Subject: Value: why not float support?
Posted by kohait00 on Mon, 30 Aug 2010 08:54:35 GMT
View Forum Message <> Reply to Message

hey all,

Value implicit support is provided for bool, double, int, in64.
why not for float? int is supported in 2 precisions? so why not saying for float the same..

this would imply some extension of code i know, but what was the reason for leaving it out?

Subject: Re: Value: why not float support?
Posted by mirek on Tue, 31 Aug 2010 13:49:49 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Mon, 30 August 2010 04:54hey all,

Value implicit support is provided for bool, double, int, in64.
why not for float? int is supported in 2 precisions? so why not saying for float the same..

this would imply some extension of code i know, but what was the reason for leaving it out?

Nobody uses float. The only reason for using float is for compacting storage in specific cases. It is like int16.

There is no point for supporting everything in Value. Specially float you can always represent as double.

Subject: Re: Value: why not float support?
Posted by kohait00 on Tue, 31 Aug 2010 14:11:10 GMT
View Forum Message <> Reply to Message

thats right. but thinking of porting upp to embedded world, float is more common there than double (size, less cycles). on x86 world, it has no point. double is calculated just (almost) as fast as double (or am i wrong here?) but think of porting to android

btw: related.
the unsigned things are not needed that much. but how is that one, i.e. parsing from string, (ScanInt) to an unsigned long? could it be done with it?

Subject: Re: Value: why not float support?
Posted by mirek on Tue, 31 Aug 2010 16:49:28 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Tue, 31 August 2010 10:11thats right. but thinking of porting upp to embedded world, float is more common there than double (size, less cycles). on x86 world, it has no point. double is calculated just (almost) as fast as double (or am i wrong here?) but think of porting to android

Are you aware that C/C++ always performs all FP arithmetics in double precision by standard definition?

float really is only about storage.

(OTOH, nothing prevents C/C++ implementation to use 32-bit doubles).

Quote:
btw: related.
the unsigned things are not needed that much. but how is that one, i.e. parsing from string, (ScanInt) to an unsigned long? could it be done with it?

No. But you still have stou / stou64...

Subject: Re: Value: why not float support?
Posted by kohait00 on Tue, 31 Aug 2010 20:02:44 GMT
View Forum Message <> Reply to Message

ok, thats a word thanks for clarification.
(didnt know that, always learning)

Subject: Re: Value: why not float support?
Posted by gprentice on Wed, 01 Sep 2010 11:52:18 GMT
View Forum Message <> Reply to Message

Quote:Are you aware that C/C++ always performs all FP arithmetics in double precision by standard definition?

As far as I can see this is not true. The C++ standard incorporates part of the ISO C standard by reference, including 5.2.4.2.2 where this is specified. I don't have the C90 standard but the C99 standard specifies that FLT_EVAL_METHOD IN float.h (cfloat in C++) has value zero if the calculation precision is the same as the operand precision and value 1 if the calculation precision is double. FLT_EVAL_METHOD wasn't part of C90 but it's very unlikely that C90 required double precision and C99 doesn't.

Also, double is required to have at least 10 significant decimal digits (DBL_DIG in cfloat) and

floats at least 6 digits(FLT_DIG).

Graeme

Subject: Re: Value: why not float support?
Posted by kohait00 on Wed, 01 Sep 2010 14:13:46 GMT
View Forum Message <> Reply to Message

this spots a new light
it would be no huge thing to support float as well, would make a lot of code complient out of the box (when porting to upp or taking over from other projects) without the need to also care for float (which would need a n explicit (double) cast at least to shut up the warnings)
but i also can understand mireks point. Value intrinsic stuff only for the most needed.

Subject: Re: Value: why not float support?
Posted by mirek on Thu, 02 Sep 2010 07:40:55 GMT
View Forum Message <> Reply to Message

gprentice wrote on Wed, 01 September 2010 07:52Quote:Are you aware that C/C++ always performs all FP arithmetics in double precision by standard definition?

As far as I can see this is not true. The C++ standard incorporates part of the ISO C standard by reference, including 5.2.4.2.2 where this is specified. I don't have the C90 standard but the C99 standard specifies that FLT_EVAL_METHOD IN float.h (cfloat in C++) has value zero if the calculation precision is the same as the operand precision and value 1 if the calculation precision is double. FLT_EVAL_METHOD wasn't part of C90 but it's very unlikely that C90 required double precision and C99 doesn't.

Also, double is required to have at least 10 significant decimal digits (DBL_DIG in cfloat) and floats at least 6 digits(FLT_DIG).

Graeme

Correct, I was wrong. Not sure where I got that info...

