Subject: Dark theme issues
Posted by cbpporter on Thu, 18 Apr 2019 10:43:10 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Thu, 18 April 2019 11:30Hi cbpporter,

Dark theme is enabled in Windows 10 using Settings > Colors > Choose your default app mode > Dark

Then restart TheIDE.

Best regards,

Tom
Thanks! I mean more like in code, how to turn on CtrlLib dark mode.

Found it in GIT: Ctrl::SetDarkThemeEnabled();

But... sweet baby Jesus, my eyes! I do not mean to be rude or anything, but those colors... That contrast... That Cyan text if you switch code editor to dark mode in TheIDE. That color scheme is completely unusable. It makes my eyes bleed. And literally hurt.

Again, not to be rude, but to avoid personal bias, I showed the theme to about 5 people and we had a good time. Except for the eye bleeding...

Not to mention that it is buggy and once I switched CodeEditor to dark mode, switching it to light mode doesn't work and restoring defaults.

I need to delete settings to get things back to normal...

We need something like SetDarkIconsEnabled which is independent on SetDarkThemeEnabled, maybe support for two color icons...

And a serious pass on the colors...

I tried to change the colors but it is not as simple as it sounds because of TheIDE: I noticed that several parts of TheIDE and CodeEditor work differently when together or used separately, probably because TheIDE micros and sets stuff multiple times in different points. So changing test application was easy, changing TheIDE very hard.

Chameleon needs refactoring.

Still, after about 15 minutes of work, see the attached screenshot.

I am not a graphics designer...

File Attachments

1) darkmode.png, downloaded 480 times

Subject: Re: U++ 2019.1.rc4 released
Posted by cbpporter on Thu, 18 Apr 2019 11:01:04 GMT
View Forum Message <> Reply to Message

Oh, and calling Ctrl::SetDarkThemeEnabled() with Windows set to light theme causes problems both in TheIDE and test app. Calling Ctrl::ReSkin() after fixes these problems but now the theme is white.

Chameleon really needs to read and write themes once, respect what you tell it to do and have a clear point after which you can make changes that are never lost by random CtrlCore behavior.

Calling SetDarkThemeEnabled once in GUI_APP_MAIN should be enough for that application to be dark forever. Then you should be able to change ink from black to Color(51, 51, 51) and this change to be respected forever.

If this can't be done with the current API, we need a new Theme class that is always respected and just change themes as monoliths.

This reminds me of my work many years ago in bazaar/Theme, where you had to Apply the theme in a very specific semi-magical way for it reliably work... I no longer remember too many details about working with Chameleon and no longer have an intuitive grasp on what a "color shadow" means, so I can't pick the colors perfectly...

Subject: Re: U++ 2019.1.rc4 released
Posted by mirek on Thu, 18 Apr 2019 11:33:34 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Thu, 18 April 2019 12:43Tom1 wrote on Thu, 18 April 2019 11:30Hi cbpporter,

Dark theme is enabled in Windows 10 using Settings > Colors > Choose your default app mode > Dark

Then restart TheIDE.

Best regards,

Tom
Thanks! I mean more like in code, how to turn on CtrlLib dark mode.

Found it in GIT: Ctrl::SetDarkThemeEnabled();

But... sweet baby Jesus, my eyes! I do not mean to be rude or anything, but those colors... That contrast... That Cyan text if you switch code editor to dark mode in TheIDE. That color scheme is completely unusable. It makes my eyes bleed. And literally hurt.

Again, not to be rude, but to avoid personal bias, I showed the theme to about 5 people and we had a good time. Except for the eye bleeding...

Not to mention that it is buggy and once I switched CodeEditor to dark mode, switching it to light mode doesn't work and restoring defaults.

I need to delete settings to get things back to normal...

We need something like SetDarkIconsEnabled which is independent on SetDarkThemeEnabled, maybe support for two color icons...

And a serious pass on the colors...

Your post is confusing, I am really not able to tell what you are speaking about...

Are you annoyed by colors in theide editor in the dark mode? (If so, welcome to the club, I dislike it too. But then I really dislike dark theme itself, so who am I to judge...).

SetDarkThemeEnabled tells U++ that application is ready to handle the dark theme. It in fact has any meaning only in windows. If you activate that, app starts following theme setting in Windows 10.

Subject: Re: U++ 2019.1.rc4 released
Posted by mirek on Thu, 18 Apr 2019 11:38:44 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Thu, 18 April 2019 13:01Oh, and calling Ctrl::SetDarkThemeEnabled() with Windows set to light theme causes problems both in TheIDE and test app. Calling Ctrl::ReSkin() after fixes these problems but now the theme is white.

