Subject: Know what you're using. Size of some common types.
Posted by Lance on Mon, 27 Dec 2021 18:33:50 GMT
View Forum Message <> Reply to Message

Test result

Now I have redone the test in Release mode, the result is not as eye-catching.

Event<> is of the same size as void *, this is better than I had expected. Of course actual memory used might be more than that: a thisfn with the sizeof of member function pointer and an object pointer for this will have difficulty to fit in the room for a void *.

Using 64 bit for context. I would think a ScrollBar is too big for the job it does. Ideally it should be done without containing 4 Buttons or Button should somehow be compacted to use significantly less room and leave some functions to derived class or optionally(pay per use) memory allocated from heap.

Anyway, the result is quite satisfying and reassuring.

BTW, test program:#include <CtrlLib/CtrlLib.h>
#include <GridCtrl/GridCtrl.h>
#include <TabBar/TabBar.h>

using namespace Upp;
#define SZ(t) "\n"#t"\t" << sizeof(t) /*<<"\t"<<alignof(t)*/

```
GUI_APP_MAIN
{
 String s;
 s << SZ(void *)
  << SZ(Value)
  << SZ(String)
  << SZ(Event<>)
  << SZ(Vector<int>)
  << SZ(Button)
  << SZ(EditField)
  << SZ(ScrollBar)
  << SZ(TabBarCtrl)
  << SZ(ArrayCtrl)
  << SZ(GridCtrl);
 RLOG(s);
}
```

File Attachments

1) a.png, downloaded 1326 times

SizeOf	CLANG	CLANG64	MSBT22	MSBT22x64
void *	4	8	4	8
Value	16	16	16	16
String	16	16	16	16
Event<>	4	8	4	8
Vector<int>	12	16	12	16
Button	160	224	160	224
EditField	320	440	320	440
ScrollBar	808	1136	816	1136
TabBarCtrl	728	1024	728	1000
ArrayCtrl	2680	3784	2688	3752
GridCtrl	4912	6784	4904	6712

Subject: Re: Know what you're using. Size of some common types.
Posted by Klugier on Mon, 27 Dec 2021 18:58:22 GMT
View Forum Message <> Reply to Message

Hello Lance,

Could you tell us more what is the root cause of your problems? Today, you created several threads about optimization. What is the reason of it? Do you want to write application on some embedded system?

In order to understand Button size problem it would be good to know the size of Ctrl (The class from which all controls inherits) and Pusher (Base class for Button). I am also analyzing ScrollBar code and it seems that for most themes we do not need prev2 and next2 buttons:

Button prev, prev2, next, next2;

I could imagine themes without buttons (like current KDE one). In this case keeping four buttons on stack seems like a waste. It should be replaced with something like std::optional<Button> (Upp::One):

One<Button> prev, prev2, next, next2;

Klugier

Subject: Re: Know what you're using. Size of some common types.
Posted by Lance on Mon, 27 Dec 2021 19:32:31 GMT
View Forum Message <> Reply to Message

Hi Klugier:

Thank you for being responsive.

No I am not doing embedding developing. Just out of curiosity.

Regards,
Lance

Subject: Re: Know what you're using. Size of some common types.
Posted by Lance on Mon, 27 Dec 2021 19:37:46 GMT
View Forum Message <> Reply to Message

BTW, theming is something I almost have 0 knowledge. Any suggestion on which part of uppsrc or examples/references/tutorials etc, I should take a first look at?

Subject: Re: Know what you're using. Size of some common types.
Posted by Klugier on Mon, 27 Dec 2021 20:42:48 GMT
View Forum Message <> Reply to Message

Hello Lance,

You should read documentation page about Chameleon. You could find it here. Code sample could be find in reference/Chameleon.

Klugier

Subject: Re: Know what you're using. Size of some common types.
Posted by Lance on Tue, 28 Dec 2021 01:29:56 GMT
View Forum Message <> Reply to Message

Thanks, Klugier! It's going to take me a while to digest the materials.

Subject: Re: Know what you're using. Size of some common types.
Posted by Novo on Tue, 28 Dec 2021 04:17:29 GMT
View Forum Message <> Reply to Message

It is possible to reduce size of data structures by eliminating padding gaps ...

Subject: Re: Know what you're using. Size of some common types.
Posted by Lance on Tue, 28 Dec 2021 19:27:43 GMT
View Forum Message <> Reply to Message

Novo:

By moving similarly algined items together (combined two seperated bitfield section, and move int8 item together with them)

```
 int8    push;
 int8    light;
 bool    horz:1;
 bool    jump:1;
 bool    track:1;
 bool    autohide:1;
 bool    autodisable:1;
 bool    is_active:1;
```

We can save like 16 bytes on 64 bit platform. If we try harder, like declaring linesize, etc as int8, we can save some more bytes. But these are all marginal.

What I have in mind is to get rid of the 4 Buttons completely. That way we can save around 1K in each ScrollBar object. ScrollBar is definitely a class that is worth rewritten. The rewriting work might even not be too difficult.

Here is a function from ScrollBar.cpp

```
int  SkrollBar::GetMousePart()
{
 int q = -1;
 for(int i = 2; i >= 0; i--)
  if(HasMouseIn(GetPartRect(i))) {
   q = i;
   break;
  }
 return q;
}
```

The slider area is divided into 3 parts, the upper blank area, the slider button, the bottom blank area. We can divide it into 5(or seven), with addition to accomodate prev,next (and even prev2, next2: anybody can ecudate me on what these two buttons are doing? I don't see it on the GUI at all)

These kind of refinement do not add functionalities but still contribute to a better U++ experience.

Subject: Re: Know what you're using. Size of some common types.
Posted by Novo on Tue, 28 Dec 2021 22:49:48 GMT

If one is allocating millions of inefficiently aligned structures, then he/she is wasting a lot of memory.

BTW, there are tools which visualize amount of wasted memory caused by inefficient alignment of data.

Subject: Re: Know what you're using. Size of some common types.
Posted by Lance on Wed, 29 Dec 2021 00:42:34 GMT

Agreed. Here the waste on suboptimal alignment (or possible gain by rearrange member vars to arrive at the least waste on padding) is insignificant comparing to the size of the object(of the class).

I do believe we should pay some more attention to the order we declare struct/class member variable to arrive at more efficient memory usage. I come across bitfields separated by other type of variable once in a while. The ScrollBar class is an example of this.

```
private:
 int     thumbpos;
 int     thumbsize;
 bool    horz:1;
 bool    jump:1;
 bool    track:1;
 int     delta;
 int8    push;
 int8    light;

 Button  prev, prev2, next, next2;
 int     pagepos;
 int     pagesize;
 int     totalsize;
 int     linesize;
 int     minthumb;
 bool    autohide:1;
 bool    autodisable:1;
 bool    is_active:1;
```

But this are overall of a less degree of concern. The percentage saving is usual immaterial.