Subject: Re: Value: why not float support?
Posted by kohait00 on Thu, 02 Sep 2010 09:24:06 GMT
View Forum Message <> Reply to Message

so is there any hope for float:)

Subject: Re: Value: why not float support?
Posted by mirek on Mon, 06 Sep 2010 08:40:29 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Thu, 02 September 2010 05:24so is there any hope for float:)

It still does not have any point. What benefits is it supposed to do? Save 4 bytes per Value? - Would not happen, allocator would align the block to 16bytes multiples anyway.

Maybe would save conversion runtime, but if you are using Value for HPC, you are doing something wrong anyway.

However, keep asking. I guess your question help to clarify things and design intentions.

Subject: Re: Value: why not float support?
Posted by kohait00 on Mon, 06 Sep 2010 09:54:09 GMT
View Forum Message <> Reply to Message

ok, float is burried
dont worry, i'll keep asking things, maybe more than you like...

Subject: Re: Value: why not float support?
Posted by kohait00 on Mon, 13 Dec 2010 14:47:43 GMT
View Forum Message <> Reply to Message

sorry to bother again here..

in my struggle with the OSC protocol, a again came across this issue. OSC supports, together with int, int64, bool, etc.. the float AND double extra.

since i am using Value as the prefered container for all of it, i so far was working with the double as float Value, simply ignoring the fact that float doesnt exist.

i could use double for both, double and float, question, i need to distinguish them, because they need to be sent with different markers. so i have no means to distinguish them..

this is not possible when using double for both double and float:

...
if(v.Is<double>()) { /*send as double, "/osc/message,d 137884828388288.23828" */ }

else if(v.Is<float>()) { /*send as float, "/osc/message,f 123.43" */ }

and, also, the implicit conversions would be not possible.

any idea how to do that?

EDIT:
meanwhile i tried to extend value accordingly. here is a patchfile to revision 2902 to show where changes would occure.

File Attachments
1) Value_float.patch, downloaded 352 times

Subject: Re: Value: why not float support?
Posted by mirek on Wed, 15 Dec 2010 15:15:49 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Mon, 13 December 2010 09:47
any idea how to do that?

Use something else than Value, or add some explicit parameter to distinguish it?

Seriously, my bet is that OSC does not really care whether you are sending double or float. At least, this was ever the case with most protocols...

Mirek

Subject: Re: Value: why not float support?
Posted by kohait00 on Wed, 15 Dec 2010 15:41:31 GMT
View Forum Message <> Reply to Message

it's not the problem whith receiving..
on receive, i could destinguis if it's a float, or a double, and map both to use Value double..

but since the API uses Value directly, which is unbelievably easy. it determines what to send by simply checking the type.

nevertheless, it's important to differ them, since there are old target devices that only understand spec 1.0, which only has float. and newer devices also support spec 1.1 double as extra type, and maybe even expect it for some parameters, while still demanding float for some other..and how to destinguish them?

hope to have cleared it up a bit (and a bit of a bit convinced you

i still could use RichValue<float>, and all the stuff for my own, but it would lack things like Value::IsNumber() and the implicit conversions, which are upp gloabl, but are reaaly appealing..

please, rethink it, float is an own type, just as int64, even if on many systems its breaks down to double in assembler/cpu arch, but compiler still differs them..

Subject: Re: Value: why not float support?
Posted by rylek on Mon, 20 Dec 2010 22:08:52 GMT
View Forum Message <> Reply to Message

Hi there!

I'm afraid this is a difficult nut to crack. At first glance it seems that by adding the support of a new type (like float) to Value we are just enriching the U++ environment and not losing anything. Yet in reality I'm afraid we are losing some of Value's clarity. I, for instance, on numerous occasions make a switch over a Value's GetType(). Now, for instance, when we agreed some time ago to add BOOL_V and INT64_V, I had to manually adjust them to be able to accept a new value subtype. Some code broke in a very tricky way back then.

All right, my switches might not be the cleanest programming technique; at least I should finally ask Mirek to agree to add a series of inline functions to Core/Value.h to check Value types for these convertible groups of datatypes; like

```
inline bool IsNumberType(int value_type)
{ return value_type == BOOL_V || value_type == INT_V
|| value_type == INT64_V || value_type == DOUBLE_V; }
```

- equivalents of .IsNumber() etc. Value member functions, just operating on the type constants; but imagine what the above is going to look like when, after FLOAT_V, we continue to add LONG_DOUBLE_V and [U]INT8/16/32/64_V.

