NVM Express®

Boot Specification

Revision 1.0a

July 29th, 2024

Please send comments to info@nvmexpress.org

NVM Express® Boot Specification, Revision 1.0a is available for download at https://nvmexpress.org. The NVM Express Boot Specification, Revision 1.0a incorporates TP8012, ECN120, and ECN122.

SPECIFICATION DISCLAIMER

LEGAL NOTICE:

NVM Express Workgroup
c/o VTM, Inc.
3855 SW 153rd Drive
Beaverton, OR 97003 USA
info@nvmexpress.org

Table of Contents

Table of Figures

This page is intentionally left blank.

1 Introduction

1.1 Overview

The NVM Express® (NVMe®) Base Specification defines an interface for host software to communicate with a non-volatile memory subsystem (NVM subsystem) over a variety of memory-based transports and message-based transports.

This document defines constructs and guidelines for booting from NVM Express interfaces.

The current revision defines extensions to the NVM Express specification family that enable booting over NVMe over supported fabric transports. This specification includes:

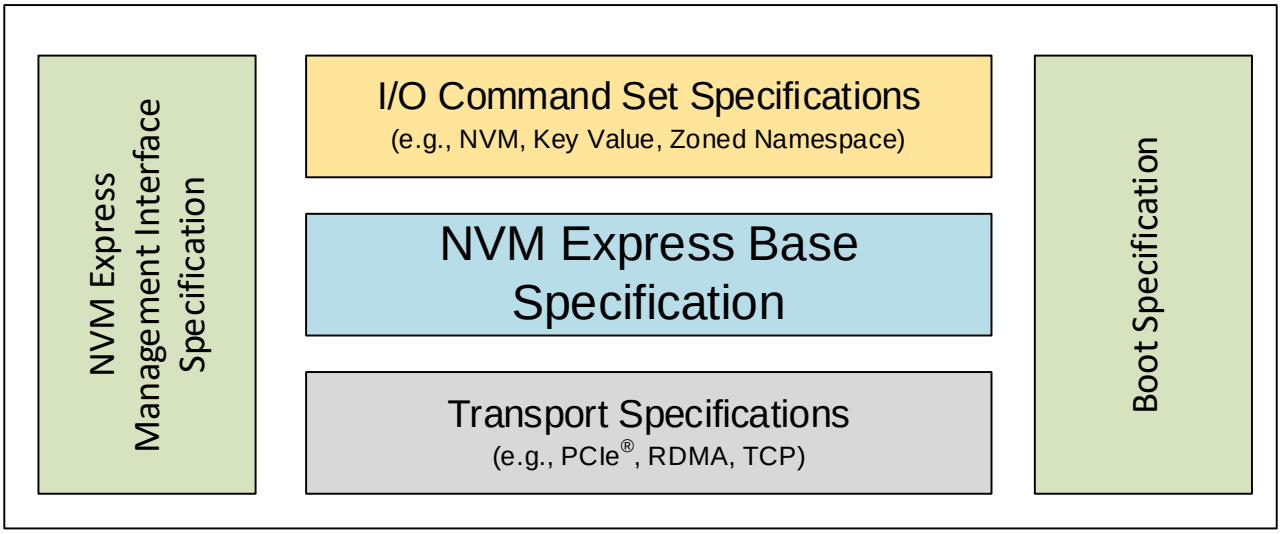
- material specific to the NVMe-oF TCP transport; and
- operating system best practices and their consumption of pre-boot configuration (e.g., UEFI).

Other published specifications referred to in this document, even if required for compliance, are outside the scope of this specification; this includes published specifications for fabrics and other technologies referred to by this document or any NVM Express Transport binding specification.

1.2 Scope

Figure 1 shows the relationship of the NVM Express® Boot Specifications to other specifications within the NVMe® family of specifications.

Figure 1: NVMe Family of Specifications



The NVM Express specification family structure shown in Figure 1 is intended to show the applicability of NVM Express specifications to each other, not a hierarchy, protocol stack, or system architecture.

The NVM Express Base Specification defines a protocol for host software to communicate with non-volatile memory subsystems over a variety of memory-based transports and message-based transports.

The NVM Express Management Interface Specification defines an optional management interface for all NVM Express Subsystems.

NVM Express I/O Command Set specifications define data structures, features, log pages, commands, and status values that extend the NVM Express Base specification.

NVM Express Transport specifications define the binding of the NVMe protocol including controller properties to a specific transport.

The NVM Express Boot Specification (i.e., this specification) defines constructs and guidelines for booting from NVM Express interfaces.

1.3 Definitions

Key Terms and acronyms used throughout this document. Definitions pertinent to domain specific terms are additionally in Annex A.1.

1.3.1 Boot Device

The storage volume which contains the relevant instructions and necessary next step content that a Platform may load an operating system or preform other basic functions beyond that of a pre-OS environment. In UEFI, this is commonly a device with an EFI System Partition.

1.3.2 Discovery Controller (DC)

A controller that exposes capabilities that allow a host to retrieve a Discovery log page. A Discovery controller does not implement I/O Queues or provide access to a non-volatile memory storage medium. Refer to the NVM Express Base Specification.

1.3.3 Diffie-Hellman (DH)

A method of securely exchanging cryptographic keys (refer to RFC 2631).

1.3.4 Discovery Log Page (DLP)

A log page is provided by a Discovery controller contains one or more entries. Each entry indicates information necessary for the host to connect to an NVM subsystem. Refer to the NVM Express Base Specification.

1.3.5 EFI System Partition (ESP)

An EFI defined partition in a UEFI compliant system (refer to the Unified Extensible Firmware Interface Specification) in which all OS loaders and applications will be stored and read by pre-OS Firmware for the purposes of loading or booting an UEFI compatible operating system. On Linux systems this might be represented by a directory structure under /boot/efi for example. For UEFI complaint Platforms, the presence of an ESP makes a namespace bootable.

1.3.6 Heap Space

Refer to section 3.1.1.1.

1.3.7 Host Fabric Interface (HFI)

A Host Fabric Interface is a structure that contains information about various supported fabric interfaces and their transport specific characteristics. These are logical or physical interfaces to local or remote access as defined by the HFI.

1.3.8 iSCSI Boot Firmware Table (iBFT)

A block of information that contains parameters that are useful to the iSCSI Boot process defined in the iSCSI Boot Firmware Table (iBFT) Specification. The iBFT is the mechanism by which iSCSI Boot parameter values are conveyed to the operating system.

1.3.9 Namespace (NS)

The representation of a data device that NVMe-oF hosts attach to in order to locate Bootable and Non-bootable UEFI Block or File Entries.

1.3.10 Namespace ID (NSID)

The namespace ID is used by a controller to provide access to a namespace as defined in the NVM Express Base Specification.

1.3.11 Namespace Identifier (NID)

The Namespace Identifier is a globally unique value defined in the Namespace Identification Descriptor list (i.e., CNS 03h) in the NVM Express Base Specification.

1.3.12 NVM Subsystem

An NVM subsystem includes one or more domains, one or more controllers, zero or more namespaces, and one or more NVM subsystem ports. An NVM subsystem may include a non-volatile memory storage medium and an interface between the controller(s) in the NVM subsystem and non-volatile memory storage medium. Refer to the NVM Express Base Specification.

1.3.13 NVM Subsystem Port

An NVMe over Fabrics protocol interface between an NVM subsystem and a fabric. An NVM subsystem port is a collection of one or more physical fabric interfaces that together act as a single interface. Refer to the NVM Express Base Specification.

1.3.14 NVMe Boot Firmware Table (NBFT)

The NVMe Boot Firmware Table is the data structure defined by this specification as a means of passing context from a pre-OS Boot environment to an OS runtime.

1.3.15 NVMe over Fabrics (NVMe-oF)

An implementation of the NVM Express interface that complies with either the message-only or the message/memory-based implementation of the memory-based transport model definition. Refer to the NVM Express Base Specification.

1.3.16 NVMe Qualified Name (NQN)

Used to uniquely describe a host or NVM subsystem for the purposes of identification and authentication. Refer to the NVM Express Base Specification.

1.3.17 Platform

A computer or hardware device and/or associated operating system, or a virtual environment, on which software is installed or run (refer to ISO19770-2:4.1.17). Examples of platforms include a computer system running Linux®, Microsoft Windows 10®, or Java™.

An NVMe-oF Platform should be a Platform that is capable of instantiating a NVMe-oF association as an NVMe host. Refer to NISTIR 7695 from ISO/IEC 19770-2.

1.3.18 Pre-OS Driver

The NVMe-oF environment that is responsible for initializing any host specific NVMe configuration and populating that configuration data into the NBFT.

1.3.19 Storage Subsystem

An NVM subsystem that may include a non-volatile memory storage medium and exposes namespaces, but is not a Discovery Service.

1.3.20 Subsystem and Namespace Descriptor (SSNS)

An NBFT structure that contains information about how to locate a specific namespace by its globally unique CNS 03h NID value.

1.3.21 Unified Extensible Firmware Interface (UEFI)

Firmware interface between the platform and the operating system. Defined by the UEFI Forum (refer to uefi.org).

1.3.22 UEFI Human Interface Infrastructure (UEFI HII)

UEFI core functionality describes the data and APIs used to manage the system's configuration and their human interfaces including basic mechanisms for managing both user input and UEFI protocols. Refer to the Unified Extensible Firmware Interface (UEFI) Specification.

1.3.23 Extended System Description Table (XSDT)

The XSDT is the first table used by an operating systems ACPI subsystem and contains addresses of most other ACPI tables on the system. If a system has an NBFT, its address will be in the system's XSDT. Refer to the Advanced Configuration and Power Interface Specification.

1.4 References

NVM Express® Base Specification, Revision 2.0c. Available from https://www.nvmexpress.org.

NVM Express® Command Set Specification, Revision 1.0b. Available from https://www.nvmexpress.org.

NVM Express® TCP Transport Specification, Revision 1.0b. Available from https://www.nvmexpress.org.

Win 32 API: NL_PREFIX_ORIGIN enumeration, Accessed April 19, 2022. Available from https://docs.microsoft.com/en-us/windows/win32/api/nldef/ne-nldef-nl_prefix_origin.

RFC 7143, M. Chadalapaka, J. Satran, K. Meth, D. Black, "Internet Small Computer System Interface (iSCSI) Protocol (Consolidated)", April 2014. Available from https://datatracker.ietf.org/doc/html/rfc7143.

RFC 3720, J. Satran, K. Meth, C. Sapuntzakis, M. Chadalapaka, E. Zeidner, "Internet Small Computer Systems Interface (iSCSI)", April 2004. Available from https://datatracker.ietf.org/doc/html/rfc3720.

RFC 4173, P. Sarkar, D. Missimer, C. Sapuntzakis, "Bootstrapping Clients using the Internet Small Computer System Interface (iSCSI) Protocol", April 2005. Available from https://datatracker.ietf.org/doc/html/rfc4173.

iSCSI Boot Firmware Table (iBFT) as Defined in ACPI 3.0b Specification, Version 1.03. Available from https://download.microsoft.com/download/1/e/5/1e5a2287-366c-431a-8e05-b958540230b1/iBFT.docx.

IEEE Guidelines for Use of Extended Unique Identifiers, August 2017. Available from https://standards.ieee.org/wp-content/uploads/import/documents/tutorials/eui.pdf.

IEEE 802.1q-2018: IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks, August 2018. Available from https://standards.ieee.org/ieee/802.1Q/6844/.

IETF RFC 2132, S. Alexander, R. Droms,"DHCP Options and BOOTP Vendor Extensions", March 1997. Available from https://rfc-editor.org/rfc/rfc2132.txt.

RFC 2631, E. Rescorla, "Diffie-Hellman Key Agreement Method", June 1999. Available from https://www.ietf.org/rfc.html.

R. Hinden, B. Carpenter, L. Masinter, "Format for Literal IPv6 Addresses in URL's", December 1999. Available from https://rfc-editor.org/rfc/rfc2732.txt.

RFC 3986, T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", January 2005. Available from https://www.ietf.org/rfc.html.

IETF RFC 4038, M-K. Shin, Ed., Y-G. Hong, J. Hagino, P. Savola, E. M. Castro, "Application Aspects of IPv6 Transition", March 2005. Available from https://rfc-editor.org/rfc/rfc4038.txt.

IETF RFC 4249, B. Haberman, "IP Forwarding Table MIB", April 2006. Available from https://www.ietf.org/rfc/rfc4292.txt.

IETF RFC 4291, R. Hinden, S. Deering, "IP Version 6 Addressing Architecture", February 2006. Available from https://rfc-editor.org/rfc/rfc4291.txt.

DMTF DSP0270, "Redfish Host Interface Specification", Revision 1.3. Available from https://www.dmtf.org/sites/default/files/standards/documents/DSP0270_1.3.0.pdf.

DMTF DSP8010 "Redfish 2021.4 Schema Bundle", Version 2021.4, December 2021. Available from https://www.dmtf.org/sites/default/files/standards/documents/DSP8010_2021.4.zip

Tianocore, "EDK2 Build Specification", Version 1.0.0. Available from https://edk2-docs.gitbook.io/edk-ii-build-specification/.

ACPI "Advanced Configuration and Power Interface (ACPI) Specification", Version 6.4. Available from https://uefi.org/specs/ACPI/6.4/.

UEFI, "Unified Extensible Firmware Interface (UEFI) Specification", Version 2.9. Available from https://uefi.org/specifications.

IANA TLS Parameters Registry, TLS Cipher Suites, accessed April 26th, 2022. Available from https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml#tls-parameters-4.

IANA TLS Parameters Registry, TLS Supported Groups, accessed April 26th 2022. Available from https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml#tls-parameters-8.

NISTIR 7695 from ISO/IEC 19770-2. Available from https://nvlpubs.nist.gov/nistpubs/Legacy/IR/nistir7695.pdf

PCI Express Base Specification, Revision 6.0. Available from https://members.pcisig.com/wg/PCI-SIG/document/16609

PCI Firmware Specification, Revision 3.3. Available from https://members.pcisig.com/wg/PCI-SIG/document/15350

1.5 Conventions

1.5.1 Keywords

Several keywords are used to differentiate between different levels of requirements.

1.5.1.1 mandatory

A keyword indicating items to be implemented as defined by this specification.

1.5.1.2 may

A keyword that indicates flexibility of choice with no implied preference.

1.5.1.3 obsolete

A keyword indicating functionality that was defined in a previous version of the NVM Express specification and that has been removed from this specification.

1.5.1.4 optional

A keyword that describes features that are not required by this specification. However, if any optional feature defined by the specification is implemented, the feature shall be implemented in the way defined by the specification.

1.5.1.5 R

“R” is used as an abbreviation for “reserved” when the figure or table does not provide sufficient space for the full word “reserved”.

1.5.1.6 reserved

A keyword referring to bits, bytes, words, fields, and opcode values that are set-aside for future standardization. Their use and interpretation may be specified by future extensions to this or other specifications. A reserved bit, byte, word, field, property, or register shall be cleared to 0h, or in accordance with a future extension to this specification. The recipient of a command or a register write is not required to check reserved bits, bytes, words, or fields. Receipt of reserved coded values in defined fields in commands shall be reported as an error. Writing a reserved coded value into a controller property field produces undefined results.

1.5.1.7 shall

A keyword indicating a mandatory requirement. Designers are required to implement all such mandatory requirements to ensure interoperability with other products that conform to the specification.

1.5.1.8 should

A keyword indicating flexibility of choice with a strongly preferred alternative. Equivalent to the phrase “it is recommended”.

1.5.2 Numerical Descriptions

A 0’s based value is a numbering scheme in which the number 0h represents a value of 1h, 1h represents 2h, 2h represents 3h, etc. In this numbering scheme, there is no method to represent the value of 0h. Values in this specification are 1-based (i.e., the number 1h represents a value of 1h, 2h represents 2h, etc.) unless otherwise specified.

Size values are shown in binary units or decimal units. The symbols used to represent these values are as shown in Figure 2.

Figure 2: Decimal and Binary Units

Decimal		Binary	
Symbol	**Power (base-10)**	**Symbol**	**Power (base-2)**
kilo / k	10^3	kibi / Ki	2^{10}
mega / M	10^6	mebi / Mi	2^{20}
giga / G	10^9	gibi / Gi	2^{30}
tera / T	10^{12}	tebi / Ti	2^{40}
peta / P	10^{15}	pebi / Pi	2^{50}
exa / E	10^{18}	exbi / Ei	2^{60}
zetta / Z	10^{21}	zebi / Zi	2^{70}
yotta / Y	10^{24}	yobi / Yi	2^{80}

The ^ operator is used to denote the power to which that number, symbol, or expression is to be raised.

Some parameters are defined as an ASCII string. ASCII strings shall contain only code values 20h through 7Eh. For the string “Copyright”, the character “C” is the first byte, the character “o” is the second byte, etc. The string is left justified and shall be padded with spaces (ASCII character 20h) to the right if necessary. A hexadecimal ASCII string is an ASCII string that uses a subset of the code values: “0” to “9”, “A” to “F” uppercase, and “a” to “f” lowercase.

Hexadecimal (i.e., base 16) numbers are written with a lower case “h” suffix (e.g., 0FFFh, 80h). Hexadecimal numbers larger than eight digits are represented with an underscore character dividing each group of eight digits (e.g., 1E_DEADBEEFh).

Binary (i.e., base 2) numbers are written with a lower case “b” suffix (e.g., 1001b, 10b). Binary numbers larger than four digits are written with an underscore character dividing each group of four digits (e.g., 1000_0101_0010b).