Subject: Re: Know what you're using. Size of some common types.
Posted by Lance on Thu, 30 Dec 2021 01:39:57 GMT

I manage to create a ScrollBar twin (SkrollBar) that should look and act exactly the same (to be

safe, I leave untouched some old code that could benifit from rewritten with new facilities but I am not very sure yet).

Class Size
------- -----
Button 224
SkrollBar 232
ScrollBar 1136
TabBarCtrl 1000
ArrayCtrl 3752
GridCtrl 4904

See above table. SkrollBar is now almost same size of Button. Above size of GridCtrl is after both ScrollBar objects in it has been redefined as of type SkrollBar. Imagine howmany GridCtrl/ArrayCtrl you will be using in your program

The code is still very rough. I dare not to touch the original Slider() portion's Paint & mouse event (I wasn't able to understand it very well). I figure, push and light can be do without, linesize and minthumbsize will be more than enough with an int8. I haven't tries (removing light and push will require rewriting some code), chance is we can get SkollBar of the same size of Button.

Attached is a test that use ScrollBar/SkrollBar Vert()/Horz() side by side. They should look the same and behave the same. The Outer ones are SkrollBar. They respond to MouseMove in debug mode to report the section number that the mouse is currently in.

With a vertical SkrollBar, section 0 is the prev button, section 2 is the prev2 button (mostly invisble), section 3 is the portion of slider above the thumb, section 4 is the thumb, section 5 is the portion of slider under the thumb, section 5 is the next2 button (mostly invisible), section 6 is the next button.

File Attachments

1) SkrollBar.zip, downloaded 319 times

Subject: Re: Know what you're using. Size of some common types.
Posted by Lance on Fri, 31 Dec 2021 18:51:15 GMT
View Forum Message <> Reply to Message

A more presentable state, see attached zip file. Now sizeof(ScrollBar)=sizeof(Button).

Please make a copy of your existing uppsrc/CtrlLib/ScrollBar.*, and and unpack the zip file to overwrite existing ScrollBar.{h,cpp} in the uppsrc/CtrlLib folder. The revision is transparent to library users. Your program should feel no difference, except some savings on executable size and for each ScrollBar object you used, you will save around 900 bytes of memory.

File Attachments

1) ScrollBar.zip, downloaded 336 times

Subject: Re: Know what you're using. Size of some common types.
Posted by Lance on Fri, 31 Dec 2021 19:20:48 GMT
View Forum Message <> Reply to Message

In the spirit of previous discussion with Novo, the following minor change to CtrlCore/CtrlCore.h should decrease the sizeof(Ctrl) and that of all its derivative by 8 bytes on a 64-bit platform. While on 32-bit system there is no gain(Ctrl has been perfectly fine tuned for 32-bit platform), and there should be no penalties either.

Current code:

```
 Top         *top;
 int         exitcode;

 Ctrl        *prev, *next;
 Ctrl        *firstchild, *lastchild;//16
 LogPos       pos;//8
 Rect16       rect;
 Mitor<Frame> frame;//16
 String       info;//16
 int16        caretx, carety, caretcx, caretcy;//8

 byte         overpaint;
```

Proposed change:

```
 Top         *top;

 Ctrl        *prev, *next;
 Ctrl        *firstchild, *lastchild;//16
 LogPos       pos;//8
 Rect16       rect;
 Mitor<Frame> frame;//16
 String       info;//16
 int16        caretx, carety, caretcx, caretcy;//8

 int         exitcode; // move the line here

 byte         overpaint;
```

After the change, sizeof(Ctrl) is reduced from 152 bytes to 144 bytes on 64bit platform (both MSBT22x64 and CLANG64), while on 32bit platform, it remains unchanged with CLANG, but increases by 8 bytes with MSBT22. This increase is unexpected. If anybody can explain it or figure out a way to avoid it, it will be fully appreciated.

@mirek or @klugier, please consider apply the change after identifying and fixing the unexpected behavior with MSBT. The change is too simple to have potential danger and will affect all objects of Ctrl and its derivatives.

PS: By making use of MSC 32 bit flag _M_IX86, the above problem could be circumvented as

follows:

```
...
 Top        *top;
#if defined(_M_IX86) // 32bit MSC compiler
 int   exitcode;
#endif

 Ctrl       *prev, *next;
 Ctrl       *firstchild, *lastchild;//16
 LogPos      pos;//8
 Rect16      rect;
 Mitor<Frame> frame;//16
 String      info;//16
 int16       caretx, carety, caretcx, caretcy;//8

#if !defined(_M_IX86)
 int         exitcode;
#endif
 byte        overpaint;

 bool        unicode:1;

 bool        fullrefresh:1;

 bool        transparent:1;
 bool        visible:1;
 bool        enabled:1;
 bool        wantfocus:1;
 bool        initfocus:1;
 bool        activepopup:1;
 bool        editable:1;
 bool        modify:1;
 bool        ignoremouse:1;
...
```

File Attachments

1) a.png, downloaded 1166 times

	MSBT22		CLANG	
	32bit	64bit	32bit	64bit
Original Ctrl	104	152	104	152
Revised Ctrl	112	144	104	144

Subject: Re: Know what you're using. Size of some common types.
Posted by Lance on Fri, 31 Dec 2021 19:28:10 GMT
View Forum Message <> Reply to Message

And in the same spirit, move

```
protected:
 bool    monoimg;
 byte    type;
```

out of class Button, to its base class Pusher, ending with something like

```
class Pusher : public Ctrl {
public:
 virtual void   CancelMode();
 virtual void   LeftDown(Point, dword);
 virtual void   MouseMove(Point, dword);
 virtual void   MouseLeave();
 virtual void   LeftRepeat(Point, dword);
 virtual void   LeftUp(Point, dword);
 virtual void   GotFocus();
 virtual void   LostFocus();
 virtual void   State(int);
 virtual String GetDesc() const;
 virtual bool   Key(dword key, int);
 virtual bool   HotKey(dword key);
 virtual dword  GetAccessKeys() const;
 virtual void   AssignAccessKeys(dword used);

private:
 bool    push:1;
 bool    keypush:1;
 bool    clickfocus:1;
protected:
 bool    monoimg;
 byte    type;
```

Should not harm Pusher but decrease sizeof(Button) and that of its derivatives by 8 bytes on 64-bit platform and 4 bytes on 32-bit platform.

The changes are too trivial to be of danger.

Subject: Re: Know what you're using. Size of some common types.
Posted by mirek on Wed, 05 Jan 2022 09:47:55 GMT
View Forum Message <> Reply to Message

BTW, there are two benchmarking packages for this purpose already:

benchmarks/sizeof

bemchmarks/sizeof_gui

Anyway, this is definitely a good effort! Keeping sizeof(Ctrl) low is important.

