Subject: Global style changes using Chameleon...
Posted by tvanriper on Fri, 11 Apr 2008 14:48:31 GMT
View Forum Message <> Reply to Message

Introduction

I wanted my application to use custom graphics and colors for a lot of things, to include my buttons, text boxes, and tab headers.

I found it very tedious to do this for every single control in every single dialog in my application, though. This gets extremely rough if your application has quite a few dialogs in it, and you want everything to look uniform.

So, I figured the whole point of Chameleon was to make this kind of thing easier, and I set about figuring out how to use it this way. Documentation is scarce, since it's pretty new... but I can read source code well enough, so I thought I'd give it a try.

Hopefully, I can describe what I learned in some fashion, so others can more easily use this bit of technology. If I have more time, and if nobody else beats me to it, I'll try my hand at writing a decent guide for the help topics later.

Using Chameleon for global style changes

If you're looking to use Chameleon for this sort of purpose, you need to remember the word 'Write'. All of the Chameleon functions intended to help you modify the initial values tend to have the word 'Write' in it somewhere.

So, for example, if you want to change the color of the 'paper' attribute, so anything using SColorPaper will use your color instead of the default, you have to call SColorPaper_Write([your color here]).

You can even use LabelBoxTextColor_Write to help change the default color of your label box texts.

Here's a quick, but non-exhaustive, list of attributes whose colors you can change:

SColorPaper_Write
SColorText_Write
SColorHighlight_Write
SColorHighlightText_Write
SColorMenu_Write
SColorMenuText_Write
SColorDisabled_Write
SColorFace_Write
SColorShadow_Write
LabelBoxTextColor_Write

You may want to hunt around for others, as needed.

This said, sometimes you might want to modify the default graphics for a particular item... maybe you want a different look to your progress bar, or your buttons.

The best approach for this, I think, involves calling the Write() function for your particular style, and modifying the object directly.

Here's an example using the Write() function:

```
Button::Style& button_style = Button::StyleNormal().Write();
button_style.look[0] = MyButton::Get(0);
button_style.look[1] = MyButton::Get(1);
button_style.look[2] = MyButton::Get(2);
button_style.look[3] = MyButton::Get(3);
button_style.textcolor[0] = White;
button_style.textcolor[1] = White;
button_style.textcolor[2] = White;
button_style.textcolor[3] = Gray;
```

Edit:

Edited to reflect some of the information mentioned in this thread.

Subject: Re: Global style changes using Chameleon...
Posted by mrjt on Fri, 11 Apr 2008 16:09:30 GMT
View Forum Message <> Reply to Message

I'll add what I know to this, since your guide idea is a good one.

I have a feeling that the XXX_Write functions are not meant to be comprehensive, just present where necessary for matching native themes.

The .Write() function should be present on every Chameleon style (they are all inherited from ChStyle), so I'm not sure why you need the const_cast<> stuff. Is there an example where this doesn't work?. All the Write() function does is return a non-const reference to the style so that you can edit it.

A stightly more concise version of your button example for instance:

```
Button::Style &button_style = Button::StyleNormal().Write();
button_style.look[0] = MyButton::Get(0);
button_style.look[1] = MyButton::Get(1);
```

```
button_style.look[2] = MyButton::Get(2);
button_style.look[3] = MyButton::Get(3);
button_style.textcolor[0] = White;
button_style.textcolor[1] = White;
button_style.textcolor[2] = White;
button_style.textcolor[3] = Gray;
```

Quote:I think your XXX::Style object needs to remain instantiated for the duration of your application... I typically handle this by creating these objects in main, allowing them to go out of scope when main() ends. Perhaps someone may correct me if I'm wrong.
You are correct. Controls store a pointer to the object, so it should not run out of scope. My preferred solution is this (for single ctrls):

```
const EditField::Style &MyEditStyle() {
    static EditField::Style mystyle = EditField::StyleDefault();
    mystyle.focus = SYellow();
    return mystyle;
}

// Somewhere else
username.SetStyle(MyEditStyle());
```

This is more-or-less what the CH_STYLE macro does I believe.

