NVMe:

Beyond the Basics

Non-Volatile Memory Express

Including Revision 1.2.1

NVMe Management Interface
NVMe Over Fabrics

NVMe:
Beyond the Basics

Non-Volatile Memory Express

Authored by
Hugh Curley

Published by
KnowledgeTek

Many of the terms used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designators appear in this book, and the author or publisher were aware of the trademark claim, the designations have been marked with a trademark sign ™. The author and publisher make no claim that every trademarked term is so noted. Only the first occurrence of trademark names is indicated. If you find a trademarked term in this book that is unmarked, please notify the author or publisher (addresses below) and it will be included in the next printing.

Check with your legal counsel regarding trademarks and patents before using the material in this text or in the NVMe or PCIe Specifications. The author and publisher make no claims about the existence, or lack of existence, of any trademarks or patents, other than stated in the text.

The author and publisher have taken care in preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information contained herein. Non-Volatile Memory Express (NVMe) is a work in progress; hence not all issues have been discovered and the ones discovered may not have been resolved. This book attempts to report the current state of NVMe.

This book is intended for training to get engineers familiar with NVMe and PCIe so they can better understand the official documentation. Hence, there are many simplifications in this book and many cases not covered.

ISBN: 987-0-9661008-4-6

31.K.19

Cover Design: Cherie Helms

For more information, please contact:
Hugh Curley via e-mail at Hugh@HughCurley.com
970-344-6053

KnowledgeTek, Inc.
8690 Wolff Court, Suite 110
Westminster, Co 80031
www.knowledgetek.com
303-465-1800

Dedicated to the students of

KnowledgeTek's

NVMe classes,

who make creating this book worthwhile

Audience

This book on Non-Volatile Memory Express (NVMe) is intended for three different audiences.

1. For people who need only an overview of NVMe and for those who need an overview before delving into NVMe more deeply, read all of the first chapter and the first subsection of the subsequent chapters.

2. For people who need the more in-depth study of NVMe, read the first subsection of each chapter and study the chapters of interest in their entirety. Tables and charts from the specifications or standards are shown in abbreviated form in this book; the complete forms are in the companion Reference Manual available from KnowledgeTek at www.knowledgeTek.com.

3. For people who need a reference only, all the tables and charts mentioned in the individual chapters are reproduced in the reference section in a companion book available from KnowledgeTek at www.knowledgeTek.com.

This text is meant to be a learning document. Rather than placing some information in only one area, and referring to it when necessary, information is placed where it is needed. At times, information is repeated as necessary to make the material more understandable. Much detailed information is omitted to help clarify the material - the omitted information can be found in the specification or standards.

In the Overview Chapter, there are many references to other chapters. The purpose of that chapter is to introduce information, not perform an exhaustive study of it. The more exhaustive study is in later chapters as referenced.

Conventions

Throughout this book, numbers are given in either binary form followed by "b," decimal form with no designation, or hexadecimal form followed by "h." The numbers 110001b, 49 and 31h are equivalent. With any good rule there are exceptions: the term 8b/10b refers to an encoding scheme that encodes 8 bits into 10 bits on transmit and decodes the 10 bits back into 8 bits on receive.

This document uses the American convention for designating numbers: a comma is used for thousands separators and a period is used as a decimal point. The NVMe Specification appears to show:
binary numbers grouped in four bits separated by spaces and indicated by a "b",
decimal numbers (maximum 4096) written without comma or space to separate thousands, and
hex numbers written in groups of 4bytes (8 digits) with groups of 8 separated by an underline and followed by an "h".

In this text, the term "Device" or "NVMe Device" or their plurals refers to NVMe controllers and the non-volatile storage media.

In this text, when referring to bytes or bits by abbreviation, upper case "B" means Bytes, lower case "b" means bits. Therefore MB/s is Mega-Bytes per second and Gb/s is Giga-bits per second. MSB is Most Significant Byte, MSb is Most Significant bit. Caution: The NVMe specification uses MSB as Most Significant bit.

The abbreviation "ms" means milliseconds (10^{-3} seconds). Microseconds (10^{-6} seconds) is represented by "µs." This text and the NVMe Specification both use "K" to mean 2^{10} or 1024 decimal.

When referencing a figure, the term "above" means either above on this page or on a previous page in this text. The term "below" means either lower on this page or on a page later in this text.

References to the "specification," "PCIe Specification" or "NVMe Specification" all refer to the following Specifications:

NVM-Express-1_2.1_Gold.pdf or
PCI_Express_Base_r3.1_November07-2013.pdf.

These specifications may have errata or supplements that must be referenced separately.

Copies of NVMe Specification is available on the Internet at nvmexpress.org.

Copies of the PCIe Specifications are available on the Internet at pcisig.com .

Always reference the most recent standard to obtain the latest information.

Caution

This book was current when written. However, because the specifications are constantly evolving, check the websites to obtain the most up-to-date specifications.

In any case of disagreement between this text and the specification, the specification must prevail. Emails to the author about the disagreement are appreciated.

This book is not intended to take the place of the PCI or NVMe specifications, but to prepare the reader with enough understanding of NVMe that finding specific answers in the specification will become easier. To that end, the use of the "standards' vocabulary" of *Shall*, *Should* and *Prohibited* are minimized. Instead, this text uses the words of *May*, *Can*, *Does* along with their negatives. It is hoped that by avoiding "Standardese" the book is easier to read and comprehend.

Accuracy is the most important concern in writing this book. If a choice had to be made between readability or convention on one hand and accuracy on the other, the author chose accuracy. In a choice between completeness and readability, the author readability. Some of the charts and tables in this text are complete, others are not. The reason to include them in this text is to show enough to give the reader an idea of how to use them. The complete information is in the Reference Manual (as of the date published).

This book was reviewed by experts and their comments have been used to alter the original text. If you find any omissions or errors please notify the author with the specifics at hcurley@indra.com. The changes will be in the next edition of this book.

A Matter of Style

Acronyms are usually all caps. The exceptions are acronyms that have entered the language as words.

Figures are either drawings or data structures. Text related to figures was positioned before or after the figure to promote understanding. Usually, the text describing the figure is before the figure, but the details of each field in a data structure follow the figure.

The word "may" is defined in the NVMe Specification as "Optional, no preference is indicated." In this text, it uses the dictionary definition of "may" meaning "permission".

The notation for footnotes and the exponents look the same. Footnotes are at the bottom of the page they are noted or in a few cases, the bottom of the next page. Footnotes are numbered sequentially through the book. Exponents are usually for a base 2, and the value of the exponents are usually 16, 24, 32, or 64.

We want your feedback.

Hugh Curley and KnowledgeTek value your feedback, comments, corrections and suggestions. Please contact us in a manner most convenient to you.

Hugh Curley
e-mail hcurley@indra.com
Phone 970-344-6053

KnowledgeTek, Inc.
e-mail info@knowledgetek.com
Web www.knowledgetek.com
Phone 303-465-1800
Fax 303-317-8973

Mail KnowledgeTek at:

KnowledgeTek, Inc.
8690 Wolff Court, Suite 110
Westminster, Colorado 80031

Part One: NVMe over PCIe

Overview

Chapter Contents

Why a New Interface

The SCSI and ATA (IDE) command sets and transports have been used for three decades with constant refinement and adjustment for new technology and requirements. They have proven themselves very well.

But both have a lot of legacy baggage that was put in when it was applicable, but in today's world is no longer needed. Both T10 (SCSI) and T13 (ATA) have been removing much of the obsolete functionality in the newer standards, but device manufacturers, both hardware and software, must continue to support the previous definitions or risk losing customers who don't want to upgrade.

Also SCSI and ATA command sets were developed for devices with long mechanical operations such as cylinder seek (average times are measured in single digit ms) and rotational latency (average times are about 3 ms). Against the long seek times, the command decode and setup is minuscule. As we move to SSDs the seek times are 0; the decoding and setup of complex commands becomes important to performance.

From an efficiency standpoint, the command, data, and status for SCSI and ATA already has to be transported across the PCIe transport, NVMe removes the necessity of converting it to the SCSI or ATA transport.

What is NVMe

NVMe is a new command set and queuing mechanism designed to utilize the PCIe transport.

Command Sets

NVMe has designed two new command sets specifically for SSDs connected directly to the system bus of PCIe. The commands sets are:

Admin to configure the NVMe controller and device, and
NVMe commands for accessing user data.

The NVMe command set is under the umbrella of I/O command sets which allow additional command sets to be developed later, perhaps for additional device types.

Admin

The Admin command set, covered in detail in "Admin Commands" on page 69, allows for configuring the controller or device. Facilities are provided to create and delete queues for I/O commands and status, read or write features to control the operation of the controller or device, update firmware, abort commands and get the device or namespace characteristics. Version 1.2 added the ability to create namespaces.

I/O

The I/O command title covers one or more command sets for accessing user data. Currently, the only I/O command set defined is called the NVMe command set covered in detail in "NVMe Commands" on page 125. It includes commands to read or write user data, control reservations to ensure certain command operations can be completed without other hosts intervening, marking blocks are unusable, and flushing cache.

Queue

Please refer to "Queuing" on page 151 for a complete treatment of NVMe queues.

Merriam-Webster defines a queue as (among other things):
1: <definition not applicable>,
2: a waiting line especially of persons or vehicles,
3a: <definition not applicable>, and

3b: a data structure that consists of a list of records such that records are added at one end and removed from the other.

If we consider the queue at the bus stop, people "queue-up" by moving to the tail-end of the line. When the bus arrives, the people board starting with those at the head of the line. If the people in the queue remain orderly, those arriving earlier at the bus stop board earlier than those who arrived later - a First-in First-out queue.

The NVMe queues work the same way, First-in First-out.

NVMe queues are located in the host memory address space. Rev 1.2 adds the capability of creating the queues in either host physical memory or controller physical memory. The host places the command into a submission queue (SQ) for the drive to fetch. The drive will place the command status into the completion queue (CQ) for the host to read. Data transfers do not use the queues.

Each queue has two pointers:

Tail points to the next slot to where information will be added, and
Head points to the next slot from where information will be removed.

Caution: Disk Drive command queuing is very different. SCSI Tag Command Queuing and ATA Native Command Queuing are drive side queuing. When the drive receives a command, it places the command into a queue based on host provided information and drive provided algorithms. Commands will probably get executed in an order other than the order received by the drive.

Admin Queues

Admin queues are for submission of Admin commands to the controller and for receiving completion status from the controller. There is a single submission queue (SQ ID = 0) and a single completion queue (CQ ID = 0) for each controller.

Submission

The Admin SQ contains up to 4096 entries of 64 bytes each to hold the commands from the host to the controller.

Completion

The Admin CQ contains up to 4096 entries of 16 bytes each to hold the completion status sent from the controller to the host.

I/O Queues

I/O Queues hold the I/O commands from the host to the controller to access user data or the completion status. For each controller, there must be at least one SQ and one CQ.

Submission

I/O SQ hold the I/O commands from the host to the controller. The host can provide for each controller to have from 1 to 64K queues, each queue can have from 2 to 64K entries. Each entry holds one NVMe command of 64 bytes. The controller can specify a maximum number of supported queues and a maximum number of entries per queue.

Each SQ is associated with a specific CQ at the time of queue creation. This may be a 1 to 1 association or a many to one (more than 1 SQ per CQ).

Completion

I/O CQ hold the statuses from the controller to the host. The host can provide for each controller to have from 1 to 64K queues, each queue can have from 2 to 64K entries. Each entry holds one NVMe completion status of 16 bytes. The controller can specify a maximum number of supported queues and a maximum number of entries per queue.

Queue Priority

Submission Queues can be assigned a priority. Commands than can be submitted to the appropriate priority queue such that high priority commands should be executed before lower priority commands. I/O SQ priorities are Urgent, High, Medium and Low.

There are two priority mechanisms defined in NVMe: Round Robin which is mandatory for all controllers, and Weighted Round Robin with Urgent which is optional. Also, controllers may support a vendor specific priority mechanism. The priority mechanisms supported are in the Controller Capabilities Registers, bits 18:17. The priority mechanism currently selected is in the Controller Configuration Register, bits 13:11.

For details about Queue Priority, see “Queue Priority” on page 160.

Doorbells

Doorbells are to get your attention.

In NVMe, there is one doorbell per queue, rung by the host to get the controller’s attention. In both cases (SQ and CQ) the host is notifying the controller that the host has updated queue pointers and the controller may need to know about the updates.

Doorbells are NVMe registers; actually they are host memory locations located on the NVMe device that when written have a side effect. See “NVMe Registers” on page 39 for details.

Example Command Execution

Figure 1 shows the sequence of an example command execution. This assumes the SQ and CQ are in the host physical memory.

1. The host will build the necessary buffers, construct the command and write the command into the SQ at the tail. The host may write one or more commands at a time.

2. The host will write the new tail value into the appropriate doorbell register to indicate the new ending of the queue to the NVMe controller.

3. The NVMe controller will issue a PCIe memory read transaction to fetch the commands from the head of the SQ and place the commands into its command buffer.

4. The NVMe controller will select a command to process and process it. If the command indicates data transfer, the NVMe controller will use PCIe memory read or memory write transactions to move the data. A read command indicates that data will move from the device to the host so the NVMe controller will use a PCIe memory write transaction.

5. Upon completion of the command, the controller will use a PCIe write transaction to place the status onto the CQ. The controller may place one or more statuses on the CQ at a time.

6. The NVMe controller will interrupt the host using legacy pin-based interrupts (INT A/B/C/D), Message Signaled Interrupt (MSI), or Message Signaled Interrupt Extended (MSI-X).

7. The host will process the completions. The P bit in the status entry indicates to the host the end of the new status entries. This will be covered in the Common Command Fields chapter.

8. The host will write the CQ head pointer into the CQ doorbell. The controller needs this information to determine if the queue is full or can accept more status entries.

