

# Re: Singles to Doubles

---

*Source:*

<http://www.tech-archive.net/Archive/VB/microsoft.public.vb.general.discussion/2007-07/msg01579.html>

---

- *From:* "Ralph" <[nt\\_consulting64@xxxxxxxxxx](mailto:nt_consulting64@xxxxxxxxxx)>
  - *Date:* Wed, 18 Jul 2007 09:35:54 -0500
- 

"dpb" <[none@xxxxxxxx](mailto:none@xxxxxxxx)> wrote in message [news:f715rp\\$iqn\\$1@xxxxxxxxxxxx](mailto:news:f715rp$iqn$1@xxxxxxxxxxxx)

Michael C wrote:

"dpb" <[none@xxxxxxxx](mailto:none@xxxxxxxx)> wrote in message [news:f7k4fe\\$vv8\\$1@xxxxxxxxxxxx](mailto:news:f7k4fe$vv8$1@xxxxxxxxxxxx)

If, after all this you still want more, do a google on "What Every Computer Scientist...Floating Point" and you'll be led quickly to a

very

fine paper on all the details of floating point. I thought I had a bookmark, but can't seem to find it at present and I'm plumb tuckered

out,

sorry... :)

What you say makes sense although I'm still trying to get my head fully around it. A binary number such as 1.11 should be converted to 1.1100 as these 2 values are equal. \*If\* we assume the number came from a base 10 number then we could provide some optimisation to the binary conversion

to

make some sort of guess as to what information was lost converting it to binary. But what we have is a binary number which could have been

obtained

from any source and might not have been a base 10 number at some stage.

## Re: Singles to Doubles

Therefore to assume base 10 could be faulty.

I'm confused by your nomenclature but let me see if I can paraphrase and add and get us closer...

A "binary number 1.11 should be converted to 1.1100 as they are equal".

If you're saying an internal representation of 1.11 (decimal) should be displayed as 1.1100 externally under all circumstances, that can't be unless it is one of the set of reals than can be represented exactly (and I didn't check for sure). In that case it will be preserved both in i/o and in promotion.

If otoh it can't be represented exactly as a Single, then the likelihood that it can in Double isn't zero, but the problem of translation to/from the external representation arises, as does that of promotion. The VB i/o rtl is in a way a culprit in masking these problems because Print (and friends) do incorporate logic in them to try their darnedest to make "ugly" values "pretty" because, just as in Karl's case, that's what is often wanted. Consequently, when the "problem" does appear, as this thread has amply demonstrated, it is more confusing than it might otherwise be in a more rigorous (in some sense) language such as C or Fortran where "what you see is what you get" makes the internal representations a little more visible.

The part about assuming whether the number came from base 10 or was possibly from somewhere else I don't understand at all. In the context of VB, all external representations are base 10 unless explicitly coded as Hex\$ or Oct\$ in which case the base is also known and explicit. There is no assumption of base ever required, the only "assumptions" in what I was trying to describe had to do with what one is to assume trailing digits are for added precision when there isn't any information providing them.

I'm repeating myself, but that's the situation w/ promotion or translation. In both cases the course of action that is followed is to extend w/ zeros, but that leads to two different results, depending on whether one is extending zeros in the external or internal representation.

To see what is going on internally, find Tony's post and paste his code in and look at the actual bit patterns and it will probably become much more clear. Or, google on "floating point conversion" or similar and amongst the zillion of hits pretty early on there will be sites that have floating point calculators that you can type numbers into and see their internal representations.

Again, I heartily recommend at least reading through Goldberg's paper even if you don't study it in depth...

[http://docs.sun.com/source/806-3568/ncg\\_goldberg.html](http://docs.sun.com/source/806-3568/ncg_goldberg.html)

## Re: Singles to Doubles

--

This thread highlights the importance, for any program of consequence, of defining business rules for scope and precision whenever one deals with floating point math and storage. For example, had the app defined stored values as significant to 5 places, Karl would have home free a long time ago.

In the absence of any definitions the existing routine that calculates a value becomes the de facto "Business Rule". Any modification of that routine is thus a change to the BRs – and as such requires testing and sign-offs beyond a simple programming decision or technical fix-up.

Programmers intuitively grasp the idea that converting a double to a single will likely lead to a loss of information. The fact that performing a conversion in the opposite direction will also lose information is less obvious but just as real.

–ralph

.