Mirek

Subject: Re: Know what you're using. Size of some common types.
Posted by Lance on Fri, 07 Jan 2022 16:25:51 GMT
View Forum Message <> Reply to Message

mirek:

Thanks. The output from above test program works better with Excel like utility so that record keeping and comparison are easier.

While there will not be a fixed ratio between total Ctrl used and that of ScrollBar used, I have test run some examples to get a feel of a rough ratio.

Examples/AddressBook(Up to the mainwindow is open): Max Ctrl Used: 96, Max ScrollBar used: 11 (9:1)

Examples/HomeBudget(Up to the mainwindow is open): Max Ctrl Used: 277, Max ScrollBar used: 22 (13:1)

Reference/GridCtrlTest(Up to the maindown is open): Max Ctrl Used: 1011, Max ScrollBar used: 132 (8:1)

UppSrc/ide(open a blank CtrlLib application): Max Ctrl Used: 802, Max ScrollBar used: 217 (4:1);

Again ide, but this time open UppSrc/ide, and in it, click the very last file ide.lay: Max Ctrl Used: 22001, Max ScrollBar used: 2181 (10: 1)

Considering the absolute and percentage saving we derived from the new implementation of ScrollBar (well, only an insignificant part of it to be more precise), accepting new ScrollBar would be as beneficial as compacting Ctrl, if not more: mirek mentioned in another discussion that he could replace a String with a const char *, resulting in additional saving of 8 bytes on 64 bit platforms and 12 bytes on 32 bit ones. Combining with that derived from rearranging member variables to minimize padding, we end up with 16 bytes each on both 32 and 64 bit platforms. That's about it if we don't want to lose any functionalities.

ScrollBar is pretty isolated: I don't think many people will need to derive from it. As long as we maintain the user interface stable/untouched, and test it on different platforms/settings, it should pose very low risk of messing up things (to replace it). These are all gain at no cost: by the way, new ScrollBar results in smaller executable too. Now my question is: Why not?

PS: Even if you derive from ScrollBar, I don't think you will be affected: the functions/member

variables changed are all private (as far as I can remember).

Subject: Re: Know what you're using. Size of some common types.
Posted by Lance on Fri, 07 Jan 2022 16:44:17 GMT
View Forum Message <> Reply to Message

Assuming the proposed data reorganization for Ctrl and Pusher & Button are both applied, sizeof(Button) will be reduced 16 bytes from 224 to 208 (on 64bit platforms). By ridding of the 4 contained Buttons, each ScrollBar object will be using 4x204=832 less bytes.

Use second case of uppsrc/ide on 64 bit platform for example, compacting Ctrl to save 8 bytes each will have a total memory saving of 22001x8=176,008 bytes. The new ScrollBar implementation will have an incremental memory saving of 2181x832=1,814,592 bytes.

I am not saying 2M or even 10M of memory saving will make much a difference in now-a-days hardware, but reducing sizeof(ScrollBar) to 1/5 of what it is (actually even less) might not be less important than reducing sizeof(Ctrl) by 8 - 24 bytes from practical perspective.

Subject: Re: Know what you're using. Size of some common types.
Posted by mirek on Fri, 07 Jan 2022 17:32:00 GMT
View Forum Message <> Reply to Message

Well, I do not want to dive into this now - we need to release soon to stabilise huge changes done and there is a lot of more important things to fix.

Anyway, after that, this is quite important and moreover fun.

The situation with ScrollBar buttons repeats itself in other places, e.g. with SpinButtons. I am thinking there could be some nice generic solution, something like "Buttons" partly abstract class that would represent "embedded buttons" using just virtual methods of derived class (similar fashion to your ScrollBar implementation, but in generic way). EditIntWithSpin is quite big sizeof as well and it is even more important (as it is has higher chance to be used in huge quantities).

But that all is pennies compared to DropList sizeof. That one needs converting PopUpTable list; to One<PopUpTable> list; and only create when needed and then delete. Unfortunately, it is delicate work, a lot of things there could go wrong.

Another things I would like to see reduced is String Ctrl::info. const char * would work there with some effort. 8 bytes saved (Maybe add some flag that it points to Layout ID only, then it could point to character literal in layout widget, even more savings).

Subject: Re: Know what you're using. Size of some common types.

Posted by mirek on Fri, 07 Jan 2022 17:34:29 GMT
View Forum Message <> Reply to Message

Lance wrote on Fri, 07 January 2022 17:44Assuming the proposed data reorganization for Ctrl and Pusher & Button are both applied, sizeof(Button) will be reduced 16 bytes from 224 to 208 (on 64bit platforms). By ridding of the 4 contained Buttons, each ScrollBar object will be using 4x204=832 less bytes.

Use second case of uppsrc/ide on 64 bit platform for example, compacting Ctrl to save 8 bytes each will have a total memory saving of 22001x8=176,008 bytes. The new ScrollBar implementation will have an incremental memory saving of 2181x832=1,814,592 bytes.

I am not saying 2M or even 10M of memory saving will make much a difference in now-a-days hardware, but reducing sizeof(ScrollBar) to 1/5 of what it is (actually even less) might not be less important than reducing sizeof(Ctrl) by 8 - 24 bytes from practical persperctive.

Frankly, I am not that concerned about saving memory in such normal situations.

However, I have seen/used ArrayCtrls with thousands of embedded DropLists. There it could be huge....

Subject: Re: Know what you're using. Size of some common types.
Posted by Lance on Fri, 07 Jan 2022 18:01:25 GMT
View Forum Message <> Reply to Message

Thanks for the feedback, mirek. Please stay focused on important/urgent matters.

And come back to consider these more marginal issues when you have time & interests.

Thank you again for such a great product named Ultimate++. I wish it would stay around as long as there are people using C++. I feel sad that such great work is not as popular as it deserves. I tried (quite hard and persistently) to sell it to my teenage son and his friends but haven't been very successful. More good projects that's written with UPP facilities need to be around to spread its influence. E.g. pgAdminIII could benefit from a better/more responsive GUI. Our community members should be able to come up with better candidates. Downstream demands will also help to find directions of Core Upp development.

Subject: Re: Know what you're using. Size of some common types.
Posted by Lance on Fri, 07 Jan 2022 18:26:35 GMT
View Forum Message <> Reply to Message

I kind of understand what you intends to do, because I have also thought in that direction. I call it FakeCtrl: Ctrl-like objects that rely on containing Ctrl's Paint(...), LeftDown(...) etc to mimic an actual Ctrl. Problems is, if we want to generalize it, we will need to have almost all state (and hence member variable) a Ctrl has: the saving won't be significant. While a generic solution is hard to come up with, case by case is easy.

Subject: Re: Know what you're using. Size of some common types.
Posted by mirek on Fri, 07 Jan 2022 21:22:00 GMT
View Forum Message <> Reply to Message

Lance wrote on Fri, 07 January 2022 19:26I kind of understand what you intends to do, because I have also thought in that direction. I call it FakeCtrl: Ctrl-like objects that rely on containing Ctrl's Paint(...), LeftDown(...) etc to mimic an actual Ctrl. Problems is, if we want to generalize it, we will need to have almost all state (and hence member variable) a Ctrl has: the saving won't be significant. While a generic solution is hard to come up with, case by case is easy.

