Subject: Chameleon again
Posted by andrei_natanael on Mon, 18 Jan 2010 01:17:31 GMT
View Forum Message <> Reply to Message

I still have graduation thesis to do and want to improve U++ Chameleon but i need a deeply understanding of it. If someone may explain how chameleon internals i would be glad and i may continue my thesis and work on U++ else i have to give up and choose something else as the end of first semester is almost here and i want to finish on time.

As i understand now, ChHostSkin() is called once at application startup and it initialize widgets ChStyle's structures with widget images read from OS. After that even if widgets get resized ChHostSkin() isn't called anymore(only if theme changed). I think the key point for that are Hotspots from U++ images. I guess that widgets are resized based on Hotspots from images, is that true? If so some mysteries are gone, if it's not true may someone give some information about how's done?

Any information about how Chameleon work is needed. Thanks in advance.

Andrei

Subject: Re: Chameleon again
Posted by mr_ped on Mon, 18 Jan 2010 08:24:22 GMT
View Forum Message <> Reply to Message

I'm not exactly Cham expert (never used it actually except default behaviour), but AFAIK the hotspots in images are just a hint for scaling them. If the GUI needs 300% size of some pic, it's not scaled 300% proportionally, but the area outside of hotspots is scaled only in the direction it really must be scaled, rest is kept 1:1, area inside hotspots is used to fill the rest to get 300% size of image. (I'm not 100% about this to be correct, but this is how I remember it from one forum post)

So you can think about hotspots as a smart picture resize, only graphics thing, nothing more.

If I'm wrong, I hope somebody will correct me soon enough to not mislead you too much.

Subject: Re: Chameleon again
Posted by cbpporter on Mon, 18 Jan 2010 08:34:46 GMT
View Forum Message <> Reply to Message

Well for starters let us ignore system skins and just consider custom skins. For every widget you theoretically have a structure that describes how it should be painted. I'll use Button as an example. I has 4 states: normal, focused, pushed and disabled. For every state you would like to have a way to draw it without having Button be aware of the intimate details of rendering. So you use images. I'll use this very ugly image:

The two red spots are the hot spots. This divides the image into 9 areas. This image will be applied to all sizes of Button. The hot spots describe which areas will be resized. In this image, the cyan areas will remain constant size, the orange areas will be resized in one dimension, and the gray/purple one will be resized in both dimensions. Resizing algorithm is smooth and uses a filter which is appropriate for making good looking pictures while resizing. You will never get a blocky image, but sometimes you get a blurry one.

Assign this image to look[0] for Button and you will get the result described above. But you can assign other things. Color is an obvious example, but for Windows another kind of data is used. This data is obtained from the system and allows U++ applications to look almost 100% native (unfortunately no animations for Vista and Win 7). This System skins are not images and do not have hot spots. It is more like "take this are and please fill it with the look of widget X in state Y". It is a lot more complicated than this. And there was probably a healthy amount of guess work and trial and error involved until look under Windows got as good as it is now.

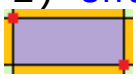
Under Linux there is a different mechanism which I believe instantiates the equivalent Gtk widget and obtains the correct look.

Now the internals of Chameleon are ugly and nobody except the one who wrote them understands them 100%. That part could be indeed improved, but it would be a pretty big task. The structures that provide the looks for all the widgets are easy to work with once you understand them, but there are a lot of hidden rules. If field a is present, but not b and c, then use a, otherwise use b for this and c for that. There is zero documentation for this right now (sorry). You can look over Skulpture code to see how the look is changed for most widgets.

This is just a short intro. You can post more questions and I'll try to answer the best I can.

File Attachments

1) Untitled4.png, downloaded 1878 times



Subject: Re: Chameleon again
Posted by andrei_natanael on Mon, 18 Jan 2010 10:00:31 GMT
View Forum Message <> Reply to Message

Thank you for your answers.
So basically the control hold a pointer to main ChStyle class for his style or a pointer to another ChStyle if main style get replaced for that control (only) with another style?
Buttons use the same small image (12x22) and the resizing algorithm is called every time the control Paint is called? Isn't used a cache to store new resized image for a control so not having to resize a small image every time when Paint is called?