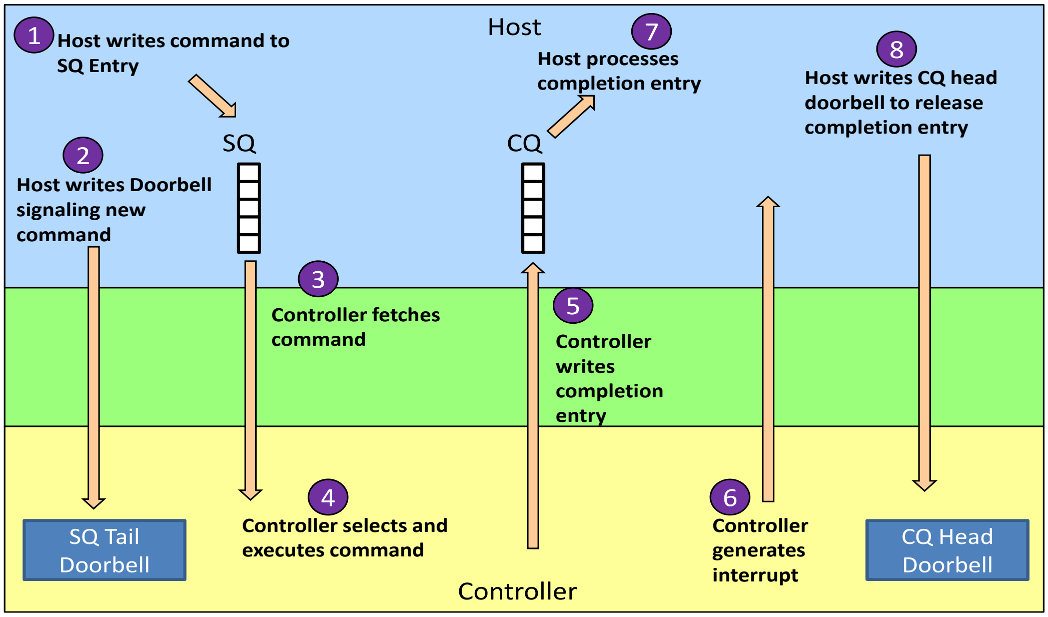

Host
1
Host writes command to SQ Entry
SQ
2
Host writes Doorbell signaling new command
CQ
7
Host processes completion entry
8
Host writes CQ head doorbell to release completion entry
3
Controller fetches command
5
Controller writes completion entry
4
Controller selects and executes command
6
Controller generates interrupt
SQ Tail Doorbell
CQ Head Doorbell
Controller

Figure 1: NVMe Example Command Execution

End to End Data Protection

Within computer systems there are many data protection mechanisms to cover certain parts. For example, data stored on media is usually protected by ECC, and data traveling on serial interfaces between devices is usually protected by CRC. However, there are many places where the data may or may not be protected such as internal to switches, routers, computers or storage devices. See Figure 2. To solve this possibility of undetected data lost or corruption, several groups added end-to-end data protection.

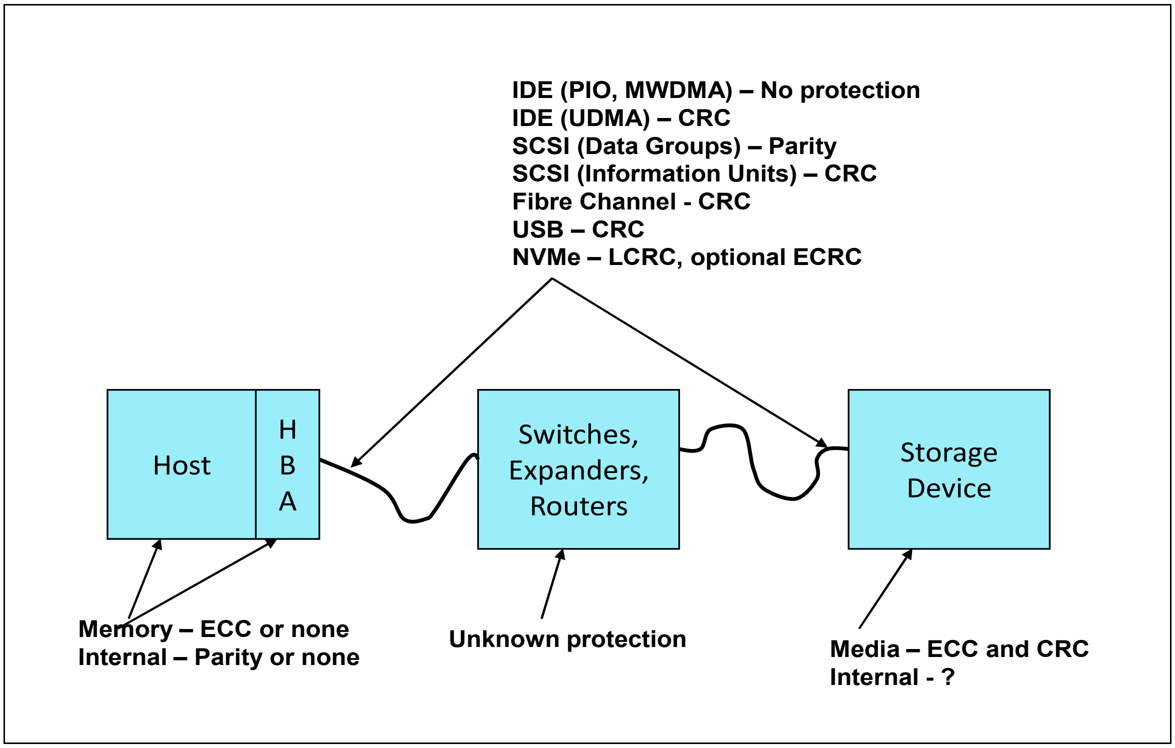

IDE (PIO, MWDMA) – No protection
IDE (UDMA) – CRC
SCSI (Data Groups) – Parity
SCSI (Information Units) – CRC
Fibre Channel - CRC
USB – CRC
NVMe – LCRC, optional ECRC
Host
H
B
A
Switches, Expanders, Routers
Storage Device
Memory – ECC or none
Internal – Parity or none
Unknown protection
Media – ECC and CRC
Internal - ?

Figure 2: End to End Data Protection Problem

PCIe ECRC

One version of end to end data protection is the optional End-to-End CRC of PCIe. It is added by the transaction layer of the source device to the Transaction Layer Packet (TLP) and is verified and removed by the transaction layer of the final destination device.

Problems with the PCIe ECRC are:

It does not protect the data before the PCIe transaction layer of the source device,
It is optional,
It can be removed or recalculated enroute if multi-cast overlay is used, and
It does not protect the data after the transaction layer of the destination device.

An example of this last point is on a write command, the data is not protected between the storage devices PCIe transaction layer and the device's ECC generation circuitry. On a read command, the data is not protected between the device's ECC checking circuitry and the PCIe transaction layer.

PCIe LCRC is from link layer on one end of a physical link to the link layer on the other end of the physical link only.

NVMe can get the benefits of this PCIe facility.

DIF

The SCSI group under ANSI (www.t10.org) created a Data Integrity Field (DIF) which is added in the computer at the Host Bus Adapter and travels with the data on writes, is stored with the data on the media, and is returned with the data on reads. Data is protected from the time the data leaves the computer with a write command until it is read by the computer with a read command.

The protection field contains three items (See Figure 3):
16-bit CRC to detect corruption,
16-bit Application Tag for use by the application, and
32-bit Reference Tag, a sequential counter to detect lost packets or frames.

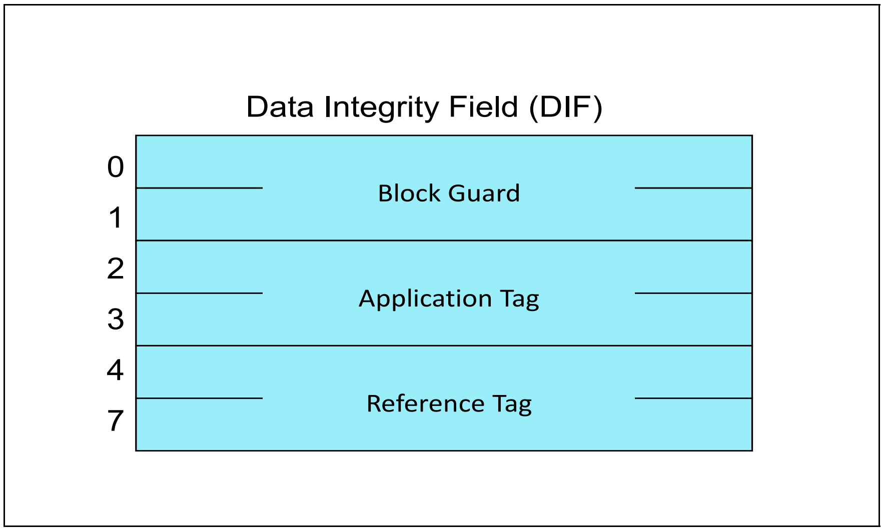


Figure 3: DIF Information

T10 defined four types of protection is shown in “: DIF Protection Types” on page 10.

Type 0 means there is no DIF protection provided.
Type 1 means that both the Block Guard (16-bit CRC) is checked for corruption and the Reference Tag (32-bit Sequence number) is checked. When using type 1, the first block’s reference tag is the last four bytes of the LBA.
Type 2 checks the Block Guard and all the Reference Tags except the first one. This was created for use primarily behind disk array controllers. When using type 2, the first block’s reference tag is assigned by the host.
Type 3 checks the Block Guard and not the References Tags.

The Application Tag is marked as “AS” meaning Application Specific.

The following NVMe commands are designed for using the DIF fields in DWords 14 and 15 of the commands:
Read,
Write,
Compare, and
Write Zeros.

Type	0	1	2	3
Block Guard	No	Yes	Yes	Yes
Application Tag	No	AS	AS	AS
Reference Tag	No	Yes	Yes, except 1st block	No

Table 1: DIF Protection Types

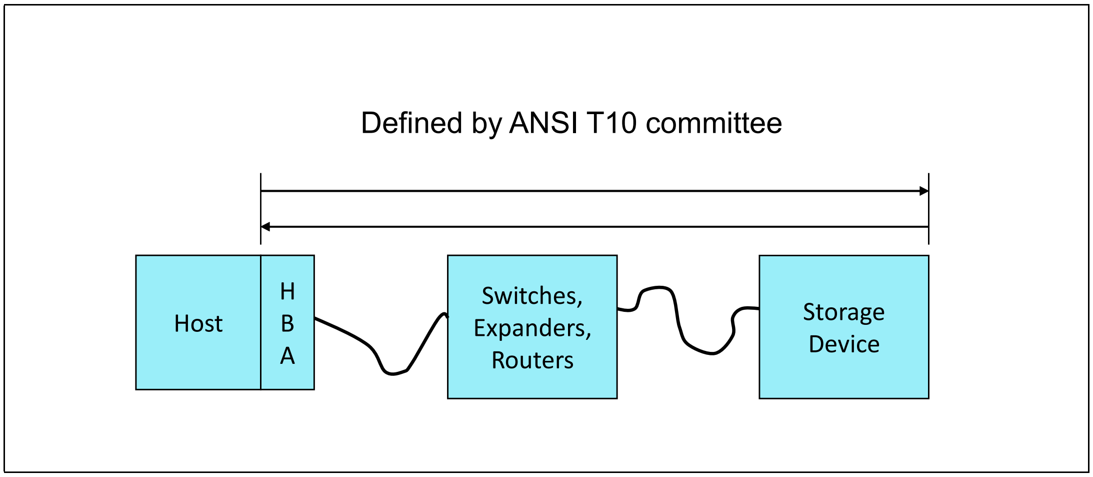


Figure 4: DIF Limits

This information was added to and stored with the data, requiring that a 512 byte logical block be formatted on the media as 520 bytes. Also, because this was defined by the T10 organization of ANSI, the limits of its protection was from the HBA on write to the HBA on receive. To get around these two problems, another group (SNIA - SCSI Networking Industry Association) defined Data Integrity Extensions (DIX).

DIX

Data Integrity Extension (DIX) was created to overcome two problems with DIF. For the most part, they copied DIF where DIF worked for them.

First, the DIX protection information is created in the host, and checked in the host. This extends the checking over DIF because in DIF the HBA, or the HBA - Computer interface, could have a non-detected problem. DIX should catch these problems.

Second, DIX is designed to store the protection data in a different area than the user data. That means that:

a 512 byte block is still formated for 512 bytes and

additional information such as creating department or expiration date can be stored with the protection information.

The additional information is called metadata. NVMe commands have a 64-bit field, called Metadata, for defining a separate buffer for storing the metadata.

Caveats

Base 0/Base 1

Some fields are labeled as “0’s based” meaning they start counting with 0. Therefore 0 = 1, 1 = 2, etc. This allows a natural binary number (such as 4096) to be specified in the field (FFFh = 4096d).

Others fields are labeled as “1’s based” meaning they start counting with 1. Therefore 1 = 1, 2 = 2, etc. This allows a value of 0 = none, nothing, or invalid. Also it is easier for a human to read the number. I/O queue identifiers start with 1 because queue 0 refers to the Admin queues.

Sometimes there is a good reason for a field being 0’s based or 1’s based. Sometimes it seems arbitrary.

Outside the Scope

The NVMe specification defines a register interface and command set. These may be used with various types of storage media, transports, programming languages and structures, CPU architectures, or usage models. Many questions asked when reading the specification are listed as “outside the scope” meaning they may be defined elsewhere.

ECN

Engineering Change Notices are the normal way to update a completed and approved specification for NVMe. That means that to be current, you must get the latests specification and all the ECNs that apply to it. At the time of this writing, the current specification is NVMe 1.2 and there are seven ECNs that apply to it each with multiple changes.

Chapter Summary

- Why a New Interface
- What is NVMe
    - Command Sets
        - Admin
        - I/O
    - Queue
        - Admin Queues
        - I/O Queues
        - Queue Priority
    - Doorbells
    - Example Command Execution
    - End to End Data Protection
        - PCIe ECRC
        - DIF
        - DIX
- Caveats
    - Base 0/Base 1
    - Outside the Scope
    - ECN

PCIe Overview

Chapter Contents

PCIe Header and Capability Structures

Question: How does the host find the NVMe controller?
Answer: That’s a PCIe question.

Question: How does the host discover the NVMe’s characteristics and control the configuration?
Answer: That’s a PCIe question.

Question: How do I learn PCIe?
Answer: Keep reading.

This section discusses only the PCIe registers and operations that are relevant to NVMe. For other PCI, PCI-X or PCIe registers, check with the PCIe Specification or KnowledgeTek’s PCIe course.

PCIe Configuration Space

PCIe puts all the configuration information in a single place and uses special transactions to read or write it. There is a Configuration Space for every PCIe function. A PCIe function corresponds to the NVMe controller. See Figure 5.

The first 64 bytes of Configuration Space are called the Configuration Header and are shown in Figure 6. Configuration space with offsets greater than 40h contain Capability structures.

PCI and PCI-X allowed 256 bytes (0-FFh) for Configuration Space. PCIe extended the space to 4K bytes (0-FFFh). The configuration information above FFh is not available to legacy (PCI and PCI-X) systems.

The Configuration Space is read with Configuration Read transactions and written with Configuration Write transactions. The addressing is by Bus-Device-Function, called ID Routing. The location to read within the Configuration Space is by “offset,” called Register in PCIe specification. Each Register is on a DWord offset, but Byte Enables in the transaction header specify which bytes the host wants with this access.

There are two “types” of Configuration Read (CfgRd0 and CfgRd1) and Configuration Write (CfgWr0 and CfgWr1). As the transaction is flowing through the topology, it will be set to type 1. The output port of the last bridge/switch (the port connected directly to the end endpoint) will change it to type 0. Caution: Configuration Space Headers are also type 0 (endpoint/function) or type 1 (bridge/switch) but it is used for a different purpose.

