Subject: Re: ASSERT when using ValueMap
Posted by mirek on Tue, 07 Mar 2017 11:10:20 GMT
View Forum Message <> Reply to Message

NilaT wrote on Tue, 07 March 2017 11:54cbpporterLong story short, Value/ValueMap is not debug-able. Period.
mirek it is simply the fact that Value code is immensely complicated. Really, absolutely totally fucked hard.

This may be the most accurate description about U++ ever made!

Well, I believe not the whole U++. Just this piece of code.

Quote:
but it's not documented.

To be fair, there is shitload of related documentation and examples:

http://www.ultimatepp.org/src$Core$Value$en-us.html

even reading the first couple of senteces would save you a lot of troubles.

http://www.ultimatepp.org/srcdoc$Core$Tutorial$en-us.html

chapter 4 explains everything related to Value.

http://www.ultimatepp.org/reference$JSON$en-us.html

here is how to work with JSON...

I would sort of say that there is perception that documentation is completely missing which was true 5 years ago. Nowadays, there definitely are areas that need improvement, but it is not THAT bad.

Quote:
Ok, anyway... Thanks for fixing it, I don't know what RawToValue exactly does, so I use it everywhere. As I understand it, Value is some kind of container which can hold almost anything. But to do so, you have to "convert" it, and that's what RawToValue does... At least that's my belief.

Well, when type is aware about Value (or Value about type, which is the case of some basic types like 'int'), you do not need any explicit conversions. You are only supposed to use RawToValue for types when implicit conversion does not work.

Quote:

One thing I'm still curious about... What the heck is this assert:
ptr()->GetType() >= 255 || !svo[ptr()->GetType()]
and what does it have in common with

This assert basically checked that you are not using RawToValue for "SVO optimized types". Which I agree is borderline - it might happen this is needed in some template code, so I have changed the rule (I mean, you can now use RawToValue even for types that are alredy supported on higher level).

Quote:
```
inline _Uint4_t _Fetch_add_seq_cst_4(volatile _Uint4_t *_Tgt, _Uint4_t _Value)
 { /* add _Value to *_Tgt atomically with
   sequentially consistent memory order */

 return (_INTRIN_SEQ_CST(_InterlockedExchangeAdd)((volatile long *)_Tgt, _Value));
 }
```

Thanks

Now hard to say about this, but this is probably related to reference counting of one form of internal Value storage. Very likely things got messed up and Value destructor thought it has to release reference count while internal storage was NOT reference counted.