Chameleon really needs to read and write themes once, respect what you tell it to do and have a clear point after which you can make changes that are never lost by random CtrlCore behavior.

Calling SetDarkThemeEnabled once in GUI_APP_MAIN should be enough for that application to be dark forever. Then you should be able to change ink from black to Color(51, 51, 51) and this change to be respected forever.

Really not sure what you are trying to do. U++ is designed in a way that the look&feel is set at the start of app and should not change after that point.

theide works quite well in both dark and light.

There is one issue which is maybe confusing. The editor highlighting settings are INDEPENDENT. Yes, on the first start light or dark theme is loaded based on current schema, but then it stays at colors selected. The reason is simple: colors are user settings, so you cannot change them at start. Users who want to adjust settings would not be happy about it...

Mirek

Subject: Re: U++ 2019.1.rc4 released
Posted by cbpporter on Thu, 18 Apr 2019 12:02:08 GMT
View Forum Message <> Reply to Message

The issues is that TheIDE CtrlCore constantly overrides your changes. In a very counter-intuitive way. Chameleon has been doing this for over a decade now. If you want to correct something, you can't easily do it. If you want to implement your own theme on top of the current dark mode? It is fairly hard. It is fairly hard to force a dark app on a light windows or the other way around. Making TheIDE do the same thing is even harder because it also constantly overrides you changes in many places. The same for code editor.

Chameleon has always been a bit hard to use and counter-intuitive.

As an example, the newest addition: SetDarkThemeEnabled. This should set your app to dark theme. Always. Without asking the operating system in 1000 places if IsDarkTheme.

Quote:SetDarkThemeEnabled tells U++ that application is ready to handle the dark theme. It in fact has any meaning only in windows. If you activate that, app starts following theme setting in Windows 10.
IMHO that is a counter-intuitive and problematic design. SetDarkThemeEnabled means "Set my theme to dark": change colors and skins.

Why cant U++ detect multiple themes and be coded to religiously follow those themes. So that if you change them, I don't have to track down dozen of places to make things just work.

I would use a design where you have a Theme, all theme aware components just read from the theme, nobody can write, the theme knows if it is dark or now (you can use heuristics if not sure) and then people are free to update that theme. The system would auto-detect the theme before GUI_APP_MAIN, produce both light an dark versions, and then you in GUI_APP_MAIN can produce a new theme based on both, set the theme without ruining the originals and things should just work.

In short, two phases:
- auto-detect and build several themes
- use the theme (just set a pointer and force the gui to refresh)

And negative icons must be turned on or off independently from any setting. All my hand inverted dark mode icons are now auto-inverted and made ugly.

Been though this multiple times, but never arrived to a final perfect dark mode skin.

I'm currently googling samples, this is the best I found for full GUI.
https:// www.reddit.com/r/Unity3D/comments/9reqtm/i_wasnt_planning_on
_sharing_this_so_early_but/

I'll continue searching and am open to suggestions. If anybody knows a good dark UI that is easy to copy, please let me know.

Subject: Re: U++ 2019.1.rc4 released
Posted by mirek on Thu, 18 Apr 2019 12:13:34 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Thu, 18 April 2019 14:02The issues is that TheIDE CtrlCore constantly overrides your changes. In a very counter-intuitive way. Chameleon has been doing this for over a decade now. If you want to correct something, you can't easily do it. If you want to implement your own theme on top of the current dark mode? It is fairly hard. It is fairly hard to force a dark app on a light windows or the other way around. Making TheIDE do the same thing is even harder because it also constantly overrides you changes in many places. The same for code editor.

Chameleon has always been a bit hard to use and counter-intuitive.

Maybe because the problem itself is hard and counterintiutive?

Anyway, missing docs about Chameleon is not helping. Will try to fix than.

Quote:
Quote:SetDarkThemeEnabled tells U++ that application is ready to handle the dark theme. It in fact has any meaning only in windows. If you activate that, app starts following theme setting in Windows 10.
IMHO that is a counter-intuitive and problematic design. SetDarkThemeEnabled means "Set my theme to dark": change colors and skins.

OK, we can argue about the name and I am willing to change it. But I really need that function to do exactly what it does.

Quote:
Why cant U++ detect multiple themes and be coded to religiously follow those themes. So that if you change them, I don't have to track down dozen of places to make things just work.

I would use a design where you have a Theme, all theme aware components just read from the theme, nobody can write, the theme knows if it

Thats centralized model which is not extensible - that is the issue I wanted to avoid.