Only for buttons. That is not that hard.

Mirek

Subject: Re: Know what you're using. Size of some common types.
Posted by Lance on Sun, 09 Jan 2022 19:36:10 GMT
View Forum Message <> Reply to Message

If the intended modified Button will continue to be a Ctrl derivative so that it works seamlessly with existing code, the minimum size it will have is sizeof(Ctrl). In this particular situation, the Ctrl member variable LogPos(and maybe more candidates) can be done without, we can reuse it to put a Style * st; and probably WhenPush event variable somewhere else. In any case, the minimum size will be sizeof(Ctrl). The savings won't be quite as good (significantly less than the case when buttons or even frames are ridden of), even though it indeed involves least work and least chance of surprise.

Another way is to define it something like

```
class FakeButton
{
public:
   void Paint(Draw& w, Rect& where, Button::Style& style, int
state/*normal,hot,pressed,disabled*/);

   Event<> WhenPush;
};
```

This way sizeof(FakeButton)==sizeof(Event<>)(==sizeof(void*)), but the effort involved in revising containing Ctrl (ScrollBar, EditWithSpin, DropList, etc)'s Paint,LeftDown,LeftRepeat, etc, will be quite similar to hardcoding each of the Ctrl's method that we need to change to allow the savings to take place.

Subject: Re: Know what you're using. Size of some common types.
Posted by mirek on Mon, 10 Jan 2022 00:07:26 GMT
View Forum Message <> Reply to Message

Lance wrote on Sun, 09 January 2022 20:36If the intended modified Button will continue to be a Ctrl derivative so that it works seamlessly with existing code, the minimum size it will have is sizeof(Ctrl). In this particular situation, the Ctrl member variable LogPos(and maybe more candidates) can be done without, we can reuse it to put a Style * st; and probably WhenPush event variable somewhere else. In any case, the minimum size will be sizeof(Ctrl). The savings won't be quite as good (significantly less than the case when buttons or even frames are ridden of), even though it indeed involves least work and least chance of surprise.

Another way is to define it something like

```
class FakeButton
{
public:
   void Paint(Draw& w, Rect& where, Button::Style& style, int
state/*normal,hot,pressed,disabled*/);

   Event<> WhenPush;
};
```

This way sizeof(FakeButton)==sizeof(Event<>)(==sizeof(void*)), but the effort involved in revising containing Ctrl (ScrollBar, EditWithSpin, DropList, etc)'s Paint,LeftDown,LeftRepeat, etc, will be quite similar to hardcoding each of the Ctrl's method that we need to change to allow the savings to take place.

This would be too long. What we need is class Buttons like this:

```
class Buttons : Ctrl {
   virtual int  GetButtonCount() = 0;
   virtual Rect GetButtonRectint i) = 0;
   virtual int  ButtonMouseDown(Point p);
   virtual int  ButtonMouseUp(Point p);
   virtual int  ButtonMouseMove(Point p);
}
```

Subject: Re: Know what you're using. Size of some common types.
Posted by Lance on Mon, 10 Jan 2022 01:00:13 GMT
View Forum Message <> Reply to Message

Great. For WithSpin<> which in turn contains SpinButtons, this can roughly reduce the object size by 2*sizeof(Button) - sizeof(Ctrl). That's a great achievement with minimal impact on existing code. It will just work.(Well, SpinButtons exposed Button inc, dec; but it should not be referenced much except in the library implementation anyways).

And it's a generic solution. No matter how many buttons it fakes, it will take up room of only sizeof(Ctrl).

For WithSpin in particular, another route I was considering (I am not versed with Upp enough to know whether it will work) is to start from EditField. Basically add 2 bitfield bool

```
class EditField:....
{
....
public:
	void Paint(Draw& d)override{
		PaintSpinButtons();
		DoOriginalEditFieldPaintOnReducedSize();
	}
	void LeftDown(Point p, dword f)override{
		if(p not in SpinPart)
			Parent::LeftDown(p,f);
		else if(p in UpperPart of Spin)
			WhenSpin(false);
		else// (p in LowerPart of Spin)
			WhenSpin(true);
	}
	void LeftRepeat(Point p, dword f)override{
		LeftDown(p,f);
	}
	Image CursorImage(Point p, dword)override{
	   // Image according to part of the Ctrl
	}
	//... maybe more to be override'd or rewrite to take care of Fake SpinButton part.

	Event<bool> WhenSpin;
private:
	bool with_spin : 1;
	bool spin_visible : 1;

	Size GetReducedSize(){
		 Size sz=GetSize();
		 if(with_spin && spin_visible)
			 reduce_size_to_leave_room_for_spin_buttons(sz);
	   }
}
```

Then in actual types(EditIntWithSpin, EditInt64WithSpin, etc) that needs SpinButtons, we just turn the flags on in respective constructors, and connect to WhenSpin event.

I am not sure if we claim part of EditField as Frame without actully AddFrame, etc, will work as wished.

If you figure this route is worth considering, I can do a preliminary implementation. Otherwise (if you prefer the more normal Buttons route), I will wait and see.

Subject: Re: Know what you're using. Size of some common types.
Posted by Lance on Mon, 10 Jan 2022 20:43:41 GMT
View Forum Message <> Reply to Message

Buttons established a concept. If implemented with due care, it will arrive at same Ctrl size (eg, ScrollBar, SpinButtons) as when hardcoded (like the way I do ScrollBar). That being said, Buttons interface are not expected to save much work (than hard coded way), if any. Personally I will do without: I don't see the benefit here, except the visual clue that they are doing similar things.

I have performed a simple test on adding SpinButtons to EditField without resorting to a CtrlFrame, and it works like a charm. I know this is not conceptually right, but it will work in practice and save memory of a little over sizeof(Ctrl) for each EditWithSpin object than the Buttons way mirek is considering. The only functionality it lose is when there are multiple frames, you cannot choose to put the SpinButtons frame on anywhere except innermost, because it's not a real CtrlFrame.

The Test is very simple, start a CtrlLib project with a Main Window with a EditField and a EditInt64WithSpin. Now we start to play around with CtrlLib.