Andrei

Subject: Re: Chameleon again
Posted by andrei_natanael on Mon, 18 Jan 2010 14:47:10 GMT
View Forum Message <> Reply to Message

I don't find Skulpture code, only some portions of it in forum.

Subject: Re: Chameleon again
Posted by andrei_natanael on Mon, 18 Jan 2010 17:59:32 GMT
View Forum Message <> Reply to Message

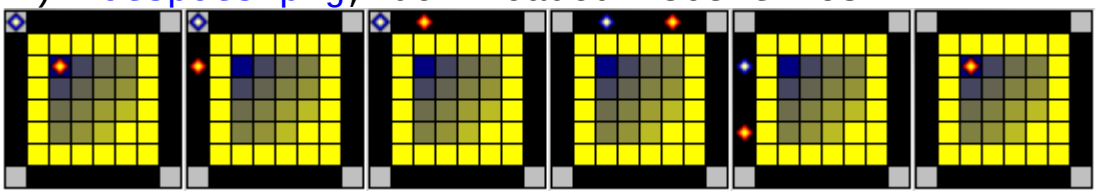
cbpporter wrote on Mon, 18 January 2010 10:34
The two red spots are the hot spots. This divides the image into 9 areas. This image will be applied to all sizes of Button. The hot spots describe which areas will be resized. In this image, the cyan areas will remain constant size, the orange areas will be resized in one dimension, and the gray/purple one will be resized in both dimensions. Resizing algorithm is smooth and uses a filter which is appropriate for making good looking pictures while resizing. You will never get a blocky image, but sometimes you get a blurry one.

In that case with both hotspots having distinct locations and x,y != 0 seems logic how each part created by hotspot position is used to compose the button, but what about the following cases?

Here hotspots split image in 2, 3 or 4 parts not 9 as in your case.
I think when hotspots coincide in x,y=0,0 the image is scaled equally in both directions (it doesn't appear in image), but in other cases which part is used for which button part?

File Attachments

1) hotspots.png, downloaded 1863 times



Subject: Re: Chameleon again
Posted by andrei_natanael on Mon, 18 Jan 2010 19:20:42 GMT
View Forum Message <> Reply to Message

Quote:This System skins are not images and do not have hot spots. It is more like "take this are and please fill it with the look of widget X in state Y". It is a lot more complicated than this. And there was probably a healthy amount of guess work and trial and error involved until look under

Windows got as good as it is now. Isn't simple and also portable to ignore hotspots at all?
I mean, if we use system paint (DrawBackgroundTheme and gtk equivalent [don't have one but it's possible to "emulate"]) during resize events (how often we resize? this doesn't introduce too much overhead) and once the size is stabilized store images in cache and paint them from there when Over, Clicked etc. events happen. I know it doesn't introduce overhead because widgets in gtk and msw use this intensely, no cache is used.
That have drawbacks anyway. If i understand well, during Paint event in U++ the Button image is scaled to the current size of the button and painted on Drawing area. It means after the resized image is drawn it's freed from memory, so it use less memory how it's implemented now but use more CPU time because of scaling. If method described by me would be used then CPU used time will be minimum but memory usage is increased. My method is more efficient if we have to deal with custom themes where we don't know where a hotspot should be placed in image in order to have a good look. AFAIK hotspots are fixed so if a theme use a different larger border or different style for a button the chameleon will fail to acquire system look.

Andrei

Subject: Re: Chameleon again
Posted by dolik.rce on Mon, 18 Jan 2010 19:31:45 GMT
View Forum Message <> Reply to Message

andrei_natanael wrote on Mon, 18 January 2010 15:47I don't find Skulpture code, only some portions of it in forum.
It is located in bazaar/Theme package and the skins are in bazaar/Themes directory.