I myself, when working with ActiveX, for instance, sometimes find myself longing for better type distinction in Value in order to be able to provide a better mapping between Value and VARIANT. But then I ask myself: do I want U++ to become such pile of mess as the COM and Value such monster as VARIANT?

And yet, it's evident that on numerous previous occassions we didn't adhere to the position we now hold. We have already extended the numeric type set from the original INT_V / DOUBLE_V pair (not mentioning that even INT_V is in fact superfluous) to INT64_V and BOOL_V (see? we didn't upgrade INT_V to 64-bit, we added a different value type). We have STRING_V and WSTRING_V (here there is some justification because conversion of long binary blocks transferred through Strings, which was always seen as a sound option under U++, is time and memory consuming and potentially even lossy), DATE_V and TIME_V.

This whole discussion would be much easier if the value type was two-dimensional; the above type families (reflected in the member functions IsNumber(), IsString() and IsDateTime()) could then represent a 'principal' value type which would have a 'biggest' or 'most general' representant (double, WString and Time in the above case) and a (perhaps extensible) family of derived subtypes which would be able to convert themselves to and from the type family representant.

But it's not and, as I see it, it would cost all of us many a hair to seamlessly rework it like this. Even so, you still have situations like serializing Values where it's extremely unpleasant to have the type family growing all the time.

Regards

Tomas

Subject: Re: Value: why not float support?
Posted by kohait00 on Tue, 21 Dec 2010 07:20:44 GMT
View Forum Message <> Reply to Message

generally you are right about that, we should aim to keep u++ core slim and not extend it for every purpose. i also agree that, having Value 'user-extensible', like you mentioned it in the 2-dimensional way would do a great deal, so everyone who really needs it and cant avoid converts, can extend it. my propose actually was exactly because IsNumber global functions could not be made float-aware in user way.

OTOH, we should not take the risk to spare out at wrong places. we are talking about intrinsic types, not about bloated class libraries. they are there and *are* still widely used as API, especially for embedded systems. compare it here to a screwdrivers set, where one size is missing. while one generally can spare out some other tools in a collection, there are some general things that *always* need to be there.
missing float *is* like a missing screwdriver.

and one more: design aspects for extension of upp should not only be concerned about not making code possibly bricked (this can be fixed) but also to think about 'what would be the logical, right and most usefull and most performant way'..

so, please, consider float again

Subject: Re: Value: why not float support?
Posted by mirek on Fri, 24 Dec 2010 11:39:59 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Tue, 21 December 2010 02:20
and one more: design aspects for extension of upp should not only be concerned about not making code possibly bricked (this can be fixed) but also to think about 'what would be the logical, right and most usefull and most performant way'..

so, please, consider float again

I am still considering it. I even added value type testers already.

Anyway, to demonstrate a few things, this code would break immediately with float Value type:

```
float x;
SQL.Execute("select FOO from BAR where VALUE > ?", x);
```

which sounds quite dangerous to me. Plus we would have to define a new Null constant etc...

Perhaps it would help me if I understood what you do plan to do with it. Frankly, to depend only on Value type to induce call signature seems like a little bit dangerous practice to me... To easy to make things go very very bad.

Subject: Re: Value: why not float support?
Posted by kohait00 on Fri, 24 Dec 2010 12:55:12 GMT
View Forum Message <> Reply to Message

i admit i have no idea about sql layer in Upp, havent needed it so far. so i dont really understand how this would break it..maybe you can explain it in some 2 phrases..

the new Null value is imho not really a problem.

well, my problem, like described above, is, that i use the cool upp types, which can be packed into Value, as direct feed to the OSC layer, which basicly supports the intrinsic types, but this depends on the spec version, 1.0, or additinoally, 1.1.

1.0 supports int, float, string and blob (binary).
1.1 *additionally* supports int64, double, bool, datetime, etc..

so i decided to simply use the Value as container type, also because there are many implicit conversion possibilities, which spare a lot of work. thats why the extension is crucial, for me at least

i was going well with the 'float as double' handling, as long as keeping spec 1.0 support only, but when, cleanly, extending to 1.1, i need both types to be distinct.

that'd be no problem, i could define and use a custom RichValue<float> and use it just like that, but the conversion mechanisms cant be enriched with user code to support it. IsNumber(), and the implicit converters.

if that'd were possible, no hassle.

but nevertheless, float beeing intrinsic type was finally a motivation to extend it.

i have to admit, char, unsigned types and all that aren't there as well, but they are at least, binary compatible types, means
a char could be banary transported as int, and be interpreted as is later. this holds true for unsigned types as well. but float and double, when transported / serialized over net etc. are *not* binary compatible. though they might be treated equal in cpu, on the net and serialsation, they arent. float is 4 byte, double is 8. this is another pro for float.

if i understood what else might brick with it, i could help thinking of a solution. but i'd rather prefer to make sql layer aware of float than to spare out on float (IMO ofcorse)

afaik, sql also supports both (and some more)
http://dev.mysql.com/doc/refman/5.0/en/numeric-type-overview .html

so let me know, where i can help, if so.