Quote:
 - use the theme (just set a pointer and force the gui to refresh)

void Ctrl::SetSkin(void (*_skin)())

Mirek

Subject: Re: U++ 2019.1.rc4 released
Posted by cbpporter on Thu, 18 Apr 2019 12:29:43 GMT
View Forum Message <> Reply to Message

OK, let's go in order.

Before the dark mode support, setting dark mode in Windows didn't work. So clearly it is not a "simple" issue of applying a system theme, because most applications ignore that windows option.

So U++ creates this new dark theme.

So my questions are:
 - how do you activate dark theme in any app once, and have it work forever? Even if Windows is in light mode. Same for light mode.
 - how do you detect which mode it is. I want: if (isdarkmode) setdarkmode else setlightmode. That is a nice intuitive model.
 - how do you do a few basic changes to it, like SColorPaper_Write(Color(51, 51, 51)) and force this change forever?
 - how do you do the same for TheIDE, because things that work in normal apps need special extra work for TheIDE.

Quote:Thats centralized model which is not extensible - that is the issue I wanted to avoid.
It looks like to me like you achieved the exact opposite, with Chameleon doing its own thing and being super hard to reliably and intuitively override. Is there anybody who ever had luck with doing anything in Chameleon except for me 10 years ago?

This invisible magic macro system is IMHO in no way superior to a non-extensible simple deign and the non-extensible bit is arguable. There is nothing non-extensible about it because you don't need to extend upon it. For years now people have been able to define incredibly diverse themes with a CSS and a bunch of pictures.

Case in point:
void Ctrl::SetSkin(void (*_skin)())

It is not clear what it does and there are 100 more ways to touch skins. I remember now, but my bazaar/Theme did not call this function and in consequence you couldn't reliably change the theme after program launch. So GUI elements and colors updated, some didn't. So I decided back then to set the them as the first thing in the program.

Subject: Re: U++ 2019.1.rc4 released
Posted by mirek on Thu, 18 Apr 2019 12:54:08 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Thu, 18 April 2019 14:29OK, let's go in order.

Before the dark mode support, setting dark mode in Windows didn't work. So clearly it is not a "simple" issue of applying a system theme, because most applications ignore that windows option.

Well, the real secret is that Windows does not really support dark theme for Win32 apps....

So what we do is to detect the fact that win10 is in dark theme, then roll out our own based on DarkTheme versions of system colors.

Quote:
So my questions are:
 - how do you activate dark theme in any app once, and have it work forever? Even if Windows is in light mode. Same for light mode.

Do not call DarkThemeEnabled. It does something else.

Create dark theme skin. That must be a function that sets everything to your desired colors.

Use SetSkin.

Quote:
Case in point:
void Ctrl::SetSkin(void (*_skin)())

It is not clear what it does and there are 100 more ways to touch skins.

There are 100 ways to touch the skin, BUT you are supposed to do them in skin routine.

So basically it does what you would want from SetTheme(Theme *). But the advantage is that you inherit some default skin which you are changing and you can even call "subskin" routines there.

I agree that more docs and some cleanup would be nice, but the basic idea is IMO sound.

Mirek

Subject: Re: U++ 2019.1.rc4 released
Posted by mirek on Thu, 18 Apr 2019 13:01:50 GMT
View Forum Message <> Reply to Message

P.S.: WRT to .iml being automatically converted, I agree that is rough edge and I have introduced that with great hesitation and still feel uneasy about it. What I know is that 'something like that' is needed. Also note that the whole issue is recent addition and perhaps it really was a mistake to rush into it. What is done is done though...

The problem there is that the number of icons increases quickly. For UHD we have two variants, now with dark mode, it would be 4. Maybe it was mistake to introduce .iml model with "HiDPI" / "Normal" / "Raw" setting per icon. Perhaps we should rather have whole .iml sets and icons would default to "normal" set and be converted only if not found in given set?

Subject: Re: U++ 2019.1.rc4 released
Posted by cbpporter on Thu, 18 Apr 2019 18:31:36 GMT
View Forum Message <> Reply to Message

mirek wrote on Thu, 18 April 2019 15:54
I agree that more docs and some cleanup would be nice, but the basic idea is IMO sound.

Let's agree to disagree on this one :).

Everything is a singleton and U++ often updates stuff in a surprising fashion. And there are macros involved. When choosing color at runtime, I can't write code like: oh, it looks like I'm using black ink on black paper. This is what the curent theme asks of me. The curent theme is dark. Let me see if the system theme does not have a light color for this element?

