Subject: Intentionally inconstistent use of "Atomic" in Mt.h?
Posted by Werner on Sat, 16 Feb 2008 12:19:06 GMT
View Forum Message <> Reply to Message

All the Atomic... functions in Mt.h (be it WIN32 or POSIX) return int - but shouldn't it be Atomic?

The type of Atomic depends:

MT & POSIX: _Atomic_word -> int
MT & WIN32: LONG -> long
ST & POSIX: int
ST & WIN32: int

Indeed the called WIN32 functions Interlocked... return long. So, to avoid implicit casts it might be preferable to adapt the return type of the Atomic... functions. If I then assign an Atomic to an int, I am seeing what I am doing.

Werner

Subject: Re: Intentionally inconstistent use of "Atomic" in Mt.h?
Posted by mirek on Sat, 16 Feb 2008 16:03:21 GMT
View Forum Message <> Reply to Message

Werner wrote on Sat, 16 February 2008 07:19All the Atomic... functions in Mt.h (be it WIN32 or POSIX) return int - but shouldn't it be Atomic?

The type of Atomic depends:

MT & POSIX: _Atomic_word -> int
MT & WIN32: LONG -> long
ST & POSIX: int
ST & WIN32: int

Indeed the called WIN32 functions Interlocked... return long. So, to avoid implicit casts it might be preferable to adapt the return type of the Atomic... functions. If I then assign an Atomic to an int, I am seeing what I am doing.

Werner

Well, but you should not directly assign Atomic to int, you should rather use AtomicRead...

Mirek

Subject: Re: Intentionally inconstistent use of "Atomic" in Mt.h?
Posted by Werner on Sat, 16 Feb 2008 16:36:24 GMT
View Forum Message <> Reply to Message

luzr wrote on Sat, 16 February 2008 17:03Well, but you should not directly assign Atomic to int, you should rather use AtomicRead...

A misunderstanding? If so, sorry for expressing the issue insufficiently!

I'm talking about the all the "Atomic..." functions in Mt.h, i. e.

AtomicDec,
AtomicInc,
AtomicRead,
AtomicWrite,
AtomicXAdd.

They all have an "Atomic&" as parameter but treat it with an "int" and/or return an "int".

If - and this is the case when using the MSC - "Atomic" is a "long" in disguise, namely typedef'd, implicit casts are inevitable.

I would prefer explicit casts or at least casts which are obvious to me. That is why I suggest to replace the "int" arguments and/or return types with "Atomics".

Werner

P. S.:
"AtomicDec", "AtomicInc", and "AtomicXAdd" are "global" functions. If you discourage their usage, you should document that.

W.

Subject: Re: Intentionally inconstistent use of "Atomic" in Mt.h?
Posted by mirek on Wed, 20 Feb 2008 18:39:40 GMT
View Forum Message <> Reply to Message

Werner wrote on Sat, 16 February 2008 11:36luzr wrote on Sat, 16 February 2008 17:03Well, but you should not directly assign Atomic to int, you should rather use AtomicRead...

A misunderstanding? If so, sorry for expressing the issue insufficiently!

I'm talking about the all the "Atomic..." functions in Mt.h, i. e.

AtomicDec,
AtomicInc,

AtomicRead,
AtomicWrite,
AtomicXAdd.

They all have an "Atomic&" as parameter but treat it with an "int" and/or return an "int".

First of all I think this is miniscule problem... not worthy of such detailed examination.

Anyway, notice this: The set of functions is complete. You never need to assign int directly to Atomic or back.

Imagine some platform that e.g. lacks interlocked operations. In that case you would need to simulate Atomic using mutex:

struct Atomic { Mutex lock; int value; }

There are no direct conversions to/from Atomic now... But using AtomicRead/Write and others, returning int, everything works.

Sure, things are not ideal, perhaps Atomic should rather be class from the beginning and all operations its methods....

Mirek

Subject: Re: Intentionally inconstistent use of "Atomic" in Mt.h?
Posted by Werner on Wed, 20 Feb 2008 20:47:11 GMT

View Forum Message <> Reply to Message

luzr wrote on Wed, 20 February 2008 19:39First of all I think this is miniscule problem... not worthy of such detailed examination.

... but then again I have seen - in other cases and under worse conditions - that minuscule inconstistencies led to major problems ...

But never mind, in this specific case you are most likely right, so thank you very much for still answering anyway.

I mainly posted this question out of curiosity, just to learn why you coded it that way.

Actually I bumped into this issue when analyzing Mt.h, Mt.cpp, CoWork.h, and CoWork.cpp. Unfortunately multithreading with Ultimate++ is poorly documented so far, and I am the guy who wants to know exactly what's going on when using someone else's code.

Werner