Subject: BackPaint question
Posted by hojtsy on Fri, 20 Jan 2006 16:45:37 GMT
View Forum Message <> Reply to Message

While analyzing the AnimatedHello example I found this method:

```
HelloWorld::HelloWorld()
{
  SetTimeCallback(-40, THISBACK(Animate));
  BackPaint();
  Zoomable().Sizeable();
  SetRect(0, 0, 260, 80);
}
```

Why is the BackPaint() there? It is supposed to mean that the framework should clear the full widget to the backround color before calling Paint, right? There seems to be no need for that, since the HelloWorld::Paint starts with painting the whole area with the white color.

A related question: Would it be logical for flickeing to occur in the AnimatedHello application, especially since the background seems to be painted two times with different color? I don't see any flicker, but how is it avoided?

Subject: Re: BackPaint question
Posted by unodgs on Fri, 20 Jan 2006 21:07:01 GMT
View Forum Message <> Reply to Message

Backpaint is here to avoid flickering between that white rectangle and the text (because recatnle cover the whole window area)! If you have fast computer you may not see that flickering.

Subject: Re: BackPaint question
Posted by hojtsy on Fri, 20 Jan 2006 23:27:48 GMT
View Forum Message <> Reply to Message

Invoking BackPaint() withouth parameters is equivalent to BackPaint(FULLBACKPAINT). Any and all documentation about this enum value is "Whole area of Ctrl is backpainted". This short description can be understood in a whole lot of ways. My understanding was that it instructs the Ctrl class to paint the full area of the widget to the background color before calling the overloaded Paint method. I tried to search in the upp sources to find where and how this FULLBACKPAINT is used. I found that it is used in the Ctrl::CtrlPaint method. I tried to decrypt how that method works, and it seems that it uses the undocumented BackDraw class to buffer the drawing operations done in Ctrl::Paint. Now I see that this can avoid flickering, but still don't understand what is that connection with "Whole area of Ctrl is backpainted"?! It is quite possible to do multiple drawing operations (like overlapping images or polygons) in a sequence which could result in flickering even if you do not backpaint the whole Ctrl. In such case you would also activate this mode, and not because of backpainting. So is it possible that both the name and the documentation of this mode is missleading? Or am I missunderstanding something?

Subject: Re: BackPaint question
Posted by mirek on Sun, 22 Jan 2006 20:18:58 GMT
View Forum Message <> Reply to Message

FULLBACKPAINT just paints it in the "backpaint" buffer and transfers the result to the screen. Means, if some areas in the window are to be repainted (U++ always cumulates are "damaged" areas and defers it as much as possible, in Win32 that is done automatically and repainting is done when WM_PAINT for top-level Ctrl is recieved, in X11 a lot of additional code is inbolved).

In other words, without FULLBACKPAINT all Paint routines draw directly to the screen, which can result in flickering.

With FULLBACKPAINT, help buffer is used, Paint paints to it and then it is transfered to the screen.

Alternative mode backpaints areas covered by transparent Ctrls only, as those are places where flickering is most visible. This mode is default (and it is also the only reason for "Ctrl::Transparent" flag to exist).

(Remaining option, EXCLUDEPAINT, is just "misuse" that reuses the flag to support special cases, like OLE controls that have WM_PAINT based painting).

Sorry for confusion. Please feel free to improve on docs if you are going to fix grammar there.