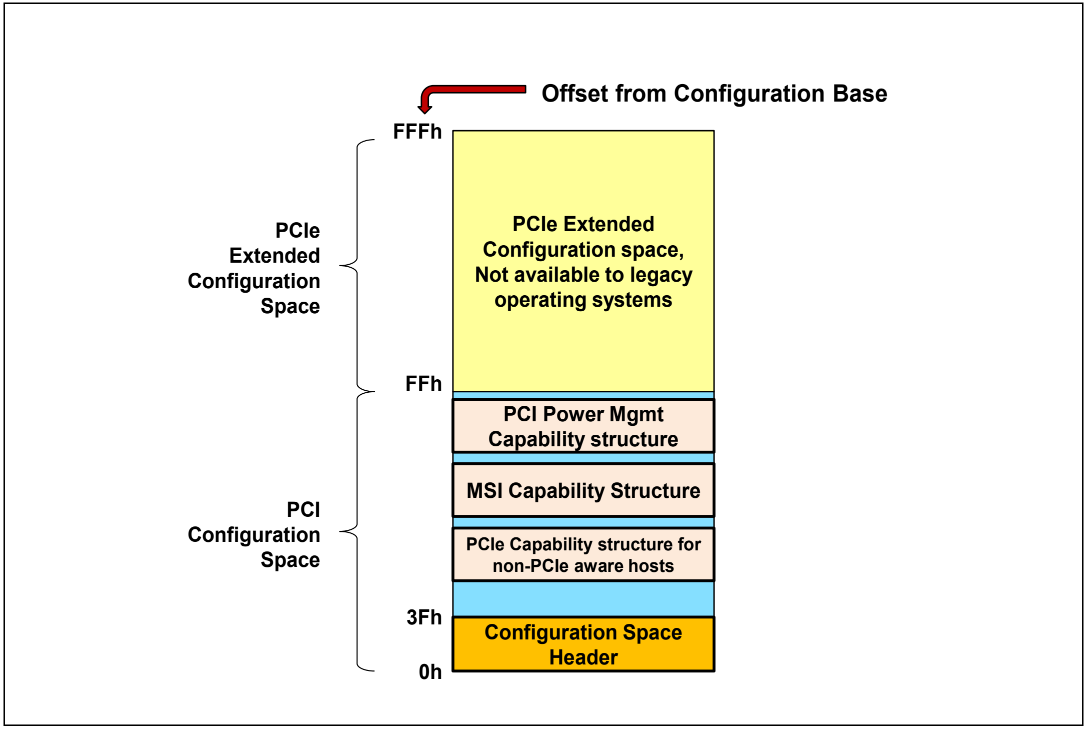
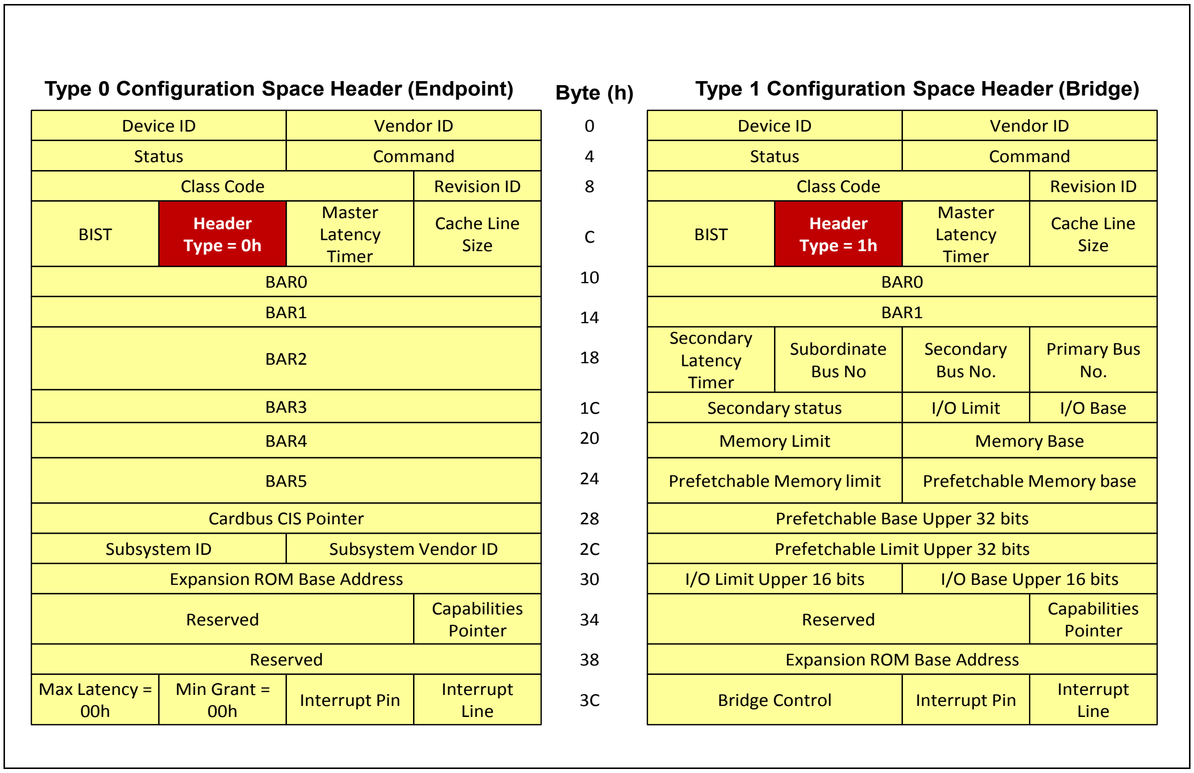

Offset from Configuration Base
FFFh
FFh
3Fh
0h
PCIe Extended Configuration Space
PCI Configuration Space
PCIe Extended Configuration space, Not available to legacy operating systems
PCI Power Mgmt Capability structure
MSI Capability Structure
PCIe Capability structure for non-PCIe aware hosts
Configuration Space Header

Figure 5: PCIe Configuration Header and Capability Structures

Configuration Headers



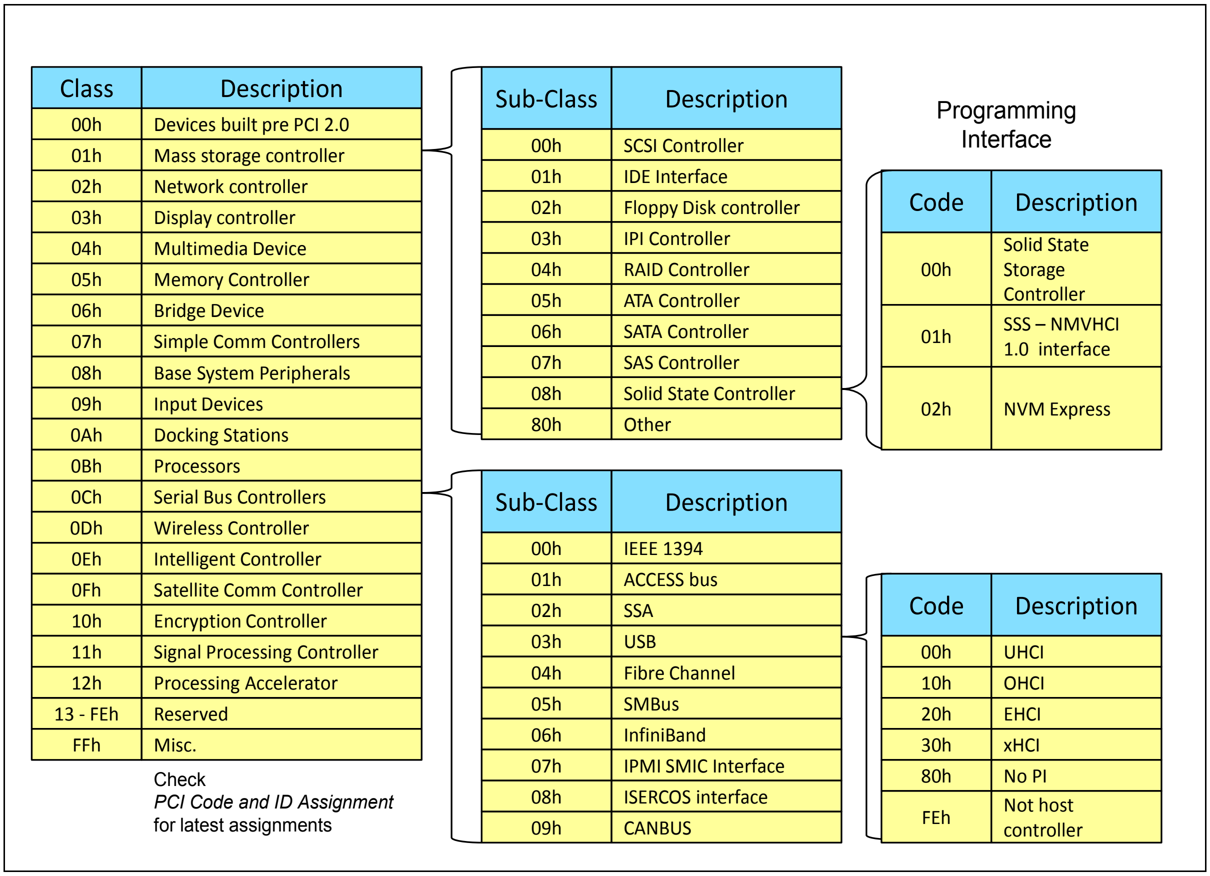
Type 0 Configuration Space Header (Endpoint)				Byte (h)	Type 1 Configuration Space Header (Bridge)			
Device ID		Vendor ID		0	Device ID		Vendor ID	
Status		Command		4	Status		Command	
Class Code			Revision ID	8	Class Code			Revision ID
BIST	**Header Type = 0h**	Master Latency Timer	Cache Line Size	C	BIST	**Header Type = 1h**	Master Latency Timer	Cache Line Size
BAR0				10	BAR0			
BAR1				14	BAR1			
BAR2				18	Secondary Latency Timer	Subordinate Bus No	Secondary Bus No.	Primary Bus No.
BAR3				1C	Secondary status		I/O Limit	I/O Base
BAR4				20	Memory Limit		Memory Base	
BAR5				24	Prefetchable Memory limit		Prefetchable Memory base	
Cardbus CIS Pointer				28	Prefetchable Base Upper 32 bits			
Subsystem ID		Subsystem Vendor ID		2C	Prefetchable Limit Upper 32 bits			
Expansion ROM Base Address				30	I/O Limit Upper 16 bits		I/O Base Upper 16 bits	
Reserved			Capabilities Pointer	34	Reserved			Capabilities Pointer
Reserved				38	Expansion ROM Base Address			
Max Latency = 00h	Min Grant = 00h	Interrupt Pin	Interrupt Line	3C	Bridge Control		Interrupt Pin	Interrupt Line

Figure 6: Configuration Header

Notice in Figure 6 the Header Type field to differentiate the two types of Configuration Space Header.

The first 4 bytes of both header types contain the PCIe assigned Vendor ID and a Device ID. They answer the questions of who made this product and what product is it. The header type 0 also has in bytes 2C - 2Fh a Subsystem Vendor ID and Subsystem ID to allow the function to report two levels of manufacture; for example the motherboard and the internal chip.

Both header types have a Class Code which is actually three bytes of information: Class, Sub-class and Programming Interface. These values are assigned by PCI-SIG. Figure 7 shows a subset of the class codes. NVM Express (NVMe) is Class 01h, Sub-class 08h, Programming Interface 02h. The document “PCI Code and ID Assignment” available from PCI-SIG (www.pcisig.com) has the most complete and up-to-date list of codes.



Class	Description
00h	Devices built pre PCI 2.0
01h	Mass storage controller
02h	Network controller
03h	Display controller
04h	Multimedia Device
05h	Memory Controller
06h	Bridge Device
07h	Simple Comm Controllers
08h	Base System Peripherals
09h	Input Devices
0Ah	Docking Stations
0Bh	Processors
0Ch	Serial Bus Controllers
0Dh	Wireless Controller
0Eh	Intelligent Controller
0Fh	Satellite Comm Controller
10h	Encryption Controller
11h	Signal Processing Controller
12h	Processing Accelerator
13 - FEh	Reserved
FFh	Misc.

Check
PCI Code and ID Assignment
for latest assignments

Sub-Class	Description
00h	SCSI Controller
01h	IDE Interface
02h	Floppy Disk controller
03h	IPI Controller
04h	RAID Controller
05h	ATA Controller
06h	SATA Controller
07h	SAS Controller
08h	Solid State Controller
80h	Other

Programming Interface

Code	Description
00h	Solid State Storage Controller
01h	SSS – NMVHCI 1.0 interface
02h	NVM Express

Sub-Class	Description
00h	IEEE 1394
01h	ACCESS bus
02h	SSA
03h	USB
04h	Fibre Channel
05h	SMBus
06h	InfiniBand
07h	IPMI SMIC Interface
08h	ISERCOS interface
09h	CANBUS

Code	Description
00h	UHCI
10h	OHCI
20h	EHCI
30h	xHCI
80h	No PI
FEh	Not host controller

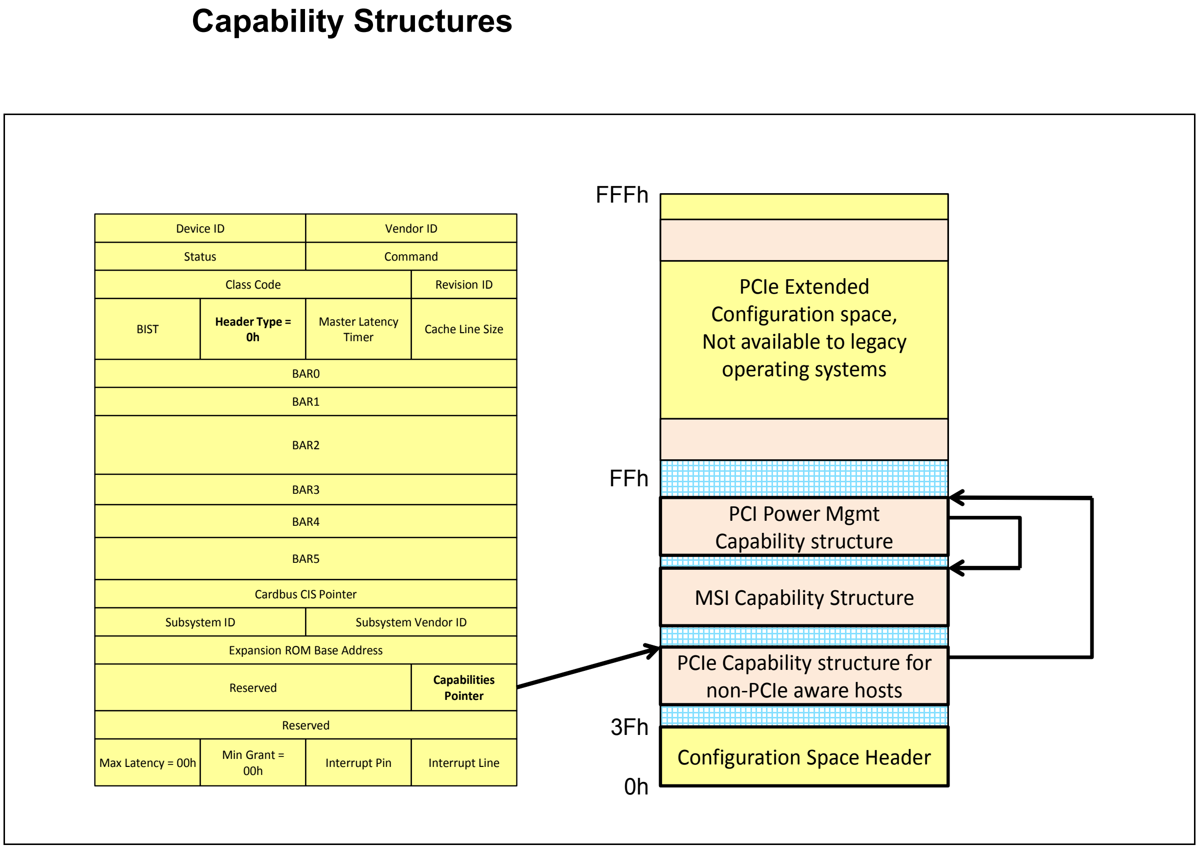
Figure 7: Class Codes

Header Type 0 has six Base Address Registers (BAR), Header Type 1 has two BARs. During initialization, the host will write memory addresses into the BARs. The memory addresses are above any the system would normally use. That memory is physically on the endpoint/function (for type 0) but addressed and treated like host memory. That way, by adding many more functions each function supplies the memory it needs and does not take any of the system memory. A single BAR can be used for a 32-bit memory address, or two adjacent BARs can be used for a 64-bit memory address. NVMe controllers use BAR0 and BAR1 as the base address for the NVMe defined registers, and BAR1 for an identifier to facilitate legacy I/O addressing. See “Find the Registers” on page 40.