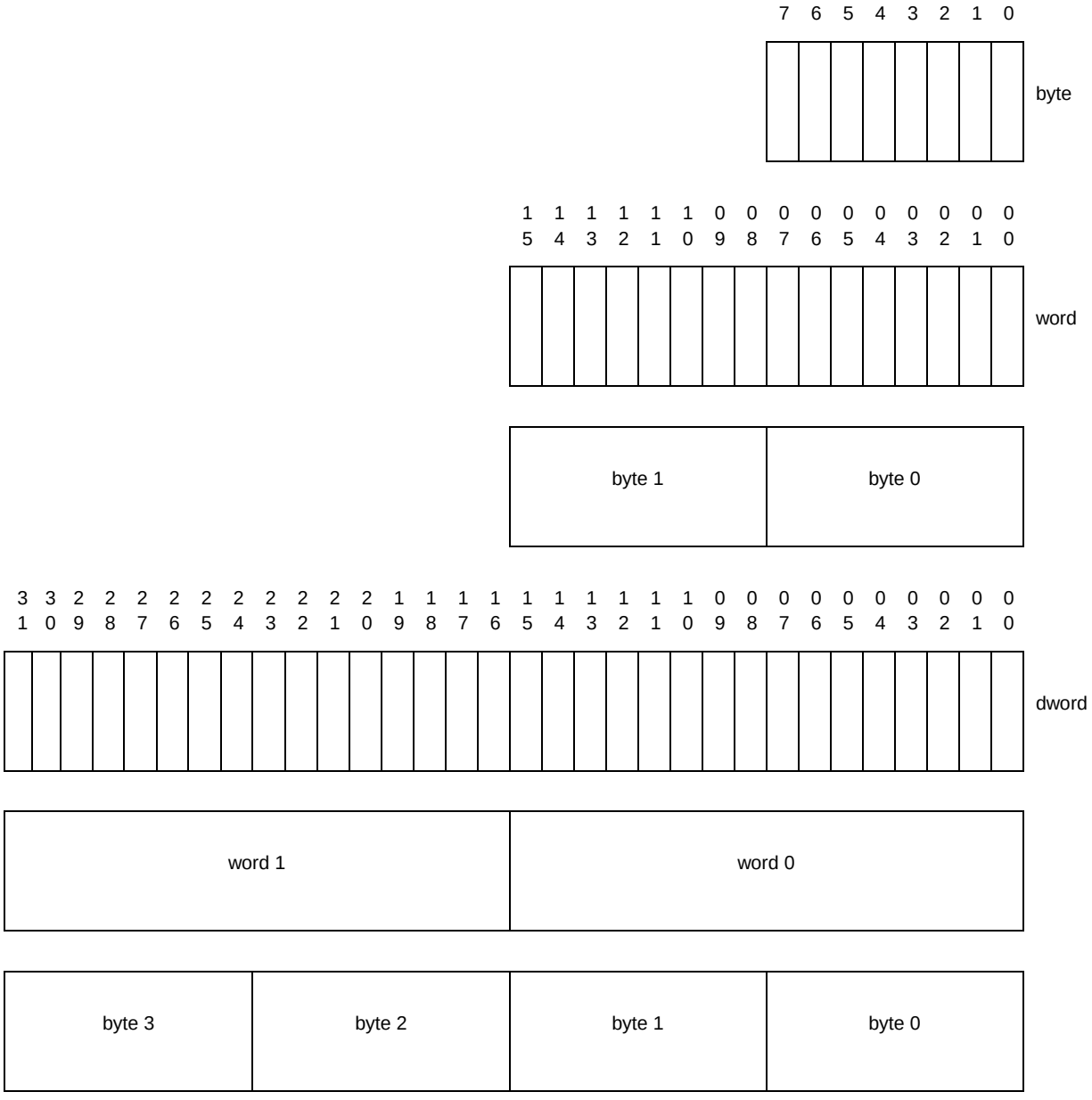
All other numbers are decimal (i.e., base 10). A decimal number is represented in this specification by any sequence of digits consisting of only the Western-Arabic numerals 0 to 9 not immediately followed by a lower-case b or a lower-case h (e.g., 175). This specification uses the following conventions for representing decimal numbers:

a) the decimal separator (i.e., separating the integer and fractional portions of the number) is a period;
b) the thousands separator (i.e., separating groups of three decimal digits in a portion of the number) is a comma;
c) the thousands separator is used in only the integer portion of a number and not the fractional portion of a number; and
d) the decimal representation for a year does not include a comma (e.g., 2019 instead of 2,019).

1.5.3 Byte, Word, and Dword Relationships

Figure 3 illustrates the relationship between bytes, words and dwords. A qword (quadruple word) is a unit of data that is four times the size of a word; it is not illustrated due to space constraints. Unless otherwise specified, this specification specifies data in a little-endian format.

Figure 3: Byte, Word, and Dword Relationships



1.5.4 Data Endian-ness

This specification conforms to the Conventions section, Keywords section, and Byte, Word, and Dword Relationships section of the NVM Express Base Specification.

1.5.5 IP Addresses and Formatting

Throughout this document, IP Addresses as identified in this document shall follow the common conventions specified in IETF RFC 4291, RFC 4038, RFC 3986, and RFC 2732. Excerpts and examples included here are done so for readability and do not supersede those RFC's unless explicitly stated.

1.5.5.1 IPv6 and IPv4 Byte Representations

Within this specification, when IP addresses are used in byte fields, then those fields shall consist of reserving 16 bytes to accommodate IPv6 addresses.

Additionally, IPv4 addresses shall be encapsulated in IPv6 address following IETF RFC 4291 and RFC 4038 where IPv4 is modeled inside the IPv6 Scope (which is included in a v4-to-v6 range) in "IPv4-mapped IPv6 address" that looks like:

Figure 4: IPv4 in IPv6 Format

MSB		LSB
80 bits	16 bits	32 bits
0000	FFFF	IPv4 Address

This field has the MSB starting at the left decreasing to right in the data.

1.5.5.2 String Representation

When IP literals are used in string arrays, those IP literals shall follow the IEFT RFC 3986 and RFC 2732 grammar. Specifically, IPv6 Addresses shall follow IETF RFC 2732 square bracket encapsulation of IPv6 String Literals with port and prefix outside square bracket encapsulated literals.

Examples of such are:

"[2001:0DB8:7654:3210:FEDC:BA98:7654:3210]:4420"

"[::FFFF:129.144.52.38]:8009"

Within string arrays, IPv4 Addresses may be encapsulated or not. Field requirements will state if a URI is expected to be a string array.

1.5.6 Host Name String Format

Throughout this document, when a well-formed string representation of a Host Name is required, the string shall follow the following common host representation as specified in RFC 3986, including UTF-8 formatting of the Host Name String.

Host name if supplied shall be in one of the following formats:

- <HOST NAME>
- <HOST NAME>.
- <HOST NAME>.<DOMAIN>

1.5.7 NVMe-oF URI Format

Throughout this document, when well-formed string representations of an address URI are required to be specified, the following URI format shall be adhered to.

An NVMe-oF scheme, that conforms to IETF RFC 3986, consists of two parts: the nvme class, and a +<trtype> transport specific identifier. A valid URI shall be formatted in UTF-8.

Valid Transport types string representation are found in Figure 7.

When referencing IP based transports, Host and Port of an Authority statement shall be valid IP literals, decimal formatted addresses, or resolvable names. Allocation, and registration of which are outside the scope of this document.

Examples of well-formed string representations of an address URI using a tcp transport specific identifier include:

"nvme+tcp://192.168.1.1:4420/"

"nvme+tcp://[FE80::1010]:4420/"

1.5.8 Optional and Mandatory Fields in Tables

Tables and Structures within this document may contain a 'O/M' column with entries in that column that indicate if the use or content of specific fields in that Table or Structure is Optional or Mandatory. If a record includes an 'M' then that field is mandatory. If a record includes a 'O', it indicates the field is optional.

Optional tables may be absent unless specified otherwise. Optional fields may be cleared to 0h unless otherwise specified. Optional referenced objects, such as heap objects, may be absent and their reference fields may be cleared to 0h unless otherwise specified.

1.5.9 Namespace Identifier String Formats

This section specifies representations that contain lexical decoration and tokenization that may be used when well-known textual representations of Namespace Identifiers are necessary for interoperability. In all cases, the contents of internal and in-memory formats are defined elsewhere, and this section only relates to their textual representations associated with the NVM Express Boot Specification. Further example uses of the Namespace Identifier string formats are described in Section A.3.2.3.

1.5.9.1 NVMe EUI-64 String Format

The EUI-64 textual representation shall follow the IEEE RA EUI-64 hyphenated textual format (refer to the IEEE Guidelines for Use of Extended Unique Identifiers) with the generic prefix decorator "eui:".

This format consists of:

- The string "eui:"; and
- a string representation of the 8 byte IEEE RA formatted EUI-64.

The following is an example of such an NVMe EUI-64 formatted string: "eui:AC-DE-48-23-45-67-01-9F".

1.5.9.2 NVMe NGUID String Format

The NGUID textual representation shall be a prefix of "nvme-nguid:" followed by the NGUID tokens: vendor specific extension identifier, IEEE OUI, and an extension identifier, with a ASCII '-' as a field separator between tokens.

This format consists of:

- The string "nvme-nguid:";
- a string representation of the 8 byte vendor specific extension identifier;
- a ASCII "-";
- a string representation of the 3 byte IEEE OUI;
- a ASCII "-"; and
- a string representation of the 5 byte extension identifier.

The following is an example of such an NVMe NGUID formatted string: “nvme-nguid:FEDCBA9876543210-ABCDEF-0123456789”.

1.5.9.3 NVMe UUID String Format

The UUID textual representation shall follow the IETF RFC 4122 textual format.

The following is an example of such an NVMe UUID formatted string:

“urn:uuid:4eff7f8e-d353-4e9b-a4ec-deea8eab84d7”.

2 Background and General Methodology

2.1 Booting from NVMe

To address the complexities of boot from NVMe, especially NVMe-oF, this document contains normative content that describes generic requirements of information sharing as well as introducing domain specific mechanics such as ACPI tables to provide reliable ways for pre-OS and OS drivers to share configuration context.

2.1.1 Boot from NVMe: PCIe Transport

For the PCIe-based controllers, NVMe storage behaves very similarly to other types of block storage as indicated in Annex A.2.7. Expansion ROM images or boot environment firmware initializes and configures the PCI functions which are the controllers. The controlling firmware enumerates namespaces as block storage devices to the boot environment. Similar to SCSI-based storage, the controlling firmware converts the basic block storage device interface (select/read/write/deselect) used by the boot environment to the NVM Express protocol commands and PCIe transport interface for use with the namespaces. When the OS environment assumes control, its own drivers detects, initializes, and configures the NVMe controllers and enumerate the NVM subsystem's namespaces with the OS.

As the NVM subsystem configuration is predominantly static (i.e., changes only with explicit administrative action taken on the NVM subsystem), the storage devices enumerated to the boot environment will be the same set of storage devices enumerated to the OS. No interaction between the boot environment and OS environment is necessary to ensure device presence.

2.1.2 Boot from NVMe-oF: FC Transport

NVMe-oF Boot from Fibre Channel (FC) storage currently leverages the implementation for Boot from SAN on Fibre Channel. Rather than speaking the SCSI protocol to FC devices, the FC host adapters utilize the NVMe-oF and FC-NVMe protocols with the FC devices. Expansion ROM images for the FC host adapter detects NVMe-oF Subsystems, both discovery and storage subsystems, and creates NVMe-oF associations (including an NVMe controller) to NVM subsystems. Namespaces on the NVM subsystems will then be enumerated to the boot environment.

Note: As a FC endpoint may support both the SCSI and NVMe protocols, an FC host adapter may enumerate storage devices that are both SCSI and NVMe with the boot environment and OS environment.

When creating NVMe-oF FC associations, the FC host adapter typically creates a Host NQN and a HostID value to be used for the associations while in the boot environment. Expansion ROM images typically present administrative interfaces to view and/or set the values to be used for the boot environment. Values for OS-based associations are typically assigned by the OS.

FC supports connectivity to a large number of FC-NVMe devices and NVM subsystems, and the NVM subsystems may enumerate a large number of storage devices. As such, FC host adapters have leveraged the same management methodologies developed for SCSI to allow administrators to view and select the NVMe-oF-based storage devices that are to be enumerated to the boot environment.

FC storage is self-discovering. The host adapter typically sees the same devices regardless of whether it is the boot environment or the OS environment. The host adapter expansion ROM may filter the devices presented to the boot environment. However the devices presented to the OS environment are typically all of the devices, which includes those presented to the boot environment. Therefore, device presentation for

boot devices occurs in both the boot environment and the OS environment without requiring the two environments to interact.

2.1.3 Boot from NVMe-oF: TCP Transport

Boot from NVMe-oF storage on TCP, behaves similarly to NVMe PCIe-based storage and NVMe-oF FC-based storage. Once an NVMe-oF association is created and a controller exists, expansion ROM software or boot environment software enumerates namespaces on the NVMe subsystem and convert the basic block storage device interface (select/read/write/deselect) used by the boot environment to the NVM Express protocol commands.

Similar to historical requirements in iSCSI storage (See further detail in Annex A.2.4), NVMe-oF TCP associations are not self-discovering. The host adapter, or NVMe-oF stack utilizing a network adapter, requires information to locate and/or create the NVMe-oF TCP associations. Information includes IP addresses and port numbers for the initial TCP connections, Host NQN and Host ID values, and Subsystem NQN values to be connected to. To aid in the amount of information required to be known and stored, IP services such as DHCP may be used to query for the NVMe-oF TCP boot device session information. Even in those cases, address information and service type information may have to be specified.

To address these issues, the NVMe Boot Firmware Table (NBFT) provides a mechanism to pass NVMe-oF Boot Association information from the boot environment to the OS environment using the ACPI API. The NBFT is defined in such a manner that transports other than TCP, as supported by the NVM Express Transport specifications, may create tables for boot associations such that they are passed to the OS.

2.1.3.1 NVMe Boot Firmware Table

Section 3.1 defines the NVMe Boot Firmware Table (NBFT) which is a multi-level table structure that provides a standardized means of passing configuration and connection context from a pre-OS Boot environment to an OS runtime in regard to the administratively configured run-time as specified by its user. It is formatted to follow conventions of, and to be included as, an ACPI XSDT table. The naming follows the iSCSI Boot Firmware Table (iBFT).

3 Boot Mechanisms

This section specifies different mechanisms and their requirements that may be used for booting over NVMe-oF.

3.1 ACPI NVMe Boot Firmware Table (NBFT)

The NVMe Boot Firmware Table (NBFT) structure is an ACPI structure in host read/write memory that the platform boot firmware reserves for ACPI usage. The NBFT shall be referenced in the ACPI XSDT. This structure is mandatory if the pre-OS environment supports the NVMe Boot Transport mechanism and NBFT. The presence of the table Header, Control, and Host Descriptor structures shall indicate to the OS that the system supports booting from an NVMe boot mechanism. The NBFT uses a structure approach for associating the local and remote aspects of NVMe Boot configurations. If a BIOS supports NVMe Boot with the mechanism defined in this specification, but it is not administratively configured, then a bare table without a valid control structure shall be present.

ACPI NBFT content is a one-way means of communication from the pre-OS environment to the runtime OS environment. Changes that an OS may make to a NBFT in memory are not able to be captured by the pre-boot environment. If changes to the pre-boot configuration are required, another mechanism such as DMTF's Redfish may be required, and its use and configuration are outside the scope of this document.

Note that some systems may provide more than one NBFT. If there are multiple NBFTs on a system, each of them shall be pointed to by the XSDT. Individual NBFTs are independent of each other (i.e., no cross references are possible between any separate NBFT tables on the same system).

The topic of multiple NBFTs is discussed further in section A.3.1.3 of this document.

3.1.1 NBFT Table Overview

The layout of a NBFT is divided up into representative descriptors, some mandatory and some optional that fully describe features as described in Figure 5.

The table Header, Control, and Host Descriptor structures have fixed well-known addresses, all other table structures have starting locations and lengths based on their content defined in the Control structure. Additionally, the Header, Control, and Host Descriptors are the only mandatory structures for a valid minimal NBFT, all other structures are optional unless pre-OS connectivity is established that requires a structure to be conveyed to an OS. Lastly, the Header, Control and Host Descriptors shall be present in the order defined in Figure 8, however all other structures are referenced by offset in the Control structure and may be contiguously added to the NBFT in any order.

Figure 5: NBFT Elements

Element	Descriptor Type Designator	Description	Reference
Header	N/A	An ACPI structure header with some additional NBFT specific info.	3.1.2.1
Control Descriptor	1h	Indicates the location of host, HFI, SSNS, security, and discovery descriptors.	3.1.2.2
Host Descriptor	2h	Host information.	3.1.2.3
HFI Descriptor	3h	An indexable table of HFI Descriptors, one for each fabric interface on the host.	3.1.2.4
Subsystem Namespace Descriptor	4h	An indexable table of SSNS Descriptors.	3.1.2.5

Figure 5: NBFT Elements

Element	Descriptor Type Designator	Description	Reference
Security Descriptor	5h	An indexable table of Security descriptors.	3.1.2.6
Discovery Descriptor	6h	An indexable table of Discovery Ddescriptors.	3.1.2.7
Heap Resident Structures			
HFI Transport Descriptor	7h	Indicated by an HFI Descriptor, corresponds to a specific transport for a single HFI..	3.1.2.4.1.1
Reserved	8h		
SSNS Extended Info Descriptor	9h	Indicated by an SSNS Descriptor if required.	3.1.2.5.5

Figure 6 shows graphically the relationships of the elements of the NBFT. In addition to the two elements shown in the Heap, other elements are placed in the Heap as described in subsequent sections.

