Subject: Re: Graduation thesis - Camouflage - a replacement of(a part of) Chameleon
Posted by andrei_natanael on Thu, 11 Feb 2010 19:11:36 GMT
View Forum Message <> Reply to Message

luzr wrote on Thu, 11 February 2010 20:12Hm, I propose you to check existing Styles of CtrlLib first.

Start with ScrollBar or MultiButton...

I would say the problem of Chameleon is not anything that could be solved by proposed API. The problem is how to get required information from host-OS.

That's what my API is trying to solve, how to get required information from OS and present it to U++ in a portable way.
Checking existing Styles would help me to decompose controls in smaller parts which would get returned by my API if they are requested by U++.

Quote:
Plus, such fixed API is somewhat too rigid. Note that Style system is naturaly 'cascading' and easily extensible, everything has a nice default etc... I can say "this button is supposed to look like normal button if not said otherwise" etc....

I will keep Style in every Ctrl, i will not even modify Style structure if it is complete (having all controls states), so 'cascading' of styles will be preserved. My API only action as a replacement of ChWin32.cpp and ChGtk.cpp - first part of these files which are responsible for getting required information from the OS.
What my API is trying to solve is hiding complexity of getting information from OS and present to U++ in a portable manner. For example what my API is trying to do is to hide this:

```
int efp = EP_EDITTEXT;
int efs = 1;
int ebsx = max(2, XpInt(XP_EDIT, efp, efs, 2403/*TMT_BORDERSIZE*/));
int ebsy = max(1, XpInt(XP_EDIT, efp, efs, 2403/*TMT_BORDERSIZE*/));
Image ee = XpImage(XP_EDIT, efp, efs, SColorFace(), Size(10 * ebsx, 10 * ebsx));
ImageBuffer eb(ee);
eb.SetHotSpot(Point(ebsx, ebsy));
ee = eb;
EditField::Style& s = EditField::StyleDefault().Write();
s.activeedge = false;
s.edge[0] = ee;
```

into something like this:

```
unsigned char buffer[12*12*4]; // => imagecx * imagecy * sizeof(RGBA)
Point p1, p2;
CamoGetImage(/* unsigned char */ &buffer, EditText, State_Normal, 12, 12, &p1, &p2);
ImageBuffer ib = RGBAToImageBuffer(buffer);
ib.SetHotSpot(p1);
Image ee = ib;
```

```
EditField::Style& s = EditField::StyleDefault().Write();
s.activeedge = false;
s.edge[0] = ee;
```

I've only changed how things are get from OS, the real get will be in my dll/so implementation.

Quote:
In fact, I was thinking about something completely different. What I would like to see is that any widget somehow fits with target platform, including custom widgets you develop.

Custom widgets will fit in platform if the programmer use available colors provided by OS (through Camo interface - somehow like GetThemeColor).
Quote:
Instead of providing drawing api, I would perhaps tried to "classify" host GUI, like buttons are rectangular, slightly rounded, extremely rounded, base dialog background, base button face, platicity etc.. Having these colors and visual features, you could easily choose the style of any widget, even if you do not have enough info from host os.

Mirek
IMO having something like that will fail to provide the same look and feel as the OS have. Take for example MacOSX Aqua theme, it's animated, buttons have a blue "wave" on them.
It's possible to provide also an API which can tell if the button is rectangular, a bit rounded, extremely rounded, that would be a plus but not the base for widgets.
Why then not use our own widgets without taking the look from OS take only colors and colorize widgets based on that? I think that simple wouldn't fit.

If you'll have to start from scratch developing a GUI toolkit which way you'll choose - current way of U++ and Qt of drawing every widget or wxWidgets way of being a wrapper over OS API? Drawing widgets have the advantage that you may customize them as you want while using a wrapper have the advantage that your application will look and feel exactly as other native applications.

Andrei