Header Type 1 has three bus number fields in bytes 18h, 19h, and 1Ah. The Primary Bus No. is the upstream (toward the Root Complex) bus number. It will be the lowest number of the three. The Secondary Bus No. is the next downstream bus of the bridge. The Subordinate Bus No. is the highest Bus Number that can be reached by going through this bridge. When a bridge sees a PCIe transaction on its upstream port that uses ID Based addressing, the bridge will pass that transaction only if it is between the Secondary Bus No. and the Subordinate Bus No. inclusive. This is covered in more detail later, but to see an example, see Figure 14.

Type 1 Header, bytes 1C-33h (except Secondary status field) define the I/O and memory address limits that can be reached through this bridge. They are specified as a base address and a limit. Bytes 20-23h are the base and limit for Non-prefetchable Memory Addresses.

At Configuration Header Space address 34h is the Capabilities Pointer. As shown in Figure 8, it points to the first Capability Structure, which points to the second, which points to the third, etc. in a linked list format.



Device ID		Vendor ID	
Status		Command	
Class Code			Revision ID
BIST	**Header Type = 0h**	Master Latency Timer	Cache Line Size
BAR0			
BAR1			
BAR2			
BAR3			
BAR4			
BAR5			
Cardbus CIS Pointer			
Subsystem ID		Subsystem Vendor ID	
Expansion ROM Base Address			
Reserved			**Capabilities Pointer**
Reserved			
Max Latency = 00h	Min Grant = 00h	Interrupt Pin	Interrupt Line

FFFh
PCIe Extended Configuration space, Not available to legacy operating systems
FFh
PCI Power Mgmt Capability structure
MSI Capability Structure
PCIe Capability structure for non-PCIe aware hosts
3Fh
Configuration Space Header
0h

Figure 8: Capability Pointer and Linked List

There are two formats for Capability Structure: Legacy and PCIe Extended. The formats are shown in Figure 9.

These data structures are drawn as in the PCIe specification. Note that the Configuration Space structure on the left has the lowest address on the bottom and addresses increase as the picture goes up the page. The two Compatibility Data Structures on the right have their lowest address on top and the address increases as the picture goes down the page. Be careful.

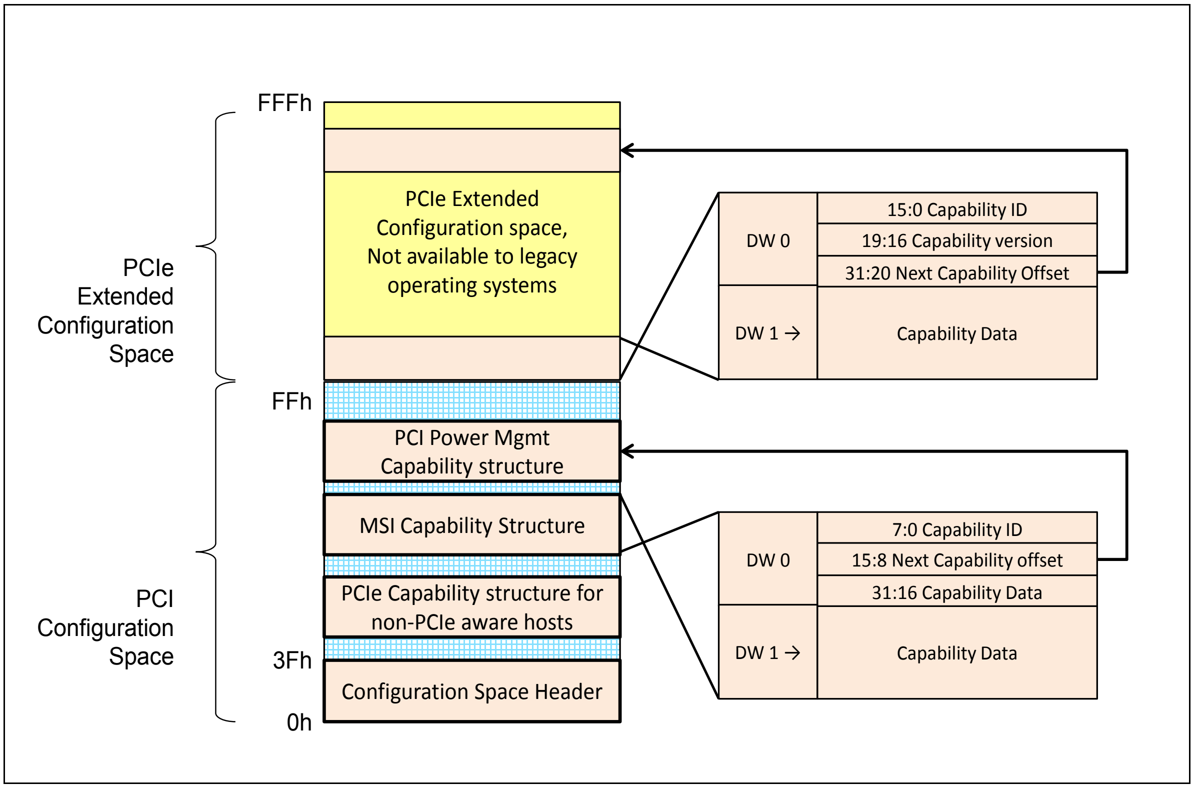

FFFh
PCIe Extended Configuration Space
PCIe Extended Configuration space, Not available to legacy operating systems
FFh
PCI Power Mgmt Capability structure
MSI Capability Structure
PCI Configuration Space
PCIe Capability structure for non-PCIe aware hosts
3Fh
Configuration Space Header
0h
DW 0
15:0 Capability ID
19:16 Capability version
31:20 Next Capability Offset
DW 1 →
Capability Data
DW 0
7:0 Capability ID
15:8 Next Capability offset
31:16 Capability Data
DW 1 →
Capability Data

Figure 9: Capability Structures

In the legacy capability structure, the first byte is the Capability ID, the second byte is the next capability offset and the remainder of the data as defined by the Capability ID.

In the PCIe Extended Capability Structure the first two bytes are the Capability ID, the next 4 bit are a version code, and the next 12 bits are the Next Capability Offset. Starting with the second DWord is the information for that Capability.

The legacy structure must fit into the configuration space of 00 - FFh. Its Next Capability Offset pointer is only 8 bits long. The PCIe Extended Capability Structure has a 12-bit address linking to the next structure so can address the full 4K of addresses.

The length of each structure can be known by the Capability ID (legacy) or Capability ID and Version (Extended).

Table 6 lists the legacy Capability IDs.

Cap ID	Name
01h	Power Management
03h	Vital Product Data
04h	Slot Identification
05h	MSI Capability
09h	Vendor Specific
0Ah	Debug Port
0Ch	Hot-Plug Controller
0Dh	SSID/SSVID Registers in a Bridge
0Fh	Secure Device
10h	PCI Express Capability
11h	MSI-X Capability

Table 2: PCI Capabilities List

Many NVMe controllers will have Capability Structures for Cap IDs of 01h, 05h, 10h and 11h. Figure 10 is an example of a Capability Structure that is used frequently in NMVe. It defines many PCIe features and capabilities yet is in the first 256 bytes of the Configuration space so is accessible to hosts that at not PCIe aware.

PCIe Capability Structure for non-PCIe aware hosts

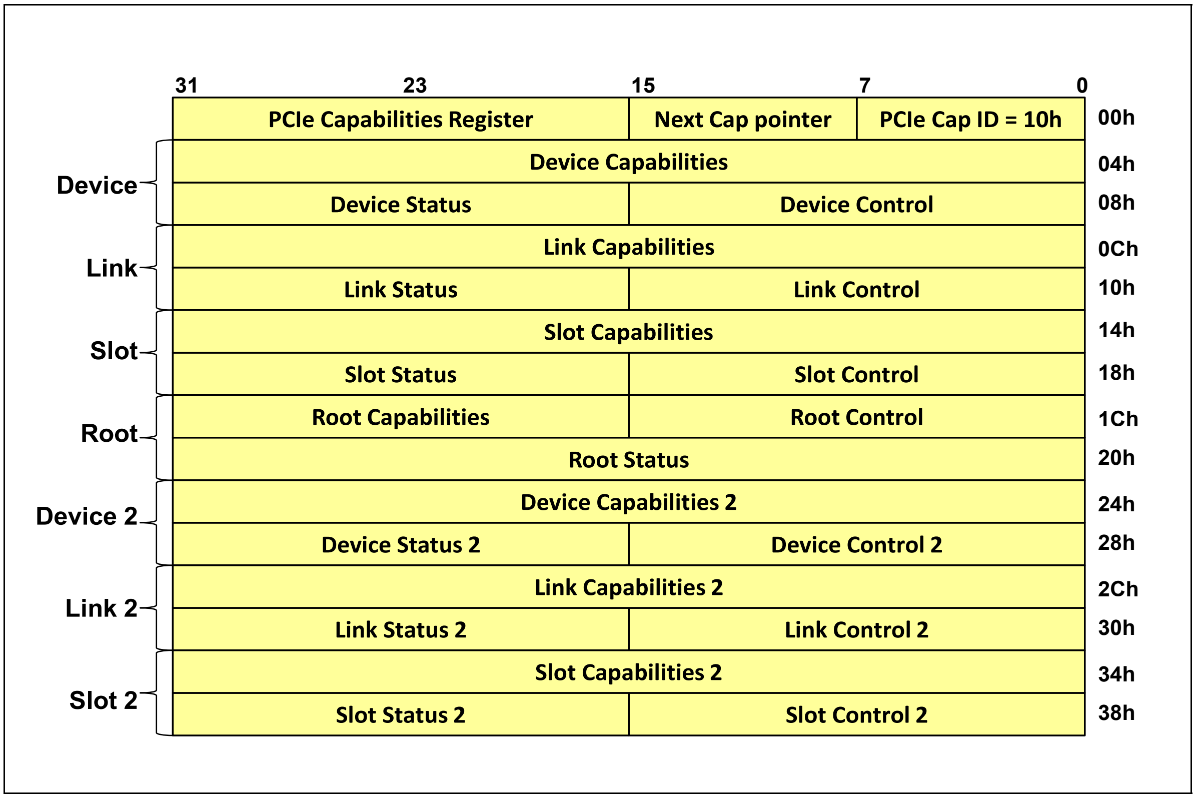


Figure 10: PCIe Capability for non-PCIe hosts

This Capability Structure has sets of 2 DWords each for the Device, Link, Slot, and Root. Later because of the desire to add additional information, a second set of DWords was added for the Device, Link, and Slot. Each set of 2 DWords has space for capabilities, status, and control.
Capabilities tell the host, "This is what I can do."
Status tells the host, "This is my current configuration or state."
Control tells the host, "Here is how you can change my operational characteristics."

For the full list of capabilities, status and control for the Device, Link, Slot and Root refer to the PCIe specification at www.PCISIG.org or the KnowledgeTek's PCIe Reference Manual.

MSI and MSI-X interrupts

The Message Signaled Interrupt (MSI) capability and Message Signaled Interrupt - Extended (MSI-X) are interrupts designed to replace the PC's four legacy pin-based shared interrupts of INT A/B/C/D. These are two different structures; a function may support either one or both. Only one can be enabled are a time.

PCI initializes with both MSI and MSI-X disabled. Interrupts must be submitted via pin-based interrupt until either MSI or MSI-X is configured and enabled.

MSI and MSI-X are unfortunate choice of names. There are messages in PCIe, but they are completely different in form and function than MSI and MSI-X. It is less confusing to view these as interrupts and not as messages.[1]

Both MSI and MSI-X were defined in PCI_LB3.0-2-6.04 and modified by PCIe. The PCIe specification gives only the differences from the LB specification to the PCIe specification.

MSI Capability Structures

The MSI (Message Signaling Interrupt) capability (MSICAP) has four different structures with different lengths. The options are 32- or 64-bit addresses and with or without Per-Vector Masking. The Message Control field has bits to let the host know the format of the MSICAP structure. See Table 3 on page 23.

Figure 11 and Figure 12 show the MSICAP using 64-bit addressing, and with and without Per-Vectoring Masking. NVMe requires 64-bit addressing, if you will be using 32-bit addressing the "Message Upper Address" DWord will not be present.

The MSI interrupt allows each function its own 32, non-shared interrupts (vectors). If MSI is enabled (bit 0 of Message Control, see Table 3 on page 23) the host will write a 64-bit address into the Message Address and Message Upper Address DWords and a System Identifier into Message Data bits 15:00 (see Table 4 on page 23). When the Function wants to send an interrupt to the host, the Function will overwrite the low order bits of the Message Data with the interrupt vector (1-32) and write that to the 64-bit Message Address. The host is monitoring that address and can tell which function interrupted and why.

In Table 4 below, x will be 4 if the host allocates 32 interrupt vectors, 3 if the host allocates 16 interrupt vectors, 2 if the host allocates 8 interrupt vectors, 1 if the host allocates 4 interrupt vectors, 0 if the host allocates 2 interrupt vectors. For example, if the host allocates four interrupt vectors (Multiple Message Enable = 010b), then bits 1 and 0 will have the interrupt vector and bits 15:02 will have the system identifier.

It is possible, and maybe even probably, that a function may have a single, non-shared, interrupt only.

[1] To be fair, MSI and MSI-X are carried over from PCI. Message transactions were added later in PCIe.