Figure 6: NBFT Layout

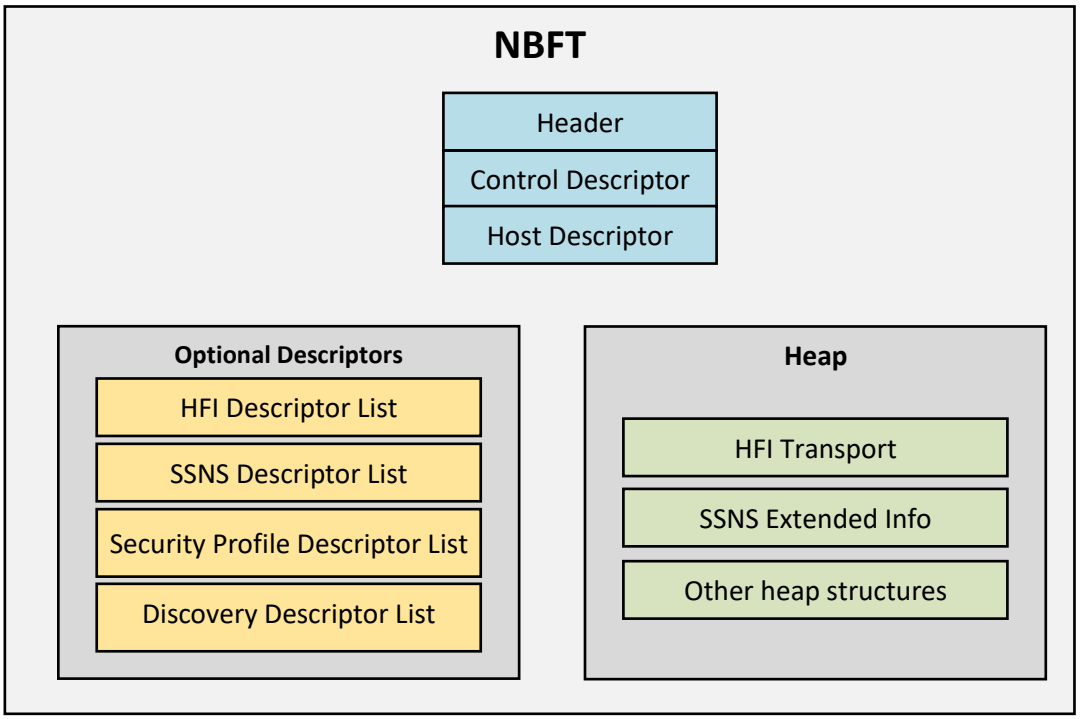


Table elements are typically compacted in memory, but this is not required, and the tree shall be traversed by the links (i.e., offset fields).

3.1.1.1 Heap

The Heap is a structure within the NBFT and exists as a contiguous range of addressable variable storage space used by other NBFT structures. A heap contains objects that may be binary data, a null-terminated string, or structured data that may be variable length in size. The null termination of a string shall count towards the length of referenced heap object. The heap may or may not be compact. If a pre-OS NBFT driver removes an entry, the pre-OS NBFT driver may or may not recompact the heap. Order of entries in the heap shall not be assumed. Offsets to heap objects in NBFT descriptors shall be specified as a relative offset from the start of the NBFT. When addressing a valid object in the heap, length and offset address

shall be set to a non-zero value. For unused and optional heap objects, length and offset address shall both be cleared to 0h in NBFT descriptors.

3.1.1.2 Index Values

List elements in NBFT descriptors are specified as Index values. Index values are used as part of the structure offsets for providing the enumerated elements in the system with discrete configuration. As part of resolving the unique access model of a device, HFI, SSNS, and their security profiles shall be paired together. Index values unless otherwise stated, shall start from 1. Index values for NBFT descriptors are not used in any specific order, and index order should not denote priority to consumers of the data.

3.1.1.3 Transport Types

The NVM Express Base Specification defines Numeric Designator values for Transport types. Throughout this document these Numeric Designators are used for transport specific structures. Transport types supported by the NBFT are listed in Figure 7. In addition, for fields that use string data such as URIs, a string representation is also provided in Figure 7.

Figure 7: NBFT Interface Transport Types

Numeric Designator	Transport	String Designator	Default Transport Service Identifier
3h	NVMe/TCP (802.3 + TCP/IP)	“tcp”	34343230h (i.e.,“4420”) with a length of 4 Bytes

3.1.1.4 Driver Signature

The Driver Signature is a NBFT Header structure that shall contain a per-system unique identification of the driver source of the NBFT structure data. In the NBFT Table Header, the Driver Signature is a heap object consisting of a UTF-8 string for the unique identity of the driver source of NBFT structure data in that system. On UEFI-based systems, the Driver Signature shall consist of UEFI Driver Image Device Path string. Examples of this are found in Annex A.3.1.3.4.

3.1.2 NBFT Structure

3.1.2.1 NBFT Header Structure

Structure of the NBFT Header Table (refer to Figure 8) follows the normal conventions of an ACPI system description table.

The host or OS software that is compliant with this specification should ignore any new identifiers and fields indicating larger minor revision numbers.

Figure 8: NBFT Table

Bytes	Description
	Header
03:00	**Signature:** An ASCII string representation of the table identifier. This field shall be set to the value 4E424654h (i.e., “NBFT”) for the NVMe-oF Boot Firmware Table.
07:04	**Length:** The length of the table, in bytes, including the header, starting from offset 0h. This field is used to record the size of the entire table.
08	**Major Revision:** The major revision of the structure corresponding to the Signature field. Larger major revision numbers should not be assumed backward compatible to lower major revision numbers with the same signature. This field shall be set to ‘1’.

Figure 8: NBFT Table

<table>
<tr><th>Bytes</th><th>Description</th></tr>
<tr><td>09</td><td>Checksum: The entire table, including the Checksum field, shall sum to 0h to be considered valid. This checksum shall follow ACPI table checksum requirements. Refer to the Advanced Configuration and Power Interface (ACPI) Specification.</td></tr>
<tr><td>15:10</td><td>OEMID: OEMID shall be populated by the NBFT driver writer by an OEM-supplied string that identifies the OEM. All trailing bytes shall be null.</td></tr>
<tr><td>23:16</td><td>OEM Table ID: This field shall be populated by the NBFT driver writer with an OEM-supplied string that the OEM uses to identify the particular data table. This field is particularly useful when defining a definition block to distinguish definition block functions. The OEM assigns each dissimilar table a new OEM Table ID.</td></tr>
<tr><td>27:24</td><td>OEM Revision: An OEM-supplied revision number. Larger numbers are assumed to be newer revisions.</td></tr>
<tr><td>31:28</td><td>Creator ID: Vendor ID of utility that created the table. For instance, this may be the ID for the ASL Compiler.</td></tr>
<tr><td>35:32</td><td>Creator Revision: Revision of utility that created the table. For instance, this may be the ID for the ASL Compiler.</td></tr>
<tr><td>39:36</td><td>Heap Offset (HO): This field indicates the offset in bytes of the heap, if any, from byte offset 0h of the NBFT Table Header.</td></tr>
<tr><td>43:40</td><td>Heap Length (HL): The length of the heap, if any.</td></tr>
<tr><td>49:44</td><td>Driver Signature Heap Object Reference: This field indicates the offset in bytes of a heap object containing the Driver Signature, if any, from byte offset 0h of the NBFT Table Header. For more detail on Driver Signature, refer to section A.3.1.3.4.
<table>
<tr><th>Bytes</th><th>Description</th></tr>
<tr><td>03:00</td><td>Offset: Offset in bytes of the heap object, if any, from byte offset 0h of the NBFT Table Header.</td></tr>
<tr><td>05:04</td><td>Length: Length in bytes of the heap object, if any.</td></tr>
</table></td></tr>
<tr><td>50</td><td>Minor Revision: The minor revision of the structure corresponding to the Signature field. If the major revision numbers are the same, any minor revision number differences shall be backwards compatable with the same signature.

This field shall be cleared to ‘0’.</td></tr>
<tr><td>63:51</td><td>Reserved</td></tr>
<tr><th colspan="2">Control Descriptor</th></tr>
<tr><td>64</td><td>Structure ID: This field specifies the element (refer to Figure 5). This field shall be set to 1h (i.e., Control).</td></tr>
<tr><td>65</td><td>Major Revision: The major revision of the structure corresponding to the Signature field. Larger major revision numbers should not be assumed backward compatible to lower major revision numbers with the same signature.

This field shall be set to ‘1’.</td></tr>
<tr><td>66</td><td>Minor Revision: The minor revision of the structure corresponding to the signature field. If the major revision numbers are the same, any minor revision number differences shall be backwards compatable with the same signature.

This field shall be cleared to ‘0’.</td></tr>
<tr><td>67</td><td>Reserved</td></tr>
<tr><td>69:68</td><td>Control Structure Length (CSL): This field indicates the length in bytes of the Control Descriptor.</td></tr>
<tr><td>70</td><td>Flags:
<table>
<tr><th>Bits</th><th>Feature</th></tr>
<tr><td>07:01</td><td>Reserved</td></tr>
<tr><td>00</td><td>Block Valid: If set to ’1’, then this structure is valid. If cleared to ’0’, then this structure is not valid.</td></tr>
</table></td></tr>
<tr><td>71</td><td>Reserved</td></tr>
</table>

Figure 8: NBFT Table

<table>
<tr><th>Bytes</th><th>Description</th></tr>
<tr><td>77:72</td><td>Host Descriptor (HDESC): This field indicates the location and length of the Host Descriptor (refer to Figure 9).
<table>
<tr><th>Bytes</th><th>Description</th></tr>
<tr><td>03:00</td><td>Offset: Offset in bytes of the Host Descriptor from byte offset 0h of the NBFT Table Header. The Offset field shall be set to a non-zero value.</td></tr>
<tr><td>05:04</td><td>Length: Length in bytes of the Host Descriptor. The Length field shall be set to a non-zero value.</td></tr>
</table></td></tr>
<tr><td>78</td><td>Host Descriptor Version (HSV): This field indicates the version of the Host Descriptor. This field shall be set to ’1’.</td></tr>
<tr><td>79</td><td>Reserved</td></tr>
<tr><th colspan="2">Host Fabric Interface Descriptor Information</th></tr>
<tr><td>83:80</td><td>HFI Descriptor List Offset (HFIO): If this field is set to a non-zero value, then this field indicates the offset in bytes of the HFI Descriptor List (refer to Figure 10), if any, from byte offset 0h of the NBFT Table Header. If the NumHFI field is cleared to 0h, then this field is reserved.</td></tr>
<tr><td>85:84</td><td>HFI Descriptor Length (HFIL): This field indicates the length in bytes of each HFI Descriptor, if any. If the NumHFI field is cleared to 0h, then this field is reserved.</td></tr>
<tr><td>86</td><td>HFI Descriptor Version (HFIV): This field indicates the version of each HFI Descriptor. This field shall be set to ’1’.</td></tr>
<tr><td>87</td><td>Number of Host Fabric Interface Descriptors (NumHFI): This field indicates the number of HFI Descriptors (refer to Figure 11) in the HFI Descriptor List (refer to Figure 10), if any. If no interfaces have been configured, then this field shall be cleared to 0h.</td></tr>
<tr><th colspan="2">Subsystem Namespace Descriptor Information</th></tr>
<tr><td>91:88</td><td>SSNS Descriptor List Offset (SSNSO):: This field indicates the offset in bytes of the SSNS Descriptor List (refer to Figure 14), if any, from byte offset 0h of the NBFT Table Header. If the NumSSNS field is cleared to 0h, then this field is reserved.</td></tr>
<tr><td>93:92</td><td>SSNS Descriptor Length (SSNSL): This field indicates the length in bytes of each SSNS Descriptor, if any. If the NumSSNS field is cleared to 0h, then this field is reserved.</td></tr>
<tr><td>94</td><td>SSNS Descriptor Version (SSNSV): This field indicates the version of the SSNS Descriptor. This field shall be set to ’1’.</td></tr>
<tr><td>95</td><td>Number of Subsystem and Namespace Descriptors (NumSSNS): This field indicates the number of Subsystem Namespace (SSNS) Descriptors (refer to Figure 15) in the SSNS Descriptor List (refer to Figure 14), if any.</td></tr>
<tr><th colspan="2">Security Profile Information</th></tr>
<tr><td>99:96</td><td>Security Profile Descriptor List Offset (SECO): This field indicates the offset in bytes of the Security Profile Descriptor List (refer to Figure 20), if any, from byte offset 0h of the NBFT Table Header. If the NumSec field is cleared to 0h, then this field is reserved.</td></tr>
<tr><td>101:100</td><td>Security Profile Descriptor Length (SECL): This field indicates the length in bytes of each Security Profile Descriptor, if any. If the NumSec field is cleared to 0h, then this field is reserved.</td></tr>
<tr><td>102</td><td>Security Profile Descriptor Version (SECV): This field indicates the version of the Security Profile Descriptor. This field shall be set to ’1’.</td></tr>
<tr><td>103</td><td>Number of Security Profile Descriptors (NumSec): This field indicates the number of Security Profile Descriptors (refer to Figure 21), if any, in the Security Profile Descriptor List (refer to Figure 20).</td></tr>
<tr><th colspan="2">Discovery Descriptor Information</th></tr>
<tr><td>107:104</td><td>Discovery Descriptor Offset (DISCO): This field indicates the offset in bytes of the Discovery Descriptor List (refer to Figure 23), if any, from byte offset 0h of the NBFT Table Header. If the NumDisc field is cleared to 0h, then this field is reserved.</td></tr>
<tr><td>109:108</td><td>Discovery Descriptor Length (DISCL): This field indicates the length in bytes of each Discovery Descriptor, if any. If the NumDisc field is cleared to 0h, then this field is reserved.</td></tr>
<tr><td>110</td><td>Discovery Descriptor Version (DISCV): This field indicates the version of the Discovery Descriptor. This field shall be set to ’1’.</td></tr>
<tr><td>111</td><td>Number of Discovery Descriptors (NumDisc): This field indicates the number of Discovery Descriptors Figure 24), if any, in the Discovery Descriptor List (refer to Figure 23), if any.</td></tr>
<tr><td>127:112</td><td>Reserved</td></tr>
</table>

3.1.2.1.1 Optional: HFI Descriptor Offsets

If the Number of HFI Descriptors (NumHFI) field is greater than 1h, the offset for each additional HFI Descriptor is calculated from the descriptor information. That calculation is based on the two-byte location value offset by the number of HFI's specified in the NumHFI field.

An example calculation for the offset of a specific HFI Descriptor Index is:

IF(NumHFI >1) { HFIO + (HFIL * NumHFI)}.

3.1.2.1.2 Optional: Subsystem Namespace Descriptor Offsets

If the number of Subsystem and Namespace Descriptors (NumSSNS) field specified in the NumSSNS field, is greater than 1h, then additional Namespace records are present and are located at an offset after the first Namespace record. The offset calculation is based on the two-byte location value of Namespaces specified in NumSSNS.

An example calculation for the Offset is:

IF(NumSSNS >1) { SSNS_1_Offset + (SSNSL * NumSSNS)}.

3.1.2.1.3 Optional: Security Profile Descriptor Offsets

If the Number of Security Descriptors (NumSec) field referenced in the NumSec field, is greater than 0h, then the user has specified security profiles and this optional structure shall be populated. The offset for the number of security profiles is a calculated value based on the number of Security Profiles specified in the NumSec field.

An example calculation for the Offset is:

IF(NumSec > 1) { SECO + (SECL * (NumSec - 1))}.

3.1.2.1.4 Optional: Discovery Descriptor Offsets

If the Number of Discovery Descriptors (NumDisc) field referenced in the NumDisc field, is greater than 0h, then the user has specified discovery entries and the corresponding optional structure shall be populated. The offset for the number of Discovery Descriptor entries is a calculated value based on the number of Discovery Entries specified in the NumDisc field.

An example calculation for the Offset is:

IF(NumDisc > 1) { DISCO + (DISCL * (NumDisc - 1))}.

3.1.2.2 Control Descriptor

The Control Descriptor (refer to Figure 8) presents global configuration options that are meaningful to the system as a whole. This includes offset reference designators for the configured system elements. Byte Offset in the Control Descriptor is specified as a relative offset from the start of the NBFT.

The Control Descriptor provides lookup features into the offsets of each additional Table structure via structure indices. The Host Descriptor defines the identity of the host known by the pre-OS environment. Lastly, the heap space, an area for string and configuration attributes of a non-fixed length resides at the end of the NBFT and is defined in the NBFT Table Header.

3.1.2.3 Host Descriptor

The Host Descriptor (refer to Figure 9) specifically defines details that are scoped globally to the host. Only one instance of this descriptor shall exist per NBFT for a physical or logical system that implements boot from NVMe-oF. The byte offset in the Host Descriptor Table is specified as a relative offset from the start of this Descriptor rather in reference from the start of the NBFT.

Figure 9: Host Descriptor

<table>
<tr><th>Bytes</th><th>Description</th></tr>
<tr><td>00</td><td>Structure ID: This field shall be set to 2h (i.e., Host Descriptor; refer to Figure 5).</td></tr>
<tr><td>01</td><td>Host Flags:
<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>00</td><td>Descriptor Valid: If set to ’1’, then this descriptor is valid. If cleared to 0, then this descriptor is reserved.</td></tr>
<tr><td>01</td><td>HostID Configured: If set to ’1’, then the Host ID field contains an administratively-configured value. If cleared to 0, then the Host ID field contains a driver default value.</td></tr>
<tr><td>02</td><td>Host NQN Configured: If set to ’1’, then the Host NQN indicated by the Host NQN Heap Object Reference field contains an administratively-configured value. If cleared to 0, then the Host NQN indicated by the Host NQN Offset field contains a driver default value.</td></tr>
<tr><td>04:03</td><td>Primary Administrative Host Descriptor: This field indicates whether the Host Descriptor in this NBFT was selected as the primary NBFT for administrative purposes of platform identity as a hint to the OS.

If multiple NBFT tables are present, only one NBFT should be administratively selected. There is no enforcement mechanism for this to be coordinated between multiple NBFT tables, but this field should be set to Selected (i.e., 10b) if more than one NBFT is present.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>00b</td><td>Not Indicated by Driver: The driver that created this NBFT provided no administrative priority hint for this NBFT.</td></tr>
<tr><td>01b</td><td>Unselected: The driver that created this NBFT explicitly indicated that this NBFT should not be prioritized over any other NBFT.</td></tr>
<tr><td>10b</td><td>Selected: The driver that created this NBFT explicitly indicated that this NBFT should be prioritized over any other NBFT.</td></tr>
<tr><td>11b</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>05:07</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>17:02</td><td>Host ID: This field shall be set to the Host Identifier (refer to the NVM Express Base Specification).