And this extends to CodeEditor too. If you make those colors work and be different for several programming languages, like when writing an IDE, dark and light theme, plus user profiles you can load or save, the whole updating singletons does not look like a good idea to me. I've been struggling with this for a long time. I remember for months when a single background color would not update because you had to update it outside of CodeEditor or vice-versa.

Tomorrow I'll continue investigating dark mode and see if I can can turn this mess into a half decent looking skin.

So how do you activate the dark theme in an application using SetSkin, let's say in AddressBook.

Other things:
- Icons: at the very least icons must have a flag. If checked, U++ should draw them as they are. And also a way to disable all UHD and sizing shenanigans.
- Not just icons, but everything else needs to have light mode/dark mode override. Like in TheIDE, as you described and I have noticed, CodeEditor and GUI light/dark mode are separate. This is perfectly fine, as long as each knows if it is dark/light and can access both light and dark colors at the same time. So singleton SColorFace just won't do.
- The standard U++ set skin code must set the controls/icons dark/light flag on request and only then.

Tomorrow I will go over all the colors, replace them with red and blue, see how they work and what they affect and hopefully correct this theme problem for my apps, if not for TheIDE.

The problem is that I'm not good at graphics design. I would certainly pay an affordable to designer to create a professional GUI theme. Until then all I can do is look at screenshots on the Internat and try out colors and skins.

I'm also thinking of replacing icons with SVG ans scale them on app activation, but like I said in a post, SVG are drawn with white background and that is nonsensical. Default should be Null color. Another thing to fork.

Subject: Re: U++ 2019.1.rc4 released
Posted by cbpporter on Thu, 18 Apr 2019 19:49:48 GMT
View Forum Message <> Reply to Message

PS: We also need a UI Demo with most widgets and quick toggles for testing all skins.

That's how I'll start working tomorrow.

My latest plan for a first trial: rip off MacOS...

Subject: Re: U++ 2019.1.rc4 released
Posted by mirek on Fri, 19 Apr 2019 10:30:19 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Thu, 18 April 2019 20:31
- Icons: at the very least icons must have a flag. If checked, U++ should draw them as they are. And also a way to disable all UHD and sizing shenanigans.

Yeah, well, but they do.... Right-click on the icon and select "Raw".

Mirek

Subject: Re: U++ 2019.1.rc4 released
Posted by cbpporter on Fri, 19 Apr 2019 10:57:57 GMT
View Forum Message <> Reply to Message

mirek wrote on Fri, 19 April 2019 13:30cbpporter wrote on Thu, 18 April 2019 20:31
- Icons: at the very least icons must have a flag. If checked, U++ should draw them as they are. And also a way to disable all UHD and sizing shenanigans.

Yeah, well, but they do.... Right-click on the icon and select "Raw".

Mirek
Sorry, can't find it.

I attached in screenshot what I see when right-clicking or double clicking.

File Attachments
1) icon.png, downloaded 383 times

Subject: Re: U++ 2019.1.rc4 released
Posted by mirek on Fri, 19 Apr 2019 11:09:10 GMT
View Forum Message <> Reply to Message

Sorry, "None".

Mirek

Subject: Re: U++ 2019.1.rc4 released
Posted by mirek on Fri, 19 Apr 2019 11:11:51 GMT
View Forum Message <> Reply to Message

(I have moved topic where it belongs...)

Subject: Re: U++ 2019.1.rc4 released
Posted by cbpporter on Fri, 19 Apr 2019 12:01:37 GMT
View Forum Message <> Reply to Message

mirek wrote on Fri, 19 April 2019 14:09Sorry, "None".

Mirek
So:
1. If I want to turn off icon inverting as a whole in a dark mode application, how do I do that? You can't invert icons too easily because inverting for dark mode requires the inversion of fades and shadows, but only slight adjustments to non fades. (sample)
2. After I turned off only icon inverting for the dark mode application, how do I detect if it is dark mode so I can change all my icons to proper dark mode variants?
2. How do I turn off invert for a single icon without changing UHD?

Subject: Re: U++ 2019.1.rc4 released
Posted by cbpporter on Fri, 19 Apr 2019 12:15:58 GMT

View Forum Message <> Reply to Message

SetDarkThemeEnabled needs a lot of explanations.

Test scenario.

```
 Ctrl::SetDarkThemeEnabled();
 Ctrl::SetSkin(ChClassicSkin);
```

Scenario explanation: app start in dark mode and looks ugly. You select ChClassicSkin to return to a usable UI. Doesn't work.

Interesting behavior:

```
 Ctrl::SetSkin(ChClassicSkin);
 Ctrl::SetDarkThemeEnabled();
```

The app looks completely different if I just swap the two statement around.