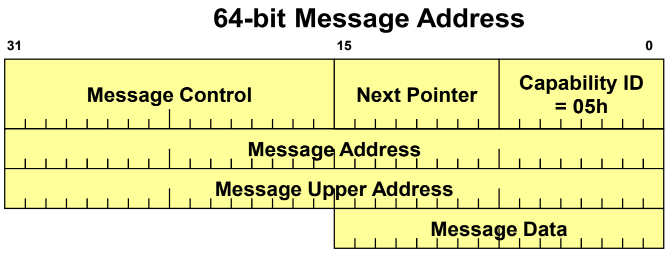


Figure 11: MSI Capability without Per-Vector Masking

Using Per-Vector Masking[2], the host can mask off chosen interrupts by writing a 1b to that bit on the Mask DWord. See Figure 12. As shown on the top of the structure, Interrupt 0 is masked with the right most bit in the mask field, increasing to interrupt 31 in the left most bit of the mask field.

If the Function has an interrupt to present that is currently masked, it will set the pending bit for that interrupt. It will remain pending until the software unmasks it and the interrupt is presented to the host.

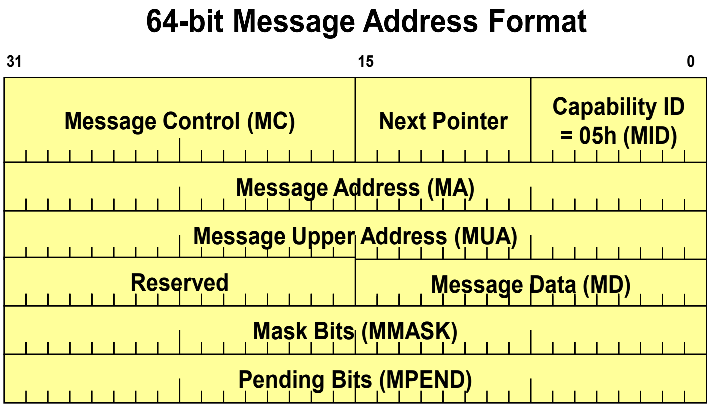


Figure 12: MSI Capability with Per-Vector Masking

Message Control (MC) fields are shown in “: MSI Capable Message Control bits” on page 23

[2] The name “pre-vector masking” is used because hardware may present an interrupt when the software is not expecting it - called a spurious interrupt. Likewise, hardware may not present an interrupt when the software is expecting it - called a lost interrupt. To manage and control this, software can set the mask bits to accept only expected interrupts (eliminating spurious interrupts). When all expected interrupts are processed, software can check the pending bits to see if any other interrupts are waiting to be processed and if so, unmask them when it is ready.

NVMe specifications identify the capability structure, fields and bits differently than PCI and PCIe specifications. NVMe references this structure as MSICAP. Next append the field name shown in parentheses above, e.g.MSICAP.MC. If the field has individually defined bits, then append the bit names, e.g. MSICAP.MC.MME. This is read as, “The MME bit (Multiple Message Enable bit) of the MC (Message Control) field of the MSI Capability data structure.”

.

Bits	Type	Reset	Description
15:09	RO	0	Reserved
08	RO	Impl Spec	**Per-Vector Masking Capable (PVM):** Specifies whether controller supports MSI Per-Vector Masking
07	RO	1	**64 bit Address Capable (C64):** Specifies whether the controller is capable of generating 64-bit messages. NVMe controllers shall be 64-bit capable.
06:04	RW	000	**Multiple Message Enable (MME):** Host writes this field to indicate the number of interrupt vectors the Function may use. This field is a power of 2 and is read/write.
03:01	RO	Impl Spec	**Multiple Message Capable (MMC):** Read by the host to determine the number of interrupt vectors the Function requests. This field is the exponent (power of 2). This field is read only.
00	RW	0	**MSI Enable (MSIE):** If set to ‘1’, MSI is enable and the traditional interrupt pins are not used to generate interrupts. If cleared, MSI operation is disable and the traditional interrupt pins are used.

Table 3: MSI Capable Message Control bits

Bits	Description
15:x+1	Host assigned number for controller identification
x:0	MSI Interrupt Vector

Table 4: Message Data (MD)

MSI-X Capability Structure

Another form of interrupt defined by PCIe is MSI-X (Message Signaled Interrupt - Enhanced).

Where MSI has up to 32 interrupt vectors per function, MSI-X has up to 2K interrupt vectors per function. For the structure, see Figure 13.

The second DWord of the structure points to a MSI-X interrupt table. The table is pointed to by the BIR (Base Address Register Indicator Register) which is one of the function’s BARs starting at offset 10h of the Configuration Space. See Figure 6. The other bits specify the offset from that BAR to the beginning of the Interrupt Vector Table.

The NVMe Registers are defined as being off BAR0/1. This Interrupt vector table and the Pending Bit Array can also be off BAR0/1 with the offset for each starting after the NVMe registers. OR the interrupt vector table and/or Pending Bit Array can be off other available BARs perhaps with offset 0.

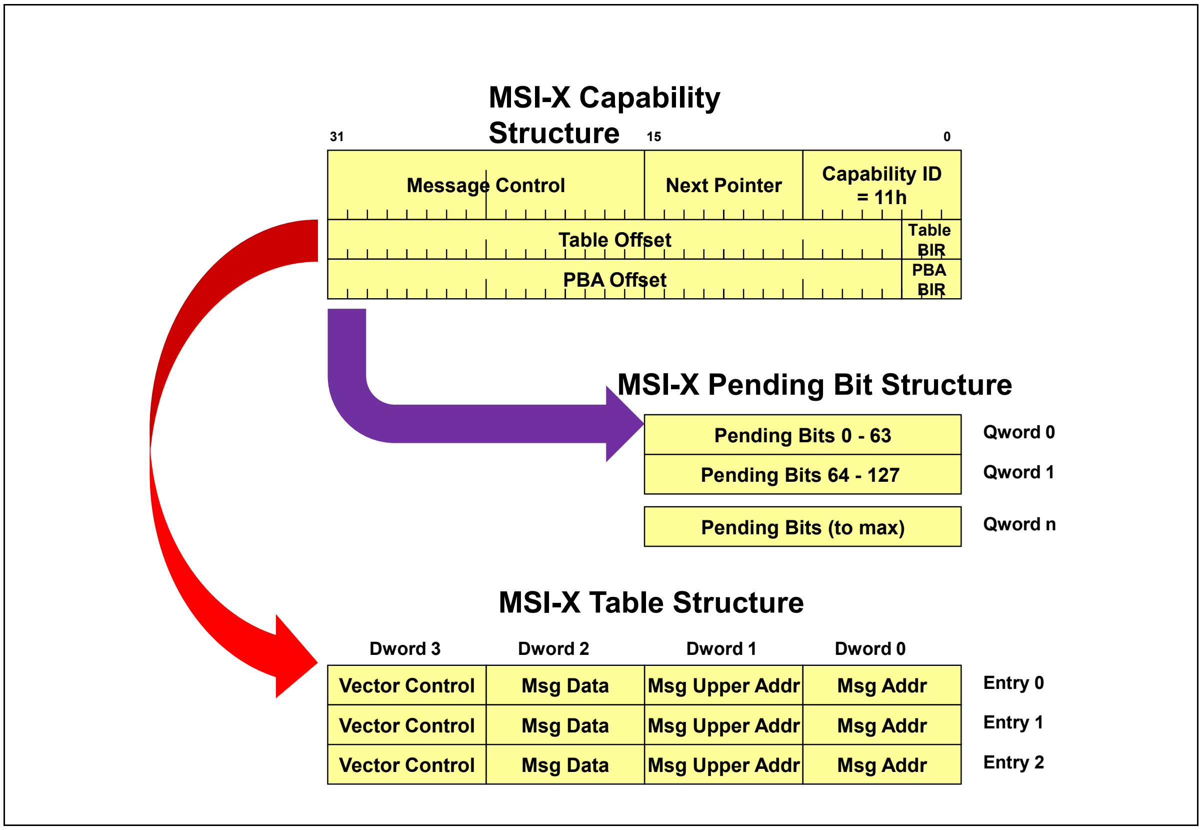


Figure 13: MSI-X Capability Structure

Each entry in the MSI-X vector table is made up of four DWords as shown in Figure 13. There is one entry per interrupt vector. The maximum number of interrupt vectors per table is given in Message Control, Bits 10:00 (0's based).

DWord 0 and DWord 1 create a 64-bit memory address for the controller to write the message data (DWord 2).

DWord 3 bit 0 is the Vector Mask bit; when set, this entry (vector) cannot interrupt the host. Other vectors, even with the same Message Data and Message Address may interrupt if not masked off. Message Control bit 14 is the Function Mask bit. If the Function Mask bit is set, the Vector Mask bits are ignored; if the Function Mask bit is cleared, the Vector Mask bits are used.

The remaining bits are used for Steering hints (bits 31:16) or reserved (bits 15:1).

There is one bit in the Pending Bit Structure (also called Pending Bit Array and PBA) for each entry in the MSI-X Table Structure. For each interrupt vector masked off, the controller cannot interrupt. If it has an interrupt to send, it will set the corresponding bit in the PBA.

The MSI-X Table Structure is read/write by the host; the PBA is read only.

PCIe Extended Configuration Capability

To provide more status, capabilities and control, PCIe created the PCIe Extended Capability structures. Table 5 provides a list of these PCIe capability structures.

Note that Capability ID 2 and 9 both are named the same. To maintained backward compatibility the PCIe specification authors left Capability 2 stand as it was when they needed to add new features and created Capability 9. The rule is that if Capability 8 is implemented the function must implement Capability 9; if Capability 8 is not implemented, then the function must implement Capability 2 to define is virtual channels.

Always be sure to use the latest specification available. Capability 19h was defined in PCI Express Base 3.0 and has major updates in 3.1. Capability 1B through 22h were added 3.1.

Capability ID	Name
0001h	Advanced Error Reporting
0002h	Virtual Channel Capability
0003h	Device Serial Number Capability
0004h	Power Budgeting Capability
0005h	Root Complex Link Declaration Capability
0006h	Root Complex Internal Link Control
0007h	Root Complex Event Collector Capability
0008h	Multi-function Virtual Channel Capability
0009h	Virtual Channel Capability
000Ah	RCRB (Root Complex Register Block) Header Capability
000Bh	Vendor Specific Capability
000Dh	ACS (Access Control) Extended Capability
000Eh	ARI (Alternative Routing ID Interpretation) Capability
0010h	SR-IOV
0011h	MR-IOV
0012h	Multicast Capability
0015h	Resizable BAR Capability
0016h	Dynamic Power Allocation Capability
0017h	TPH (TLP Processing Hints) Requester Capability
0018h	Latency Tolerance Reporting Capability
0019h	Secondary PCIe Extended Capability
001Bh	PASID Extended Capability
001Ch	LNR Extended Capability
001Dh	DPC Extended Capability
001Eh	L1 Power Management Substates Extended Capability
001Fh	Precision Time Management (PTM) Capability
0020h	M-PCIe Extended Capability
0021h	Function Readiness Status (FRS) Queuing Extended Capability
0022h	Readiness Time Reporting Extended Capability

Table 5: PCIe Extended Capabilities

PCIe Transaction Processing

PCIe Transaction Addressing

PCIe defines three methods of addressing. ID Routing allows the host to address a specific Function on a Bus to execute PCIe transactions or for the Function to tell the host which Function is posting ending status.

Memory addressing allows the host to define to the device a memory address for the Function to transfer data to or from for read commands or write commands. PCIe can make use of the full 64-bit addressing range available, and it not limited to the DRAM memory in the host.

Implicit addressing is used only for messages. The destination is implied by the message type so ID Routing and Memory addressing are not required. A broadcast message goes to all downstream (away from the root complex) Functions. A message from a Function goes upstream to the host.

ID Routing

Bus - Device - Function (BDF) Addressing

This addressing method was called Bus-Device-Function (BDF) Addressing.

Configuration Reads and Writes and all completions use ID Routing. Figure 14 shows an example of a PCIe computer with its Root Complex and three downstream (going away from the Root Complex) ports. The left most port and the right most port are connected to switches which, in turn, are connected to endpoints with Functions. The endpoint connected to Bus 4 has two Functions and the endpoint connected to Bus 9 has three Functions. The other endpoints have a single Function each. A PCIe Function is equivalent to an NVMe controller.

The Buses are number during initialization, starting with the Root Complex internal Bus being addressed as Bus 0. The other Buses are numbered sequentially as they are discovered. The discovery process goes deep before it goes wide. Note that the discovery and enumeration process finds and numbers all the Buses associated with Switch A before it moves to the next Root Complex port.

If this was PCI or PCI-X, each device as it was discovered would be assigned the next sequential device number (hence the “D” name in BDF addressing). PCIe eliminated the device address, and added Alternate Routing Interpretation (ARI) to allow for 256 Function per endpoint on each bus (hence the name change to ID Routing added in PCIe 1).

Function numbers are assigned at the time of manufacture by the manufacturer. They are discoverable by the host, but not changeable. Legacy systems can only address functions 0-7; PCIe systems can address functions 0-256.

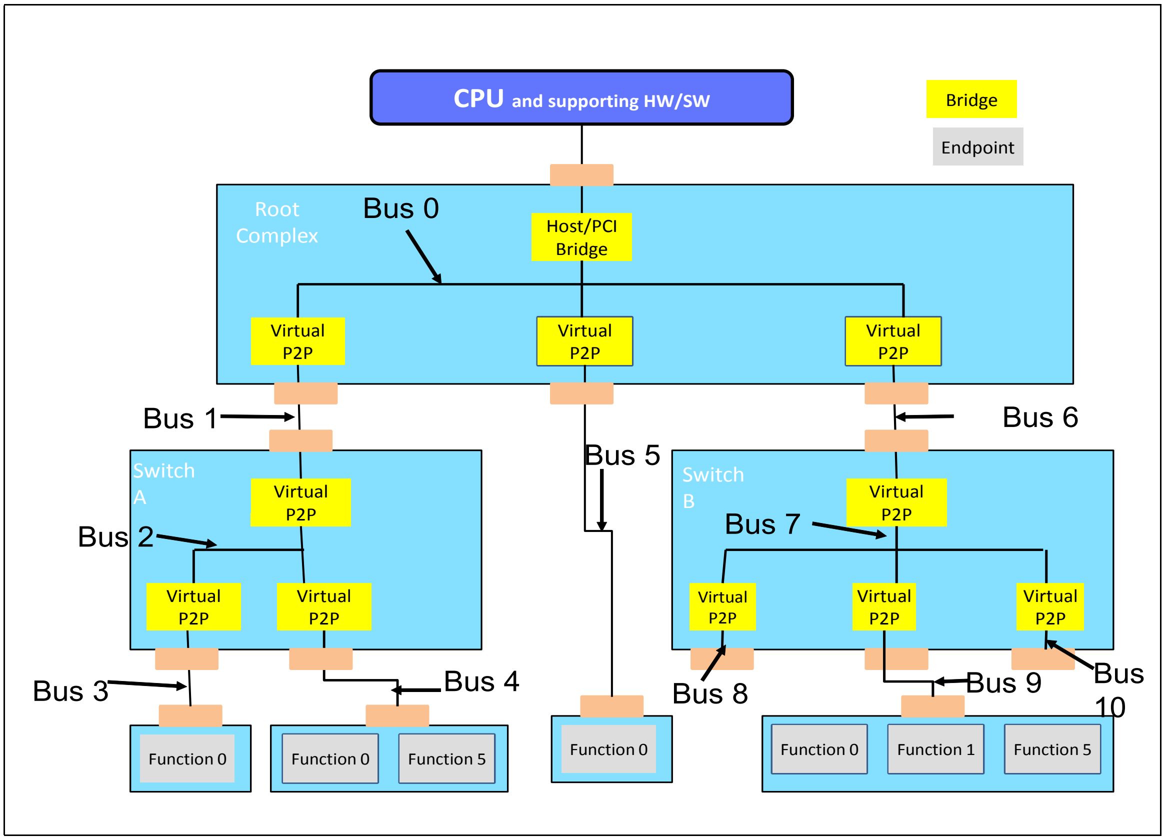


Figure 14: BDF Topology

Figure 15 shows the Transaction Layer Packet (TLP) header when using Configuration Read and Configuration Write transactions. Fields left blank are not applicable for this discussion; most of them are defined for use in other circumstances.