This field shall not be empty if the NBFT and NVMe Boot are supported by the Platform.</td></tr>
<tr><td>23:18</td><td>Host NQN Heap Object Reference: this field indicates a heap object containing a Host NQN (refer to the NVM Express Base Specification)

This object shall not be empty if the NBFT and NVMe Boot are supported by the Platform.
<table>
<tr><th>Bytes</th><th>Description</th></tr>
<tr><td>03:00</td><td>Offset: Offset in bytes of the heap object, if any, from byte offset 0h of the NBFT Table Header. This field shall be set to a non-zero value.</td></tr>
<tr><td>05:04</td><td>Length: Length in bytes of the heap object, if any. This field shall be set to a non-zero value.</td></tr>
</table></td></tr>
<tr><td>31:24</td><td>Reserved</td></tr>
</table>

The value of the Host ID field in the NBFT Host Descriptor shall be set to the value used by the pre-OS driver that populates the NBFT and was used as part of any Connect command by the pre-OS driver. Upon reaching run-time, an OS or application may use a different value for the Host ID, the value in the Host ID field is to represent the Host ID used as part of the boot lifecycle.

If the driver value used was administratively configured, the HostID Configured bit shall be set to '1'. If the driver assumed a value or used a non-administrative authoritative source (e.g., default values with respect to a platform or device), then the HostID Configured bit shall be cleared to '0'. If no authoritative sources are present to supply a value for the HostID field, the HostID field shall be cleared to 0h by the driver.

The Host NQN is used by the Connect command to identify the host in this transaction. Upon reaching run-time, an OS or application may use a different Host NQN than the Host NQN which was used as part of the Boot lifecycle. If the value of Host NQN was not administratively configured, the Host NQN Configured bit shall be cleared to '0' and the value of Host NQN in the Host Descriptor shall be populated with the value used by the driver.

3.1.2.4 Host Fabric Interface (HFI) Descriptor

This Host Fabric Interface Descriptor List (refer to Figure 10) consists of an array of contiguous Host Fabric Interface Descriptors (refer to Figure 11). Each Host Fabric Interface Descriptor indicates the transport specific parameters associated with one fabric interface. Fabric interfaces may be a representation of specific host adapters with specific transports as defined by the HFI Transport Type flag. The same fabric interface or host adapter may support multiple transport types with their own unique configuration, which results in multiple HFI Descriptors in sequence within the NumHFI defined in the Control Descriptor.

Transport specific details are addressed in section 3.1.2.4.1.

Figure 10: Host Fabric Interface (HFI) Descriptor List

Bytes	Description
31:00	**Host Fabric Interface Descriptor 1:** The first Host Fabric Interface Descriptor (refer to Figure 11), if any.
64:32	**Host Fabric Interface Descriptor 2:** The second Host Fabric Interface Descriptor (refer to Figure 11), if any.
…	
(n*32)-1:(n-1)*32	**Host Fabric Interface Descriptor n:** The last Host Fabric Interface Descriptor (refer to Figure 11), if any.

Figure 11: Host Fabric Interface (HFI) Descriptor

<table>
<tr><th>Bytes</th><th>Description</th></tr>
<tr><td>00</td><td>Structure ID: This field shall be set to 3h (i.e., Host Fabric Interface Descriptor; refer to Figure 5).</td></tr>
<tr><td>01</td><td>HFI Descriptor Index: This field indicates the number of this HFI Descriptor in the Host Fabric Interface Descriptor List (i.e., the number 1h represents HFI Descriptor 1, 2h represents HFI Descriptor 2, etc.).</td></tr>
<tr><td>02</td><td>HFI Descriptor Flags:
<table><tr><th>Bits</th><th>Description</th></tr><tr><td>07:01</td><td>Reserved</td></tr><tr><td>00</td><td>Descriptor Valid: If set to '1', then this descriptor is valid. If cleared to 0, then this descriptor is reserved.</td></tr></table></td></tr>
<tr><td>03</td><td>HFI Transport Type: Refer to Figure 7.</td></tr>
<tr><td>15:04</td><td>Reserved</td></tr>
</table>

Figure 11: Host Fabric Interface (HFI) Descriptor

<table>
<tr><th>Bytes</th><th>Description</th></tr>
<tr><td>21:16</td><td>HFI Transport Info Descriptor Heap Object Reference: If this field is set to a non-zero value, then:
a) this field indicates the location and size of a heap object containing a HFI Transport Info Descriptor (refer to section 3.1.2.4.1); and
b) the Offset field and the Length field shall be set to non-zero values.
If this field is cleared to 0h, then this field is reserved.
<table>
<tr><th>Bytes</th><th>Description</th></tr>
<tr><td>03:00</td><td>Offset: Offset in bytes of the heap object, if any, from byte offset 0h of the NBFT Table Header.</td></tr>
<tr><td>05:04</td><td>Length: Length in bytes of the heap object, if any.</td></tr>
</table>
</td></tr>
<tr><td>31:22</td><td>Reserved</td></tr>
</table>

3.1.2.4.1 HFI Transport Info Descriptor

This section contains all the HFI Transports that are supported. All HFI Transport Info Descriptors consist of a common header format with NBFT structure ID of 7h with a HFI Transport Type specific payload that corresponds to a Transport type described in the HFI Transport Type field (refer to Figure 7). Figure 12 shows the common representation of what a Transport Specific HFI Transport Info Descriptor header shall contain.

Figure 12: HFI Transport Info Descriptor Header Template

Bytes	Description
00	**Structure ID:** This field shall be set to 7h (i.e., HFI Transport Info; refer to Figure 5).
01	**Version:** This field shall be set to 1h.
02	**HFI Transport Type:** Refer to Figure 7.
03	**Transport Info Version:** Implementations compliant to this specification shall set this field to 1h.
05:04	**HFI Descriptor Index:** The value of the HFI Descriptor Index field of the HFI Descriptor (refer to Figure 11) whose HFI Transport Info Descriptor Heap Object Reference field indicates this HFI Transport Info Descriptor.

3.1.2.4.1.1 HFI Transport Info Descriptor – NVMe/TCP

This section defines the HFI Transport details specific to the NVMe/TCP transport defined in The NVM Express TCP Transport Specification in Figure 13.

Figure 13: HFI Transport Info Descriptor – NVMe/TCP

Bytes	O/M[1]	Description
00	M	**Structure ID:** This field shall be set to 7h (i.e., HFI Transport Info; refer to Figure 5).
01	M	**Version:** This field shall be set to 1h.
02	M	**HFI Transport Type:** This field shall be set to 03h (i.e., NVMe/TCP; refer to Figure 7).
03	M	**Transport Info Version:** Implementations compliant to this specification shall set this field to 1h.
05:04	M	**HFI Descriptor Index:** The value of the HFI Descriptor Index field of the HFI Descriptor (refer to Figure 11) whose HFI Transport Info Descriptor Heap Object Reference field indicates this HFI Transport Info Descriptor.

Figure 13: HFI Transport Info Descriptor – NVMe/TCP

<table>
<tr><th>Bytes</th><th>O/M[1]</th><th>Description</th></tr>
<tr><td>06</td><td>M</td><td>HFI Transport Flags:
<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>07:03</td><td>Reserved</td></tr>
<tr><td>02</td><td>DHCP Override: If set to ’1’, then HFI information was populated by consuming the DHCP on this interface. If cleared to ’0’, then the HFI information was set administratively by a configuration interface to the driver and pre-OS envrionment.</td></tr>
<tr><td>01</td><td>Global Route vs. Link Local Override Flag: If set to ’1’, then the pre-OS driver utilized this interface described by HFI to be the default route with highest priority. If cleared to 0, then routes are local to their own scope. Refer to section 3.1.2.4.1.3.</td></tr>
<tr><td>00</td><td>Descriptor Valid: If set to ’1’, then this descriptor is valid. If cleared to 0, then this descriptor is reserved.</td></tr>
</table></td></tr>
<tr><td>10:07</td><td>M</td><td>PCI Express Routing ID for the HFI Transport Function: This field indicates the PCI Express Routing ID as specified in the PCI Express Base Specification.

Note: If the platform supports PCIe ARI, then the Device bits and the Function bits may be concatenated for that purpose.
<table>
<tr><th>Bits</th><th>Description</th><th>Refer to</th></tr>
<tr><td>31:16</td><td>Segment Group Number</td><td>PCI Firmware Specification</td></tr>
<tr><td>15:08</td><td>Bus</td><td rowspan="3">PCI Express Base Specification</td></tr>
<tr><td>07:03</td><td>Device</td></tr>
<tr><td>02:00</td><td>Function</td></tr>
</table></td></tr>
<tr><td>16:11</td><td>M</td><td>MAC Address: The MAC address of this HFI, in EUI-48™ format, as defined in the IEEE Guidelines for Use of Extended Unique Identifiers. This field shall be set to a non-zero value.</td></tr>
<tr><td>18:17</td><td>O</td><td>VLAN: If this field is set to a non-zero value, then this field contains the VLAN identifier if the VLAN associated with this HFI, as defined in IEEE 802.1q-2018.

If no VLAN is associated with this HFI, then this field shall be cleared to 0h.</td></tr>
<tr><td>19</td><td>O</td><td>IP Origin: If this field is set to a non-zero value, then this field indicates the source of Ethernet L3 configuration information used by the driver for this interface. Valid values are defined in the Win 32 API: NL_PREFIX_ORIGIN enumeration specification.

This field should be cleared to ‘0’ if the IP Origin field is unused by driver.</td></tr>
<tr><td>35:20</td><td>M</td><td>IP Address: This field indicates the IPv4 or IPv6 address of this HFI. This field shall be set to a non-zero value.

The format of this field is defined by section 1.5.5.1.</td></tr>
<tr><td>36</td><td>M</td><td>Subnet Mask Prefix: This field indicates the IPv4 or IPv6 subnet mask in CIDR routing prefix notation.</td></tr>
<tr><td>52:37</td><td>O</td><td>IP Gateway If this field is set to a non-zero value, this field indicates the IPv4 or IPv6 address of the IP gateway for this HFI. If this field is cleared to 0h, then no IP gateway is specified.</td></tr>
<tr><td>53</td><td></td><td>Reserved</td></tr>
<tr><td>55:54</td><td>O</td><td>Route Metric: If this field is set to a non-zero value, this field indicates the cost value for the route indicated by this HF. This field contains the value utilized by the pre-OS driver when chosing among all availible routes. Lower values relate to higher priority. Refer to IETF RFC 4249.

If the pre-OS driver supports routing and did not configure a specific route metric for this interface, then the pre-OS driver should set this value to 500.

If the pre-OS driver does not support routing, then this field should be cleared to 0h.</td></tr>
<tr><td>71:56</td><td>O</td><td>Primary DNS: If this field is set to a non-zero value, this field indicates the IPv4 or IPv6 address of the Primary DNS server for this HFI, if any, from byte offset 0h of the NBFT Table Header. If this field is cleared to 0h, then no Primary DNS is specified.</td></tr>
<tr><td>87:72</td><td>O</td><td>Secondary DNS: If this field is set to a non-zero value, this field indicates the IPv4 or IPv6 address of the Secondary DNS server for this HFI, if any, from byte offset 0h of the NBFT Table Header. If this field is cleared to ’0’, then no Secondary DNS is specified.</td></tr>
</table>

Figure 13: HFI Transport Info Descriptor – NVMe/TCP

<table>
<tr><th>Bytes</th><th>O/M[1]</th><th>Description</th></tr>
<tr><td>103:88</td><td>O</td><td>DHCP Server: If the DHCP Override bit is set to ’1’, then this field indicates the IPv4 or IPv6 address of the DHCP server used to assign this HFI address. If that bit is cleared to ’0’, then this field is reserved.</td></tr>
<tr><td>109:104</td><td>O</td><td>Host Name Heap Object Reference: If this field is set to a non-zero value, then:
a) this field indicates the location and size of a heap object containing a Host Name string (refer to section 1.5.6); and
b) the Offset field and the Length field shall be set to non-zero values.
If this field is cleared to 0h, then this field is reserved.
<table>
<tr><th>Bytes</th><th>Description</th></tr>
<tr><td>03:00</td><td>Offset: Offset in bytes of the heap object, if any, from byte offset 0h of the NBFT Table Header.</td></tr>
<tr><td>05:04</td><td>Length: Length in bytes of the heap object, if any.</td></tr>
</table></td></tr>
<tr><td>127:110</td><td></td><td>Reserved</td></tr>
<tr><td colspan="3">Notes:
1. O/M definition: O = Optional, M = Mandatory.</td></tr>
</table>

3.1.2.4.1.2 HFI Info Payload Host Name

In a DHCP scenario the HFI Info Payload Host Name may be the name stored as the Option 12 host name following the Host Name String Format (refer to section 1.5.6).

3.1.2.4.1.3 HFI Global and Local Routes

An HFI may optionally specify an IP Gateway, which indicates that a pre-OS driver route entry for a specific HFI descriptor exists. If the Global Route vs. Link Local Override Flag bit is cleared to ‘0’ in the HFI Transport Flags field (refer to Figure 13), the HFI route specified by that descriptor is link local. If the Global Route vs. Link Local Override Flag bit is set to ‘1’, the HFI route specified by that descriptor is the default route that was used by the pre-OS driver for the purposes of reachability. There shall be at most one descriptor per NBFT that sets the Global Route vs. Link Local Override Flag bit to ‘1’.

3.1.2.5 Subsystem Namespace (SSNS) Descriptor

The Subsystem Namespace Descriptor List (refer to Figure 14) consists of an array of Subsystem and Namespace (SSNS) descriptors, defined in Figure 15, with NumSSNS contiguous entries. Each SSNS descriptor defines the namespace specific parameters required to uniquely identify the attached namespace that was available during system initialization and the boot process. If multiple transports correspond to the same SSNS descriptor there may be multiple SSNS descriptors for the same NID with at least one per transport type.

Figure 14: Subsystem Namespace Descriptor List

Bytes	Description
127:00	**Subsystem Namespace Descriptor 1** The first Subsystem Namespace Descriptor (refer to Figure 15), if any.
255:128	**Subsystem Namespace Descriptor 2:** The second Subsystem Namespace Descriptor (refer to Figure 15), if any.
…	
(n*128)-1:(n-1)*128	**Subsystem Namespace Descriptor n:** The last Subsystem Namespace Descriptor (refer to Figure 15), if any.

Figure 15: Subsystem Namespace (SSNS) Descriptor

<table>
<tr><th>Bytes</th><th>O/M[1]</th><th>Description</th></tr>
<tr><td>00</td><td>M</td><td>Structure ID: This field shall be set to 4h (i.e., SSNS; refer to Figure 5).</td></tr>
<tr><td>02:01</td><td>M</td><td>SSNS Descriptor Index: This field indicates the number of this Subsystem Namespace Descriptor in the Subsystem Namespace Descriptor List (i.e., the number 1h represents Subsystem Namespace Descriptor 1, the number 2h represents Subsystem Namespace Descriptor 2, etc.).</td></tr>
<tr><td>04:03</td><td>M</td><td>SSNS Flags: Refer to Figure 16.</td></tr>
<tr><td>05</td><td>M</td><td>Transport Type: Refer to Figure 7.</td></tr>
<tr><td>07:06</td><td>M</td><td>Transport Specific Flags: Refer to Figure 17.</td></tr>
<tr><td>08</td><td>O</td><td>Primary Discovery Controller Index: The Discovery Descriptor Index field of the Discovery Descriptor (refer to Figure 24) that is associated with this SSNS Descriptor.

If a Discovery controller was used to establish this record this value shall be set to a non-zero value.

If this namespace was associated with multiple Discovery controllers, those Discovery controllers shall have records in the Discovery Descriptor to facilitate multi-path rediscovery as required.

If no Discovery controller was utilized to inform this namespace record, this field shall be cleared to 0h.</td></tr>
<tr><td>09</td><td></td><td>Reserved</td></tr>
<tr><td>15:10</td><td>M</td><td>Subsystem Transport Address Heap Object Reference: This field indicates the location and size of a heap object containing the Subsystem Transport Address.

For IP based transports types, shall be an IP Address, following the format described in section 1.5.5.1.

<table><tr><th>Bytes</th><th>Description</th></tr><tr><td>03:00</td><td>Offset: Offset in bytes of the heap object from byte offset 0h of the NBFT Table Header. The Offset field shall be set to a non-zero value.</td></tr><tr><td>05:04</td><td>Length: Length in bytes of the heap object. The Length field shall be set to a non-zero value.</td></tr></table></td></tr>
<tr><td>21:16</td><td>M</td><td>Subsystem Transport Service Identifier Heap Object Reference: This field indicates the location and size of a heap object containing an array of bytes indicating the Subsystem Transport Service Identifier.

Valid values are found in Figure 7.

<table><tr><th>Bytes</th><th>Description</th></tr><tr><td>03:00</td><td>Offset: Offset in bytes of the heap object from byte offset 0h of the NBFT Table Header. The Offset field shall be set to a non-zero value.</td></tr><tr><td>05:04</td><td>Length: Length in bytes of the heap object. The Length field shall be set to a non-zero value.</td></tr></table></td></tr>
<tr><td>23:22</td><td>M</td><td>Subsystem Port ID: Port in the NVM subsystem associated with this transport address used by the pre-OS driver (refer to the NVM Express Base Specification).</td></tr>
<tr><td>27:24</td><td>O</td><td>Namespace ID: This field indicates the namespace identifier (NSID) of the namespace indicated by this descriptor.

This field shall be cleared to 0h if not specified by the user. If this value is cleared to 0h, then consumers of the NBFT shall rely on the NID.</td></tr>
<tr><td>28</td><td>M</td><td>Namespace Identifier Type (NIDT): This field contains the value of the Namespace Identifier Type (NIDT) field in the Namespace Identification Descriptor (refer to the NVM Express Base Specification) for the namespace indicated by this descriptor.

If a namespace supports multiple NIDT entries for uniqueness, the order of preference is NIDT field value of 3h (i.e., UUID) before 2h (i.e., NSGUID), and 2h before 1h (i.e., EUI-64).</td></tr>
<tr><td>44:29</td><td>M</td><td>Namespace Identifier (NID): This field contains the value of the Namespace Identifier (NID) field in the Namespace Identification Descriptor (refer to the NVM Express Base Specification) for the namespace indicated by this descriptor.</td></tr>
</table>

Figure 15: Subsystem Namespace (SSNS) Descriptor

<table>
<tr><th>Bytes</th><th>O/M[1]</th><th>Description</th></tr>
<tr><td>45</td><td>O</td><td>Security Profile Descriptor Index: If the Use Security Flag bit in the SSNS Flags field is set to ’1’, then this field indicates the value of the Security Profile Descriptor Index field of the Security Profile Descriptor (refer to Figure 21) associated with this namespace.