In <CtrlLib/EditCtrl.h, add a new virtual function to class EditField

```
class EditField : public Ctrl, private TextArrayOps {
public:
 virtual void  Layout();
 virtual void  Paint(Draw& draw);
 virtual void  LeftDown(Point p, dword keyflags);
       ... omitted
 virtual void   State(int);


       // newly introduced
 virtual Size   GetReducedSize()const{
            // EditField::GetReducedSize() should just return GetSize();
            // it's the WithSpin derived class that should override this
            // function to reserve room for SpinButtons.
            //
            // here we reduce it as if allocating space for SpinButtons
            Size sz = GetSize();
```

```
        sz.cx -= 30;
        if(sz.cx < 0)
            sz.cx=0;
        return sz;
    }[/b]
...
```

In <CtrlLib/EditField.cpp>, search all occurence of "GetSize()", if it's not "GetSize().cy", and is not "GetParent()->GetSize()", repace it with "GetReducedSize()". Run the test program, notice the room for SpinButton has been allocated, and play with some input in the EditField, see it scroll horizontally properly when the text gets really long. Now we can be certain this route is feasible. And it will come out with more compact EditXXWithSpins than to retain Buttons in a CtrlFrame. I would say the amount of coding involved are quite similar in both ways.

I will otherwise leave EditField (except probably add a spin_visible bitfield member intended for EditXXwithSpin). Then override Paint, relevant mouse event virtual function in WithSpin<> template class. It's quite similar to what I did with ScrollBar. No impact will be felt by libary users except smaller EditXXWithSpin objects---class hierarchy and interfaces remain unchanged.

Subject: Re: Know what you're using. Size of some common types.
Posted by Lance on Tue, 11 Jan 2022 01:29:27 GMT
View Forum Message <> Reply to Message

WithSpin<> has method named OnSides() which allows the SpinButtons be splitted on both side of the EditField. Not putting SpinButtons in a CtrlFrame will require a lot more work to modify EditField::Paint() etc. Housing it in a CtrlFrame is cleaner and safer.

Subject: Re: Know what you're using. Size of some common types.
Posted by mirek on Tue, 11 Jan 2022 12:41:17 GMT
View Forum Message <> Reply to Message

Lance wrote on Mon, 10 January 2022 21:43That being said, Buttons interface are not expected to save much work (than hard coded way), if any.

You wanna bet?

Mirek

Subject: Re: Know what you're using. Size of some common types.
Posted by Lance on Tue, 11 Jan 2022 14:19:52 GMT

View Forum Message <> Reply to Message

I see. You will provide underlying facilities to support the Buttons interface so that by implementing the interface, underlying CtrlLib or CtrlCore facilities will draw it correctly and deliver mouse event etc as if each button is a actual Ctrl.

For WithSpin<>.OnSides, the painting can be taken care of by Draw.Offset. Not sure if there will be more complications.

Subject: Re: Know what you're using. Size of some common types.
Posted by Lance on Tue, 11 Jan 2022 16:40:56 GMT
View Forum Message <> Reply to Message

This idea might worth further generalization:

For example, we have an ArrayCtrl who is housing thousands of child Ctrl: edits, buttons, etc. Technicallu, each child don't need to worry about its location and size which will be set to whatever the ArrayCtrl decides. I am sure there will be a lot more states a resident child doesn't need to worry about, similar to the Buttons case, but more generic: each child is determined by a row and a column, unlike in Buttons case we only need an index to identify a fake Ctrl. As these interface may be required by both ArrayCtrl and GridCtrl, ideally the support can be built into Ctrl base class. And Edits, Button, etc need to provide a fake version (not derived from Ctrl, while the real version ,like
what they are, is just fake controls contained in Ctrl which they also derive from.

Subject: Re: Know what you're using. Size of some common types.
Posted by Lance on Tue, 11 Jan 2022 21:07:07 GMT
View Forum Message <> Reply to Message

If ArrayCtrl limits children to Fake Ctrls only, SyncLineCtrls etc will be dispensible.

Back to Buttons, some chain of thoughts. Instead of inheriting from Ctrl, it should be leave as an interface (no data members); concrete Ctrls will decide whether to inherit from Ctrl, ArrayCtrl, EditField or other Ctrl derivatives and implementing the interface to use the Ctrl's Paint & other virtual functions to fake out Buttons.

This will add a cost of sizeof(void *) (pointer to interface vtbl) to each object of Classes that implement the interface. Why not add the interface (a set of virtual functions) to Ctrl directly to save this vtble pointer?

```
class Ctrl : public ...
{
public:
    virtual void Paint(Draw& w)
    {
```

```
        for(int i = 0; i < GetFakeCtrlsCount(); ++i)
        {
            Rect r=GetFakeCtrlsRect(i);
            w.Clipoff(r);
            PaintFakeCtrls(w, i);
            w.End();
        }
    }
    virtual void LeftDown(Point p, dword flag)
    {
        for(int i = 0; i < GetFakeCtrlsCount(); ++i)
        {
            Rect r=GetFakeCtrlsRect(i);
            if( MouseInRect(r) )
            {
                p.x -= r.left; p.y -= r.top;
                FakeCtrlsLeftDown(p, flag, i);
                break;
            }
        }
    }
    ...
     // added to support fake Ctrls
     virtual int GetFakeCtrlsCount()const{ return 0; }
     virtual void PaintFakeCtrls(Draw& w, int index){}
     virtual void FakeCtrlsLeftDown(Point p, dword flag, int index){}
     ... and some more virtual function of Ctrl with an extra parameter index.
};
```

This way we don't explicitly limit the type of fake Ctrls (Not even by a non-restrictive but suggestive name "Buttons"). It's totally up to each derivative that actually implements the interface to decide what Ctrls it wants to fake.

PS: instead of repeatedly calling multiple virtual functions to Paint each fake Ctrl, it make sense to instead

```
class Ctrl: ...
{
public:
    ...
     virtual void PaintFakeCtrls(Draw& w);
     ...
```

Then the PaintFakeCtrls() will be quite like ScollBar::Paint(). I don't know, let's see how you do it clearly and efficiently.

Subject: Re: Know what you're using. Size of some common types.
Posted by Lance on Tue, 11 Jan 2022 21:25:05 GMT
View Forum Message <> Reply to Message

The cost is the size of the vtable of Ctrl and each of its derivatives almost doubles. With the number of classes in the hierarchy, this could be quite big to ignore. Then we get something like suggested by you

```
class CtrlWithFakeCtrls : public Ctrl
{
...
}
```

Only Ctrls derived from the class are able to make use of the facility. If any existing Ctrl is believed to be able to benefit from it, we just change its base class.

Now I get your point. It's nice.

