Subject: Manual for custom Chameleonized Ctrls
Posted by kohait00 on Thu, 24 Jun 2010 20:34:35 GMT
View Forum Message <> Reply to Message

hi people,

what about having a manual on how to create own Ctrl's which are using chameleon stuff *properly*. as far as i know there is few info about it, and reading code on it is bothersome, freaky sideways of win32 / GTK stuff. few people want to dive that deep.

chameleon is sophisticated enough to announce it kind of publicly.
the bazaar would have lots more Ctrl's already adapted to chameleon.

part of manual should be a short description on how the whole Style stuff is made up, later introducing most important chameleon functions. i would do that if i knew about the techniques..but so far i can't quite get the big picture. or is there already manuals provided?

it's time to demystify the chamelon...

cheers

Subject: Re: Manual for custom Chameleonized Ctrls
Posted by cbpporter on Fri, 25 Jun 2010 08:11:09 GMT
View Forum Message <> Reply to Message

Well there are two levels to this issue.

On high level we have the styles, which right now allow you to render almost every imaginable control both with system skins and custom skins. There is no Win32/Gtk programing involved. Using styles or creating new controls to use existing styles is easy once you learn it, but there is a serious learning curve. No documentation for this, but enough code in bazaar to learn it.

At the deeper level is introducing new ways to get system styles. This is only needed if you can find a case where "Windows can render this control, but in U++ it looks bad". This is Win32/Gtk dependent. I think currently Chameleon is almost perfect, except for the lack of animations.

So what exactly did you have in mind when you said that you would like to use it properly? I consider myself knowledgeable with Chameleon, but you won't get me to write documentation for it anytime soon .

Subject: Re: Manual for custom Chameleonized Ctrls
Posted by kohait00 on Fri, 25 Jun 2010 09:47:06 GMT
View Forum Message <> Reply to Message

Quote:
you won't get me to write documentation for it anytime soon .

DOOO..! pity..

the problem is actually that the examples spread only a glimpse of style stuff each, but there is no consistent model / picture of it.

i'd like to know which steps to take to make an own Ctrl's with ChSyle<> support the proper way, what is considered part of Style (colors, no fonts i.e. or something the like)

i'll provide some steps i could find out so far...maybe you could check if they are correct, logical, whether relation to chameleon is correct..

1) define what you consider style for your control

```
//.h
 struct Style : ChStyle<Style> {
  Color paper;
  Color disabled;
  Color focus;
  Color invalid;
  Color text, textdisabled;
  Color selected, selectedtext;
  Value edge[4];
  bool  activeedge;
  int   vfm;
 };
```

2) setup a default style, it is registered globally in Chameleon database for this special control (thats why MACRO needs Ctrl class name, Style class name, generates the function with name 'StyleDefault'', use defaul color descriptions, like SColorFace, SColorPaper, to remain consistant to global gui design where logical. (What meanings do the SCloro... and related stuff have, how are they related to Chameleon?

```
//.cpp
CH_STYLE(EditField, Style, StyleDefault)
{
 paper = SColorPaper();
 disabled = SColorFace();
 focus = paper;
 invalid = Blend(paper, Color(255, 0, 0), 32);
 text = SColorText();
 textdisabled = SColorDisabled();
 selected = SColorHighlight();
 selectedtext = SColorHighlightText();
 for(int i = 0; i < 4; i++)
```

```
  edge[i] = CtrlsImg::EFE();
 activeedge = false;
 vfm = 2;
}
```

3) use a const Style *style to referece the currently used style in your control code, dont forget to initialize the pointer to your default style

```
//.h
 const Style *style;
 static const Style& StyleDefault();
 EditField&  SetStyle(const Style& s);
//.cpp
 style = &StyleDefault(); //ctor

EditField& EditField::SetStyle(const Style& s)
{
 style = &s;
 RefreshLayout();
 RefreshFrame();
 return *this;
}
```

