Subject: Overriding Display methods too complicated due to high amount of arguments
Posted by Klugier on Sat, 17 Oct 2020 22:01:56 GMT
View Forum Message <> Reply to Message

Hello,

I would like to tell you what I don't like about Display class (CtrlLib). It's two virtual methods are complicated to override due to the need to explicitly repeat the parameter list. It is exactly 6 parameters. Let's take a look at what we have now:

```
class Display {
public:
...
virtual void PaintBackground(Draw& w, const Rect& r, const Value& q,
                    Color ink, Color paper, dword style) const;
virtual void Paint(Draw& w, const Rect& r, const Value& q,
    Color ink, Color paper, dword style) const;
```

The solution for that is to introduce simply DisplayPaintContext that will contain all parameters variables:

```
class Display {
public:
...
virtual void PaintBackground(const DisplayPaintContext& ctx) const;
virtual void Paint(const DisplayPaintContext& ctx) const;

// Or
virtual void PaintBackground(Draw& w, const DisplayPaintContext& ctx) const;
virtual void Paint(Draw& w, const DisplayPaintContext& ctx) const;
```

Of course, it can be implemented in alternative way for example by providing first Draw parameter and rest as a context. Of course we can not simply change the declaration of this method. To fix the problem we should introduce the new one and deprecate the old.

Sample replacment in Display and Array examplex:

```
struct FontFaceDisplay : Display {
 virtual void Paint(Draw& w, const DisplayPaintContext& ctx) const override {
            auto r = ctx.GetRect();

            Font fnt = Font(ctx.GetValue(), r.Height() - 2);
            String txt = Font::GetFaceName(ctx.GetValue());
            w.DrawRect(r, ctx.Paper());
            w.DrawText(r.left + 2, r.top + (r.Height() - GetTextSize(txt, fnt).cy) / 2, txt, fnt,
```

```
ctx.GetInk());
 }
};

struct MyDisplay : public Display {
   virtual void Paint(Draw& w, const DisplayPaintContext& ctx) const override
      w.DrawRect(ctx.GetRect(), paper);
      w.DrawEllipse(ctx.GetRect(), ctx.GetValue());
   }

};
```

Reference:
- Coding Revolution - Long Parameter List
- Refactoring Guru - Long Parameter List
- SonarSource - Static analyzer - Functions should not have too many parameters (4 is maximum limit)

Klugier

Subject: Re: Overriding Display methods too complicated due to high amount of arguments
Posted by Oblivion on Sun, 18 Oct 2020 17:16:38 GMT
View Forum Message <> Reply to Message

As a user, my opinion:

I personally don't find Display paint methods confusing at all.

If the existing methods are going to be deprecated, however, I would suggest passing the reference of Draw and Value in DisplayDrawContext:

```
struct DisplayDrawContext
{
   Draw&       draw;
   const Value& value;
   Rect        rect;
   Color       ink   = SColorText;
   Color       paper = SColorPaper;
   dword       style = 0;
   DisplayDrawContext(Drww& w, const Value& q, const Rect& r) : draw(w), value(q), rect(r) {}
};

MyDisplay().Paint(DisplayDrawContext(w, q, r));
```

Copying the value is not always a good idea as it can contain large stuff.

Best regards,
Oblivion

Subject: Re: Overriding Display methods too complicated due to high amount of arguments
Posted by mirek on Mon, 19 Oct 2020 13:59:10 GMT
View Forum Message <> Reply to Message

Oblivion wrote on Sun, 18 October 2020 19:16

Copying the value is not always a good idea as it can contain large stuff.

Incorrect, does not depend on how large the stuff is.

You would have to try hard to create Value that is expensive to copy because it is large. In fact, I think it is impossible.

If it fits within 12 bytes, you copy just 16 bytes. If it does not, it is created once and then reference counted. Now reference counting is certainly a bit expensive (but not prohibitively), but definitely does not depend on how large the stuff is... :)

Mirek

Subject: Re: Overriding Display methods too complicated due to high amount of arguments
Posted by Oblivion on Mon, 19 Oct 2020 14:53:21 GMT
View Forum Message <> Reply to Message

Quote:Incorrect, does not depend on how large the stuff is.

You would have to try hard to create Value that is expensive to copy because it is large. In fact, I think it is impossible.