That's why I'm saying: dark mode needs to be more of its own thing that you can select instead of having such a complex an unmanageable interaction with everything else. Dark mode just needs to set things to dark without having complex system wide repercussions and changing the way some colors are computed.

I'll go over all the widgets and make a complete list of things that need to be updated by dark mode.

So far all frames need to be updated and made (more) Chameleon aware.

Subject: Re: U++ 2019.1.rc4 released
Posted by cbpporter on Fri, 19 Apr 2019 12:55:40 GMT
View Forum Message <> Reply to Message

I'm close to getting a workable system for themeing that does not need U++ changes (maybe something minor) but lives peacefully together.

What I want to do is have choice in what theme to use.

```
 if(Ctrl::IsDarkThemeEnabled() && IsSystemThemeDark() &&
!IsDark(Color::FromCR(GetSysColor(COLOR_WINDOW))))
  sEmulateDarkTheme = true;
```

So basically if IsDarkThemeEnabled and the system is dark themed, we use dark theme.

Here is where I want more control. I want all valid combinations of system dark theme and app dark theme, so 4 combinations.

We need to adjust this condition a bit.

I propose we rename IsDarkThemeEnabled, in something like IsDarkModePermited and add a new IsDarkThemeEnabled, geter and seter, where the user can choose if the skin is dark. SetDarkThemeEnabled would be called set as is today, with IsSystemThemeDark() && !IsDark(Color::FromCR(GetSysColor(COLOR_WINDOW))), but NOT in ChSysInit. Before, on system startup, so auto-detection is correct, but I can override it later.

Of course, this is just an option. Here is what I want:

```
// autodetect light or dark
void SetHostSkin() {
 Ctrl::SetDarkThemeEnabled();
 Ctrl::SetSkin(ChHostSkin);
}

// always light
void SetLightHostSkin() {
 Ctrl::SetDarkThemeEnabled(false);
 Ctrl::SetSkin(ChHostSkin);
}

// always dark
void SetDarkHostSkin() {
 Ctrl::SetDarkThemeEnabled(true);
 Ctrl::ForceDarkSkin(true);
 Ctrl::SetSkin(ChHostSkin);
}

// old, unchaged
void SetStdSkin() {
 Ctrl::SetDarkThemeEnabled(false);
 Ctrl::SetSkin(ChStdSkin);
}

// old, unchaged
void SetClassicSkin() {
 Ctrl::SetDarkThemeEnabled(false);
 Ctrl::SetSkin(ChClassicSkin);
}

// my fixed version
```

```
void SetNewDarkHostSkin() {
 Ctrl::SetDarkThemeEnabled(false);
 Ctrl::SetSkin(ChHostSkin);

 // do fixes

 Ctrl::ReSkin();
}
```

I managed to do this, except for forcing dark skin on light skin windows. Hence the Ctrl::ForceDarkSkin(true); or the proposed rename mentioned above.

Subject: Re: U++ 2019.1.rc4 released
Posted by mirek on Fri, 19 Apr 2019 13:29:33 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Fri, 19 April 2019 14:01mirek wrote on Fri, 19 April 2019 14:09Sorry, "None".

Mirek
So:
1. If I want to turn off icon inverting as a whole in a dark mode application, how do I do that? You can't invert icons too easily because inverting for dark mode requires the inversion of fades and shadows, but only slight adjustments to non fades. (sample)
2. After I turned off only icon inverting for the dark mode application, how do I detect if it is dark mode so I can change all my icons to proper dark mode variants?
2. How do I turn off invert for a single icon without changing UHD?

Long story short: Wait till next week....

Mirek

Subject: Re: U++ 2019.1.rc4 released
Posted by mirek on Mon, 22 Apr 2019 08:33:24 GMT
View Forum Message <> Reply to Message

https://www.ultimatepp.org/forums/index.php?t=msg&th=106 59&start=0&

Subject: Re: U++ 2019.1.rc4 released
Posted by cbpporter on Mon, 22 Apr 2019 10:47:46 GMT
View Forum Message <> Reply to Message

mirek wrote on Mon, 22 April 2019 11:33
https://www.ultimatepp.org/forums/index.php?t=msg&th=106 59&start=0&
It is certainly a step in the right direction. Got the latest nightly.

I still have eye pain from staring at the TheIDE for the last few days, so I'll get back to you in a couple of days.

But I need a break first. I'm going into the code and turning off dark mode so I can work in piece for a bit.

Which brings me to: we need and option to disable dark mode in TheIDE, even if a restart is needed.

