**# Title:**

Clarify NVMe and Infiniband device path EUI-64 byte order

**# Status:**

Draft

**# Document:**

UEFI Specification 2.9 (Future Errata)

**# License:**

SPDX-License-Identifier: CC-BY-4.0

**# Submitter:**

* Samer El-Haj-Mahmoud, ARM
* TianoCore Community (<https://www.tianocore.org>)

**# Summary of the change**

Clarify NVMe and Infiniband device path EUI-64 byte order. See details below

**# Benefits of the change**

The UEFI Spec is ambiguous when it comes to the byte-order of EUI-64 fields for NVMe (and Infiniband) device path nodes

See: <https://bugzilla.tianocore.org/show_bug.cgi?id=3291>

and <https://bugzilla.tianocore.org/show_bug.cgi?id=3292>

The UEFI specification for the NVMe() node refers to NVM Express Specification.

**Section 10.3.4.22***“Refer to the latest NVM Express specification for descriptions of Namespace Identifier (NSID) and IEEE Extended Unique Identifier (EUI-64):See “Links to UEFI-Related Documents” (http://uefi.org/uefi under the headings “NVM Express Specification”.”*

**Table 10-61 says:**

*“  
NVMe(NSID,EUI)  
…*

*The EUI is the IEEE Extended Unique Identifier (EUI-64) that is displayed in hexadecimal format represented as a set of octets separated by dashes (hexadecimal notation), e.g., FF-FF-FF-FF-FF-FF-FF-FF.”*

In the "NVM Express Base Specification", NVM Express Revision 1.4b ,September 21, 2020, chapter 7.10.4 IEEE Extended Unique Identifier (EUI64) we find:

"EUI64 is defined in big endian format."

But TianoCore implemented NVMe device path as Little endian:

\* DevPathToTextNVMe()

\* DevPathFromTextNVMe() - both in BaseTools and MdePkg

\* TranslatePciOfwNodes()

Similar issue exists for Infiniband:

**10.3.4.15 InfiniBand Device Path**

*"IOC GUID/Service ID : 64-bit unique identifier to remote IOC or server process. Interpretation of field specified by Resource Flags (bit 0) Target Port ID : 64-bit persistent ID of remote IOC port Device ID: 64-bit persistent ID of remote device"*

**Table 10-61:**

*"ServiceId, TargetId and DeviceId are 64-bit unsigned integers."*

**15.5.4 InfiniBand Device Path Example**

*" The SCSI Host Adapter is a PCI device that is located at PCI device number 0x07 and PCI function 0x00, and is directly attached to a PCI root bridge. The SCSI device is addressed by the (IOU X, IOC Y, DeviceId Z) in the InfiniBand Network. (X, Y, Z are EUI-64 compliant identifiers)."*

Just like NVMe, in this case, it is not clear if the UEFI spec intended for InfiniBand to big or little endian. But the Infiniband spec (<https://cw.infinibandta.org/document/dl/7161> ) uses big-endian for all of these EUI-64 IDs. The EDK2 implementation DevPathFromTextInfiniband () seems to also be using little endian.

Looking more into the spec, I see that there was conscious effort to use little endian in other device path types that contain 64-bit values, even though the original spec defined it as big-endian. See for example:

**10.3.4.3 Fibre Channel Device Path**

*"When an application client displays or otherwise makes a 64-bit LUN visible to a user, it should be done in conformance with SAM-4. SAM-4 requires a LUN to be displayed in hexadecimal format with byte 0 first (i.e., on the left) and byte 7 last (i.e., on the right) regardless of the internal representation of the LUN. UEFI defines all data structures a “little endian” and SCSI defines all data structures as “big endian”."*

**10.3.4.20 Serial Attached SCSI (SAS) Ex Device Path**

*"This section defines the extended device node for Serial Attached SCSI (SAS) devices. In this device path the SAS Address and LUN are now defined as arrays to remove the need to endian swap the values."*

*...*

*"When an application client displays or otherwise makes a 64-bit LUN (8 byte array) visible to a user, it should be done in conformance with SAM-4. SAM-4 requires a LUN to be displayed in hexadecimal format with byte 0 first (i.e., on the left) and byte 7 last (i.e., on the right) regardless of the internal representation of the LUN. UEFI defines all data structures a “little endian” and SCSI defines all data structures as “big endian”."*

Given that EDK2 implementation of NVMe and Infiniband device paths is little endian, and the explicit sentences on using little endian for SAS and Fibre channel, I am inclined to say the UEFI spec intended for NVMe and Infiniband EUI-64 to be little endian.

If this is the case, then we need a clarification in the UEFI spec similar to the statements added for SAS and Fibre Channel.

**# Impact of the change**

Change is compatible with existing TianoCore EDK2 implementations. No EDK2 FW or OS code changes are expected.

UBoot UEFI implementation may need to change (not sure) to ensure compatibility.

**# Detailed description of the change [normative updates]**

* Insertions in **green**
* Removals in ~~red~~

10.3.4.22 NVM Express namespace messaging device path node

…  
Refer to the latest NVM Express specification for descriptions of Namespace Identifier (NSID) and IEEE Extended Unique Identifier (EUI-64):See “Links to UEFI-Related Documents” (http://uefi.org/uefi under the headings “NVM Express Specification”.

*"When an application client displays or otherwise makes the EUI-64 identifiers visible to a user, the values should be displayed in hexadecimal format with byte 0 first (i.e., on the left) and byte 7 last (i.e., on the right) regardless of the internal representation of the EUI-64. UEFI defines all data structures as “little endian” and IEEE defines EUI-64 as “big endian”."*

15.5.4 InfiniBand Device Path Example

Table 15-4 shows an example device path for a SCSI device in an InfiniBand Network. This SCSI device is connected to a single SCSI channel generated by a SCS Host Adapter, and the SCSI Host Adapter is an end node in the InfiniBand Network. The SCSI Host Adapter is a PCI device that is located at PCI device number 0x07 and PCI function 0x00, and is directly attached to a PCI root bridge. The SCSI device is addressed by the (IOU X, IOC Y, DeviceId Z) in the InfiniBand Network. (X, Y, Z are EUI-64 compliant identifiers).

*"When an application client displays or otherwise makes the EUI-64 identifiers visible to a user, the values should be displayed in hexadecimal format with byte 0 first (i.e., on the left) and byte 7 last (i.e., on the right) regardless of the internal representation of the EUI-64. UEFI defines all data structures as “little endian” and IEEE defines EUI-64 as “big endian”."*