Subject: FIX: UPP::Scan returns int64 for INT_V
Posted by kohait00 on Wed, 13 Apr 2011 21:01:00 GMT
View Forum Message <> Reply to Message

hi

i ran into the issue, that the ConvertInt actually returns an int64
this is not correct i think, since one should be able to rely on the type that is returned in Value with GetData from an EditInt to be int and not int64.

supplied is the fix, i also augmented the wchar versions for scan functions for completeness sake.

replace to current revision 333x

cheers.

File Attachments
1) Convert.h, downloaded 407 times
2) Convert.cpp, downloaded 297 times

Subject: Re: FIX: UPP::Scan returns int64 for INT_V
Posted by Mindtraveller on Fri, 15 Apr 2011 19:44:27 GMT
View Forum Message <> Reply to Message

About two years ago I've met the same issue and asked Mirek why it works this way (you may search this forum). Mirek answered something like "it's not a bug, it's a feature, just don't use ValueTo<>, instaed use (int) v. And everything will be fine".
Since then I use c-style cast for Value and everything is fine.

Subject: Re: FIX: UPP::Scan returns int64 for INT_V
Posted by kohait00 on Sun, 17 Apr 2011 12:57:42 GMT
View Forum Message <> Reply to Message

could you provide the link to the thread? couldnt find it..

i understand, that for best/secure performance ConvertInt should be able to keep the *parsed* data type as complete as possible (int64, if enough info provided from string), even if 'later' in usage the int64 is downgraded to int.

but then, the problem should be solved in EditInt, because, at most, there, one could expect to have a true int value returned. but it is not easy, since EditInt is a EditMinMax<> typedef.

i took a look in the code again. the int64 seems to be there to be able to perform range test after conversion, and return ErrorValue. there is also the solution to the problem. if no error, the range is ok and an int can be generated.

```
Value ConvertInt::Scan(const Value& text) const {
 Value v = UPP::Scan(INT_V, text);
 if(IsError(v)) return v;
 if(IsNull(v)) return notnull ? NotNullError() : v;
 int64 m = v;
 if(m >= minval && m <= maxval) return int(m);//v; <====
 return ErrorValue(UPP::Format(t_("Number must be between %d and %d."), minval, maxval));
}
```

Subject: Re: FIX: UPP::Scan returns int64 for INT_V
Posted by mirek on Sun, 17 Apr 2011 19:23:15 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Sun, 17 April 2011 08:57
but then, the problem should be solved in EditInt, because, at most, there, one could expect to have a true int value returned. but it is not easy, since EditInt is a EditMinMax<> typedef.

One should not. int/int64/double/bool are "poly" types, convertible one to each other. ValueTo is just implementation thing. You are expected to use automatic casts with Value wherever possible, not ValueTo. This is how things are designed, end of story. Coding is easier this way, as you do not have to care about exact type when not necessarry.

BTW, you have also to think about more complex context like Sql, where in fact the number type might not be known until the actual data is fetched from DB.

(Having said that, perhaps we would not really need both EditInt and EditInt64, but that is another story...)

Subject: Re: FIX: UPP::Scan returns int64 for INT_V
Posted by kohait00 on Mon, 18 Apr 2011 08:46:22 GMT
View Forum Message <> Reply to Message

i'm actually not using ValueTo..
it was merely of logical convenience. the stuff i'm pointing to is what i stumble upon, while dealing with environments where the data type issue is still a matter. i'm not planning to change upp to my will, don't get me wrong here.
to have a 'predictable' interface is another design feature.. thats why i said that, dealing with EditInt, i expect to be dealing with int, no matter which container it is actually in, dealing with

EditDouble with double.. this is user POV.

i agree, that EditInt probably is not needed, due to same reason why float is not in Value (remember . it's a 'legacy', but here again, legacy means compatibility of types as well (as of me).

as of the SQL stuff. i think at this point, the data has been already analyzed and can continue to be dealt with as int, isnt it?

if(m >= minval && m <= maxval) return int(m);//v; <====

the other changes in the files are, again, only for logical completeness..

Subject: Re: FIX: UPP::Scan returns int64 for INT_V
Posted by mirek on Tue, 19 Apr 2011 05:28:32 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Mon, 18 April 2011 04:46

as of the SQL stuff. i think at this point, the data has been already analyzed and can continue to be dealt with as int, isnt it?

No. Consider:

```
Sql sql;
sql * Select(NUMBER).From(TABLE);
sql.Fetch();
int x = sql[NUMBER];
```

At the line of assignement to 'x', at compile phase nothing is known about what data type DB returned - was it 'int' or 'double' or 'int64'?