Subject: Re: U++ 2019.1.rc4 released
Posted by cbpporter on Mon, 22 Apr 2019 14:09:48 GMT
View Forum Message <> Reply to Message

Yeah I'm having a real hard time figuring out what is actually taken from Windows and what is U++ convention when it comes to Chameleon, especially for dark mode.

And probably things are different on Gtk and Mac...

Subject: Re: U++ 2019.1.rc4 released
Posted by mirek on Mon, 22 Apr 2019 18:03:42 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Mon, 22 April 2019 16:09Yeah I'm having a real hard time figuring out what is actually taken from Windows and what is U++ convention when it comes to Chameleon, especially for dark mode.

And probably things are different on Gtk and Mac...

Exactly.

Subject: Re: U++ 2019.1.rc4 released
Posted by cbpporter on Tue, 23 Apr 2019 06:58:20 GMT
View Forum Message <> Reply to Message

mirek wrote on Mon, 22 April 2019 21:03cbpporter wrote on Mon, 22 April 2019 16:09Yeah I'm having a real hard time figuring out what is actually taken from Windows and what is U++ convention when it comes to Chameleon, especially for dark mode.

And probably things are different on Gtk and Mac...

Exactly.

Well I'm having a hard time figuring it out but you know the system. :)

Speaking off, I implemented live skin switching. Pretty hard to do because a lot of controls cache several values and there is no universal OnThemeChanged virtual method with deep GUI traversal.

And I'm also having this problem: I'm in dark mode, I switch to light mode once, icons are still dark. I switch a second time to (the same) light mode, icons are light. Is this the dark mode icon cacher? If yes can we force an invalidate in it with some API?

Anyway, here is an early demo, compiled for Win because I had to do a few small changes to U++:
 https://drive.google.com/open?id=18sCUaufbYdv1XBS8W21mcc61uL qK-YKL

The "skin" buttons on the toolbar change the skin.

Subject: Re: U++ 2019.1.rc4 released
Posted by mirek on Tue, 23 Apr 2019 07:04:12 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Tue, 23 April 2019 08:58mirek wrote on Mon, 22 April 2019 21:03cbpporter wrote on Mon, 22 April 2019 16:09Yeah I'm having a real hard time figuring out what is actually taken from Windows and what is U++ convention when it comes to Chameleon, especially for dark mode.

And probably things are different on Gtk and Mac...

Exactly.

Well I'm having a hard time figuring it out but you know the system. :)

Speaking off, I implemented live skin switching. Pretty hard to do because a lot of controls cache several values

That is exactly the reason why we support setting skin on startup.

See, I guess there is a certain mismatch of expectations. For me and for U++ users I care about, skinning is not that important. I just want the app look as much integrated into host as possible, do not care much above that. And most users do not switch themes 3 times per hour.

Quote:
And I'm also having this problem: I'm in dark mode, I switch to light mode once, icons are still dark. I switch a second time to (the same) light mode, icons are light. Is this the dark mode icon cacher? If yes can we force an invalidate in it with some API?

Fixable, just forgot to fix it.

Mirek

Subject: Re: U++ 2019.1.rc4 released
Posted by cbpporter on Tue, 23 Apr 2019 07:55:40 GMT

View Forum Message <> Reply to Message

---

mirek wrote on Tue, 23 April 2019 10:04
That is exactly the reason why we support setting skin on startup.

See, I guess there is a certain mismatch of expectations. For me and for U++ users I care about, skinning is not that important. I just want the app look as much integrated into host as possible, do not care much above that. And most users do not switch themes 3 times per hour.

Yeah, but that only works if you expect nobody to fine tune the skin. If you are fine with the high contrast eye-strain default skin.

You could fine tune it before TopWindow starts, but TheIDE for example, sets the skin after loading the settings, so all changes done before the main window is created are lost. So maybe if proper live skinning won't be supported, TheIDE should be fixed so you can:

```
#ifdef flagMAIN
GUI_APP_MAIN
#else
void AppMain___()
#endif
{
// Ctrl::ShowRepaint(50);

 Logi() << UPP_FUNCTION_NAME << "(): " << SplashCtrl::GenerateVersionInfo(' ');

 Ctrl::SetUHDEnabled();
 Ctrl::SetDarkThemeEnabled();
 ...snip...
 LoadSkinSetting();

 // HERE IS A CLEAR POINT WHERE I CAN CHANGE THE SKIN ONCE
```

// first line in ide/main.cpp where the GUI is used

This won't work when the user changes skins from settings, but as you said, people don't change skins that often.

But if you:

mirek wrote on Tue, 23 April 2019 10:04
Fixable, just forgot to fix it.