Subject: Re: Know what you're using. Size of some common types.
Posted by mirek on Wed, 11 May 2022 07:42:24 GMT
View Forum Message <> Reply to Message

So here I am after one month of optimizing:

Current U++:

* C:\upp\out\benchmarks\CLANGx64.Gui\sizeof_gui.exe 29.04.2022 17:35:45, user: cxl

```
sizeof(Image) = 8
sizeof(RichText) = 248
sizeof(Ctrl::LogPos) = 8
sizeof(LabelBase) = 136
=============
sizeof(Ctrl) = 152
sizeof(ScrollBar) = 1136
sizeof(HeaderCtrl) = 1456
sizeof(Button) = 224
sizeof(Switch) = 224
sizeof(Label) = 296
sizeof(EditField) = 440
sizeof(EditString) = 456
sizeof(EditInt) = 472
sizeof(SpinButtons) = 472
sizeof(EditIntSpin) = 952
sizeof(DisplayPopup) = 256
sizeof(PopUpTable) = 3824
```

sizeof(WithDropChoice<EditString>) = 4920
sizeof(DropList) = 4488
sizeof(ArrayCtrl) = 3784
sizeof(TreeCtrl) = 3656
sizeof(TreeCtrl::Node) = 80
sizeof(FileSel) = 26056
sizeof(RichTextView) = 1632
sizeof(ColumnList) = 1752
sizeof(RichEdit) = 69464

gui_sizeof branch:

* C:\upp\out\gui_sizeof_benchmarks\CLANGx64.Blitz.Gui\sizeof_gui.exe 11.05.2022 09:39:05, user: cxl

sizeof(Image) = 8
sizeof(RichText) = 248
sizeof(Ctrl::LogPos) = 8
sizeof(LabelBase) = 64
==========
sizeof(Ctrl) = 104
sizeof(ScrollBar) = 192
sizeof(HeaderCtrl) = 464
sizeof(Button) = 176
sizeof(Switch) = 176
sizeof(Label) = 176
sizeof(EditField) = 320
sizeof(EditString) = 336
sizeof(EditInt) = 352
sizeof(SpinButtons) = 376
sizeof(EditIntSpin) = 384
sizeof(DisplayPopup) = 16
sizeof(PopUpTable) = 1552
sizeof(DropList) = 440
sizeof(WithDropChoice<EditString>) = 760
sizeof(ArrayCtrl) = 1512
sizeof(TreeCtrl) = 1288
sizeof(TreeCtrl::Node) = 80
sizeof(FileSel) = 8536
sizeof(RichTextView) = 640
sizeof(ColumnList) = 520
sizeof(RichEdit) = 22728

Subject: Re: Know what you're using. Size of some common types.
Posted by Novo on Thu, 12 May 2022 06:26:58 GMT
View Forum Message <> Reply to Message

FYI.
I've attached three tab-separated tables having a header below:
Type, Name, TypeIndex, Nested, Size, Padding, Padding/Size

This info is for TheIDE, Release, Windows. It is retrieved from debug info (this can be done in Windows and Unix)

Sorted by Size:
struct Upp::MemoryProfile 11203 0 278616 20 0.0000717834
struct Upp::MemoryProfile 99836 0 278616 20 0.0000717834
struct Ide 10 0 232488 108 0.0004645401
struct Ide 43920 0 232488 108 0.0004645401
class BuildMethods 71641 0 174176 0 0
class BuildMethods 825 0 174176 0 0
struct ZSTD_DCtx_s 118100 0 160312 18 0.0001122811
struct ZSTD_DCtx_s 1331 0 160312 18 0.0001122811
struct Upp::HeaderFooterDlg 5573 0 141680 0 0
class Upp::PatchDiff 102483 0 135664 4 0.0000294846
class Upp::PatchDiff 6438 0 135664 4 0.0000294846
class Upp::PatchDiff 6440 0 135664 4 0.0000294846
class Upp::PatchDiff 65233 0 135664 4 0.0000294846
class Upp::GIFRaster::Data 123850 1 133624 14 0.0001047716
class Upp::GIFRaster::Data 2175 1 133624 14 0.0001047716
class Data 123927 1 133624 14 0.0001047716
class Data 123927 1 133624 14 0.0001047716
class Upp::GIFEncoder::Data 6934 1 131432 0 0
class Data 124015 1 131432 0 0
class Upp::GIFEncoder::Data 124084 1 131432 0 0
struct Upp::sPalMaker 128970 0 131076 0 0
struct Upp::sPalMaker 109 0 131076 0 0
struct CLzmaEnc 1268 0 123272 8 0.0000648971
struct CLzmaEnc 117784 0 123272 8 0.0000648971
struct Pool 87626 1 114888 11 0.0000957454
struct Pool 87626 1 114888 11 0.0000957454
struct Upp::CoWork::Pool 11093 1 114888 11 0.0000957454
class DefaultBuilderSetup 463 0 109368 0 0
class Upp::StyleManager 5023 0 104200 4 0.0000383877
class Upp::StyleManager 119901 0 104200 4 0.0000383877
class TopicEditor 2965 0 85088 7 0.0000822678
class TopicEditor 74943 0 85088 7 0.0000822678
class Upp::DirDiffDlg 102374 0 81664 0 0
class Upp::DirDiffDlg 548 0 81664 0 0
class Upp::DirDiffDlg 45948 0 81664 0 0
class Upp::DirDiffDlg 546 0 81664 0 0
struct Upp::ParaFormatDlg 9045 0 80640 0 0

class Upp::ParaFormatting 1638 0 79848 14 0.0001753331
class Upp::ParaFormatting 119816 0 79848 14 0.0001753331
struct QtfDlgEditor 6998 0 70416 0 0
struct IdeQtfDes 2039 0 70096 0 0
struct IdeQtfDes 103543 0 70096 0 0
class Upp::RichEditWithToolBar 6608 0 70072 7 0.0000998972
class Upp::RichEditWithToolBar 6606 0 70072 7 0.0000998972
struct Upp::RichEditHdrFtr 11338 0 70064 0 0
class Upp::RichEdit 7027 0 69464 41 0.0005902338
class Upp::RichEdit 7025 0 69464 41 0.0005902338
class Upp::RichEdit 7029 0 69464 41 0.0005902338
class Upp::RichEdit 120597 0 69464 41 0.0005902338
class Upp::RichEdit 68731 0 69464 41 0.0005902338
struct Upp::WordDistanceTester 109746 0 67076 0 0
struct Upp::WordDistanceTester 6526 0 67076 0 0
struct DState 7923 0 64144 9 0.0001403093
struct FormatDlg 10024 0 64136 0 0
struct OutMode 2169 0 62928 0 0
struct WithSetupFontLayout<Upp::ParentCtrl> 1817 0 58800 160 0.0027210884
struct EState 7925 0 55768 8 0.0001434514
class Gdb 81257 0 51688 28 0.0005417118
class Gdb 35 0 51688 28 0.0005417118
struct Pdb 73 0 48864 42 0.0008595285
struct Pdb 82340 0 48864 42 0.0008595285
struct AssistEditor 2647 0 48184 29 0.0006018595

