Subject: Re: DarkTheme function parameters changed
Posted by Lance on Mon, 07 Oct 2024 14:49:35 GMT
View Forum Message <> Reply to Message

I see the challenge now.

We need something like theme-dependent color.

With regard to Blend(), would it ever need more than 2 colors?

I have a hackish solution.

```
class DynColor{
  struct _combo{
     Color c;
     char  f[4];
  };

  operator Color()const{
    switch(f[3] & 3){
    case 0: // the 64 bits are a Color*
       return * reinterpret_cast<const Color*>(this);
    case 1: // a normal color hold at data.c
       return c;
    case 2: // Blend, with f[0], f[1] the index
          // of System colors, hopefully
          // we don't have more than 256 of them
          // , and f[2] hold the third paramenter of Blend
      return Blend(GetSysColorFromIndex(f[0],
                GetSysColorFromIndex(f[1]),
                f[2]);
     case 3: // let's handle the
//ctrl.Add(AttrText("Hello there!").Ink(IsDarkTheme() ? Blend(White(), LtBlue()) : Blue());
            // case here
         // theme dependant blend
         return IsDarkTheme() ?
             Blend( treat_c as byte[4], and store blend information there) :
             Blend(GetSysColorFromIndex(f[0],
               GetSysColorFromIndex(f[1]),
               f[2]);

    }
  }

private:
  combo data;
};
```

Just a prototype for conception. We need to consider endianness at least.

Now, we can

ctrl.Add(AttrText("Hello there!").Ink(ConstructDynColor(DarkBlender{...}, NormalBlender{...}));

Yes I am aware this requires changes to AttrText code. I can imagine it's quite involving.

Also this will require everywhere that requires dynamic color to use 64-bit color, a double in storage space.
