
Subject: Re: Proposal: optional int64 size/indexes
Posted by [mirek](#) on Mon, 17 Aug 2015 07:39:09 GMT
[View Forum Message](#) <> [Reply to Message](#)

Well, I spent a lot of time thinking about this in the past...

For now, my position is that yes, FileMapping should have these members fixed (probably using size_t instead of dword there). If there are any similar issues in the U++, please report them as bugs.

Anyway, situation is much less clear w.r.t. containers. For starters, consider this: Containers now support up to 2G of items. Now if you are going to store just pointer in the Vector, sizeof(void *) is 8 bytes, which results in 16GB of flat memory block. My current machines (desktop, notebook) have 8GB of physical memory... And that is just Vector, e.g. Index has inherent memory consumption of about 24 bytes per item. (-> 2G of items is now about 64G of memory).

On the other side, increasing index size from 4 to 8 bytes is not as innocent as it might sound. For example, those 24 bytes per item in Index (and e.g. VectorMap) is now 48 bytes. That is pretty high price to just have a good feeling about "hey, we are really 64-bit now!".

Also, quite minor issue is that while AMD64 ISA is fully 64-bit and int64 is as fast as int32 in most operations that count (64 bit division is slower than 32bit though, which can have impact on some hashing), their opcodes are one byte longer.

IMO, in reality, 64 bit is in fact about being able to access much more memory directly than to have arrays larger than 2G items. Which is what we do now just fine.

All that said, I understand that in some circumstances (e.g. big iron server with 256G memory doing some really serious shit...) it would be helpful to have containers without 2G restriction. My conclusion was that we in fact would need separate set of containers here, "LVector, LArray, LIndex"... etc.