Why does BitConverter.LittleEndian return false on my x86 machine?

devblogs.microsoft.com/oldnewthing/20130805-00

August 5, 2013

Raymond Chen

Welcome to CLR Week 2013, returned from its two-year hiatus.

A customer reported that when they checked with the debugger, `BitConverter.LittleEndian` reported `false` even though they were running on an x86 machine, which is a little-endian architecture.

```
ushort foo = 65280;
65280
BitConverter.IsLittleEndian
false
BitConverter.GetBytes(foo)
{byte[2]}
[0]: 0
[1]: 255
```

The bytes are extracted in little-endian order, despite the claim that the machine is big-endian. “I don’t get it.”

I didn’t know the answer, but I knew how to use a search engine, and a simple search quickly found this explanation:

Reading a member from the debugger merely reads the value of the member from memory.

That simple statement hides the answer by saying what happens and leaving you to figure out what doesn’t happen. Here’s what doesn’t happen: Reading a member from the debugger *does not execute the code to initialize that member*.

In the case of `BitConverter`, the `LittleEndian` member is initialized by the static constructor. But when are static constructors run? For C#, static constructors are run before the first instance is created or any static members are referenced. Therefore, if you never create any `BitConverter` objects (which you can’t since it is a static-only class), and if you

never access any static members, then its static constructor is not guaranteed to have run, and consequently anything that is initialized by the static constructor is not guaranteed to have been initialized.

And then when you go looking at in the debugger, you see the uninitialized value.

Why doesn't the debugger execute static constructors before dumping a value from memory? Probably because the debugger wants to avoid surprises. It would be weird if you tried to dump a value from the debugger and the program resumed execution!

Now, when you ask the debugger to evaluate `BitConverter.GetBytes(foo)`, the debugger has no choice but to execute application code, but that's okay because you explicitly told it to. But let's continue that debugging session:

```
ushort foo = 65280;
65280
BitConverter.IsLittleEndian
false
BitConverter.GetBytes(foo)
{byte[2]}
[0]: 0
[1]: 255
BitConverter.IsLittleEndian
true ← hey look
```

Your call to `BitConverter.GetBytes(foo)` caused code to execute, and then the CLR said, "Okay, but before I call this member function, I am required to run the static constructor, because those are the rules," and that resulted in the `Is&shy:LittleEndian` field being initialized to `true`.

The customer replied, "Thanks. The trick was finding the correct search terms."

I didn't think my choice of search terms was particularly devious. I simply searched for bit-converter.islittleendian false.

Bonus reading: The byte order fallacy by Rob Pike. "Whenever I see code that asks what the native byte order is, it's almost certain the code is either wrong or misguided."

Raymond Chen

Follow