Subject: Re: FIX: UPP::Scan returns int64 for INT_V
Posted by kohait00 on Wed, 27 Apr 2011 12:07:22 GMT
View Forum Message <> Reply to Message

i still can't figure out where ConvertInt comes into play here with Sql..

i understand that operator[]() returns a Value, which content is determined at runtime. is it using a ConvertInt?
(have to admit i haven't used Sql in Upp yet).

if the fix is not possible with ConvertInt, ist there a possib to have sth like

```
class EditInt : public EditMinMax<int, ConvertInt>
{
public:
Value GetData() const { return int(EditValue::GetData()); }
}
```

anyway, the question is, what purpose does the ConvertInt serve to.. to convert stuff to a Value with an int inside. it does not know anything of Sql. so there is the problem..

what about the other completions to Convert?

Subject: Re: FIX: UPP::Scan returns int64 for INT_V
Posted by mirek on Fri, 29 Apr 2011 18:17:34 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Wed, 27 April 2011 08:07i still can't figure out where ConvertInt comes into play here with Sql..

i understand that operator[]() returns a Value, which content is determined at runtime. is it using a ConvertInt?
(have to admit i haven't used Sql in Upp yet).

No. The example was to demontrate that you should never (or almost never) depend on particular number type in Value.

Use IsNumber, then assign.

Quote:
anyway, the question is, what purpose does the ConvertInt serve to.. to convert stuff to a Value with an int inside.

It already has 'int' inside... (stored in lower 32-bits of int64 . Frankly, it is not quite unlikely that at some point in future, we completely abandon INT_V and will only store int64 in Value.

Mirek

Subject: Re: FIX: UPP::Scan returns int64 for INT_V
Posted by kohait00 on Mon, 02 May 2011 08:00:26 GMT
View Forum Message <> Reply to Message

ok thanks for clearing up this stuff.

doing Python exporting currently, i see Value as sort of object and try to align them. and in python too, they have bool, int, long (64) and real (double) (and a few more complex, and ofcorse 'complex' which is not available in Upp at all currently, maybe an idea for the future).
so dont be too eager to remove INT_V too soon, since it's there in most scripting languages.

BTW: Value with its type checks is ideally suited as a data type container, where Is<>() is kind of natural approach to be certain about a type, where type differences do really matter (i.e. communication layers). i'm really lazy to invent a new layer to cope with that so i use what upp gives already..

Subject: Re: FIX: UPP::Scan returns int64 for INT_V
Posted by kohait00 on Tue, 03 May 2011 09:04:39 GMT
View Forum Message <> Reply to Message

got a compromise

fix it in EditMinMax, so all the Editors cope with their associated datatypes in the right way.

EditCtrl.h:254

```
+ Value      GetData() const                  { return DataType(EditField::GetData()); }
```

attached are the Convert Files, with some missing wchar stuff only, which is independant of that fix..

File Attachments
1) Convert.h, downloaded 356 times
2) Convert.cpp, downloaded 328 times

Subject: Re: FIX: UPP::Scan returns int64 for INT_V
Posted by mirek on Fri, 06 May 2011 07:58:31 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Tue, 03 May 2011 05:04got a compromise

fix it in EditMinMax, so all the Editors cope with their associated datatypes in the right way.

EditCtrl.h:254

+ Value GetData() const { return DataType(EditField::GetData()); }

attached are the Convert Files, with some missing wchar stuff only, which is independant of that fix..

I do not know - I see this as introducing bad practice. IMO, you should not depend on this. This change will make it possible to depend on internal type in SOME scenarios - but you would be bitten hard and more unexpectedly in other situation.

Subject: Re: FIX: UPP::Scan returns int64 for INT_V
Posted by kohait00 on Mon, 09 May 2011 16:06:24 GMT
View Forum Message <> Reply to Message

i actually can't think of anther solution. as EditInt is actually kind of 'depricated'...and to be typedef replaced with EditInt64 later anytime..

thinking of my use case (have u tried BoostPyTest already?) i can edit Int in Upp, but they show up as long longs in Python.. because of this. and i dont really come to a clue how to handle it properly.

BTW: python would be great for ide integration as well

Subject: Re: FIX: UPP::Scan returns int64 for INT_V
Posted by mirek on Fri, 13 May 2011 08:52:10 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Mon, 09 May 2011 12:06
thinking of my use case (have u tried BoostPyTest already?) i can edit Int in Upp, but they show up as long longs in Python.. because of this. and i dont really come to a clue how to handle it properly.

And is that really a problem?

http://docs.python.org/reference/expressions.html#arithmetic -conversions

I have had only short encounter with Python (about 100 lines produced for python urr client), but I believe that in Python exact type does not really matter, just as in U++...

What I would do is this: For numeric Value, simply use the minimal possible interpretation. Would that cause any problem in python?