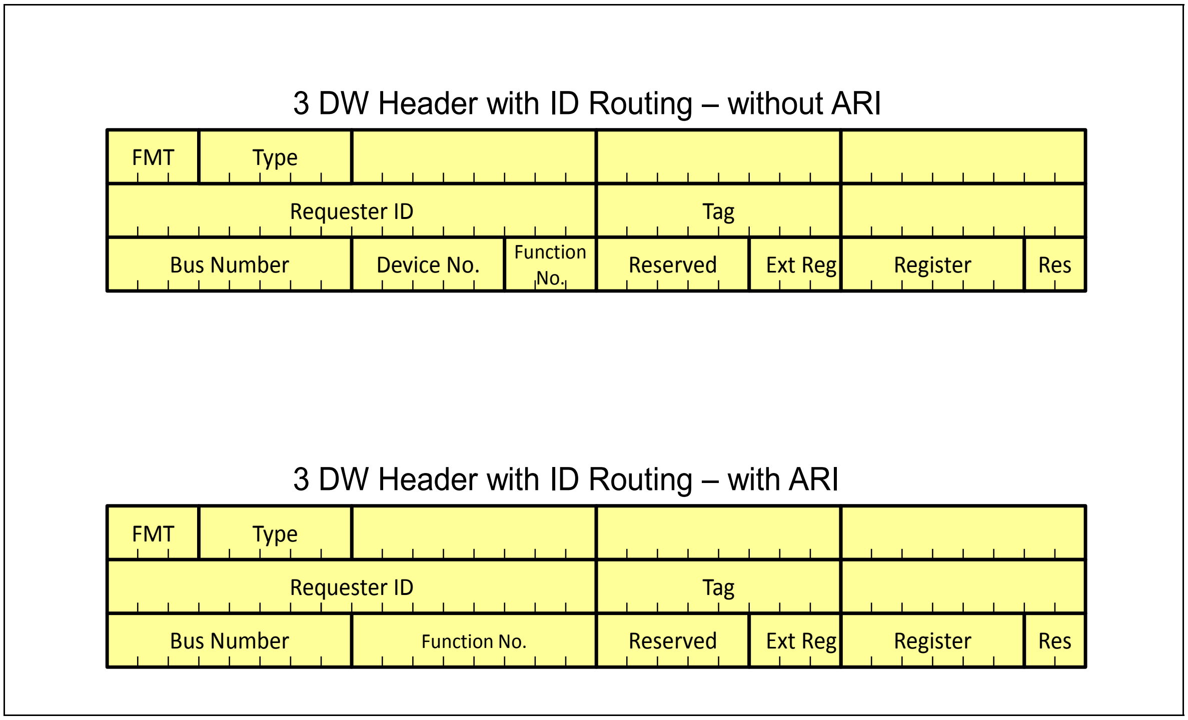


Figure 15: Configuration Request TLP Header

The format and type field are combined to identify this transaction type and format.

The Requester ID is the ID routing address of the source of this TLP. It identifies to whom the Completion Transaction will be sent.

The Tag field is a 8-bit number assigned to this request. The tag numbers of all outstanding requests between this source and destination must be unique. Each transaction will contain the tag to identify which request it answers. Posted transactions do not require a tag.

The top TLP in Figure 15 shows the older BDF addressing in the 3rd DWord. When PCIe eliminated the need for the 5-bit Device address, those bits became available to expand the Function address - as shown in the bottom TLP of Figure 15. This is called Alternate Routing Interpretation (ARI).

The offset into the Configuration Space is indicated by the concatenation of the contents of the Ext Reg and Register fields. The Res field is part of the Register field and ensures all register reads are on a DWord boundary.

Memory Addressing

Memory addressing is to pass information between the Function and host memory. To execute an NVMe read command, the host will send the read command to the device specifying the memory address via PRP or SGL and perhaps the Metadata address. The Function will generate a PCIe memory write transaction and place the requested data in memory starting at the given address. In a similar fashion, to execute an NVMe write command, the Function will generate a PCIe memory read transaction to get the information from memory.

Read commands and write commands to access user data will address memory within the address range available to the application programs. This will be limited by the size of the system's physical memory.

Other information such as configuration information or MSI or MSI-X tables may be within the 64-bit address range, but greater than the host physical memory limit. The physical memory in this case is located on the device, but addressed within the host's memory address range. Each function has a set of six Base Address Registers (BAR) that can be configured by the host to indicate the starting address for a range of memory locations the host can address. The host will set the BARs in each function to different addresses (and remember which function uses what addresses) so the host can access that space on the device.

These six BARs were originally specified to provide six 32-bit base addresses, but now can be grouped to provide three 64-bit addresses. Interestingly, in NVMe, BAR 0 and BAR 1 are combined to produce a 64-bit address capability; but BAR 2 is used as a 32-bit register called the IDBAR providing an optional Index/Data pair for I/O based accesses.

During PCIe initialization the Function will request a certain amount of memory it needs. The host memory manager will assign the memory size and base address by writing into registers on the device. All NVMe functions have their own BAR0 and BAR1 for the host to write where that function's memory begins. Every PCIe Function (NVMe Controller) will have a different base address in BAR0/BAR1.

Figure 16 shows the:

64-bit memory address range available to the host,
The system physical memory limit (labeled DRAM Limit),
The system Paging memory limit,
System memory available memory for Device 1:Function 0,
System memory available for Device 2:Function 0
System memory available for Device 3:Function 0, and
System memory available for Device 3:Function 1.

Note about Figure 16: The term “Device X” means “the device connected to bus X.”

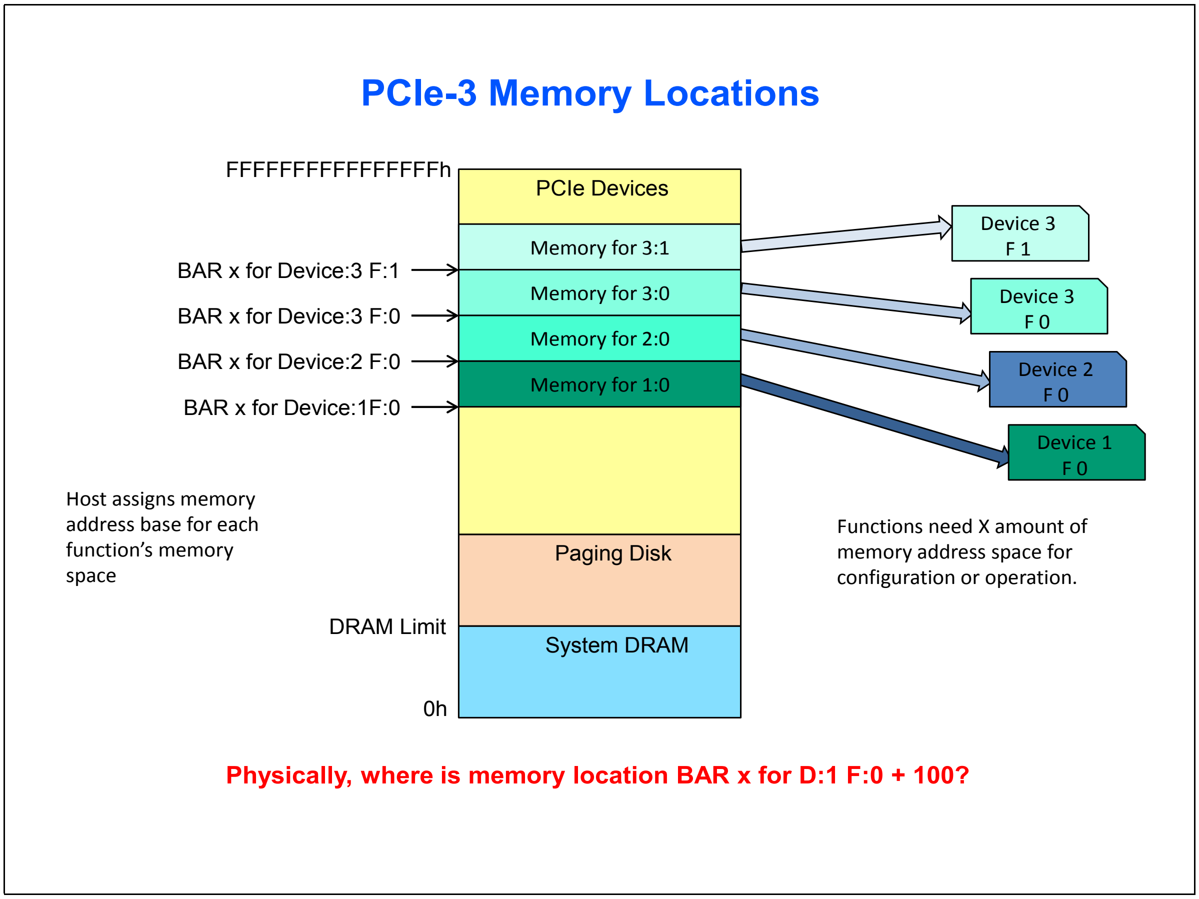


Figure 16: PCIe Memory Locations

Figure 17 shows how the host or function uses a Transaction Layer Packet header to addresses a memory location. The figure shows both 64-bit and 32-bit formats.

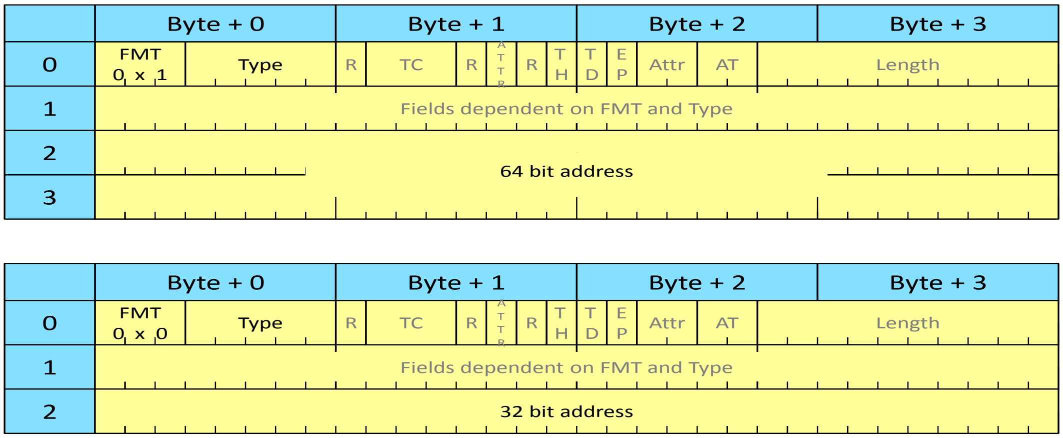


Figure 17: Memory Addressing

Implicit Addressing

Used only for certain messages. Implicit addressed messages from the host are broadcast to all Functions. Implicit addressed messages from a Function are for the host only.

PCIe Transactions

The primary means of communication between two PCIe devices across the PCIe bus is PCIe Transactions. The summary list of transactions is shown in Table 6 on page 32. PCI and PCI-X also defined I/O Read and Write but PCIe discourages their use.

Memory Write and the Messages are called “Posted” access because they do not have a response to the Transaction Layer. All others are “Non-posted” because the do have a response. All read type commands have a response because the Function must return the read data. For Configuration Write and I/O Write the Function returns ending status. Even on Posted transactions, ACKs or NAKs are returned, but only to the link layer.

Transaction type	Access	Address
Memory Read	PCIe memory space	Memory address
Memory Write	PCIe memory space	Memory address
Configuration Read	Device configuration space	Bus-Device-Function
Configuration Write	Device Configuration space	Bus-Device-Function
Message without Data		Any
Message with Data		Any

Table 6: Summary of PCIe Transactions

NVMe over PCIe Transactions

Figure 18 shows the complete TLP. All fields are optional except the header.

Prefixes are used for Processing Hints (when and how will this data be used soon?), Muli-Root - Input Output Virtualization (MR-IOV), and Process Address Space ID. If these prefixes are used, it is outside the scope of NVMe.

The header contains the Requester ID and the Destination ID (Bus - Function) for ID Routing or the Memory address. It also identifies the Transaction Type.

The Data Payload contains the NVMe command, status or data.

The TLP Digest is the optional ECRC which verifies the integrity of the header and payload of the TLP.

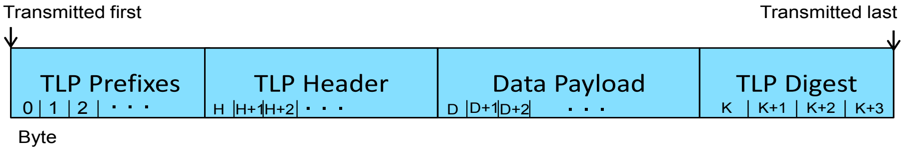


Figure 18: Complete TLP

The host will build the queues and configure the NVMe controller during initialization. To process a command:

The host will build the command, place it in a slot of the Submission Queue (SQ) and notify the controller that a command is waiting by writing the new tail pointer into the controller's SQ tail doorbell.

The controller will fetch the command by issuing a PCIe memory read transaction. The memory address will be (SQ Base Address + (slot number * slot size)).

If the command transfers data, the data address will be in the command. The controller will issue a PCIe Memory Read or Memory Write PCIe transaction the address given in the command.

The controller will then build the completion status and use a PCIe Memory Write transaction to write the status to the Completion Queue (CQ) slot pointed to by the CQ tail pointer which the controller maintains.

The controller will interrupt the host using the appropriate interrupt mechanism, INT A/B/C/D, MSI or MSI-X.

This is a very high level overview. Much more is happening on both the PCIe and the NVMe interfaces. To get the rest of the NVMe story, please continue with the next six chapters.