EDIT:
i figured out, that all major database concepts (mysql, postgre sql, oracle, mssql) support float (or single precision).

the only exception is sqlite3, which only supports double, but one could fix this very fast i think, mapping float to double there.

Subject: Re: Value: why not float support?
Posted by mirek on Sat, 25 Dec 2010 09:48:01 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Fri, 24 December 2010 07:55
i have to admit, char, unsigned types and all that aren't there as well, but they are at least, binary compatible types, means
a char could be banary transported as int, and be interpreted as is later. this holds true for unsigned types as well. but float and double, when transported / serialized over net etc. are *not* binary compatible. though they might be treated equal in cpu, on the net and serialsation, they arent. float is 4 byte, double is 8. this is another pro for float.

AFAIK, there is little difference in promoting char to int and casting back and doing the same with float and double.

Quote:
i figured out, that all major database concepts (mysql, postgre sql, oracle, mssql) support float (or single precision).

the only exception is sqlite3, which only supports double, but one could fix this very fast i think, mapping float to double there.

Sure, and every single SQL engine accepts double there (then perhaps casting it to float internally).

Anyway, that is not a problem. The issue is that existing code, which includes SQL/U++ connectors (there is about 5 of them), does not expect to get something like V_FLOAT as Value.

Currently, any float gets promoted to 'double', so it is send as double and everything works. Make Value to remember that such a type is 'V_FLOAT' and you are sudenly missing appropriate case in switch....

Which is quite easy to fix in connectors and in 'uppsrc', actually, but IMO it is little demanding to expect every U++ users to scan through his code for similar cases.

To do that, I would need some solid reason...

And you still have not provided any clue how do you plan to use this (A bit of end-user code would really be helpful). If you need to support 'char' in your interface, I do not see a single reason why you could not support 'float' the very same way.

Mirek

Subject: Re: Value: why not float support?
Posted by kohait00 on Sun, 26 Dec 2010 10:28:43 GMT
View Forum Message <> Reply to Message

well, i'm not trying to invent things to convince you of something. if i cant than maybe the reasons are not strong enough

i'm planning to release a test environment for OSC past xmas, where i have some time to clean up things, order code and make some small docus. but for that to happen i also need to know where the train in terms of value type extension goes, so as to be able to use implicitly float or to need to invent some hack to handle this. because this affects the API. (also, there is some controls issue, since OSC only makes some sense when there are good and easy menas to generate the messages, see TouchOSC for iphone).

concerning double and float in databases: i'm sure the major db's *dont* simply convert it internally. at least not in terms of storage (millions and millions of rows with wasted 4 bytes for double, where float had been enough cant be aforded). thats why one generally is to think of a good, well fitting model/schema for one's problem, not to waste space and performance when handling the data.

as i stated before, i am not fixed on float extension, when there is a possibility to 'enrich' IsNumber() and the implicit converters by a user type. maybe we should think of that, instead.

Subject: Re: Value: why not float support?
Posted by mirek on Mon, 27 Dec 2010 12:20:41 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Sun, 26 December 2010 05:28
concerning double and float in databases: i'm sure the major db's *dont* simply convert it internally. at least not in terms of storage (millions and millions of rows with wasted 4 bytes for double, where float had been enough cant be aforded). thats why one generally is to think of a good, well fitting model/schema for one's problem, not to waste space and performance when handling the data.

No, that was not what I wanted to say. Of course, if you use 4B float in the DB schema, it gets stored as 4B value (unless you are using Oracle, which has very specific way of dealing with numbers anyway).

BUT all these floats can be moved at interface level to DB as doubles.

Which is sort of similar to what I recommend w.r.t. float in C++ code - only use it for storage...

Subject: Re: Value: why not float support?
Posted by kohait00 on Mon, 27 Dec 2010 13:28:25 GMT
View Forum Message <> Reply to Message

i dont mind handling float as double in cpu context, in functions etc. but in my case, sending and receiving is done with distinct types, float and double, unfortunately not interchangeable, it's compareable to storing things .

well, my problem is actually, that i wanted to use Value as a cool implicit convertable container for arbitrary values (which it is). to save me the hassle of converting them manually, since a lot is already present. but lacking float makes it difficult to use in my case ofcorse. i might need to specify own converters which support float as well beeing a RichValue<>.

use case is indeed: an OSC Method handler receives i.e a Value as parameter, which is to be sent as float: so, if it is double, its converted, if it is int, also, if it's a String, it's tried to be parsed. etc.. thus the interface is really versatile and forgiving.

so i can set up different controls, that 'generate' internally different types (editfield a String, Option a bool/int value) but are sent as float etc.. so i dont need to care about types inside the controls already. i simply specify which type the OSC message should finally be sent as.