If the Use Security Flag bit is cleared to ‘0’, then no Security Profile Descriptor is associated with this namespace and this field is reserved.</td></tr>
<tr><td>46</td><td>M</td><td>Primary HFI Descriptor Index: This field indicates the value of the HFI Descriptor Index field of the HFI Descriptor (refer to Figure 11) for the interface associated with this namespace. If multiple HFIs are associated with this record, subsequent interfaces should be populated in the Secondary HFI Associations field (refer to section A.1.4.1).</td></tr>
<tr><td>47</td><td></td><td>Reserved</td></tr>
<tr><td>53:48</td><td>O</td><td>Secondary HFI Associations Heap Object Reference: If this field is set to a non-zero value, then:

a) this field indicates an array of bytes, in which each byte contains the value of the HFI Descriptor Index field of an HFI Descriptor in the HFI Descriptor List (refer to Figure 10 and section A.1.4.1); and
b) the Offset field and the Length field shall be set to non-zero values.

If this field is cleared to 0h, then no secondary HFI associations are specified.

<table><tr><th>Bytes</th><th>Description</th></tr><tr><td>03:00</td><td>Offset: Offset in bytes of the heap object, if any, from byte offset 0h of the NBFT Table Header.</td></tr><tr><td>05:04</td><td>Length: Length in bytes of the heap object, if any.</td></tr></table></td></tr>
<tr><td>59:54</td><td>M</td><td>Subsystem and Namespace NQN Heap Object Reference: This field indicates the location and size of a heap object containing the Subsystem and Namespace NQN (refer to the NVM Express Base Specification).

<table><tr><th>Bytes</th><th>Description</th></tr><tr><td>03:00</td><td>Offset: Offset in bytes of the heap object, if any, from byte offset 0h of the NBFT Table Header. The Offset field shall be set to a non-zero value.</td></tr><tr><td>05:04</td><td>Length: Length in bytes of the heap object, if any. The Length field shall be set to a non-zero value.</td></tr></table></td></tr>
<tr><td>65:60</td><td>O</td><td>SSNS Extended Information Descriptor Heap Object Reference: If the SSNS Extended Info In-use Flag bit is set to ’1’, then:

a) this field indicates the offset in bytes of a heap object containing an SSNS Extended Information Descriptor (refer to Figure 19) heap object from byte offset 0h of the NBFT Table Header; and
b) the Offset field and the Length field shall be set to non-zero values.

If the SSNS Extended Info In-use Flag bit is cleared to ’0’, then this field is reserved.

<table><tr><th>Bytes</th><th>Description</th></tr><tr><td>03:00</td><td>Offset: Offset in bytes of the heap object, if any, from byte offset 0h of the NBFT Table Header.</td></tr><tr><td>05:04</td><td>Length: Length in bytes of the heap object, if any.</td></tr></table></td></tr>
<tr><td>127:66</td><td></td><td>Reserved</td></tr>
<tr><td colspan="3">Notes:
1. O/M definition: O = Optional, M = Mandatory.</td></tr>
</table>

3.1.2.5.1 Subsystem and Namespace Descriptor Flags

Flags for the Subsystem and Namespace Descriptor (refer to Figure 17).

Figure 16: Subsystem and Namespace Specific Flags Field

<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>00</td><td>Descriptor Valid: If set to ’1’, then this descriptor is valid. If cleared to ’0’, then this descriptor is not valid. A host that supports NVMe-oF Boot, but does not currently have a remote Subsystem and Namespace assigned may clear this bit to ’0’.</td></tr>
<tr><td>01</td><td>Non-bootable Entry Flag: If set to ’1’, this flag indicates that this SSNS Descriptor contains a namespace of administrative purpose to the boot process, but the pre-OS may not have established connectivity to or evaluated the contents of this Descriptor. Such namespaces may contain supplemental data deemed relevent by the Administrator as part of the pre-OS to OS hand off.

This may include properties such as a UEFI device path that may not have been created for this namespace. This means an OS runtime may still require the contents of such a namespace to complete later stages of boot.

If cleared to ’0’, then this namespace did not have any special administrative intent.</td></tr>
<tr><td>02</td><td>Use Security Flag: If set to ’1’, then there is a Security Profile Descriptor associated with this SSNS record and the Security Profile Descriptor Index field is valid. If cleared to ’0’, then there is no Security Profile Descriptor associated with this SSNS record and the Security Profile Descriptor Index field is not valid.</td></tr>
<tr><td>03</td><td>DHCP Root-Path Override Flag: If set to ’1’, then this SSNS descriptor was populated by consuming the DHCP Root-Path (refer to section 3.1.2.5.3) on this interface. If cleared to ‘0’, then the DHCP Root-Path was not used in populating the SSNS descriptor.</td></tr>
<tr><td>04</td><td>SSNS Extended Info In-use Flag: If set to ’1’, then the SSNS Extended Information Offset field and the SSNS Extended Information Length field are valid.

This flag, if set to ‘1’, indicates that a Subsystem and Namespace Extended Information Descriptor corresponding to this descriptor is present.</td></tr>
<tr><td>05</td><td>Separate Discovery Controller Flag: If set to ’1’, then the Discovery controller associated with this volume is on a different transport address than the specified in the Subsystem Transport Address Heap Object Reference.

If cleared to ’0’, then the Discovery controller is the same as the Subsystem Transport Address Heap Object Reference.</td></tr>
<tr><td>06</td><td>Discovered Namespace Flag: If set to ’1’, then this namespace was acquired through discovery. If cleared to ’0’, then this namespace was explicitly configured in the system.</td></tr>
<tr><td>08:07</td><td>Unavailable Namespace Flag: This field indicates the availability of the namespace at a specific point in time. Such use is only a hint and its use does not guarantee the availablity of that referenced namespace at any future point in time.

<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>00b</td><td>Not Indicated by Driver: No information is provided.</td></tr>
<tr><td>01b</td><td>Available: A referenced namespace described by this flag was previously accessible by the pre-OS driver.</td></tr>
<tr><td>10b</td><td>Unavailable: This namespace was administratively configured but unattempted, unavailable or inaccessible when establishing connectivity by the pre-OS driver.</td></tr>
<tr><td>11b</td><td>Reserved</td></tr>
</table>
</td></tr>
<tr><td>15:09</td><td>Reserved</td></tr>
</table>

3.1.2.5.2 Transport Specific Flags

The Transport Specific Flags (refer to Figure 17) are flags which have specific meaning depending on the Transport Type of the SSNS Descriptor they belong to. Aside from the Transport Specific Flags In Use bit (i.e., Bit 0), which indicates the use of this field, all other bits shall rely on their association with the Transport column of Figure 17.

Figure 17: SSNS Transport Specific Flags Field

Transport	Bits	Description
Any	0	**Transport Specific Flags in Use:** If set to '1', then this descriptor is valid. If cleared to '0', then this descriptor is not valid.
3h	01	**PDU Header Digest (HDGST) Flag:** If set to '1', then the host or administrator required the connection described by this Subsystem and Namespace Descriptor to use the NVM Header Digest Enabled. A consumer of this information should attempt to use NVM Header Digest when recreating this connection if enabled. If cleared to '0', then the host or administrator did not require the connection described by this Subsystem and Namespace Descriptor to use the NVM Header Digest Enabled.
	02	**Data Digest (DDGST) Flag:** If set to '1', then the host or administrator required the connection described by this Subsystem and Namespace Descriptor to use the NVM Data Digest Enabled. If cleared to '0', then the host or administrator did not require the connection described by this Subsystem and Namespace Descriptor to use the NVM Data Digest Enabled. A consumer of this field should attempt to use NVM Data Digest when recreating this connection if enabled.
	15:03	Reserved

3.1.2.5.3 DHCP Root-Path

The format of this field shall follow the format defined by IETF RFC 2132, DHCP Option 17, which specifies the path name that contains the the root disk for the host. The path is formatted with a hierarchical NVMe-oF URI Format (refer to Figure 7), as defined in section 4.2.4, with UTF-8 characters and formatting otherwise as specified in RFC 3986. A definition of the valid URI tokens and their mandatory or optional requirements are specified in Figure 18.

Valid representations are:

- "NVME<+PROTOCOL>://<SERVERNAME/IP>[:TRANSPORT PORT]/<SUBSYS NQN>/<NID>"; or
- "NVME<+PROTOCOL>://<DISCOVERY CONTROLLER ADDRESS>[:DISCOVERY CONTROLLER PORT]/NQN.2014-08.ORG.NVMEXPRESS.DISCOVERY/<NID>"

Figure 18: DHCP Root-Path Elements

Token	O/M[1]	Description
<PROTOCOL>	M	Value shall correspond to valid Transport Type string designator as defined in section 2.3.1.1.1.3.
<SERVER NAME/IP>	M	The transport address of the Subsystem Port. IPv6 address shall be square brace enclosed per IETF RFC 2732.
<TRANSPORT PORT>	O	If Transport Port is blank, assume 4420 for IP Based Transports.
<DISCOVERY CONTROLLER ADDRESS>	O	If Discovery Controller Address is blank, assume integrated with transport address.
<DISCOVERY CONTROLLER PORT>	O	For protocol types: tcp If Discovery Controller Port is unused, but well known Discovery Subsystem NQN Address is populated a driver shall set this value to 8009. If unused but there is no well-known NQN, then this value may be set to the default as defined by the Transport Type in Figure 7.
<SUBSYSTEM NQN>	M	NQN of the Subsystem being indicated by this record.

Figure 18: DHCP Root-Path Elements

Token	O/M[1]	Description
<NID>	M	Desired namespace (i.e., NVMe NID) that the host shall attach to from available Namespaces
Notes: 1. O/M definition: O = Optional, M = Mandatory.		

3.1.2.5.4 Primary Discovery Controller Index Field

The Primary Discovery Controller Index field indicates the primary or first Discovery controller associated with a namespace as registered in the NBFT. In the cases of hierarchical Discovery controllers, the last controller in a topology that is encountered that refers connectivity to the owning NVM subsystem I/O controller should be specified by this field. In the cases of multipath Discovery controllers, only one Discovery controller per Namespace path shall be tracked by this field but all Discovery controllers in the chain should be registered in the Discovery Descriptor.

If this Discovery controller requires Security Parameters for connectivity, those values shall be listed in the Discovery Descriptor associated with this Index record.

3.1.2.5.5 Subsystem and Namespace Extended Information Descriptor

The Subsystem and Namespace Extended Information Descriptor, defined in Figure 19, when utilized, contains additional transport specific detail relevant to a host reestablishing connectivity to the referenced namespaces that may otherwise only be available through administrative configuration and without which some OS runtime environments may operate deficiently. This information is otherwise available and provided to the pre-OS environment through configuration means beyond the scope of this document. An OS driver should treat this information as informational and may require further advanced functionality to properly leverage.

The Subsystem and Namespace Extended Information Descriptor is structured as a standard table located in heap space as required.

Figure 19: Subsystem and Namespace Extended Information Descriptor

Bytes	O/M[1]	Description
00	M	**Structure ID:** This field shall be set to 9h (i.e., SSNS Extended Info; refer to Figure 5).
01	M	**Version:** This field shall be set to 1h.
03:02	M	**SSNS Descriptor Index:** This field indicates the value of the SSNS Descriptor Index field of the Subsystem and Namespace Descriptor (refer to Figure 15) whose SSNS Extended Information Descriptor Heap Object Reference field indicates this descriptor.

Figure 19: Subsystem and Namespace Extended Information Descriptor

<table>
<tr><th>Bytes</th><th>O/M[1]</th><th>Description</th></tr>
<tr><td>07:04</td><td>M</td><td>Flags:
<table>
<tr><th>Bits</th><th>Feature</th></tr>
<tr><td>31:02</td><td>Reserved</td></tr>
<tr><td>01</td><td>Administrative ASQSZ: If set to ’1’, then the value of the ASQSZ field was provided by administrative configuration for this SSNS record.
If cleared to 0h, then the value of the ASQSZ field was either obtained by discovery or assumed by the driver.</td></tr>
<tr><td>00</td><td>Descriptor Valid: If set to ’1’, then this descriptor is valid. If cleared to ’0’, then this descriptor is reserved.</td></tr>
</table></td></tr>
<tr><td>09:08</td><td>O</td><td>Controller ID: The controller identifier of the first controller assocaited with the Admin Queue by the driver. If a controller identifier is not administratively specified or direct configuration is not supported by the driver, then this field shall be cleared to 0h.</td></tr>
<tr><td>11:10</td><td>M</td><td>Admin Submission Queue Size (ASQSZ): The Admin Submission Queue Size utilized for the respective SSNS by the driver.</td></tr>
<tr><td>17:12</td><td>O</td><td>DHCP Root Path String Heap Object Reference: If the SSNS DHCP Root Path Override flag bit is set to ’1’, then:
a) this field indicates the offset in bytes of a heap object containing an DHCP Root Path String as defined in section 3.1.2.5.3 used by the driver.
b) the Offset field and the Length field shall be set to non-zero values.
If the SNSS DHCP Root Path Override Flag bit is cleared to ’0’, then this field is reserved.
<table>
<tr><th>Bytes</th><th>Description</th></tr>
<tr><td>03:00</td><td>Offset: Offset in bytes of the heap object, if any, from byte offset 0h of the NBFT Table Header.</td></tr>
<tr><td>05:04</td><td>Length: Length in bytes of the heap object, if any.</td></tr>
</table></td></tr>
<tr><td colspan="3">Notes:
1. O/M definition: O = Optional, M = Mandatory.</td></tr>
</table>

3.1.2.6 Security Descriptor

The Security Descriptor List (refer to Figure 20) consists of an array of Security descriptors, as defined in Figure 21, with NumSec contiguous entries. Each Security descriptor defines the various connectivity policy features of authentication and transport security related to NVMe connections to SSNS or Discovery descriptors. Security Profile Descriptors may be referenced in other descriptors, (i.e., the SSNS or discovery) using their list indices.

Figure 20: Security Profile Descriptor List

Bytes	Description
63:00	**Security Profile Descriptor 1:** The first Security Profile Descriptor (refer to Figure 21), if any.
127:64	**Security Profile Descriptor 2:** The second Security Profile Descriptor (refer to Figure 21), if any.
…	
(n*64)-1:(n-1)*64	**Security Profile Descriptor n:** The last Security Profile Descriptor (refer to Figure 21), if any.

Figure 21: Security Profile Descriptor

Bytes	O/M[1]	Description
00	M	**Structure ID:** This field shall be set to 5h (i.e., Security; refer to Figure 5).

Figure 21: Security Profile Descriptor

<table>
<tr><th>Bytes</th><th>O/M[1]</th><th>Description</th></tr>
<tr><td>01</td><td>M</td><td>Security Profile Descriptor Index: This field indicates the number of this Security Profile Descriptor in the Security Profile Descriptor List (i.e., the number 1h represents Security Profile Descriptor 1, the number 2h represents Security Profile Descriptor 2, etc.).</td></tr>
<tr><td>03:02</td><td>M</td><td>Security Profile Descriptor Flags: Refer to Figure 22.</td></tr>
<tr><td>04</td><td>M</td><td>Secret Type:
<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>07:02</td><td>Reserved</td></tr>
<tr><td>01</td><td>Redfish Host Interface URI: If set to ’1’, then the Secret Keypath Object Reference is a URI pointing to a Redfish Key Collection Object that contains the PSK. Relies on DMTF DSP 0270 v1.3 or later and DMTF DSP8010 2021.4 or later.

If cleared to ’0’, then the Secret String is not a Redfish Key Collection Object.</td></tr>
<tr><td>00</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>05</td><td></td><td>Reserved</td></tr>
<tr><td>11:06</td><td>O</td><td>Secure Channel Algorithm Heap Object Reference: If the Security Policy List field is set to 01b, then:

a) this field indicates the location and size of a heap object containing a list of secure channel algorithms from byte offset 0h of the NBFT Table Header; and
b) the Offset field and the Length field shall be set to non-zero values.

The list is an array of bytes and the values are defined in the Security Type (SECTYPE) field in the Transport Specific Address Subtype Definition in the NVMe TCP Transport Specification.

If the Security Policy List field is cleared to 0h, then this field is reserved.
<table>
<tr><th>Bytes</th><th>Description</th></tr>
<tr><td>03:00</td><td>Offset: Offset in bytes of the heap object, if any, from byte offset 0h of the NBFT Table Header.</td></tr>
<tr><td>05:04</td><td>Length: Length in bytes of the heap object, if any.</td></tr>
</table></td></tr>
<tr><td>17:12</td><td>O</td><td>Authentication Protocols Heap Object Reference: If the Authentication Policy List field is set to 01b, then:

a) this field indicates the location and size of a heap object containing a list of authentication protocol identifiers (refer to the NVM Express Base Specification) from byte offset 0h of the NBFT Table Header; and
b) the Offset field and the Length field shall be set to non-zero values.

If the Authentication Policy List field is cleared to 0h, then this field is reserved.
<table>
<tr><th>Bytes</th><th>Description</th></tr>
<tr><td>03:00</td><td>Offset: Offset in bytes of the heap object, if any, from byte offset 0h of the NBFT Table Header.</td></tr>
<tr><td>05:04</td><td>Length: Length in bytes of the heap object, if any.</td></tr>
</table></td></tr>
<tr><td>23:18</td><td>O</td><td>Cipher Suite Offset Heap Object Reference: If the Cipher Suites Restricted by Policy bit is set to ’1’, then:

a) this field indicates the location and size of a heap object containing a list of cipher suite identifiers from byte offset 0h of the NBFT Table Header; and
b) the Offset field and the Length field shall be set to non-zero values.

The list, if any, is an array of bytes and the values are defined in the IANA TLS Parameters Registry.