PCIe Packets

PCIe defines three types of packets:

Transaction Layer Packets (TLP) that go from transaction layer of the source device to the transaction layer of the destination device,

Data Link Layer Packets (DLLP) that go across the physical link only, and

Ordered Sets which go from one Physical Layer to the next Physical Layer.

TLP

The Transaction layer receives the data to be transmitted from the application layer or the operating system above the Transaction Layer. The Transaction layer adds the header (which includes the destination address) of either 3 or 4 DWords. If appropriate, the Transaction layer adds the optional End-to-End CRC[3] (ECRC) which is intended to remain with the TLP until it reaches the destination. However, the ECRC may be updated or eliminated en route if the address changes such as with the Multi-cast Overlay function.

The Data Link layer will add a sequence number and a mandatory Link CRC (LCRC). The sequence number is to assist in error recovery. If the Transaction Layer packet has to transition across several links, the sequence number is assigned by each sending link layer. The LCRC will be recalculated for each link.

The Physical Layer will add a Start TLP (STP) symbol and an END symbol[4]. These symbols are quite different between the:

8b/10b encoding of Gen 1 and Gen 2, (called K-characters) and
128b/130b encoding of Gen 3 (called tokens).

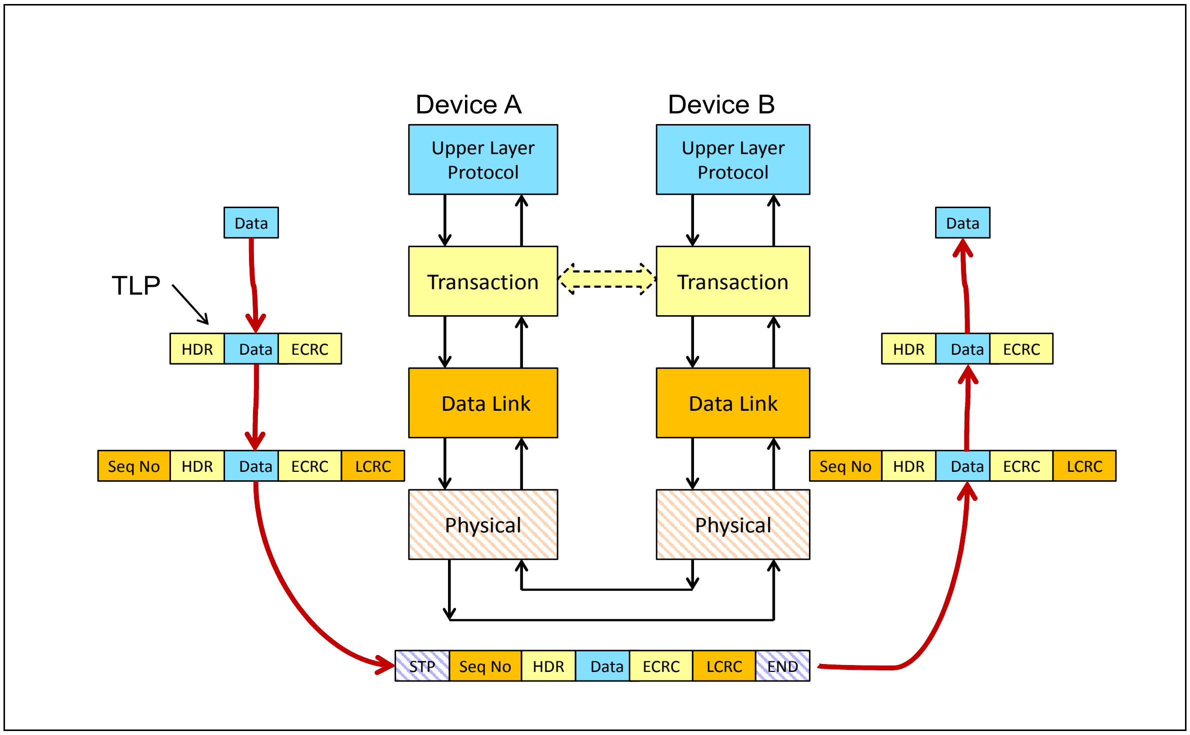


Figure 19: TLP Flow

Notice on Figure 19 how the different items are added by each layer going down the left side (transmit from source) and removed going up the right side (received by destination). The only optional item in the figure is the ECRC, if it is not being used, it just won't be there.

[3] This is different from the End-to-end Protection Information Guard CRC used in DIF or DIX.
[4] The END symbol is used only for 8b/10b encoding

DLLP

There are 16 different 6-byte packets generated by the Link Layer used for link initialization or control. DLLP's never go to the transaction layer. The packets are:
ACK and NAK,
Power control,
Flow control, and
Initialization for the link.

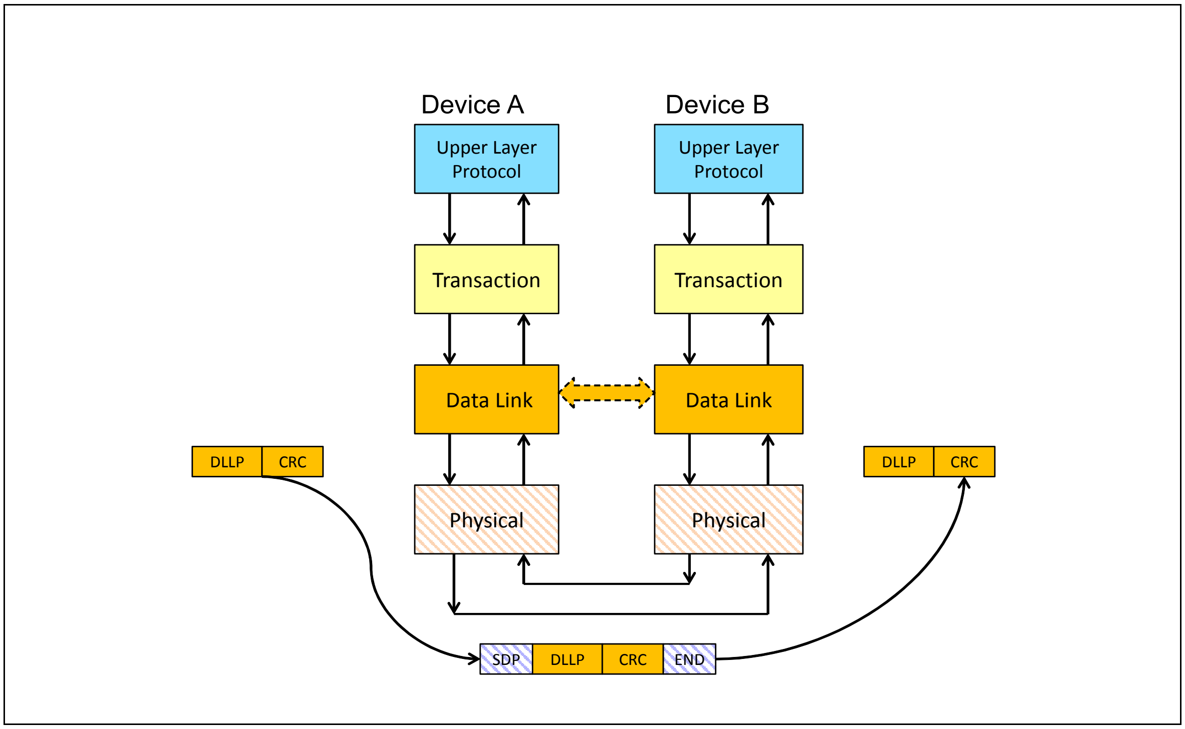


Figure 20: DLLP Flow

Figure 20 shows the flow of the DLLP from the transmit side Data Link Layer through the Physical layer, transport media, receive side Physical Layer and receiving Data Link Layer. The two byte CRC (part of the 6 bytes of DLLP) is used for integrity checking.

Notice that the Physical Layer adds a Start of DLLP (SDP) instead of the Start TLP (STP) for the TLP.

Ordered Sets

Ordered Sets are used by the Physical Layers to:

deskew lanes and elastic buffer management,
enter and exit electrical idle low power state, and
achieve bit synchronization during initialization and exiting low power states.

Figure 21 shows the flow of ordered sets from Physical Layer to Physical Layer.

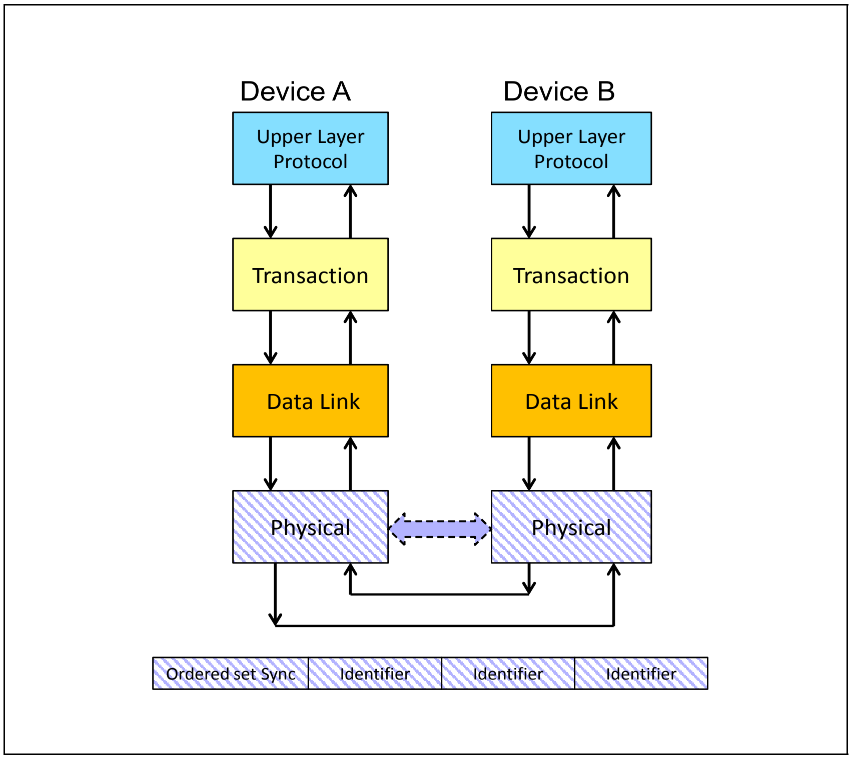


Figure 21: Ordered Sets

Chapter Summary

- PCIe Header and Capability Structures
  - PCIe Configuration Space
    - Configuration Headers
    - Capability Structures
    - PCIe Extended Configuration Capability
- PCIe Transaction Processing
  - PCIe Transaction Addressing
    - ID Routing
    - Bus - Device - Function (BDF) Addressing
    - Memory Addressing
    - Implicit Addressing
  - PCIe Transactions
    - NVMe over PCIe Transactions
  - PCIe Packets
    - TLP
    - DLLP
    - Ordered Sets

NVMe Registers

Chapter Contents

NVMe Controller Registers

Find the Registers

Step 1. PCIe Enumeration and Discovery. This assigns Bus addresses (ID Routing) to each Function and loads the Primary, Secondary, and Subordinate Bus Numbers in the Bridge Configuration space. Also, the PCIe host will read the configuration header of each Function to determine its type and capabilities, and set the BARs as necessary.

Step 2. NVMe software driver will read the PCIe Configuration Header of each NVMe capable Function. BAR 0 and Bar 1 (MLBAR/MUBAR) provide a 64 bit address for the beginning of the controller's[5] memory. The controller registers are at that 64 bit PCIe memory space address. Their physical location is on the controller.

Controller Registers List

All controller registers not defined and all reserved bits with registers are read-only and return 0h when read. Software shall not rely on 0h being returned.

The details of each register is given in the following sections. The registers listed in blue and underlined have hyperlinks to an explanation in this text of that register and details of its contents.

Remembering the Symbol listed for each register in Table 7 and the Symbol for each field in the break down of the registers will make the reading of the NVMe specification and this text much easier.

[5] An NVMe controller is equivalent to a PCIe function.

Start (h)	Length in bytes (h)	Symbol	Name
0	8	CAP	Controller Capability
8	4	VS	Version
C	4	INTMS	Interrupt Mask Set
10	4	INTMC	Interrupt Mask Clear
14	4	CC	Controller Configuration
18	4	R	Reserved
1C	4	CSTS	Controller Status
20	4	NSSR	NVM Subsystem Reset (optional)
24	4	AQA	Admin Queue Attributes
28	8	ASQ	Admin Submission Queue Base Address
30	8	ACQ	Admin Completion Queue Bass Address
38	4	CMBLOC	Controller Memory Buffer Location
3C	4	CMBSZ	Controller Memory Buffer Size
40	EC0	R	Reserved
F00	100		Command Set Specific
1000	4	SQ0TDBL	SQ 0 Tail Doorbell (Admin)
1000h + (1*(4<<CAP.DSTRD))	4	CQ0HDBL	CQ 0 Head Doorbell (Admin)
1000h + (2n*(4<<CAP.DSTRD))	4	SQnTDBL	SQ n Tail Doorbell
1000h + ((2n+1)*(4<<CAP.DSTRD))	4	CQnHDBL	CQ n Head Doorbell

Table 7: Controller Registers

The Controller Capability Register is a read-only register, set at the time of manufacturer or possibility with changing firmware levels, and defines the capability of the controller. The Controller Configuration Register is a write-read register that the host can set to turn on or off, or adjust various capabilities of the controller. The Controller Status Register is read-only from the host standpoint and contains current status of the controller.

The Admin Queue Attributes, the Admin SQ Base Address register, and the Admin CQ Base Address register a are used to define and create the Admin Queues.

The Controller Memory Buffer Location and Size registers were added in NVMe 1.2 and define memory in the host memory address space that is physically on the controller. Placing the SQ in this buffer can make the NVMe operation more efficient.

The doorbells begin at address 1000h. Each SQ has a tail doorbell and each CQ has a head doorbell. Each doorbell is four bytes long. The host will write the updated head pointer or tail pointer into the appropriate doorbell. The controller must monitor these registers for any change. The controller may monitor using a hardware implementation or a software thread. For efficiency of memory management, the doorbells may be packed in memory (beginning of one doorbell to beginning of the next doorbell is four bytes) (hardware implementation) or on a cache line (beginning of each doorbell is on the beginning of a cache line) (software implementation). That spacing is called the doorbell stride (CAP.DSTRD). Common cache line size is currently 32 bytes so the doorbell stride can be from 4 bytes to 32 bytes (limited to powers of 2).

For SQ n's Tail Doorbell the formula is 1000h + (2n* (4<<CAP.DSTRD)). This could be read as "The address for SQ n's Tail Doorbell is 1000h + two times the queue number (n) times 4 shifted left the number of positions equal to CAP.DSTRD". CAP.DSTRD is defined in the CAP (capabilities) register as an exponent that when added to 2 and raised to a power of 2 gives a value of 4 to 32. Check CAP register bits 35:32 DSTRD. "The stride is specified as (2 ^ (2 + DSTRD)) in bytes." That means that CAP.DSTRD has a value of 0, 1, 2, or 3 for a stride of 4, 8, 16 or 32 bytes. From Table 7 assume a DSTRD of 3, then 4 (0100b) shifted left three bits is 32 (10 0000b). From Table 8 bits 35:32, 2 raised to the power of 2 + 3 = 32. The formula in Table 7 and Table 8 are different but give the same results. Note: the two different formulas are given in the NVM Specification in the tables as shown here.