If it fits within 12 bytes, you copy just 16 bytes. If it does not, it is created once and then reference counted. Now reference counting is certainly a bit expensive (but not prohibitively), but definitely does not depend on how large the stuff is...

Well now, thanks for the correction, this is excellent news.

And now I've looked into the Value API doc, it is indeed pointed out there. :blush: :lol:

Best regards,
Oblivion

Subject: Re: Overriding Display methods too complicated due to high amount of arguments
Posted by mirek on Sat, 14 Nov 2020 09:02:06 GMT
View Forum Message <> Reply to Message

The rule is absolutely correct (for once :), but you should not be slave to it. It is not the whole story.

If you want one example where this does apply and where changing the code would be appropriate, check EncodeHtml.

With Display, there are some mitigating factors

- the function is rarely called by client code
- the parameter types are well different, which is very important, it greatly reduces the chance of error. It is one thing to have
Foo(int bar, int quo, int boo, int hoo, int woo) and Foo(Font bar, String quo, Color boo, int *hoo, Image wpp).
- it is just 2 more parameters over 4...

Most importantly, U++ rule number one is:

NEVER EVER OVERENGINEER THE STUFF!

Adding one class and 6 methods just to fix nothing? Are you paid by line written or what?

Mirek

Subject: Re: Overriding Display methods too complicated due to high amount of arguments
Posted by Oblivion on Sat, 14 Nov 2020 09:14:41 GMT
View Forum Message <> Reply to Message

Quote:NEVER EVER OVERENGINEER THE STUFF!

Why are my ears ringing now? :lol:

I've duly noted this.

Best regards,
Oblivion

Subject: Re: Overriding Display methods too complicated due to high amount of arguments
Posted by Klugier on Sat, 14 Nov 2020 13:23:11 GMT
View Forum Message <> Reply to Message

Hello Mirek,

I see EncodeHtml declaration. It has 9 parameters, so you should be pretty sure that nobody wants to deal with that code. This is not good for the framework to be over-complicated. We pride ourselves on simplicity, but in some places we do something opposite. Where is logic here? Only to avoid small amount of lines in the library. We are here for the users not to create library with the fewer possible lines of code. If we would follow that path we will end with things like Display API and EncodeHtml. My point of view here is simple, we should think what is the best/easiest for our users.

Quote:- the function is rarely called by client code
But this is the library public API, we should care about it.

Quote:- the parameter types are well different, which is very important, it greatly reduces the chance of error. It is one thing to have
Foo(int bar, int quo, int boo, int hoo, int woo) and Foo(Font bar, String quo, Color boo, int *hoo, Image wpp).
Sure, however you still to remember about them all, when creating new Display! The learning curve with one parameter and six are much more easier.

Quote:- it is just 2 more parameters over 4...
The limit should be reasonable. I think that the clean code assumes that maximum 3 parameters are optimal. Anyway for rare cases 4 is fine too, but more is over-complicated. You could say the same for 6 parameters plus 2, because why not. And we will finish with EncodeHtml :) We should be rational here and follow industry best practices in API design. I am fine with that on application level...

Quote:Adding one class and 6 methods just to fix nothing? Are you paid by line written or what?
If you would like to simplify it you could use structure here, then this 6 methods will not be needed. Just document it well and it should be fine. As I said before adding additional lines of code to the library is fine until end results are better. In this case it is simpler more easy to use API. So, why do not add this additional lines?

So, in this particular change request i do not see any OVERENGINEER-ing we just simplify things and reduces overenginnering on the clinet site. Do not force people to add parameters to they overridden classes to have parameters in their functions they will not needed? Do somebody pays them to have it there :)

KISS on API level :)

Klugier

Subject: Re: Overriding Display methods too complicated due to high amount of arguments
Posted by mirek on Sat, 14 Nov 2020 13:33:41 GMT
View Forum Message <> Reply to Message

Klugier wrote on Sat, 14 November 2020 14:23Hello Mirek,

I see EncodeHtml declaration. It has 9 parameters, so you should be pretty sure that nobody wants to deal with that code. This is not good for the framework to be over-complicated. We pride ourselves on simplicity, but in some places we do something opposite. Where is logic here? Only to avoid small amount of lines in the library. We are here for the users not to create library with the fewer possible lines of code. If we would follow that path we will end with things like Display API and EncodeHtml. My point of view here is simple, we should think what is the best/easiest for our users.