If the Cipher Suites Restricted by Policy bit is cleared to ’0’, then this field is reserved.
<table>
<tr><th>Bytes</th><th>Description</th></tr>
<tr><td>03:00</td><td>Offset: Offset in bytes of the heap object, if any, from byte offset 0h of the NBFT Table Header.</td></tr>
</table></td></tr>
</table>

Figure 21: Security Profile Descriptor

<table>
<tr><th>Bytes</th><th>O/M[1]</th><th>Description</th></tr>
<tr><td></td><td></td><td><table><tr><td>05:04</td><td>Length: Length in bytes of the heap object, if any.</td></tr></table></td></tr>
<tr><td>29:24</td><td>O</td><td>DH Groups Heap Object Reference: If the Authentication DH Groups Restricted by Policy List bit is set to ’1’, then:
a) this field indicates the location and size of a heap object containing a list of DH-HMAC-CHAP Diffie-Hellman (DH) group identifiers (refer to the NVM Express Base Specification) from byte offset 0h of the NBFT Table Header; and
b) the Offset field and the Length field shall be set to non-zero values.
If the Authentication DH Groups Restricted by Policy List bit is cleared to ’0’, then this field is reserved.<table><tr><th>Bytes</th><th>Description</th></tr><tr><td>03:00</td><td>Offset: Offset in bytes of the heap object from byte offset 0h of the NBFT Table Header.</td></tr><tr><td>05:04</td><td>Length: Length in bytes of the heap object.</td></tr></table></td></tr>
<tr><td>35:30</td><td>O</td><td>Secure Hash Functions Offset Heap Object Reference: If the Secure Hash Functions Policy List bit is set to ’1’, then:
a) this field indicates the offset in bytes of a heap object containing a list of DH-HMAC-CHAP hash function identifiers from byte offset 0h of the NBFT Table Header; and
b) the Offset field and the Length field shall be set to non-zero values.
The list is an array of bytes and the values are defined in the NVM Express Base Specification.
If the Secure Hash Functions Policy List bit is cleared to ’0’, then this field is reserved.<table><tr><th>Bytes</th><th>Description</th></tr><tr><td>03:00</td><td>Offset: Offset in bytes of the heap object, if any, from byte offset 0h of the NBFT Table Header.</td></tr><tr><td>05:04</td><td>Length: Length in bytes of the heap object, if any.</td></tr></table></td></tr>
<tr><td>41:36</td><td>O</td><td>Secret Keypath Offset Heap Object Reference: if this field is set to a non-zero value, then:
a) this field indicates the location and size of a heap object containing a URI; and
b) the Offset field and the Length field shall be set to non-zero values.
The type of the URI is specified in the Secret Type field.
If this field is cleared to 0h, then this field is reserved.<table><tr><th>Bytes</th><th>Description</th></tr><tr><td>03:00</td><td>Offset: Offset in bytes of the heap object, if any, from byte offset 0h of the NBFT Table Header.</td></tr><tr><td>05:04</td><td>Length: Length in bytes of the heap object, if any.</td></tr></table></td></tr>
<tr><td>63:42</td><td></td><td>Reserved</td></tr>
<tr><td colspan="3">Notes:
1. O/M definition: O = Optional, M = Mandatory.</td></tr>
</table>

Figure 22: Security Profile Descriptor Flags

Bits	Description
15:12	Reserved
11	**Secure Hash Functions Policy List:** If set to ’1’, then connections shall use one of the secure hash functions in the Secure Hash Functions Policy List is required. If cleared to ’0’, then no Secure Hash Functions Policy List is indicated and use of a secure hash function is not required.
10	**Authentication DH Groups Restricted by Policy List:** If set to ’1’, then connections shall use one of the authentication DH groups in the Authentication DH Groups List is required. If cleared to ’0’, then no Authentication DH Groups List is indicated and use of an authentication DH Group is not required.

Figure 22: Security Profile Descriptor Flags

<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>09</td><td>Cipher Suites Restricted by Policy: If set to ’1’, then the Cipher Suite Offset field and the Ciper Suite Length field indicate a list of supported cipher suites by the driver. If cleared to ’0’, then the Cipher Suite Offset field and the Cipher Suite Length field are reserved.</td></tr>
<tr><td>08:07</td><td>Security Policy List: This field indicates whether secure channel protocols were indicated by policy from driver defaults or administrative configuration.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>00b</td><td>The Offset field and Length field in the Secure Channel Algorithm Heap Object Reference field are reserved.</td></tr>
<tr><td>01b</td><td>The Heap Object specified by the Secure Channel Algorithm Heap Object Reference field indicates a list of authentication protocols used by the driver.</td></tr>
<tr><td>10b</td><td>The Heap Object specified by the Secure Channel Algorithm Heap Object Reference field indicates a list of authentication protocols that were administratively set and used by the driver.</td></tr>
<tr><td>11b</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>06:05</td><td>Secure Channel Negotiation Required: This field indicates whether secure channel negotiation (e.g., TLS) is required.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>00b</td><td>Secure channel negotiation is not supported by the NVM subsystem.</td></tr>
<tr><td>01b</td><td>Secure channel negotiation is supported by the NVM subsystem and is not required.</td></tr>
<tr><td>10b</td><td>Secure channel negotiation is supported by the NVM subsystem and is required.</td></tr>
<tr><td>11b</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>04:03</td><td>Authentication Policy List: This field indicates whether authentication protocols were indicated by policy from driver defaults or administrative configuration.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>00b</td><td>Authentication Protocols Heap Object Reference field Offset and Length are reserved.</td></tr>
<tr><td>01b</td><td>Authentication Protocols Offset field and the Authentication Protocols Length field indicate a list of authentication protocols used by the driver.</td></tr>
<tr><td>10b</td><td>Authentication Protocols Offset field and the Authentication Protocols Length field indicate a list of authentication protocols that were administratively set and used by the driver.</td></tr>
<tr><td>11b</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>02:01</td><td>In-Band Authentication Required:
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>00b</td><td>In-band authentication is not supported by the NVM subsystem.</td></tr>
<tr><td>01b</td><td>In-band authentication is supported by the NVM subsystem and is not required.</td></tr>
<tr><td>10b</td><td>In-band authentication is supported by the NVM subsystem and is required.</td></tr>
<tr><td>11b</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>00</td><td>Descriptor Valid: If set to ’1’, then this descriptor is valid. If cleared to ’0’, then this descriptor is not valid.</td></tr>
</table>

The Secure Channel Algorithm List is an array of bytes, each byte of which contains a value indicating a supported security type. The length of the list is indicated by the Secure Channel Algorithm Length field. Refer to the Security Type (SECTYPE) field in the Transport Specific Address Subtype Definition in the NVMe TCP Transport Specification.

3.1.2.7 Discovery Descriptor

The Discovery Descriptor is an indexed structure that contains all Discovery controllers that have been collected by a pre-OS driver, such as UEFI BIOS, prior to OS execution within Discovery Records. The intent of this table is to allow an OS client to both see pre-OS environment context and to serve OS clients who may not otherwise dynamically discover subsystems but are able to perform basic connections to Discovery controllers as outlined in the NVM Express Base Specification to acquire additional namespaces.

The NumDisc field in the NBFT Header Structure (refer to Figure 8) indicates how many Discovery Records are in the Discovery Descriptor List (refer to Figure 23).

Figure 23: Discovery Descriptor List

Bytes	Description
31:00	**Discovery Descriptor 1:** The first Discovery Descriptor (refer to Figure 24), if any.
63:32	**Discovery Descriptor 2:** The second Discovery Descriptor (refer to Figure 24), if any.
…	
(n*32)-1:(n-1)*32	**Discovery Descriptor n:** The last Discovery Descriptor (refer to Figure 24), if any.

Figure 24: Discovery Descriptor

<table>
<tr><th>Bytes</th><th>O/M[1]</th><th>Description</th></tr>
<tr><td>00</td><td>M</td><td>Structure ID: This field shall be set to 6h (i.e., Discovery Descriptor; refer to Figure 5).</td></tr>
<tr><td>01</td><td>M</td><td>Discovery Descriptor Flags:
<table><tr><th>Bits</th><th>Feature</th></tr><tr><td>07:01</td><td>Reserved</td></tr><tr><td>00</td><td>Descriptor Valid: If set to ’1’, then this descriptor is valid. If cleared to ’0’, then this descriptor is reserved.</td></tr></table></td></tr>
<tr><td>02</td><td>M</td><td>Discovery Descriptor Index: This field indicates the number of this Discovery Descriptor in the Discovery Descriptor List (i.e., the number 1h represents Discovery Descriptor 1, the number 2h represents Discovery Descriptor 2, etc.).</td></tr>
<tr><td>03</td><td>M</td><td>HFI Descriptor Index: This field indicates the value of the HFI Descriptor Index field of the HFI Descriptor associated with this Discovery Descriptor.

If multiple HFIs share a common Discovery controller, there shall be multiple Discovery Descriptor entries with one per HFI.</td></tr>
<tr><td>04</td><td>O</td><td>Security Profile Descriptor Index: This field indicates the value of the Security Profile Descriptor Index field of the Security Descriptor associated with this Discovery Descriptor.</td></tr>
<tr><td>05</td><td></td><td>Reserved</td></tr>
<tr><td>11:06</td><td>M</td><td>Discovery Controller Address Heap Object Reference: This field indicates the location and size of a heap object containing a URI (refer to section 1.5.7) which indicates an NVMe Discovery controller associated with this Discovery Descriptor.

If this field is cleared to 0h, then no URI is specified.
<table><tr><th>Bytes</th><th>Description</th></tr><tr><td>03:00</td><td>Offset: Offset in bytes of the heap object from byte offset 0h of the NBFT Table Header.</td></tr><tr><td>05:04</td><td>Length: Length in bytes of the heap object.</td></tr></table></td></tr>
<tr><td>17:12</td><td>O</td><td>Discovery Controller NQN Heap Object Reference: If set to a non-zero value, this field indicates the location and size of a heap object containing an NVMe Discovery controller NQN. If the NVMe Discovery controller referenced by this record requires secure authentication with a well known Subsystem NQN, this field indicates the unique NQN for that NVMe Discovery controller. This record is involved formatted as an NQN string as defined in the NVM Express Base Specification.

If this field is cleared to 0h, then:
a) this field is reserved; and
b) the OS shall use the well known discovery NQN for this record.
<table><tr><th>Bytes</th><th>Description</th></tr><tr><td>03:00</td><td>Offset: Offset in bytes of the heap object from byte offset 0h of the NBFT Table Header.</td></tr><tr><td>05:04</td><td>Length: Length in bytes of the heap object.</td></tr></table></td></tr>
<tr><td>31:18</td><td></td><td>Reserved</td></tr>
<tr><td colspan="3">Notes:
1. O/M definition: O = Optional, M = Mandatory.</td></tr>
</table>

Annex A. Booting (Informative)

This section introduces booting over NVMe-oF concepts; it also references other boot-from-SAN technologies (e.g., iSCSI and FCOE) as examples, however booting over NVMe-oF architecture is designed to be fabric-agnostic. Section A.1provides the summary of general concepts related to NVMe-oF boot.

A.1 Domain Specific Terms

This domain specific terms section contains terminology that is specific to certain domain topics, such as UEFI that are not generally relevant to the normative sections but are still important for informative content.

A.1.1 SCSI Logical Unit Number (LUN)

A LUN is a SCSI device address of the logical unit on a target device.

A.1.2 UEFI Boot Device

The device handle that corresponds to the device from which the currently executing image was loaded as defined in Unified Extensible Firmware Interface (UEFI) Specification.

A.1.3 UEFI Boot Services

Refer to Unified Extensible Firmware Interface (UEFI) Specification. The collection of interfaces and protocols that are present in the boot environment. The services minimally provide an OS loader with access to platform capabilities required to complete OS boot. Services are also available to drivers and applications that require access to platform capability. Boot services are terminated once the operating system takes control of the platform.

A.1.4 UEFI Device Handle

A handle points to a list of one or more UEFI protocols that are able to respond to requests for services for a given device referred to by the handle. Refer to the Unified Extensible Firmware Interface (UEFI) Specification.

A.1.4.1 Usage of Primary and Secondary HFI fields in the SNSS

The Secondary HFI Associations Heap Object Reference field in the SNSS provides a mechanism to associate more than one HFI for each SNSS record in an NBFT. If a pre-OS driver needs to describe two or more HFIs that represent different physical or logical interfaces with otherwise common connectivity to a common SNSS parameters, then the driver may choose to represent that connectivity by either:

- creating additional SNSS Descriptor records that only utilize a Primary HFI Descriptor Index; or
- utilizing the SNSS Secondary HFI Associations Heap Object Reference.

A driver may choose to create new SNSS Descriptor records per HFI if:

- there are any path, channel, or transport differences; or
- a pre-OS driver does not differentiate between path types.

Figure 25 shows an example of a subset of fields in an NBFT, where a driver creates additional SNSS records associated with the HFI. In this example, there are two HFI Descriptors (HFI Descriptor Index 03h and HFI Descriptor Index 04h) that represent two interfaces on a common Layer 3 TCP/IP subnet. These two interfaces, described by HFI Descriptor Index 03h and 04h can both communicate, with:

- common reachability to different SNSS Descriptors (Index 5 and Index 6); and
- an empty Secondary HFI Associations Heap Object Reference field.

Figure 25: Example NBFT with two SNSS records with two HFIs

Descriptor	Bytes	Field	Value
HRI Descriptor Index 03h			
HFI Descriptor	00	**Structure ID**	03h
	01	**HFI Descriptor Index**	03h
	…		
HFI Transport Info Descriptor – NVMe/TCP	00	**Structure ID**	07h
	01	**Version**	01h
	02	**HFI Transport Type**	05h
	03	**Transport Info Version**	02h
	05:04	**HFI Descriptor Index**	00h 03h
	…		
	35:20	**IP Address**	c0h a8h 01h 03h
	36	**Subnet Mask Prefix**	18h
	52:37	**IP Gateway**	c0h a8h 01h feh
	…		
HFI Descriptor Index 04h			
Host Fabric Interface Descriptor	00	**Structure ID**	03h
	01	**HFI Descriptor Index**	04h
	…		
HFI Transport Info Descriptor – NVMe/TCP	00	**Structure ID**	07h
	01	**Version**	01h
	02	**HFI Transport Type**	05h
	03	**Transport Info Version**	02h
	05:04	**HFI Descriptor Index**	00h 04h
	…		
	35:20	**IP Address**	c0h a8h 01h 04h
	36	**Subnet Mask Prefix**	18h
	52:37	**IP Gateway**	c0h a8h 01h feh
	…		
SNSS Record (Index 5)			
Subsystem Namespace (SNSS) Descriptor	00	**Structure ID**	04h
	02:01	**SNSS Descriptor Index**	00h 05h
	…		
	46	**Primary HFI Descriptor Index**	03h
	53:48	**Secondary HFI Associations Heap Object Reference**	00h
	…		
SNSS Record (Index 6)			
Subsystem Namespace (SNSS) Descriptor	00	**Structure ID**	04h
	02:01	**SNSS Descriptor Index**	00h 06h
	…		
	46	**Primary HFI Descriptor Index**	04h
	53:48	**Secondary HFI Associations Heap Object Reference**	00h
	…		

By utilizing the SNSS Secondary HFI Associations Heap Object Reference, the driver or administrator may optimize space or indicate multipath connectivity. Figure 26 shows this example using a subset of NBFT fields with two HFI Descriptors (Structure ID 1 and Structure ID 2) that represent common Layer 3 TCP/IP subnet interfaces, both with common reachability to the same SNSS Descriptor (Index 1).

Figure 26: Example SNSS with two HFIs

<table>
<tr><th>Descriptor</th><th>Bytes</th><th>Field</th><th>Value</th></tr>
<tr><td colspan="4">HRI Descriptor Index 01h</td></tr>
<tr><td rowspan="3">Host Fabric Interface Descriptor</td><td>00</td><td>Structure ID</td><td>03h</td></tr>
<tr><td>01</td><td>HFI Descriptor Index</td><td>01h</td></tr>
<tr><td>…</td><td></td><td></td></tr>
<tr><td rowspan="10">HFI Transport Info Descriptor – NVMe/TCP</td><td>00</td><td>Structure ID</td><td>07h</td></tr>
<tr><td>01</td><td>Version</td><td>01h</td></tr>
<tr><td>02</td><td>HFI Transport Type</td><td>05h</td></tr>
<tr><td>03</td><td>Transport Info Version</td><td>02h</td></tr>
<tr><td>05:04</td><td>HFI Descriptor Index</td><td>00h 01h</td></tr>
<tr><td>…</td><td></td><td></td></tr>
<tr><td>35:20</td><td>IP Address</td><td>c0h a8h 01h 01h</td></tr>
<tr><td>36</td><td>Subnet Mask Prefix</td><td>18h</td></tr>
<tr><td>52:37</td><td>IP Gateway</td><td>c0h a8h 01h feh</td></tr>
<tr><td>…</td><td></td><td></td></tr>
<tr><td colspan="4">HFI Descriptor Index 02h</td></tr>
<tr><td rowspan="3">Host Fabric Interface Descriptor</td><td>00</td><td>Structure ID</td><td>03h</td></tr>
<tr><td>01</td><td>HFI Descriptor Index</td><td>02h</td></tr>
<tr><td>…</td><td></td><td></td></tr>
<tr><td rowspan="10">HFI Transport Info Descriptor – NVMe/TCP</td><td>00</td><td>Structure ID</td><td>07h</td></tr>
<tr><td>01</td><td>Version</td><td>01h</td></tr>
<tr><td>02</td><td>HFI Transport Type</td><td>05h</td></tr>
<tr><td>03</td><td>Transport Info Version</td><td>02h</td></tr>
<tr><td>05:04</td><td>HFI Descriptor Index</td><td>00h 02h</td></tr>
<tr><td>…</td><td></td><td></td></tr>
<tr><td>35:20</td><td>IP Address</td><td>c0h a8h 01h 02h</td></tr>
<tr><td>36</td><td>Subnet Mask Prefix</td><td>18h</td></tr>
<tr><td>52:37</td><td>IP Gateway</td><td>c0h a8h 01h feh</td></tr>
<tr><td>…</td><td></td><td></td></tr>
<tr><td colspan="4">SNSS Record (Index 1)</td></tr>
<tr><td rowspan="5">Subsystem Namespace (SNSS) Descriptor</td><td>00</td><td>Structure ID</td><td>04h</td></tr>
<tr><td>02:01</td><td>SNSS Descriptor Index</td><td>00h 01h</td></tr>
<tr><td>…</td><td></td><td></td></tr>
<tr><td>46</td><td>Primary HFI Descriptor Index</td><td>01h</td></tr>
<tr><td>48:53</td><td>Secondary HFI Associations Heap Object Reference</td><td><table><tr><th>Bytes</th><th>Field</th><th>Value</th></tr><tr><td>03:00</td><td>Offset</td><td>400h</td></tr><tr><td>05:04</td><td>Length</td><td>01h</td></tr></table></td></tr>
<tr><td>…</td><td></td><td></td></tr>
<tr><td>Secondary HFI Associations Heap Object [Offset 400h]</td><td>03:00</td><td>n/a</td><td>02h</td></tr>
</table>

A.1.5 UEFI Device Path

A variable-length binary data structure that is composed of variable-length generic device path nodes and is used to define the programmatic path to a logical or physical device. Refer to the Unified Extensible Firmware Interface (UEFI) Specification.

