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();