...we may not have a global way to change skins, but in my apps I can emulate the "OnSkinChange" skin invalidation by calling a couple of SetSkins and recreating some borders, so no big issue for me.

For SetDarkThemeEnabled, I propose something like:

void SetDarkThemeEnabled(darkSkin = Ctrl::AutodetectDarkSkin);

with

Ctrl::AutodetectDarkSkin autodetecting and behaving like old SetDarkThemeEnabled(true)
Ctrl::DarkSkin always dark, with SetDarkThemeEnabled(true) and sEmulateDarkTheme always true
Ctrl::WhiteSkin always light, with SetDarkThemeEnabled(false) and sEmulateDarkTheme always false

And bool IsSystemThemeDark() made public.

Subject: Re: U++ 2019.1.rc4 released
Posted by Klugier on Tue, 23 Apr 2019 22:57:11 GMT

View Forum Message <> Reply to Message

Hello,

I aligned some of TheIDE icons as well as this from CtrlLib. Can you guys test and let me know (Tom and cbpporter) - what do you think about this alignment? I would be grateful for your opinions - not all icons are fixed, but some of them are (like main TheIDE toolbar now should display icons correctly on dark theme).

Sincerely,
Klugier

Subject: Re: U++ 2019.1.rc4 released
Posted by mirek on Wed, 24 Apr 2019 06:13:47 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Tue, 23 April 2019 09:55
For SetDarkThemeEnabled, I propose something like:

void SetDarkThemeEnabled(darkSkin = Ctrl::AutodetectDarkSkin);

with

Ctrl::AutodetectDarkSkin autodetecting and behaving like old SetDarkThemeEnabled(true)
Ctrl::DarkSkin always dark, with SetDarkThemeEnabled(true) and sEmulateDarkTheme always true
Ctrl::WhiteSkin always light, with SetDarkThemeEnabled(false) and sEmulateDarkTheme always false

And bool IsSystemThemeDark() made public.

Well, it is worth to keep in mind that the whole mechanism of dark theme in Windows is probably temporary solution. What we are doing there is to emulate dark mode even as Microsoft does not support it for good old win32 applications. It is very likely that this will change and the whole emulation and color conversion for S colors will be done by Win32.

Subject: Re: U++ 2019.1.rc4 released
Posted by cbpporter on Wed, 24 Apr 2019 09:37:43 GMT
View Forum Message <> Reply to Message

mirek wrote on Wed, 24 April 2019 09:13
Well, it is worth to keep in mind that the whole mechanism of dark theme in Windows is probably temporary solution. What we are doing there is to emulate dark mode even as Microsoft does not support it for good old win32 applications. It is very likely that this will change and the whole emulation and color conversion for S colors will be done by Win32.
I wouldn't place my bets on that. And dark mode is included now, so you literally opened Pandora's box. This will be for me a two week 6 hours/day project.

I'm trying to do some minimal changes to U++ that will fix some visual issues while improving usability and compatibility with my themes.

So...

Issue #1: SetDarkThemeEnabled

Problem: SetDarkThemeEnabled is IMHO poorly names, because it is more like EnableDarkModeEmulationByColorInvertingIfWindowsIsInDarkMode .

As a compromise, I added an enum that by default behaves the same, but you can force dark mode or light mode at will.

I tested things and they look compatible.

I would like this change or something that achieves the same results. Feel free to rename stuff.

Thank you!

Attached files in zip.

PS: Do you prefer a better way for patch submission?

File Attachments

1) patch1.zip, downloaded 287 times

Subject: Re: U++ 2019.1.rc4 released
Posted by cbpporter on Wed, 24 Apr 2019 10:04:32 GMT
View Forum Message <> Reply to Message

Edit:
After further testing, I think solution 2 is better.

Solution 1 was present in the attachment, so I am attaching solution 2 in a new post.

Issue #2: High contrast
I have spent a great deal of time looking over Win32 chameleon and now have a good idea on what the OS gives us when it comes to skinning. You weren't kidding when you said that dark mode was "emulation".

With so little information, inverting was pretty clever. I set up more nuanced processing for my skin, but it only works for a single windows skin. So your solution is more universal and probably the best we can do.

The problem is that it results in a high contrast themes.

The problem is that light skins is not black text on white background, only for some controls. There are gray backgrounds, shadows and overall a nice array of colors. On the other hand, if you look at our color palette colors, in dark mode, they are purely white on black background. A "normal" dark skin will be many shades of gray and some gray tinted colors. Pure white and black are taboo.

As I said above, this is not fixable with current support, but we can add some customization.

Let's start small: BlackBorder (and WhiteBorder).