4) use your style information to paint your control, either use it directly, or provide Chameleon helper functions with some of your style info

```
void EditField::Paint(Draw& w)
{
 Size sz = GetSize();

 bool enabled = IsShowEnabled();
 Color paper = enabled && !IsReadOnly() ? (HasFocus() ? style->focus : style->paper) :
style->disabled;
 if(nobg)
  paper = Null;
 Color ink = enabled ? style->text : style->textdisabled;
....
}

... //other controls use it like that (ScrollBar)
   if(i != 2 || thumbsize >= style->thumbmin)
    ChPaint(w, pr, l[i][p == i ? CTRL_PRESSED : light == i ? CTRL_HOT : CTRL_NORMAL]);
   if(i != 2)
    w.End();
  }
 }
```

```
 else
  if(style->through) {
   ChPaint(w, sz, l[0][CTRL_DISABLED]);
  }
  else
  if(IsHorz()) {
   ChPaint(w, style->arrowsize, 0, sz.cx / 2, sz.cy, l[0][CTRL_DISABLED]);
   ChPaint(w, style->arrowsize + sz.cx / 2, 0, sz.cx - sz.cx / 2, sz.cy, l[1][CTRL_DISABLED]);
  }
  else {
   ChPaint(w, 0, style->arrowsize, sz.cx, sz.cy / 2, l[0][CTRL_DISABLED]);
   ChPaint(w, 0, style->arrowsize + sz.cy / 2, sz.cx, sz.cy - sz.cy / 2, l[1][CTRL_DISABLED]);
... }
```

and THAT is exactly the point, i dont know what Ch... functions there are and how to properly use them , how to forward your style information so it can be "camelionized"

some background info is still missing, like maybe (correct please)

normally, Style struct is not alterable (thats why 'const Style *'), you can only replace ist as an entity at once (SetStyle) by a reference to another style, the referenced struct needs to exist as long as the control that's using it, exists as well (logical since it's no copy).

if you want to permanently alter the default Style for *all* controls of that type, you can disable the const lock, to edit the static global instance of your custom control's default style
(or even others if your control supports multiple global styles (how to do that??). and you can always make a preinitialisation to a Standard() style, which was defined one time as copy from the first global registered style (StyleDefault()) for your control (??? is that right). the Standard() preinit saves a lot of code when to alter only few properties. this is also how to restore an altered StyleDefault to its previous state (??)

```
 EditField::Style& es = EditField::StyleDefault().Write();

 es = es.Standard();
 es.paper = SColorPaper();
 es.disabled = SColorFace();
 es.focus = Blend(Green(), Black(), 192);
```

your main aplication window should update all instaniated controls with after finishing updating all desired styles

```
  RefreshLayoutDeep();
```

Subject: Re: Manual for custom Chameleonized Ctrls
Posted by mrjt on Fri, 25 Jun 2010 10:21:59 GMT
View Forum Message <> Reply to Message

Mainly correct ASFAIK

2) The SColor variants are the system colors as defined by the OS, ie the colours you can change in the Windows Control Panel. They pre-date Chameleon and are one compontent of it.

and THAT is exactly the point, i dont know what Ch... functions there are and how to properly use them Smile, how to forward your style information so it can be "camelionized"
ChFunctions: ChPaint is the only one you will likely need. The others are for internal use in the OS theming engine.

I'm not sure what you mean by 'forward your style'. Do you mean set it to the OS theme? If so then this is done by using the Chameleonised info from Upp (the OS only contains detailed theme info for normal controls after all).

For example: You use SColorFace for drawing background color, SColorText for text and if you need a bitmap to make a ctrl look like an OS themed control (like a Button) you can copy it from the default style of that ctrl, and possibly do some modification of it with code. That is how the Styles for TabBar are generated - the original is the TabCtrl style and the rotated ones are generated by rendering to an ImageDraw and rotating.

I'm not saying it couldn't do with more documentation, but it's not actaully all that complicated. The problem is that the underlying code is complicated so it's difficult to work out just from taht.

Subject: Re: Manual for custom Chameleonized Ctrls
Posted by mrjt on Fri, 25 Jun 2010 10:31:10 GMT
View Forum Message <> Reply to Message

In case it may be of use to other people here is example of how to modify a Chameleon style while retaining theming.

