderless requires an overridden CreateParams method. Drawing the new shape needs a new Paint method. Additionally, when the hint is activated, we need to ensure that the normal window size is enlarged slightly so that all the text can fit into the new non-rectangular shape.

Figure 1 shows our new novelty hint window in action.

```
THintWindow = class(TCustomControl)
private
  procedure CMTextChanged(var Message: TMessage);
    message CM_TEXTCHANGED;
protected
  procedure CreateParams(var Params: TCreateParams); override;
  procedure Paint; override;
public
  constructor Create(AOwner: TComponent); override;
  procedure ActivateHint(Rect: TRect; const AHint: string); virtual;
  function IsHintMsg(var Msg: TMsg): Boolean; virtual;
  procedure ReleaseHandle;
  property Caption;
  property Color;
  property Canvas;
end;
```

➤ *Listing 1*

```
procedure TMyHintWindowClass.CreateParams(var Params: TCreateParams);
begin
  Brush.Style := bsClear;                     { Make window transparent }
  inherited CreateParams(Params);            { Set up normal attributes }
  Params.Style := Params.Style and not ws_Border;       { Remove border }
end;
procedure TMyHintWindowClass.ActivateHint(Rect: TRect; const AHint: string);
begin
  { Make window bigger to accommodate new non-rectangular shape }
  Inc(Rect.Right, InflateX);
  Inc(Rect.Bottom, InflateY);
  { Set modified window size & move it on-screen }
  inherited ActivateHint(Rect, AHint);
end;
procedure TMyHintWindowClass.Paint;
var R: TRect;
    CCaption: array[0..255] of Char;
begin
  Canvas.Brush.Color := Color;
  R := ClientRect;
  Canvas.Ellipse(R.Left, R.Top, R.Right, R.Bottom);
  OffsetRect(R, InflateX div 2, InflateY div 2);
  Inc(R.Left);
  Canvas.Brush.Style := bsClear;
  StrPCopy(CCaption, Caption);
  DrawText(Canvas.Handle, CCaption, -1, R,
    dt_Left or dt_NoPrefix or dt_WordBreak);
  Canvas.Brush.Style := bsSolid;
end;
initialization
  Application.ShowHint := False;
  HintWindowClass := TMyHintWindowClass;
  Application.ShowHint := True;
end.
```

➤ *Listing 2*

Listing 2 shows the new methods as well as the initialisation section of the unit they come from (which sets up the new hint class). The project HINTWND.DPR on the disk shows this in practice. It's the extra unit NEWHINTU.PAS that implements my new TMyHintWindowClass class.

Now I'll never be able to enable hints in a Delphi application again without an involuntary shudder in anticipation of the weird and wonderful hint windows I'll find...

Brian Long is a freelance Delphi consultant and trainer based in the UK. He is available for bookings and can be contacted by email on 76004.3437@compuserve.com