Subject: Re: Value: why not float support?
Posted by mirek on Mon, 27 Dec 2010 13:44:40 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Mon, 27 December 2010 08:28i dont mind handling float as double in cpu context, in functions etc. but in my case, sending and receiving is done with distinct types, float and double, unfortunately not interchangeable, it's compareable to storing things .

well, my problem is actually, that i wanted to use Value as a cool implicit convertable container for arbitrary values (which it is). to save me the hassle of converting them manually, since a lot is already present. but lacking float makes it difficult to use in my case ofcorse. i might need to specify own converters which support float as well beeing a RichValue<>.

use case is indeed: an OSC Method handler receives i.e a Value as parameter, which is to be sent as float: so, if it is double, its converted, if it is int, also, if it's a String, it's tried to be parsed. etc.. thus the interface is really versatile and forgiving.

so i can set up different controls, that 'generate' internally different types (editfield a String, Option a bool/int value) but are sent as float etc.. so i dont need to care about types inside the controls already. i simply specify which type the OSC message should finally be sent as.

Well, if I understood well what you have just wrote, I see it as argument NOT TO introduce 'float' into Value... You can put float to Value now (as double). And you have to know the true signature of OSC method anyway, so that you can convert all parameters.

So introducing float would be no advantage here.

Mirek

Subject: Re: Value: why not float support?
Posted by kohait00 on Tue, 28 Dec 2010 12:47:47 GMT
View Forum Message <> Reply to Message

naah...that was not what i meant
anyway, permit another question..

why does Value support int? int64 would do it just the same?
why bool? it's actually an int or even int64 for the cpu / compiler.

so breaking this down, the only meaningfull types would be double and int64, as representing the superset of possible types. which i suppose would be the perfect case.. but in terms of handling it'd be a nightmare, a lot of (int)myval, (bool)myval or myval = (double)myfloat and (int64)myint. why not have Value do all this mess?

if the user decides to represent it's data as double, or as float, or int or int64 he also decides on the resolution he needs, well knowing that this might 'implicitly' reduce some resolution, if coming

from double precision i.e..

sorry to bother you with all that. i clearly can understand that you are not well with the thought of extending / breaking a working code just for the benefit of some marginal usecases. i'd do the same. mi point is, that i see a lot more potential in usage of Value than is currently possible. this is genious class that eases handling with types a LOT. but it needs some leverage of burdens.

just to demonstrate: i had to program a typed interface for a gui in our enterprise, and, just because i was 'scared' of Value, knowing it was not able to easy deal with float (because this is what our communication protocol uses the most) left it out to be used. instead, i took a templated approach, inveneted a custom database for objects, etc. etc.. now looking back, having better understood Value and having float support, i'd save me a LOOOT af work and headache.

generally, i'd recommend to enrich Value to support all types which differ in sizeof(), these are the signed variants.. this makes Value really attractive in usage in communication protocols.

bool
char
short
int
int64
float
double

Subject: Re: Value: why not float support?
Posted by kohait00 on Mon, 07 Mar 2011 13:31:04 GMT
View Forum Message <> Reply to Message

currently struggling with python incorporation in upp. and it's just to inform that python too, doesn't support float for the same reasons

i'd love to stick with that but need a good idea on how to treat float in Value maybe simply avoid transformation...concerning sending and distinguishing double and float, i dont need it anyway and the rest (computation) can happen in double.

but the one question remains: why int and int64 both supported, why not mere int64 as superset? it's same as float and double..

Subject: Re: Value: why not float support?
Posted by mirek on Mon, 07 Mar 2011 22:04:14 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Mon, 07 March 2011 08:31
but the one question remains: why int and int64 both supported, why not mere int64 as superset? it's same as float and double..

Actually, it is for historic reasons more than anything else. We have started in 1998, back then int64 was something "special".

When it was proposed to add int64 (like in 2004 , it was a good idea, but INT_V was left for BW compatibility.

Subject: Re: Value: why not float support?
Posted by kohait00 on Tue, 08 Mar 2011 07:33:52 GMT
View Forum Message <> Reply to Message

ok, that is logical
thanks for the patience