A.1.6 UEFI Device Path Protocol

A protocol used during boot services to provide the information required to construct and manage Device Paths. Also called the EFI Device Path Protocol. Refer to the Unified Extensible Firmware Interface (UEFI) Specification.

A.1.7 UEFI Protocol

A UEFI Protocol is a firmware interface between UEFI applications and the set of services offered by the UEFI firmware. Protocols serve to enable communication between separately built modules including drivers. A UEFI Protocol provides a block of function pointers and data, named by a GUID and associated with a UEFI handle. Refer to the EDK II UEFI Driver Writer's Guide.

A.2 Boot from SAN Ontology and Concepts

"Boot from SAN" is a term typically used to refer to the capability to boot from SCSI-based Block Storage where the technology of the "SCSI bus" is a network (commonly referred to as a "Storage Area Network" or "SAN"). The storage area network is typically an FC or IP-based network. The storage devices are attached to the network and not to the booting system (e.g., local storage with IDE/SATA). SANs also support multiple access paths via the fabric, through multiple SCSI host adapters and multiple SCSI targets, to the same SCSI block storage devices. This allows the booting system to support redundant and parallel connections to storage devices.

A.2.1 Boot from Block Storage Device

The boot environment traditionally begins by the system legacy BIOS or system firmware (e.g., UEFI) enumerating host adapter hardware and invoking PCIe Expansion ROM images which initialize and configure the host adapter. The Expansion ROM images on storage adapters detect and enumerate supported block storage devices as part of the Expansion ROM's initialization.

A system boot device list is configured with a small number of devices that the system attempts to boot from. The boot firmware selects the first boot device in the list and attempt to read from the device to load the OS boot image into memory. Upon a failure, the boot firmware transitions to the next boot device in the list and repeats the OS load attempt. The process repeats until the OS is successfully loaded via a device in the list. At such time, any remaining devices in the list are ignored. A failure to successfully load a device typically results in the administratively configured behavior for any boot device failure (i.e., a reboot) or a halt at prompt.

In most cases, an administratively managed list of boot devices is used to set the boot order of the system. For example, a prioritized list may start with CD/DVD device, with second USB device, followed by hard disk drive, where the transition between devices is expected to occur based on non-presence of leading devices.

In cases where the same block storage entity may be accessible via multiple host adapters, there are multiple block storage devices enumerated to the boot environment which all map to the same device. In this scenario, the boot device list may be used as a multi-path selection sequence to allow system booting, even if a host adapter fails or loses connectivity to storage.

Simplistic host adapters such as and IDE or SATA interface typically enumerate 1 or 2 storage entities based on their physical connectivity to a storage device. More complex host adapters may use the adapter's Expansion ROM image to configure the host adapter with options and settings that allow it to abstract physical connectivity. For example, an IDE or SATA host adapter may support multiple physically attached drives to be abstracted into a single block storage device presented to the boot environment. The host adapter may utilize the physical devices in one of the following manners:

- concatenated to appear as a higher capacity device;
- perform RAID 0 for striping for capacity and performance;
- perform RAID 1 for mirroring for fault-tolerance; or

- some other RAID type.

In general, each time a system boots, simplistic host adapters enumerate the same block storage devices to the boot environment as the storage devices are physically connected to the adapter. Only physical removal of the devices results in a change to the enumerated list. Given the fixed physical connectivity, the devices detected and enumerated in the boot environment are the same devices automatically detected and enumerated in the OS environment.

A.2.2 Boot from SCSI-based Block Storage

SCSI host storage adapters follow the basic behaviors described for Boot from Block Storage Devices. SCSI host adapters are typically attached to “a bus” which is able to connect to multiple SCSI storage devices. The technology of the “bus” may vary e.g., parallel SCSI, SAS, Fibre Channel Arbitrated Loop, Fibre Channel fabric, or Ethernet/IP. The storage adapter uses methods specific to the technology of the bus to detect SCSI targets, and the SCSI command set to scan for SCSI Logical Unit Numbers (LUN), detect block storage devices, and enumerate the block storage devices with the boot environment. The storage adapter converts the basic block storage device interface (i.e., select/read/write/deselect) used by the boot environment to the SCSI protocol commands for use with the SCSI LUNs.

The introduction of bus topologies loosens the physical connectivity of devices to the host adapter. A SCSI backplane with bus connections for SCSI target devices allow for dynamic device removal and insertion, resulting in possible dynamic device enumeration and changes in device enumeration order. Network topologies using switches and routers allow the storage to become even more “remote” and possibly much more dynamic in terms of addition, removal, and detection order.

Many SCSI bus topologies support connectivity to large numbers of storage devices. A single SCSI target may support many SCSI LUNs, each LUN being a block storage device. Enumerating all devices may overwhelm the boot environment. Over time SCSI adapters have expanded their Expansion ROM images to support administrative configuration which selectively identifies the SCSI target and SCSI LUN’s that are to be enumerated with the boot environment. The adapter typically saves this configuration in non-volatile store on the adapter. This change has resulted in faster boot enumeration (i.e., fewer targets/LUNs connect to and/or enumerated) and better manageability (i.e., fewer devices enumerated to the boot environment and with more consistent order).

As most SCSI topologies are self-discovering, when the OS environment re-initializes and rescans for SCSI devices, the OS environment rediscovers the same storage devices enumerated with the boot environment as those storage devices are a subset of the overall devices that are discovered by the OS (barring dynamic changes in connectivity). There is no required interaction between the boot environment and the OS to automatically re-establish access to the boot device.

A.2.3 Boot from SAN: Fibre Channel Storage

Boot from Fibre Channel (FC) storage is an example of Boot from SCSI-based Block Storage. Most references to Boot from SAN refer to the administrative settings made per host adapter to identify the FC-based SCSI target devices and the SCSI LUN numbers to enumerate with the boot environment. The FC SCSI host adapters typically perform device and LUN discovery, allow the administrator to select the block storage devices to be presented to the boot environment, and save the configuration on the adapter.

FC storage is self-discovering. The host adapter typically sees the same devices regardless of whether it is operating in the boot environment or the OS environment. The host adapter expansion ROM may filter those presented to the boot environment, however the devices presented to the OS environment are typically all devices, including those presented to the boot environment. Thus, device presentation for boot

devices occurs in both the boot environment and the OS environment without requiring the two environments to interact.

A.2.4 Boot from SAN: iSCSI Storage

Boot from iSCSI storage is another example of booting from SCSI-based block storage. Most references to Boot from SAN refer to the administrative settings made per host adapter to specify iSCSI session information for the iSCSI target(s) and SCSI LUN number for the block storage devices to enumerate with the boot environment. The settings may also include IP addresses for discovery services to help locate the iSCSI target device.

In contrast to FC storage, iSCSI storage is not self-discovering. The host adapter, or iSCSI stack utilizing a network adapter, requires information to locate and/or create the iSCSI session. Information includes IP addresses and port numbers for the initial TCP connection, iSCSI names to use for the host initiator, and iSCSI names for the iSCSI target to connect to. To aid in the amount of information that is required, IP services such as DHCP, iSNS, or SLP may be used to query for the iSCSI boot device session information. Even in those cases, address information and service type information may have to be specified. Information on such services is found in IETF RFC 4173.

As iSCSI storage is not self-discovering, when the OS environment is transitioned to, without additional aid, the OS environment does not know that it should establish an iSCSI session nor what the configuration information is for that session. As such, OS environments have to be prepared by an administrator with the same connectivity information that was in the boot environment. Not only does this require a priori knowledge and configuration of both environments, but it also presents situations where the two environments may become out of sync.

To resolve these issues, the industry created the iSCSI Boot Firmware Table (iBFT). The iBFT is a data structure that is created by the boot environment and passed to the OS environment using the ACPI API. The table contains the iSCSI boot session information that was used by the boot environment so that the OS environment realizes that it is to connect to iSCSI storage and to use the session information in the table for the connectivity.

A.2.5 Boot from NVMe-oF

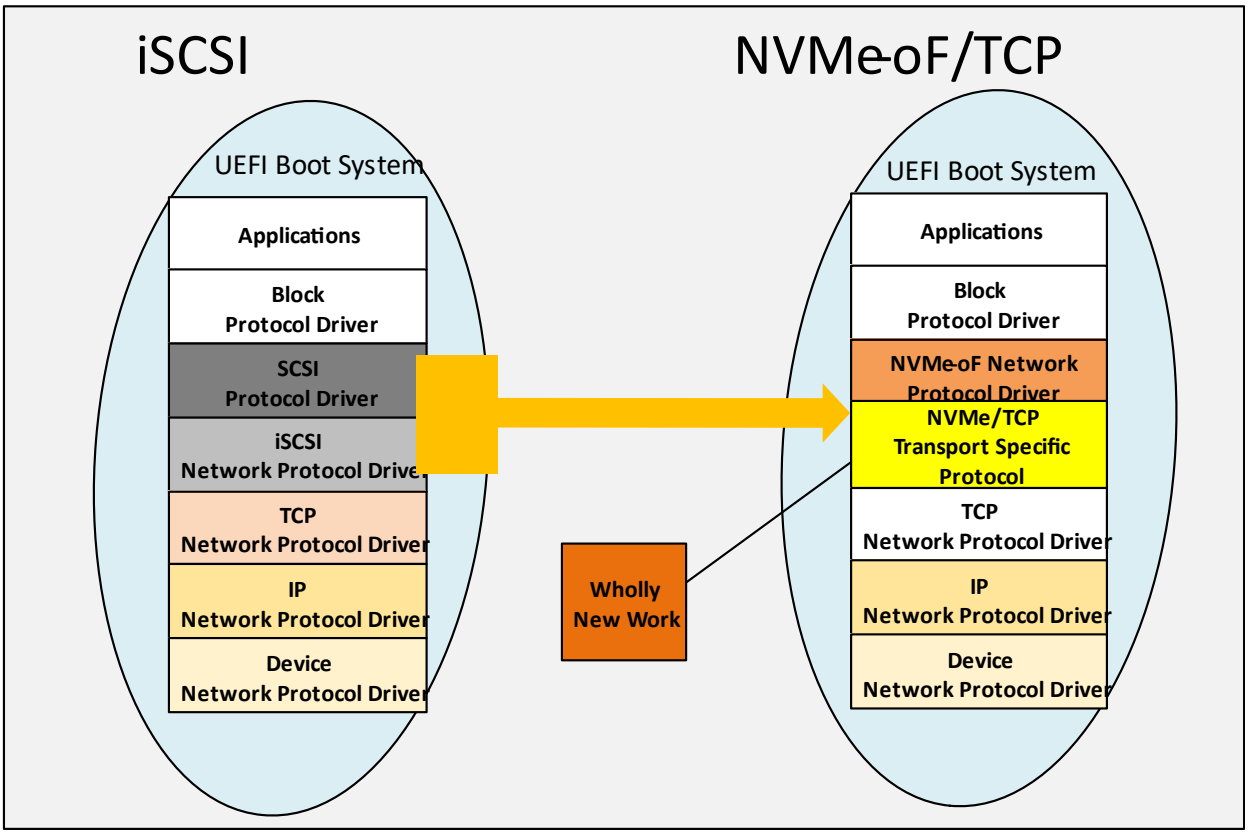
NVMe-oF boot addresses the technology requirements of remote booting from NVMe namespaces. It is similar to other remote block storage boot technologies (i.e., main concepts such as boot flow and handover mechanism are similar).

NVMe-oF provides:

- fabric agnostic capability supporting all NVMe-oF fabric transports;
- lower latency due to elimination of SCSI command translation via direct NVMe command execution; and
- increased scalability and flexibility through the NVMe-oF defined discovery mechanism.

Figure 27 outlines a hypothetical example of UEFI driver stack differences between iSCSI and NVMe-oF (using TCP as transport).

Figure 27: UEFI Network stack differences between booting from iSCSI and booting from NVMe/TCP



A.2.6 Configure Boot from NVMe-oF

Before booting from NVMe-oF, administrative configuration of the UEFI pre-OS driver is required. As an example, an administrator would configure a UEFI system through UEFI HII (Human Interface Infrastructure) to include the required information such as target Subsystem NQN, target IP address, target port number, target namespace, host NQN, security related information. How this information is stored by UEFI firmware is vendor specific, but the information is presented to the OS using an ACPI table named NBFT (NVMe Boot Firmware Table) when OS boot is taking place.

A.2.7 Boot Process

The boot process is separated as two parts: pre-OS boot and OS boot.

A.2.7.1 Pre-OS Boot

Continuing with an UEFI based example of a pre-OS environment; The seven phases in a UEFI boot sequence (refer to Figure 28 and the Tianocore: EDK2 Build Specification) are:

1. Security (SEC)
2. Pre-EFI Initialization (PEI)
3. Drive Execution Environment (DXE)
4. Boot Device Selection (BDS)
5. Transient System Load (TSL)

6. Runtime (RT)
7. After Life (AL)

The DXE driver supporting NVMe-oF boot is loaded and executed in the DXE phase. Namespaces used as boot devices are discovered using the information stored during configuration, the driver connects to the NVMe-oF subsystems, discovers the namespaces (i.e., boot devices) required for OS boot. Note that the pre-OS environment may not require persistent connections to Discovery controllers as it not intended to fully hand over connections to an OS. These namespaces are then be selected at the BDS phase as the final boot device for OS boot.

In the TSL phase, the operating system boot loader is loaded from the boot device and the UEFI firmware hands over the execution to that specific boot loader which continues the operating system boot. By this point, the NBFT has been generated either by UEFI firmware, optional drivers, or an intermediate TSL application; and stored in the main memory which is accessed by the OS as an ACPI table after UEFI ExitBootServices has been called, ending the TSL phase.

From a NVMe-oF host to controller connection perspective, a pre-OS environment may not require or utilize persistent connections to all controllers. Some pre-OS environments are not multi-threaded and may see a high burden of maintaining multiple concurrent connections.

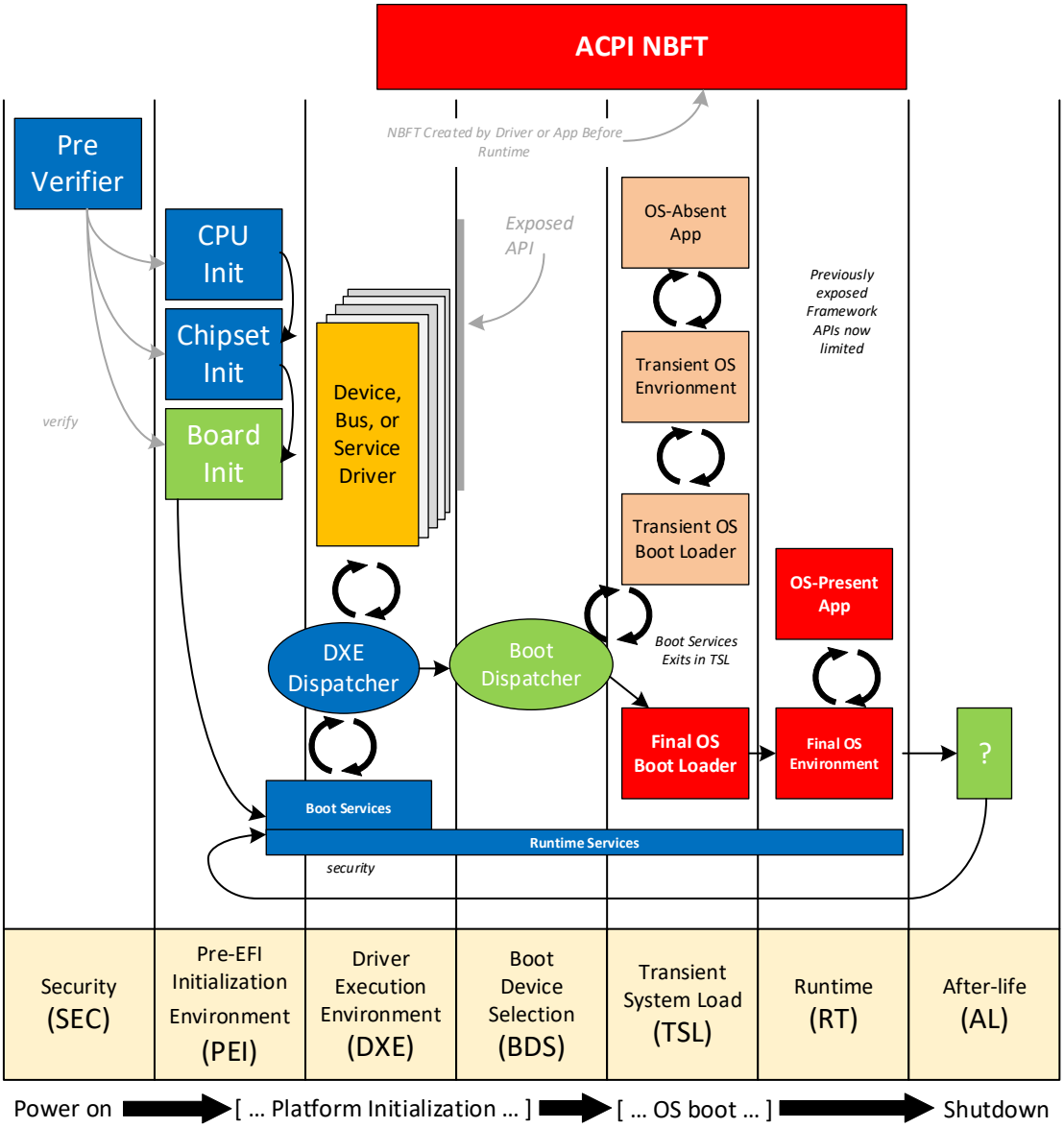
A.2.7.2 OS Boot

Typically, an operating system does not expect to rely on or reuse the pre-OS connections and connection specific data structures. Because of this, at the operating system boot stage, the OS may then use the information stored in NBFT to connect to the NVM subsystem without user intervention.