Sorted by Padding:
class Upp::CodeEditor 92617 0 19104 3468 0.1815326633
class Upp::CodeEditor 973 0 19104 3468 0.1815326633
class Upp::CodeEditor 975 0 19104 3468 0.1815326633
class Upp::CodeEditor 39471 0 19104 3468 0.1815326633
class Upp::ImagePainter 8169 0 1128 1072 0.9503546099
struct Upp::WithPaletteLayout<Upp::TopWindow> 8787 0 9648 344 0.035655058
struct WithTemplateListLayout<Upp::TopWindow> 1602 0 4576 344 0.0751748252
struct WithSimpleListLayout<Upp::TopWindow> 10165 0 4576 344 0.0751748252
struct Upp::WithMacroManagerLayout<Upp::TopWindow> 3045 0 1704 344 0.2018779343
struct WithUrepoConsoleLayout<Upp::TopWindow> 7591 0 4576 344 0.0751748252
struct WithExportTrLayout<Upp::TopWindow> 4231 0 15768 344 0.0218163369
struct Upp::WithPaletteSelectorLayout<Upp::TopWindow> 10028 0 944 344 0.3644067797
struct WithInsertAsLayout<Upp::TopWindow> 8805 0 1312 344 0.262195122
struct WithCustomLayout<Upp::TopWindow> 9241 0 5392 344 0.0637982196
struct WithMoveTopicLayout<Upp::TopWindow> 3205 0 20552 344 0.0167380304
struct WithSetupWebSearchEngineLayout<Upp::TopWindow> 6467 0 2592 344 0.1327160494
struct Upp::WithImageSizeLayout<Upp::TopWindow> 11622 0 3288 344 0.104622871
struct WithRenamePackageLayout<Upp::TopWindow> 7263 0 1840 344 0.1869565217
struct WithConfLayout<Upp::TopWindow> 3688 0 5792 344 0.0593922652
struct Upp::WithHexGotoLayout<Upp::TopWindow> 6162 0 6008 344 0.0572569907
struct WithSaveTemplateLayout<Upp::TopWindow> 6802 0 15472 344 0.0222337125

struct Upp::WithSimpleSelectLayout<Upp::TopWindow> 7940 0 4576 344 0.0751748252
struct WithCredentialLayout<Upp::TopWindow> 2412 0 8040 344 0.0427860697
struct Upp::WithImageLayout<Upp::TopWindow> 7100 0 5432 344 0.0633284242
struct Upp::WithFileSelectorLayout<Upp::TopWindow> 10310 0 18168 344 0.0189343901
struct WithNewPackageLayout<Upp::TopWindow> 9953 0 11152 344 0.0308464849
struct Upp::WithEditIntLayout<Upp::TopWindow> 4129 0 1560 344 0.2205128205
struct WithLangLayout<Upp::TopWindow> 9111 0 16384 344 0.0209960938
struct WithBaseSetupLayout<Upp::TopWindow> 9523 0 4472 344 0.0769230769
struct Upp::WithCellPropertiesLayout<Upp::TopWindow> 3301 0 16312 344 0.021088769
struct WithGotoMemoryLayout<Upp::TopWindow> 10812 0 6776 344 0.0507674144
struct WithQuickwatchLayout<Upp::TopWindow> 1587 0 9144 344 0.0376202975
struct WithMatrixLayout<Upp::TopWindow> 9943 0 5784 344 0.0594744122
struct WithAddLangLayout<Upp::TopWindow> 2193 0 9936 344 0.0346215781
struct WithTppLayout<Upp::TopWindow> 11550 0 5624 344 0.0611664296
struct WithFindInFilesLayout<Upp::TopWindow> 568 0 33064 344 0.0104040648
struct WithCharsetLayout<Upp::TopWindow> 7572 0 5576 344 0.0616929699
struct Upp::WithRescaleLayout<Upp::TopWindow> 1186 0 7048 344 0.0488081725
struct Upp::WithUnitLayout<Upp::TopWindow> 8941 0 7784 344 0.0441932169
struct WithPasskeyLayout<Upp::TopWindow> 2155 0 2304 344 0.1493055556
struct Upp::WithSplitCellLayout<Upp::TopWindow> 9882 0 3288 344 0.104622871
struct WithMoveCopyPackageLayout<Upp::TopWindow> 5037 0 6984 344 0.049255441
struct WithPrintLayout<Upp::TopWindow> 8193 0 2552 344 0.1347962382
struct WithGetPasskeyLayout<Upp::TopWindow> 7321 0 1552 344 0.2216494845
struct WithAutoSetupLayout<Upp::TopWindow> 4127 0 1768 344 0.1945701357
struct Upp::WithRichFindReplaceLayout<Upp::TopWindow> 8541 0 11912 344 0.0288784419
struct WithNestEditorLayout<Upp::TopWindow> 10780 0 5696 344 0.0603932584
struct WithRunLayout<Upp::TopWindow> 6662 0 15368 344 0.0223841749
struct WithLicenseLayout<Upp::TopWindow> 8869 0 3016 344 0.1140583554
struct WithNewTopicLayout<Upp::TopWindow> 4786 0 15472 344 0.0222337125
struct WithVisGenLayout<Upp::TopWindow> 6277 0 26104 344 0.013178057
struct WithSelectPackageLayout<Upp::TopWindow> 10725 0 11288 344 0.0304748405
struct WithAbbreviationsLayout<Upp::TopWindow> 6285 0 24648 344 0.0139565076
struct Upp::WithObjectSizeLayout<Upp::TopWindow> 206 0 7240 344 0.0475138122
struct Upp::WithFileSelectorLayout<Upp::TopWindow> 10308 0 18168 344 0.0189343901
struct WithSetupGITLayout<Upp::TopWindow> 3425 0 3280 344 0.1048780488
struct Upp::WithIDEFindReplaceLayout<Upp::TopWindow> 10873 0 14192 344 0.0242390079

Sorted by Padding/Size:
class Upp::ImagePainter 8169 0 1128 1072 0.9503546099
struct _DEBUG_EVENT 3331 0 176 160 0.9090909091
struct COFF_IMAGE_SYMBOL 3051 0 26 16 0.6153846154
struct COFF_IMAGE_SYMBOL 139682 0 26 16 0.6153846154
struct jpeg_error_mgr 260 0 168 88 0.5238095238
class Upp::CallbackN<int,bool &> 1397 0 16 8 0.5
class Upp::CallbackN<int,int> 56432 0 16 8 0.5
class Upp::CallbackN<Upp::Stream &> 9252 0 16 8 0.5
class Upp::CallbackN<int,int> 5209 0 16 8 0.5
class Upp::CallbackN<const int &> 119721 0 16 8 0.5