Before Chameleon ButtonOptions had a none themed look that I prefered, so when Chameleon was added I reproduced it:

(both buttons are pressed, I keep the OS look when not pressed)

This is the code that does it:

```
#include <CtrlLib/CtrlLib.h>
using namespace Upp;

#define IMAGECLASS Img
#define IMAGEFILE <LayoutTest/LayoutTest.iml>
#include <Draw/iml.h>
```

```
Image ApplyStyle(const Value &base, const Image &style)
{
 Size sz(30, 30);
 ImageDraw w(30, 30);
 w.DrawRect(sz, SColorFace());
 ChPaint(w, sz, base);
 ChPaint(w, sz, style);

 Image img = w;
 ImageBuffer ib(img);
 ib.SetHotSpot(style.GetHotSpot());
 ib.Set2ndSpot(style.Get2ndSpot());
 return (Image)ib;
}

void ButtonOption_HotPress(ButtonOption &bo)
{
 static ButtonOption::Style style;
 style = ButtonOption::StyleDefault();
 style.look[2] = ApplyStyle(style.look[2], Img::HotPress());;
 bo.SetStyle(style);
}

GUI_APP_MAIN
{
 TopWindow wnd;
 ButtonOption normal, hotpress;

 wnd.SetRect(Size(140, 32));
 wnd.CenterScreen();

 ButtonOption_HotPress(hotpress);

 normal.SetLabel("Normal").LeftPos(4, 64).TopPos(4, 24);
 hotpress.SetLabel("HotPress").LeftPos(72, 64).TopPos(4, 24);
 wnd.Add(normal);
 wnd.Add(hotpress);

 wnd.Run();
}
```

What it does is render the existing style value to a buffer (since it's not an Image - it's an os theme) and then draw my own image over the top.

File Attachments

1) HotPressWnd.png, downloaded 1391 times

Subject: Re: Manual for custom Chameleonized Ctrls
Posted by mrjt on Fri, 25 Jun 2010 10:33:30 GMT
View Forum Message <> Reply to Message

And here is the image I use (you have to add the HotSpots to make it scale correctly):

File Attachments
1) HotPress.png, downloaded 1377 times

Subject: Re: Manual for custom Chameleonized Ctrls
Posted by kohait00 on Fri, 25 Jun 2010 10:38:30 GMT
View Forum Message <> Reply to Message

thanks mrjt
Quote:
'forward your style'

i mean the ChPaint functions expect a value, what is part of it (its 4 images or so, what do they mean..

the description of the whole SColorX thing would be good, where it applies, what "Face" means, what "Paper" means, one can only guess by trying, but its not always best results.

Quote:
underlying code is complicated to it's difficult to work out just from taht.

absolutely, thats why i am asking, to save problems digging in code to find out things that are simple for the user if he knows some bit of basic information.

Subject: Re: Manual for custom Chameleonized Ctrls
Posted by kohait00 on Fri, 25 Jun 2010 10:40:48 GMT
View Forum Message <> Reply to Message

thanks for the example, i think with all that bits of information we can make a quite easy guide on how to extend upp with own Ctrl's while keeping up with chameleo..

Subject: Re: Manual for custom Chameleonized Ctrls
Posted by mirek on Fri, 25 Jun 2010 11:57:52 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Fri, 25 June 2010 06:38
i mean the ChPaint functions expect a value, what is part of it (its 4 images or so, what do they mean..

The beauty and cornerstone of ChPaint and Chameleon is that ChPaint expects Value... and can be extended w.r.t. Value types it is able to render.

There are two basic types supported directly in Draw: Color and Image. Image has "hotspots" logic to define inteligent scaling.

You can extend recognized types using ChLookFn. This way, e.g. XP chameleon registers its internal Value type "XpElement" and is able to use XP style rendering system to render Values from Styles...

Mirek

Subject: Re: Manual for custom Chameleonized Ctrls
Posted by kohait00 on Fri, 25 Jun 2010 12:00:31 GMT
View Forum Message <> Reply to Message

wow..

thats sort of important thing to know. thanks