The issues is that BlackBorder is SColorText, so black normally. And white in dark mode. Fair enough. But BlackBorder is used by BlackFrame in CtrlLib too as a border, and there it is similar but different from ViewBorder.

In some skins it might make sense for it to be black/white, but for a modern look, it would be probably SColorText/SColorShadow in light/dark. But this would ruin Draw.

Solution 1:
Leave things as they are, but allow us change BlackBorder color. This way I can skin this in my code.

This solution is a bit awkward since we already have:
const ColorF *BlackBorder();

Maybe there is a better way?

I have attached this solution.

Solution 2:
Make BlackBorder truly black (might or might not be useful to customize) and change BlackFrame to not use black border, but a customizable Chameleon value.

So BlackFrame would have 4 colors to customize and maybe some extra goodies, like border width customization?

And BlackBorder would be used by Draw to draw black borders and never by GUI, while the GUI would sue BlackFrame, a far more flexible solution? With image and hotspot support?

File Attachments

1) patch2.zip, downloaded 261 times

Subject: Re: U++ 2019.1.rc4 released
Posted by cbpporter on Wed, 24 Apr 2019 10:33:42 GMT
View Forum Message <> Reply to Message

Issue #3: DateCtrl an EditField fixes
I fixed some colors in DateCtrl and co and in EditField, the null image is now scaled.

File Attachments

1) patch3.zip, downloaded 259 times

Subject: Re: U++ 2019.1.rc4 released

Posted by cbpporter on Thu, 25 Apr 2019 08:57:44 GMT
View Forum Message <> Reply to Message

Issue #2: High contrast
I's attaching my new preferred method.

Draw remains unchanged. BlackBorder is black and works good on white surfaces and of course inverted for dark mode.

BlackFrame is a dark border, but used for GUI mostly. This one still has the same colors, but this one is customizable.

File Attachments
1) patch4.zip, downloaded 250 times

Subject: Re: U++ 2019.1.rc4 released
Posted by mirek on Fri, 26 Apr 2019 06:52:24 GMT
View Forum Message <> Reply to Message

Thanks, I appreciate your efforts.

That said, priority right now is to release. I do not like to change internals right now.

After that, I will be glad to do so. But perhaps it is a good time now to define desired outcomes.

Certainly there will be changes to CtrlCore/CtrlLib to make skinning better.

We might consider to abandon "skin only at the start of app" too.

It looks like the result of your work will be a bunch of new skins. Years ago I wished to have custom skins as packages in uppsrc/art. Unfortunately, the only addition there so far is "BlueBars". I am hopeful that your new skin will find a way there... That said, make sure they are self-contained (all they need must be in the package).

Mirek

Subject: Re: Dark theme issues
Posted by Oblivion on Fri, 26 Apr 2019 08:49:29 GMT
View Forum Message <> Reply to Message

Hello,

I've noticed one "paper-cut" type problem with repo (svn) dialog. Credentials button is not aligned/anchored to right (it won't update its horz. position on resize)

A more serious one is with declaration to definition conversion method, in Assist.cpp, ln. 952:

It nows removes "override" keyword from definitions, but it does not remove "final" keyword.

And IMO the method used for removing the "override" keyword is somewhat problematic:

```
String AssistEditor::MakeDefinition(const String& cls, const String& _n)
{
 String n = TrimLeft(_n);
 n.Replace("override", "");
 n.Replace("final", ""); // <-- this should be added. It's a practical solution, but is it really OK?
```

Reason:

```
my_foo::finalize();
my_foo:navigationoverridehandler();
```

Parser will remove the "final" and "override" words from the method names...

Now, this is probably a corner case, but still, it might be annoying fo the user.

Best regards,
Oblivion

Subject: Re: Dark theme issues
Posted by mirek on Fri, 26 Apr 2019 09:21:40 GMT
View Forum Message <> Reply to Message

Fixed

Subject: Re: U++ 2019.1.rc4 released
Posted by cbpporter on Fri, 26 Apr 2019 16:01:01 GMT
View Forum Message <> Reply to Message

mirek wrote on Fri, 26 April 2019 09:52
Something that can read and write system colors, you can code in it, outputs in Painter for smooth scaleable graphics, but the end result will of course be cached as an image like Chameleon does today.

I'll look over Esc/Usc and see if they can be hooked up for Painter...

This is off-topic, but I'm surprised Esc isn't integrated with Painter.

Works really well! I see a lot of potential in it! All we need is a bit of color API and helpers. And function overloading if it is not supported :).

File Attachments

1) esc.png, downloaded 491 times