James

Subject: Re: Global style changes using Chameleon...
Posted by mrjt on Fri, 11 Apr 2008 16:31:03 GMT
View Forum Message <> Reply to Message

One more thing. These functions are useful for setting ctrl looks:

```
template <class T>
void ImageLook(Value *look, int i, int n = 4)
{
 while(n--)
  *look++ = T::Get(i++);
}

template <class T>
void ImageLook(Value *look, int i, const Image& image, const Color *color, int n = 4)
{
 for(int q = 0; q < n; q++)
  *look++ = ChLookWith(T::Get(i++), image, *color++);
}

template <class T>
void ImageLook(Value *look, int i, const Image& image, int n = 4)
```

```
{
 for(int q = 0; q < n; q++)
  *look++ = ChLookWith(T::Get(i++), image, *color++);
}
```
These are templated equivalents to CtrlsImageLook used in uppsrc. Example usage (sets button style to classic):
```
Button::Style &s = Button::StyleNormal().Write();
ImageLook<ClassicCtrlsImg>(s.look, ClassicCtrlsImg::I_B);
```

Subject: Re: Global style changes using Chameleon...
Posted by tvanriper on Fri, 11 Apr 2008 20:36:00 GMT
View Forum Message <> Reply to Message

So far, in my code, the ProgressIndicator::Style lacks a 'Write()' function. I don't see anything else.. but I'm not Chameleonizing absolutely everything, so there might be others.

Subject: Re: Global style changes using Chameleon...
Posted by mrjt on Sat, 12 Apr 2008 23:33:24 GMT
View Forum Message <> Reply to Message

I've definitely got one. This compiles fine:
```
 ProgressIndicator::Style & s = ProgressIndicator::StyleDefault().Write();
```

Subject: Re: Global style changes using Chameleon...
Posted by tvanriper on Mon, 14 Apr 2008 14:27:19 GMT
View Forum Message <> Reply to Message

Hmm... I'll have to take another look. I've been running hard lately, and maybe I missed something.

Subject: Re: Global style changes using Chameleon...
Posted by mrjt on Tue, 15 Apr 2008 13:35:12 GMT
View Forum Message <> Reply to Message

Also, attached is a file that I've generated of every Style and _Write function in CtrlCore, CtrlLib and Draw. I'm using it as a template for a complete reskinning, or a near as possible.

It's generated from the 2008.beta2 release.

File Attachments
1) Skin.cpp, downloaded 634 times

Subject: Re: Global style changes using Chameleon...
Posted by tvanriper on Tue, 15 Apr 2008 18:52:06 GMT
View Forum Message <> Reply to Message

I've modified my original message to apply some of what I learned in this thread. Someone might want to check to see if I've erred.

Oh, and good work on the Skin.cpp file in the reply immediately preceding this one. That should be quite useful.

Subject: Re: Global style changes using Chameleon...
Posted by mrjt on Wed, 16 Apr 2008 13:14:26 GMT
View Forum Message <> Reply to Message

I've updated the Skin.cpp file. Somehow I missed ViewEdge_Write.

Edit: Also added MenuBar::Style. It's possible I've missed some more, I'm using the Upp parser which still has issues with some of the source files for instance this:

```
static ColorF xpmenuborder[] = {
 (ColorF)3,
 &SColorShadow, &SColorShadow, &SColorShadow, &SColorShadow,
 &SColorMenu, &SColorMenu, &SColorMenu, &SColorMenu,
 &SColorMenu, &SColorMenu, &SColorMenu, &SColorMenu,
};
```

completely borks it.

Subject: Re: Global style changes using Chameleon...
Posted by mrjt on Wed, 16 Apr 2008 15:10:52 GMT
View Forum Message <> Reply to Message

On more thing (I'm in the middle of a complete theming, so I'm kind of figuring it out as I go).

There are some images used in CtrlLib that aren't set with styles, namely the ones in CtrlImg. But you can also set these like so:

```
CtrlImg::Set(CtrlImg::I_right_arrow, MyImg::right_arrow());
```

(CtrlImg::I_whatever returns the index of the Image in CtrlImg, or any other Iml class)
This may be necessary if you plan on using a dark background color, as lots of the exrat icons are black.