There are two scenarios in the OS boot stage: operating system installation or normal operating system boot.

1. During OS installation, user may either:
    a) use the NBFT provided host NQN as its own host NQN; or
    b) set a separate host NQN if the NVM subsystem has been configured to support multiple host NQNs.
2. In normal operating system boot, as the OS may have its own mechanism to persist the required information for restore the NVMe-oF connections. The OS may either continue using the NBFT or the OS specific mechanism.

Figure 28: Modern EDK2 UEFI Boot Flow



A.3 Transport Specific and interface specific semantics

This section is intended to provide informative guidance on transport and boot mechanism specific semantics. This section includes discussions of the ACPI NBFT and UEFI semantics required to adopt UEFI or UEFI and ACPI specific boot workflows. Because every platform is different, these sections provide generalizations about how to interpret the handoff between pre-OS to OS environments.

A.3.1 ACPI NBFT Discussion and Background

The NBFT contains parameters used to connect to an NVMe-oF storage device, specifically NVMe/TCP. The NBFT allows the pre-OS boot environment and OS environment to use the same parameters. This is

important when booting from an NVMe Storage Device. The NBFT parameters are typically set in the pre-boot environment or an external management tool.

The NBFT parameters can be divided into three areas: host, fabric and storage subsystem. Host parameters are used to uniquely identify the host to the storage. These parameters are typically used for access control lists and reservations. These parameters contain the Host NQN and Host ID.

Host Fabric Interface (HFI) parameters describe the underlying connection between the host and the storage. These parameters contain the transport type (e.g., Ethernet, FC, InfiniBand), transport address, keep alive settings, etc.

Storage subsystem parameters uniquely identify the NVMe Storage Device. These parameters contain the storage NQN, controller ID, NID/Namespace ID. There may be several NVMe Storage Devices connected to the fabric. The storage subsystem parameters are used to pick a specific storage device.

The minimal set of NBFT parameters required to connect to an NVMe namespace are listed below.

A.3.1.1 Required and Information for a NVMe-oF Host

Under NVMe/TCP the following fields are required:

Figure 29: Required NVMe/TCP host NBFT information with example data

Descriptor	Field	Example
Host Descriptor	Host ID	00h 11h 22h 33h 44h 55h 66h 77h 88h 99h aah bbh cch ddh eeh ffh
Host Descriptor	Host NQN	“nqn.2014-08.com.example:nvme.host.sys.xyz”
HFI Descriptor	Transport type	03h
HFI Descriptor	Host Transport Address	192.168.1.1
SSNS	Subsystem Transport Address	192.168.1.2
SSNS	Transport Service ID	“4420”
SSNS	Subsystem NQN	“nqn.2014-08.com.example:nvme.storage.xyz”
SSNS	Controller ID	FFFFh
SSNS	Namespace Identifier Type (NIDT)	03h
SSNS	Namespace Identifier (NID)	00h 11h 22h 33h 44h 55h 16h 77h 88h 99h aah bbh cch ddh eeh ffh

A.3.1.2 Invalid HostID Values

Administrative care should be taken when choosing HostID values due to interoperability concerns with host pre-OS and OS driver versions which may be separately compliant with any revision of the NVM Express Base Specification. Host OS drivers should pay attention to the flag for primary administrative identity as defined in the Primary Administrative Host Descriptor field of host flags in Figure 9 to determine if an administratively assigned value or a default value assumed by pre-OS.

A.3.1.3 Multiple NBFTs

A.3.1.3.1 Coexistance of multiple NBFT instances

NVMe-oF boot requires that the pre-OS environment creates one NBFT ACPI table per driver, which contains information on network devices that have successfully established a connection to NVMe-oF controllers and information on the connections and NVMe-oF controllers reached. Those devices may provide different level of support for NVMe-oF network stack and may be managed by different SW drivers. A situation is possible where those drivers create two unrelated NVMe-oF stacks that co-exist on a single

platform, each providing a piece of data necessary to build a complete NBFT. The design choise to allow for the coexistence of multiple NBFTs is due to the complexity for disparate ecosystem components to share such a structure in the pre-OS environment. Shared use of a single NBFT requires considerations such as:

- simultaneous read/write access to NBFT from NVMe-oF stacks;
- support for different versions of NBFT substructures by NVMe-oF stacks; and
- UEFI context: exchange and synchronization of the TableKey parameter used for adding and removing ACPI tables via EFI_ACPI_TABLE_PROTOCOL.

To prevent issues such as these from occurring, each NVMe-oF stack implementation is advised to build and install its own NBFT table. Operating systems additionally should consider analysis of multiple NBFTs and extract necessary information from them.

For infrastructure or configuration specifics that may lead multiple drivers to establish connection to the same namespace over the same HFI (e.g., two NVMe/TCP drivers), it is possible for two or more NBFT instances to contain identical entries in NBFT substructures. While such redundant information may be used or ignored by OS, such handling is out of scope for this document.

A.3.1.3.2 OS and Post Boot handling of multiple NBFTs

If multiple NBFT tables are present, only one NBFT should be administratively selected. There is no enforcement mechanism for this to be coordinated between multiple NBFT tables, however, it is not recommended to be selected in more than one NBFT. Because an OS may encounter multiple NBFTs that assert primary administrative identity as defined in the Primary Administrative Host Descriptor field of host flags in Figure 9, the OS may correlate the Boot partition to a NBFT in order to adopt a Host NQN and HostID for its own purposes. Usually this is handled by comparing the UEFI boot variable ‘BootCurrent’ with the NBFT it is associated with, and then use the Host NQN and HostID from the host descriptor.

A.3.1.3.3 Multiple Drivers with Different Versions creating SSNS or HFI structures

Due to the complexities of a pre-OS environment that results from multiple pre-OS drivers of different levels to simultaneously exist, the possibility exists that an OS environment may encounter scenarios with multiple NBFT structures referencing the same HFI devices or SSNS endpoint. Therefore, an OS environment that interacts with multiple NBFT’s requires careful handling of potentially common or conflicting entries.

A.3.1.3.4 NBFT Header Driver Signature

The Header NBFT Driver Signature provides a mechanic for an OS driver to understand the identity of the driver that filled in a specific NBFT. These device path signatures are string representations of the issuing drivers device path.

Examples of such are:

NvmExpressDxe driver integrated into UEFI FW image on Flash:

“Fv(7CB8BDC9-F8EB-4F34-AAEA-3EE4AF6516A1)/FvFile(5BE3BDF4-53CF-46A3-A6A9-73C34A6E5EE3)”

Driver coming from OptionROM:

“PciRoot(0x0)/Pci(0x10,0x0)/Pci(0x0,0x1)/Offset(0x13800,0x31DFF)”

A.3.1.3.5 Impact of NBFT updates

In a pre-OS environment, as a driver is assembling configuration data and context for the creation of a NBFT, the driver may choose to build the structures of the NBFT in stages. As there are cross references, namely between index values for Security Profile Descriptors or Discovery Descriptors, care should be taken to ensure any creation, deletion or update of an index is done atomically.

As an example, pre-generating SSNS records before discovery is complete may result in a namespace that is associated with namespaces discovered across multiple HFIs. To handle this, a driver may determine it is required to update an existing SSNS heap object to amend a new HFI. If the driver has pre-allocated the SSNS heap object, this may result in fragmentation of the heap. An approach for drivers that want to avoid fragmentation may consider delaying allocation of the heap object until all HFI and subsequent controller discovery is completed and then come back and re-fill indices.

A.3.2 Implementation considerations

This section contains high level overview of common use cases or problems in consistent implementation and outlines steps to answer specific questions.

A.3.2.1 Host Connection Information in the absence of a Discovery Controller

If a user specifies Discovery controller(s) as the connection target, the Discovery Controller Structure should contain the Discovery controller(s) data input from the HII interface.

If a user selects the auto discovery option in the HII interface, when implemented in the HII interface, the Discovery Controller Structure should contain the Discovery controller(s) discovered.

However, if a user specifies an NVM subsystem as direct connect in the HII interface, the value in Control Structure Discovery Descriptor Offset field is required to be cleared to 0h.

If a host implementation supports a user configuration interface (i.e., UEFI HII), to allow administrator's input of NVM subsystem information (i.e., NVM subsystem qualified name, transport service identifier, port id, controller id, address family, transport type, transport address, maximum size of an admin submission queue), a host is not required to query Discovery Log Pages from the Discovery controller to obtain NVM subsystem information. For this implementation, a Discovery Descriptor is not present with

- the Discovery Offset in the Control Structure cleared to 0h; and
- the Primary Discovery Ctrl Offset field of the Subsystem and Namespace Descriptor cleared to 0h,

to indicate no Discovery controller is utilized. An example of this is shown in Figure 30.

Figure 30: Example Subsystem and Namespace descriptor

Bytes	Description	Example
03:00	**Structure ID:** 4 = Namespace	4h
05:04	**Index:** Index = 1 indicates namespace 1 … Index = n indicates namespace n	
07:06	**SSNS Flags:** 1 = Namespace Structure Valid	1h
08	**Transport Type:** 1 = NVMe/TCP	1h
09	**Primary Discovery Ctrl Offset:** 0 = No Discovery Descriptor reference.	0h
127:10	Reserved	

A.3.2.2 Redfish Keypath Object

The Security Profile Descriptor Keypath URI Object (refer to Figure 21) allows the NBFT to reference the relative URI of a well-formed Redfish™ Key Collection Object, defined in DMTF DSP8010 or later. It is assumed that the base-address of the Redfish URI is obtained through available methods to the platform or OS environment, such as DMTF DSP0270, and is outside the scope of this document.

The following is an example of such a formatted Redfish relative URI:

/redfish/v1/KeyService/NVMeoFSecrets/0

Where '0' in the Redfish® URI is the Id value of an NVMeoF Key Service Secrets Collection.

The object referenced by this Keypath includes the following important pieces of information:

Id, KeyString, NQN, and HostKeyId

Where:

- Id is again the value of the current record;
- KeyString is the key material for the belonging to Id;
- NQN is the host or Subsystem NQN depending on both HostKeyId value being present and the NQN. If the current record is not a Host NQN, Additionally the SubsystemNQN of the related Discovery or SSNS record should match this value; and
- HostKeyId is the Id value of the record that corresponds to the Host NQN associated with the current record. Its presence indicates the current record is not a Host NQN, but the record should be addressable by a relative local URI with the value of HostKeyId as the Id.

For detailed examples, please see DMTF DSP8010 mockups of NVMeoFSecrets objects.

A.3.2.3 NID decoding by drivers or environments

In some boot environments, it may be possible for administrative and configuration interfaces to digest either binary or textual representations of a NID. Without outside context an undecorated NID of type 2 (i.e., NGUID) or type 3 (i.e., UUID) may be indistinguishable for many valid values of both NID types. Because of this, a driver or environment should avoid explicitly attempting to determine if an undecorated value is one or the other without providing positive selection by a user to ensure proper NID Type attribution.

For example, the following undecorated value is both a valid NGUID and a valid UUID:

- The string "4eff7f8ed3534e9ba4ecdeea8eab84d7"

However, the following decorated string is well understood to be a UUID type NID:

- The string "urn:uuid:4eff7f8e-d353-4e9b-a4ec-deea8eab84d7"

When at all possible, it is recommended to include mechanisms to select NID type in administrative interfaces when decoration is unable to be enforced.

A.4 OS Specific Details and Application Considerations

This OS specific section details how individual OS handle namespace information handed over from pre-boot environment (e.g., virtual machines).

A.4.1 Generic OS Usage of the NBFT

A.4.1.1 Accessing NBFT in operating system

During system boot process, when the execution is handed over to the operating system, all the information, with the exception of security credentials, required for NVMe-oF boot has been presented in the form of a NBFT as an ACPI table, hence the NBFT may be accessed by the OS using OS provided ACPI API during OS boot. Security credentials may be accessed via the Security Descriptors Secret Keypath Offset Heap Object Reference (refer to Figure 21).

For further accessing of a NBFT, an OS exports an NBFT in the OS specific method (e.g., sys interface in Linux, VSI interface in VMware ESXi). However, if an NBFT is accessed as a generic ACPI table using an ACPI API, the NBFT content may be just a stream of bytes without any specific meaning. To make NBFT access more meaningful from an NVMe-oF boot point of view, the OS may implement an NBFT specific interface (e.g., API, UI, tools) to make NBFT access more convenient and user-friendly. Subsequently, other components in the OS may access NBFT without using ACPI API as the ACPI API may not be generally available to all the components.

A.4.1.2 Consuming NBFT

The information contained in an NBFT may be consumed for multiple uses, including:

- using the connect parameters to connect to NVMe subsystem on which the OS is installed, including security;
- using the connect parameters to connect/discover other NVMe subsystems; or
- using the parameters to configure the OS itself (e.g., host NQN, host ID).

The NBFT may be consumed at different stages of the OS lifecycle, including, but not limited to:

a. during system boot: The OS is able to detect if there is NBFT available and use the parameters in NBFT to connect to the NVMe-oF subsystem and present NVMe namespaces to the OS for further use; or
b. during OS installation: The OS installer is able to detect the presence of an NBFT and determine that there is at least one NVMe namespace present in the system that supports NVMe-oF boot. The administrator may then choose installing the OS on an NVMe namespace that supports NVMe-oF boot.

The OS should assume the NBFT is recreated from pre-OS context at every boot and that the NBFT is not guaranteed to remain static. Pre-OS context may change between reboots due to platform configuration or administrative changes. Therefore the OS should be tolerant of changes between reboots.

A.4.2 Linux

A.4.2.1 Linux Boot Loader

On entry, a Linux boot loader (e.g., shimx64.efi, grubx64.efi) may assume an association exists to the namespace from which it was loaded. But the boot loader is not required to be aware that it is booting from an NVMe-oF namespace or that an NBFT exists. That is, no NVMe-oF specific changes are required to an EFI boot loader, such as grubx64.efi. The boot loader may continue to use the EFI block device protocol to:

- use NVMe-oF associations made by the UEFI firmware;
- access the ESP;

- find any other partitions;
- locate systems files that contain boot loader configuration;
- locate and load the Linux kernel; and
- load the initrd/initramfs files.

The files that are used by a Linux EFI boot loader are typically located in “/boot” or the ESP. If “/boot” is located in a different NVMe-oF namespace than the ESP, that namespace must have been previously administratively configured such that required association is created during the UEFI DXE phase.

Note that there is no EFI protocol or mechanism by which an EFI boot loader creates additional NVMe-oF associations for the current boot.

A.4.2.2 Linux Kernel Boot

The EFI ExitBootServices() call is the transition from the TSL phase (e.g., EFI grub2) to the OS environment. Typically, this call is made very early in the boot flow of the Linux kernel. On return from the EFI ExitBootServices() call, any NVMe-oF associations created during the UEFI pre-OS environment are closed. Therefore, if continuation of the OS boot requires access to any NVMe-oF namespaces, the booting Linux system must recreate these associations using the Linux kernel drivers and utilities. The information in the NBFTs provided by the UEFI pre-OS environment may be used by the Linux OS to create the required boot-time NVMe-oF associations (e.g., “/”, “/boot”) that are required to continue the Linux OS boot.

Note that the presence of the NBFT does not preclude the use of OS-specific configuration files to recreate or to create new associations to any additional NVMe-oF namespaces that the OS requires. If those configuration files are somehow passed as kernel boot parameters or are located in “/boot”.

Figure 31 summarizes common Linux pre-OS associations that might utilize NBFT supported transports for NVMe-oF namespaces and possible expectations by native Linux OS boot software. When such a pre-OS association is created, parameters used to form that association are present in the NBFT.

Figure 31: Common Linux boot namespaces and expectations on NBFT content

Use of NVMe-oF Namespaces	Use of pre-OS association	Use of parameters in NBFT
ESP	Required by the boot loader[1]	May be used by OS
“/boot”	Required by the boot loader[2]	Typically used by OS
“/”	Optional for the boot loader[2]	Typically used by OS
other file system mounts, if any	OS configuration dependent[2]	OS configuration dependent
Notes: 1. Established and required by UEFI firmware. 2. If different namespace than ESP, must be administratively configured.		

A.4.2.3 NBFT access and availability

Linux makes ACPI tables available to userspace as binary files via sysfs. Since the NBFT is a standard ACPI table, if present, the sysfs representation of the NBFT is named:

/sys/firmware/acpi/tables/NBFT

If multiple NBFTs are present, then the first one has the instance number “1” appended to its name. Any additional NBFTs have ascending instance numbers, e.g.,:

/sys/firmware/acpi/tables/NBFT1

/sys/firmware/acpi/tables/NBFT2

/sys/firmware/acpi/tables/NBFT3

…

Software that reads the NBFT should check for and handle the possibility that multiple ACPI NBFT tables are present.

A.4.2.4 Requirements for EFI boot variables for NVMe-oF namespaces

The creation or modification of an EFI boot variable with a value that includes an NVMe-oF device path for an NVMe-oF namespace does not automatically trigger the update the NBFT for the next boot.

Such a boot variable is usable on the next boot, if and only if either:

1. Corresponding information about that NVMe-oF namespace is already in the NBFT; or
2. Prior to the next boot, corresponding information about that NVMe-oF namespace is administratively configured (i.e., via a RedFish API, or manually configured via EFI HII).

This has implications for an OS installer which boots from a non NVMe-oF device, e.g., network boot, and is installing the OS to an NVMe-oF namespace that was configured using OS installer provided inputs or OS-based configuration files. A Linux OS installer typically uses the efibootmgr utility to create the required EFI boot variable for the new OS installation.