Mirek

Subject: Re: FIX: UPP::Scan returns int64 for INT_V
Posted by kohait00 on Tue, 17 May 2011 09:08:25 GMT
View Forum Message <> Reply to Message

it's actually no problem in python. i do the translation as expected. int Value -> integer, int64 Value -> Long (which is not correct as well, python long: unlimited digits). this works fine. the problem arises for ConvertInt.

as i now understand ConvertInt, it's offering a margin check and returns an ErrorValue if the scanned int64 surpasses the int margin, so it's an error check feature. (the margins are the only difference between ConvertInt and ConvertInt64). ConvertInt expects you not to assume the type (int, int64, bool, actually there is not ConvertBool), it just garantees you get *some* integer value matching the expected margins of an 'int' which implies the usage of a data type as well. so i still dont quite see a problem in returning an int after error check.

since ConvertInt is used quite often (anytime scanning a String), it always generates a long in python, printing it results in '123L'.. i.e if one sets up an EditInt and expects in python to have an integer and not a long. which is not a problem, python deals with it just fine, but it's not clean. though python does the conversions as well as upp, it's still aware of the type differences
http://docs.python.org/reference/datamodel.html#the-standard -type-hierarchy

i'd really vote for doing it as in python
Quote:
Plain integers
These represent numbers in the range -2147483648 through 2147483647. (The range may be larger on machines with a larger natural word size, but not smaller.) When the result of an operation would fall outside this range, the result is normally returned as a long integer

so a ConvertInt would return the smallest representation possible *after* checking margins.

BTW: have you already checked out the BoostPyTest?

Subject: Re: FIX: UPP::Scan returns int64 for INT_V
Posted by mirek on Tue, 17 May 2011 16:51:31 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Tue, 17 May 2011 05:08
so a ConvertInt would return the smallest representation possible *after* checking margins.

Actually, why not.. (patch welcome to speed it up

Anyway, my point was that you can always do the same thing when doing Value->python 'conversion'

Quote:
BTW: have you already checked out the BoostPyTest?

Nope, and I do not plan in any time soon. Do not get me wrong, I am extremely happy you are developing it and I see it as very useful, but there is so much to do that I will not check it unless I need to... which perhaps will happen if somebody posts a patch to ide providing python support for macros or something like that.

(My priorities now is fixing U++ for linux, which I found quite broken, then finally progress with rainbow thing...)

Mirek

Subject: Re: FIX: UPP::Scan returns int64 for INT_V
Posted by kohait00 on Wed, 18 May 2011 09:14:08 GMT
View Forum Message <> Reply to Message

Quote:
Actually, why not.. (patch welcome to speed it up

how should i understand this? will you apply a change yourself or should i point to sth again? in that case this'd be it
Convert.cpp:ConvertInt::Scan

if(m >= minval && m <= maxval) return int(m);//v; <====

havent tested linux stuff for a while now, possibly you're right with it beeing broken..

sorry, didnt want to bother you with the python issue, just was curious what you think of it. and it makes good progress. in that matter, the ownership of Ctrl's is an issue again. this would ease handling of ctrls in python a lot. maybe you could spread some thoughts on it again

BTW: i'm looking forward to rainbow

Subject: Re: FIX: UPP::Scan returns int64 for INT_V
Posted by mirek on Wed, 18 May 2011 20:57:01 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Wed, 18 May 2011 05:14Quote:
Actually, why not.. (patch welcome to speed it up

how should i understand this? will you apply a change yourself or should i point to sth again? in that case this'd be it
Convert.cpp:ConvertInt::Scan

if(m >= minval && m <= maxval) return int(m);//v; <====

minval/maxval are int64....

Well, I was rather thinking about:

```
Value ConvertInt::Scan(const Value& text) const {
 Value v = UPP::Scan(INT_V, text);
 if(IsError(v)) return v;
 if(IsNull(v)) return notnull ? NotNullError() : v;
 int64 m = v;
 if(m >= minval && m <= maxval)
  if(m >= -INT_MIN && m <= INT_MAX)
   return (int)m;
  else
   return v;
 return ErrorValue(UPP::Format(t_("Number must be between %d and %d."), minval, maxval));
}
```

Mirek

Subject: Re: FIX: UPP::Scan returns int64 for INT_V
Posted by kohait00 on Thu, 19 May 2011 07:35:35 GMT
View Forum Message <> Reply to Message

you're right as usual
the int64 return needs to be addressed as well.

in this case i'd vote for separating the ConvertInt::Scan and ConvertInt64::Scan, so they both deal respectively with their proper minval maxval, so we have the INT_MAX/INT_MIN issue only in constructors. i think that's be more clean...

so i'd be

```
Value ConvertInt::Scan() { /***/ return int(v); }
Value ConvertInt64::Scan() { /***/ return v; }
```

i also thought of swapping derive order, to have ConvertInt->ConvertInt64. this would make possible to have a short ConvertInt::Scan like

```
Value ConvertInt::Scan()
{
Value v = ConvertInt64::Scan();
if(!IsErrorValue(v)) return int(m);
return v;
}
```

maybe that's an even cleaner option..

BTW: ConvertInt64 daysichain stuff returns ConvertInt& instead ConvertInt64&
