How to get the update region in a TGraphicControl

QUESTION

I made a *TGraphicControl* descendent which does fast overlayed drawings, keeping track of its own update regions, and clipping updates to those.

Problem is, if the form which the component is on gets hidden by some other window, it needs a full repaint when it is shown again. I don't have a clue how to intercept this situation on the component level.

There is the WinAPI *GetUpdateRgn*, but help on it says it's always empty as soon as *BeginPaint* is called, but if *WM_PAINT* reaches my *TGraphicControl*, *BeginPaint* has been called by its parent. Otherwise I could always just add what Windows thinks is the update region.

How would you solve this? Should I just derive from a *TCustomControl* instead?

Answer 1

I had similar problems once and did an extra check on *Canvas.ClipRect* in the *Paint* procedure. When updating the component when only parts of the component needed to be redrawn, I used:

```
{ ... }
var
  R: TRect;
begin
  if FValue <> NewValue then
  begin
    FValue := NewValue;
    R := GetClientRect;
    InflateRect(R, - 1, - 1);
    InvalidateRect(Handle, @R, not (csOpaque in ControlStyle));
  end;
end;
```

Now I could check for this rectangle in the *Paint* procedure. When the *ClipRect* was not the same as I expected it to be (eg caused by a form being moved which covered it partly or completely before the move) a full redraw was needed and performed.

```
{ ... }
with Canvas do
  if (((ClipRect.Right - ClipRect.Left) <> (Width - 2)) and
      ((ClipRect.Bottom - ClipRect.Top) <> (Height - 2)))  then
    DoDrawAll := True
  else
    {only update what needs to be updated.}
{ ... }
```

When a full redraw was not required the rest of the paint routine knew what to update by other means and only updated these parts.

The component in which I used this was a descendent of *TCustomPanel*. I'm not sure if *ClipRect* is setup the same for a *TGraphicControl* but I guess so.

Tip by Pieter Zijlstra

Answer 2

Thank you very much Pieter, it works! This is how I implemented it for the time being: In *WM_WINDOWPOSCHANGED* I assign a private field *fTestRect* like you do, but with

```
fTestRect := BoundsRect;
InflateRect(fTestRect,- 1,- 1);
```

Then I can override *TControl.Repaint*, it's natural for a user to call repaint on an animation step:

```
{ ... }
  if fUseRegions then
    if assigned(parent) then
      InvalidateRect(parent.handle, @fTestRect, False)
    else
      inherited
      else
        inherited;
{ ... }
```

fUseRegions is the flag the user can set to use those update regions for speedup. Then in *Paint*, I do as you do, and only really use the update regions if *Canvas.ClipRect* is the analogue of *fTestRect*. I can't see a slowdown. Of course, chances are that a window pops up on the control which has exactly the dimensions of *fTestRect*, but I think that chance is rather slim.

Tip by Renate Schaaf

Original resource:	The Delphi Pool
Author:	Various
Added:	2012-07-08
Last updated:	2012-07-08