class Upp::CallbackN<const Upp::String &,Upp::Vector<Upp::LineEdit::Highlight> &,const Upp::WString &> 5426 0 16 8 0.5
class Upp::CallbackN<bool,const Upp::String &,Upp::Image &> 75509 0 16 8 0.5
struct ValueType 55349 1 16 8 0.5
class Upp::CallbackN<const Upp::String &,Upp::Vector<Upp::LineEdit::Highlight> &,const Upp::WString &> 66671 0 16 8 0.5
class Upp::GateN<> 10086 0 16 8 0.5
class Upp::GateN<int,int> 2708 0 16 8 0.5
class Upp::GateN<Upp::CodeEditor::MouseTip &> 66796 0 16 8 0.5
class Upp::CallbackN<Upp::Bar &> 1485 0 16 8 0.5
class Upp::CallbackN<Upp::String &> 66756 0 16 8 0.5
class Upp::CallbackN<Upp::String> 64253 0 16 8 0.5
struct ValueType 55349 1 16 8 0.5
class Upp::CallbackN<bool,const Upp::String &,Upp::Image &> 1154 0 16 8 0.5
class Upp::CallbackN<Upp::Time &> 95128 0 16 8 0.5
class Upp::CallbackN<Upp::String &,Upp::WString &> 8703 0 16 8 0.5
class Upp::CallbackN<int,Upp::PasteClip &> 37998 0 16 8 0.5
class Upp::GateN<Upp::CodeEditor::MouseTip &> 2995 0 16 8 0.5
struct Upp::RichTxt::Part 55362 1 16 8 0.5
struct value_type 55350 1 16 8 0.5
class Upp::GateN<> 113352 0 16 8 0.5
class Upp::CallbackN<const Upp::String &,const Upp::String &> 99218 0 16 8 0.5
class Upp::CallbackN<Upp::Point_<int> > 37972 0 16 8 0.5
class Upp::CallbackN<long long> 10110 0 16 8 0.5
class Upp::GateN<int,int> 94204 0 16 8 0.5
class Upp::CallbackN<Upp::Bar &> 24631 0 16 8 0.5
class Upp::CallbackN<int> 5035 0 16 8 0.5
class Upp::CallbackN<long long> 66825 0 16 8 0.5
class Upp::CallbackN<int,Upp::PasteClip &> 4225 0 16 8 0.5
class Upp::CallbackN<int,const Upp::String &> 91852 0 16 8 0.5
class Upp::CallbackN<int,const Upp::String &> 2099 0 16 8 0.5
struct Upp::RichTxt::Part 2071 1 16 8 0.5
class Upp::CallbackN<Upp::One<Upp::Ctrl> &> 26383 0 16 8 0.5
class Upp::CallbackN<Upp::PasteClip &> 8017 0 16 8 0.5
struct value_type 138946 1 16 8 0.5
class Upp::CallbackN<Upp::EscEscape &> 250 0 16 8 0.5
class Upp::CallbackN<Upp::Point_<int> > 1598 0 16 8 0.5
struct Part 51523 1 16 8 0.5
class Upp::CallbackN<const char *> 472 0 16 8 0.5
class Upp::CallbackN<const char *> 58355 0 16 8 0.5
class Upp::CallbackN<Upp::EscEscape &> 66862 0 16 8 0.5
class Upp::CallbackN<int> 52370 0 16 8 0.5
class Upp::GateN<long long,long long> 3299 0 16 8 0.5
class Upp::GateN<long long,long long> 64649 0 16 8 0.5
class Upp::CallbackN<const Upp::String &> 66709 0 16 8 0.5
class Upp::CallbackN<const int &> 5404 0 16 8 0.5
class Upp::CallbackN<Upp::Stream &> 31135 0 16 8 0.5
class Upp::CallbackN<Upp::Time &> 6708 0 16 8 0.5

struct value_type 55350 1 16 8 0.5
struct asn1_type_st 478 0 16 8 0.5
class Upp::CallbackN<Upp::String &> 10703 0 16 8 0.5
class Upp::CallbackN<> 24613 0 16 8 0.5
class Upp::CallbackN<Upp::One<Upp::EditorSyntax> &> 92123 0 16 8 0.5
struct value_type 138946 1 16 8 0.5
class Upp::CallbackN<> 7716 0 16 8 0.5
class Upp::CallbackN<Upp::One<Upp::EditorSyntax> &> 11314 0 16 8 0.5
class Upp::CallbackN<int,bool &> 61655 0 16 8 0.5
class Upp::CallbackN<Upp::PasteClip &> 86672 0 16 8 0.5
class Upp::CallbackN<const Upp::String &,const Upp::String &> 5211 0 16 8 0.5
class Upp::CallbackN<const void *,int> 100988 0 16 8 0.5
struct Part 51523 1 16 8 0.5
class Upp::CallbackN<Upp::String> 3828 0 16 8 0.5
class Upp::CallbackN<const Upp::String &> 11143 0 16 8 0.5
class Upp::CallbackN<Upp::One<Upp::Ctrl> &> 2277 0 16 8 0.5
class Upp::CallbackN<Upp::String &,Upp::WString &> 88967 0 16 8 0.5
class Upp::CallbackN<const void *,int> 4958 0 16 8 0.5
struct AppPreview::Line 2523 1 32 15 0.46875
struct Line 31709 1 32 15 0.46875
struct ValueType 32269 1 32 15 0.46875
struct AppPreview::Line 32282 1 32 15 0.46875
struct ValueType 32269 1 32 15 0.46875
struct value_type 32270 1 32 15 0.46875
struct value_type 32270 1 32 15 0.46875
struct Upp::Iml::IImage 17989 1 16 7 0.4375
struct ValueType 18704 1 16 7 0.4375
struct Upp::sThreadParam 5869 0 16 7 0.4375
struct Upp::Iml::IImage 11268 1 16 7 0.4375
struct ValueType 18589 1 16 7 0.4375
class Upp::WaitCursor 5752 0 16 7 0.4375
struct value_type 18705 1 16 7 0.4375
struct Upp::H_l_ 9555 0 16 7 0.4375
struct value_type 17978 1 16 7 0.4375
class Pdb::CopyMenu::<lambda24> 86333 0 16 7 0.4375
class Pdb::CopyMenu::<lambda25> 86334 0 16 7 0.4375
struct value_type 17978 1 16 7 0.4375
struct Upp::TupleN<2,unsigned char,const char *> 606 0 16 7 0.4375
struct Upp::TupleN<2,unsigned char,const char *> 109882 0 16 7 0.4375
struct IImage 17819 1 16 7 0.4375
struct ValueType 18704 1 16 7 0.4375
class Pdb::CopyMenu::<lambda24> 86117 0 16 7 0.4375
struct ValueType 18589 1 16 7 0.4375
struct IImage 17819 1 16 7 0.4375
class Upp::DirDiffDlg::DirDiffDlg::<lambda0>::operator()::<lambda1> 102429 0 16 7 0.4375
struct value_type 18705 1 16 7 0.4375
class Upp::DirDiffDlg::DirDiffDlg::<lambda0>::operator()::<lambda1> 102522 0 16 7 0.4375
class Pdb::CopyMenu::<lambda25> 86125 0 16 7 0.4375

Subject: Re: Know what you're using. Size of some common types.
Posted by Lance on Sat, 14 May 2022 19:30:09 GMT
View Forum Message <> Reply to Message

Great job! Thank you, Mirek!