Subject: Re: Chameleon again
Posted by andrei_natanael on Mon, 18 Jan 2010 19:44:31 GMT
View Forum Message <> Reply to Message

dolik.rce wrote on Mon, 18 January 2010 21:31andrei_natanael wrote on Mon, 18 January 2010 15:47I don't find Skulpture code, only some portions of it in forum.
It is located in bazaar/Theme package and the skins are in bazaar/Themes directory.
Thanks, found it. It was hidden by theIDE "Main package of the first nest" and "All main packages" features.

Subject: Re: Chameleon again
Posted by tojocky on Mon, 18 Jan 2010 20:08:57 GMT
View Forum Message <> Reply to Message

Hello Andrei,

Very nice realization is Chameleon.
How about to add possibility to change/Edite theme with visual efect and save themes in external location?

Regards, Ion Lupascu (tojocky)

Subject: Re: Chameleon again
Posted by mirek on Mon, 18 Jan 2010 22:23:35 GMT
View Forum Message <> Reply to Message

andrei_natanael wrote on Mon, 18 January 2010 14:20Quote:This System skins are not images and do not have hot spots. It is more like "take this are and please fill it with the look of widget X in state Y". It is a lot more complicated than this. And there was probably a healthy amount of guess work and trial and error involved until look under Windows got as good as it is now. Isn't simple and also portable to ignore hotspots at all?
I mean, if we use system paint (DrawBackgroundTheme and gtk

But we do use DrawBackgroundTheme!

ChWin32.cpp, line 113

Chameleon is flexible enought to do that.

Hotspots are just possible solutions, when Value of background is Image. E.g. if you want do define your own skin. Or in Gtk, where in fact, DrawBackgroundTheme equivalent is not available.

Hotspots are just option. If we have a better API, like in Win32, there is no problem to use it.

Mirek

Subject: Re: Chameleon again
Posted by cbpporter on Tue, 19 Jan 2010 07:16:02 GMT
View Forum Message <> Reply to Message

andrei_natanael wrote on Mon, 18 January 2010 19:59cbpporter wrote on Mon, 18 January 2010 10:34
The two red spots are the hot spots. This divides the image into 9 areas. This image will be applied to all sizes of Button. The hot spots describe which areas will be resized. In this image, the cyan areas will remain constant size, the orange areas will be resized in one dimension, and the gray/purple one will be resized in both dimensions. Resizing algorithm is smooth and uses a filter which is appropriate for making good looking pictures while resizing. You will never get a blocky image, but sometimes you get a blurry one.

In that case with both hotspots having distinct locations and x,y != 0 seems logic how each part created by hotspot position is used to compose the button, but what about the following cases?

Here hotspots split image in 2, 3 or 4 parts not 9 as in your case.

I think when hotspots coincide in x,y=0,0 the image is scaled equally in both directions (it doesn't appear in image), but in other cases which part is used for which button part?
Well the simple answer is that you use hotspots to determine a rectangle that is going to be scaled and zones outside of that rectangle won't be scaled or scaled only in one dimension. It is useful because it allows you to have constant borders but smooth filling. You can play around around with the positions of hotspots and when they are identical you get the same result like if they were describing a symmetrical border. So the practical answer is: make a test application, use SetSkin with ChStdSkin or ChClassicSkin, edit the hotspots in CtrlLib/Ctrls.iml and ClassicCtrls.iml, and after playing around for a while, you should get the feeling for hot spot functionality.

Subject: Re: Chameleon again
Posted by mirek on Tue, 19 Jan 2010 14:38:00 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Tue, 19 January 2010 02:16and after playing around for a while, you should get the feeling for hot spot functionality.

It is worth to note that if you swap hotspots, or place them top-right, bottom left, the "internal" rectangle would fill the content with tiles (horizontal, vertical or both) instead of simple rescaling.

Mirek