Hey, you are banging on open doors here. I wanted to fix EncodeHTML for years, just never got to it. I just listed that to showcase that I understand the rule and that I agree, in most cases.

Quote:
Where is logic here?

This did not happen to save small amount of lines! This started as function with 3 parameters, but then grew over time without any real incentive to fix it. But you know there are some more rules:

- GET IT DONE FIRST
- HAVE A GOOD REASON TO FIX IT

Now EncodeHTML is ugly, but it is so seldom used that I never got to fixing it. That said, even after the fix we must keep the current version around, obsoleted but available. Because

- HAVE A REALLY REALLY GOOD REASON BEFORE BREAKING THE CLIENT CODE

Mirek

Subject: Re: Overriding Display methods too complicated due to high amount of arguments
Posted by mirek on Sat, 14 Nov 2020 13:38:29 GMT
View Forum Message <> Reply to Message

Klugier wrote on Sat, 14 November 2020 14:23
So, in this particular change request i do not see any OVERENGINEER-ing we just simplify things and reduces overenginnering on the clinet site. Do not force people to add parameters to they overridden classes to have parameters in their functions they will not needed? Do somebody pays them to have it there :)

You know, I feel like you are the only one complainting about this API decision and it even looks like the only reason you do is that you have read some well meant rule somewhere and not even read it in details.

What is said in the first link sounds very reasonable to me. Have you read it all?

Mirek

Subject: Re: Overriding Display methods too complicated due to high amount of arguments
Posted by Klugier on Sat, 14 Nov 2020 19:29:18 GMT
View Forum Message <> Reply to Message

Hello Mirek,

I am happy we are on the same page. I didn't know that this additional parameters were added in the past. Anyway the deprecation mechanism was introduced int c++ since c++14. So, even if the API mistakes were done in the past it can be reversed. The common strategy is to do not remove API immediately. Instead of that mark old method as deprecated and in the message specify that it is deprecated and it will be remove after for example three releases (Optimally it should be specify like 2022.1). This will give people time to migration and always generates warning, so they will know that they need to fix something. Without this one created API would not be able to make a change.

This is common strategy in software world to deal with old API decision that do not fit to current times. Let's just take our favorite language c++ it removes auto_ptr in c++17. It means all code that uses this kind of pointer will not compiler with c++17 standard. The migration is relatively simply and they warn that it will happen in c++14. The same is true for example for Python, they remove unnecessary API within 3 major releases.

Quote:You know, I feel like you are the only one complaining about this API decision and it even looks like the only reason you do is that you have read some well meant rule somewhere and not even read it in details.
I think I follow this rule for some time and I am quite sure that the outcomes are good. In context of software maintainability and easy to use. However to prove legitimacy, a study would have to take place. I think it will not happen soon, but if you ask people how many parameters they want to use they will always answer that less is better.

Quote:What is said in the first link sounds very reasonable to me. Have you read it all?
Yes, I have read it. However, i think there is a catch here :)

To be clear my main reason in this discussion is to make U++ API the most pleasant to use as possible. This is not about criticizing some past decisions. We are all here together and we would like to help and make U++ even better.

Klugier

Subject: Re: Overriding Display methods too complicated due to high amount of arguments
Posted by mirek on Sat, 14 Nov 2020 23:03:22 GMT
View Forum Message <> Reply to Message

Klugier wrote on Sat, 14 November 2020 20:29Hello Mirek,

I am happy we are on the same page. I didn't know that this additional parameters were added in the past.

Please do not mix HtmlEncode and Display. First one is bad and to be fixed, second one is OK.

Quote:
To be clear my main reason in this discussion is to make U++ API the most pleasant to use as possible. This is not about criticizing some past decisions. We are all here together and we would like to help and make U++ even better.

HtmlEncode is sort of past decision. Will be fixed in time. Display::Paint I would do the same no matter what.

BTW, if you read what they suggest really carefully, I think that the good indicator for change is this:

Are you using the newly created object just to fix single method, or there are more cases where this would be usable?

Does not apply always, HtmlEncode is counterargument of sorts, but really does apply to Display::Paint. Intruducing redundant object there just to satisfy the idea that function cannot have more than 4 parameters would really make me sick. Literally :) It would be really bad code smell for me.

Mirek