The DSTRD is a Read Only field set in the controller at the time of manufacturer.

More information can be found in "Head pointer" and "Tail pointer" on page 156.

Controller Capabilities Register

Bits	Type	Reset	Symbol	Description
63:56	RO	0h		Reserved
55:52	RO	Impl Spec	MPSMAX	Memory Page Size Maximum $(2^{(12+MPSMAX)})$
51:48	RO	Impl Spec	MPSMIN	Memory Page Size Minimum $(2^{(12+MPSMIN)})$
47:45	RO	0h		Reserved
44:37	RO	0h	CSS	Command Sets Supported (bit 37 = NVMe)
36	RO	Impl Spec	NSSRS	NVM Subsystem Reset Supported
35:32	RO	Impl Spec	DSTRD	Doorbell Stride $(2^{(2+DSTRD)})$
31:24	RO	Impl Spec	TO	Timeout: Worst case time in 500 ms units for controller to become ready
23:19	RO	0h		Reserved
18:17	RO	Impl Spec	AMS	Arbitration Mechanism Supported bit 17 - Weighted Round Robin w/ Urgent bit 18 - Vendor Specific
16	RO	Impl Spec	CQR	Contiguous Queues Required
15:00	RO	Impl Spec	MQES	Maximum Queue Entries Supported

Table 8: Controller Capabilities Register

Version Register

The first nibble (4 bits) is the major version; the second nibble is the minor version.

For 1.0 Controllers, the Version Register is set to 0001 0000h.
For 1.1 Controllers, the Version Register is set to 0001 0100h.
For 1.2 Controllers, the Version Register is set to 0001 0200h.
For 1.2.1 Controllers, the Version Register is set to 0001 0201h

Controller Configuration Register

Bits	Type	Reset	Symbol	Description
31:24	RO	0h		Reserved
23:20	RW	0h	IOCQES	I/O Completion Queue Entry size in bytes, power of 2
19:16	RW	0h	IOSQES	I/O Submission Queue Entry size in bytes, power of 2
15:14	RW	0h	SHN	Shutdown Notification 00b - No notification 01b - Normal shutdown notification 10b - Abrupt shutdown notification
13:11	RW	0h	AMS	Arbitration Mechanism Selected 000b - Round Robin 001b - Weighted Round Robin w/Urgent 111b - Vendor Specific
10:07	RW	0h	MPS	Memory Page Size ($2^{(12+ MPS)}$)
06:04	RW	0h	CSS	I/O Command Set Selected NVM = 000b
03:01	RO	0h		Reserved
00	RW	0h	EN	Enable

Table 9: Controller Configuration Register

The Queue Entry size fields in the Controller Configuration Register specifies the number of entires as a power of 2. Each entry in the SQ is 64 bytes; each entry in the CQ is 16 bytes. All I/O SQs on the controller must have the same number of entries and all I/O CQs on the controller must have the same number of entries. The Admin SQ and Admin CQ are stipulated in the AQA Register.

For information on the Arbitration Mechanism Selected, see “Queue Priority” on page 6.

Controller Status Register

Bits	Type	Reset	Symbol	Description
31:06	RO	0h		Reserved
05	RO	0h	PP	Processing Paused
04	RW	H/W Init	NSSRO	NVM Subsystem Reset Occurred
03:02	RO	0h	SHST	Shutdown Status 00b - Normal Operation (no shutdown requested) 01b - Shutdown processing occurring 10b - Shutdown processing complete Before issuing commands after Shutdown complete, must first issue a reset.
01	RO	H/W Init	CFS	Controller Fatal Status
00	RO	0h	RDY	Ready

Table 10: Controller Status Register

The Controller Fatal Status (CFS) bit is set to 1b when the controller has a serious error that it cannot communicate to the host via the I/O or Admin CQ. CFS is not indicated with an interrupt. If the host detects timeout conditions or repeated errors, it should check the CFS bit. If set, the hosts should reset and re-initialize the controller.

“Ready” is defined in the NVMe specification as the EN bit is set to 1b and the controller is ready to accept writes to the SQ tail doorbell (ready to accept commands).

Interrupt Mask Set and Clear Registers

Interrupt Mask Set and Clear provide the host the ability to block certain interrupts from occurring until the host is ready. Each of these registers is 32 bits long and will work with pin-based interrupts or MSI. MSI-X provides similar facilities using bit 0 in Dword 3 of each vector entry.

The register bits are bit-mapped to the 32 possible MSI interrupts.
For each bit that is written to “1b” in the Interrupt Mask Set register (IVMS), that interrupt is masked so it cannot interrupt the host. For each bit that is written to “1b” in the Interrupt Mask Clear register (IVMC), that interrupt mask is cleared so the controller may interrupt the host. Reading either register returns the current interrupt mask value in the controller, not the value of the IVMS or IVMC register.

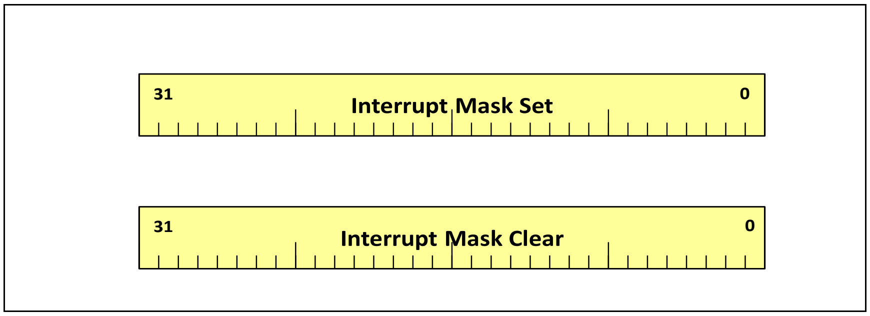


Figure 22: Interrupt Set and Clear Registers

Controller Reset Register

A write of the value 4E564D65h ("NVMe") to this field initiates an NVM Subsystem Reset. A write of any other value has no functional effect on the operation of the NVM subsystem. This register shall return the value 0h when read.

Controller Memory Buffer

Before NVMe 1.2, the following were stored in host physical memory:

data and metadata for read commands,
data and metadata for write commands,
SQ and CQ, and
PRP and SGL lists.

Beginning with NVMe 1.2, the optional Controller Memory Buffer (CMB) feature allows the host to define an area buffer in the controller to hold some or all of the mentioned data.

If the host puts this data in a buffer that is on the device, the device does not have to use PCIe facilities to fetch the data or send it to the host. The addressing of the storage facilities is part of the host address range and identified by the Base Address Registers in the device's Configuration Header.

Another benefit of using the memory in each controller is that it is more scalable. As we add more devices, they bring their own memory rather than take memory from the host DRAM.

The location of the buffer in the host address space is given with the Controller Memory Buffer Location register (CMBLOC), at location 38h in the controller registers, and shown in Table 11 below.

Bits	Type	Reset	Description
31:12	RO	Impl Spec	Offset from BAR
11:3	RO	0h	Reserved
2:0	RO	Impl Spec	BAR Indicator Register

Table 11: CMBLOC Register

To find the location, multiple the Offset from BAR times the size units in the Controller Memory Buffer Size register and add the product to the address in the BAR indicated by the Bar Indicator Register.

The size and purpose of the Controller Memory Buffer is given in the CMBSZ register shown in Table 12 below. Multiplying the contents of bits 31:12 times bits 11:8 gives the memory buffer size in bytes (KB = 1024 bytes). Bits 4:0 tell what kind of data may be stored in the CMB.

Bits	Type	Reset	Description
31:12	RO	Impl Spec	Indicates memory buffer size in multiples of the Size Unit
11:8	RO	Impl Spec	Indicates granularity of size field
			Value - Granularity
			0h - 4KB
			1h - 64KB
			2h - 1MB
			3h - 16MB
			4h - 256MB
			5h - 4GB
			6h - 64GB
7:5	RO	0h	Reserved
4	RO	Impl Spec	WDS - Write Data Support - Controller supports host storing write data and metadata in CMB
3	RO	Impl Spec	RDS - Read Data Support - Controller supports host storing read data and metadata in CMB
2	RO	Impl Spec	PRP-SGL List support - Controller supports host storing PRP lists and SGL lists in CMB
1	RO	Impl Spec	CQ support - Controller supports host storing CQs in CMB
0	RO	Impl Spec	SQ Support- Controller supports host storing SQs in CMB

Table 12: CMBSZ Register

Either Admin or I/O queues may be placed in either host memory or Controller Memory Buffer, however for each queue, the entire queue must be only in host memory or in controller memory. If placed in controller memory the queues are required to be physically contiguous.

PRP or SGL lists may be placed in either host memory or controller memory. However, the list may be placed in controller memory only if the associated command is in an SQ in the controller memory.

All data or metadata associated with a particular command shall be wholly in host memory or controller memory.

Creating the Admin Queues

Before any user data can be transferred between the host and controller, the I/O queues must be built. They are built using the Admin queues so first the Admin queues must be built. To build the Admin queues, the host will write to the Controller Registers. See Figure 23.

Each queue must have a base address, therefore set an address in the ASQ register and a different address in the ACQ register. Note: these are 64-bit registers so the queues can be located anyplace in the PCIe address space.

The submission queue must have a tail doorbell, therefore set the address in SQ0TDBL. The completion queue must have a head doorbell, therefore set the address in CQ0HDBL.

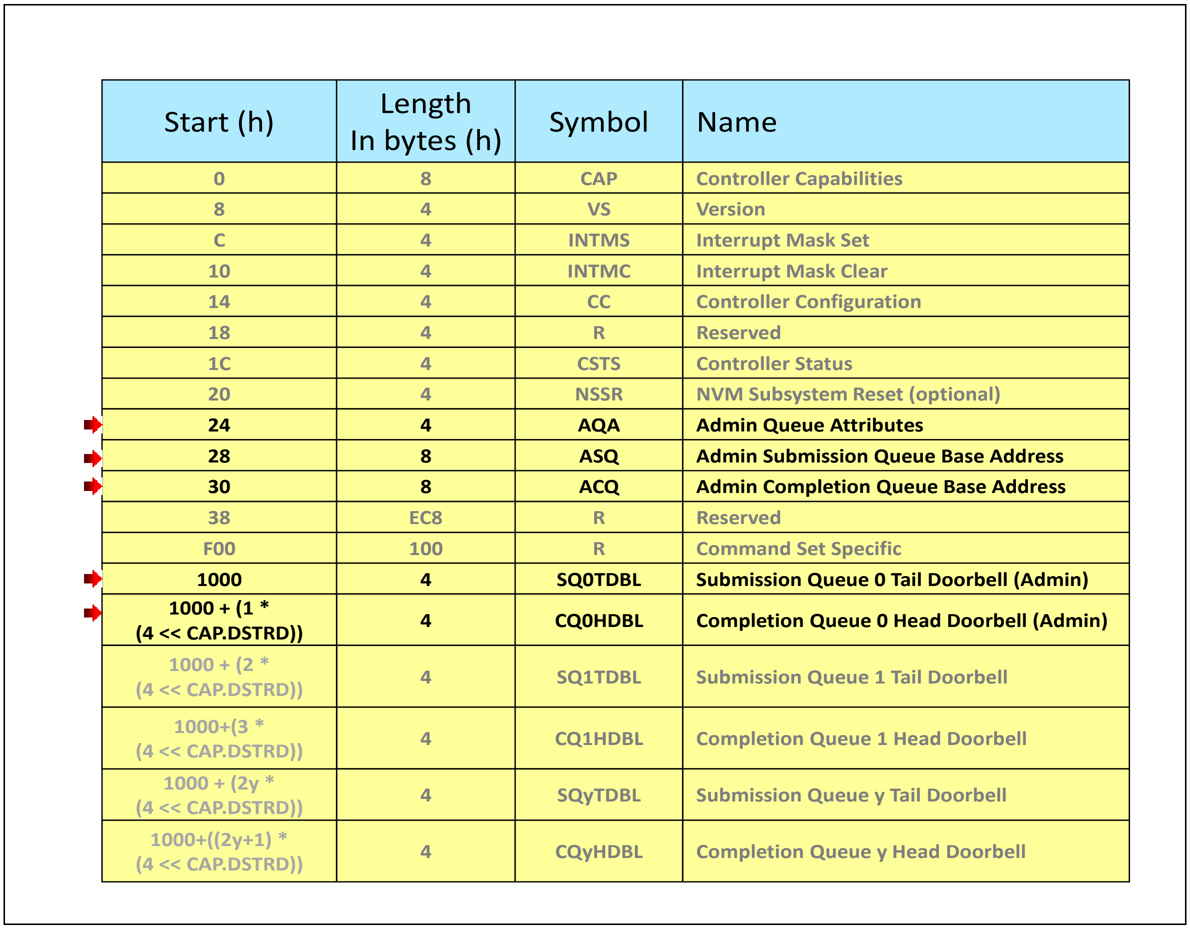
The host will define the number of entries in each Admin Queue by setting the size in the Admin Queue Attributes Register (AQA).

Bits	Type	Reset	Symbol	Description
31:28	RO	0h		Reserved
27:16	RW	0h	ACQS	Admin Completion Queue Size
15:12	RO	0h		Reserved
11:00	RW	0h	ASQS	Admin Submission Queue Size

Table 13 : Admin Queue Attributes Register

The size values are the number of entries in the queue. It is a zero based value, minimum number of entries is 2 (size = 1), maximum number is 4096 (size = 0FFFh).

Registers used



Start (h)	Length In bytes (h)	Symbol	Name
0	8	CAP	Controller Capabilities
8	4	VS	Version
C	4	INTMS	Interrupt Mask Set
10	4	INTMC	Interrupt Mask Clear
14	4	CC	Controller Configuration
18	4	R	Reserved
1C	4	CSTS	Controller Status
20	4	NSSR	NVM Subsystem Reset (optional)
24	**4**	**AQA**	**Admin Queue Attributes**
28	**8**	**ASQ**	**Admin Submission Queue Base Address**
30	**8**	**ACQ**	**Admin Completion Queue Base Address**
38	EC8	R	Reserved
F00	100	R	Command Set Specific
1000	**4**	**SQ0TDBL**	**Submission Queue 0 Tail Doorbell (Admin)**
1000 + (1 * (4 << CAP.DSTRD))	**4**	**CQ0HDBL**	**Completion Queue 0 Head Doorbell (Admin)**
1000 + (2 * (4 << CAP.DSTRD))	4	SQ1TDBL	Submission Queue 1 Tail Doorbell
1000+(3 * (4 << CAP.DSTRD))	4	CQ1HDBL	Completion Queue 1 Head Doorbell
1000 + (2y * (4 << CAP.DSTRD))	4	SQyTDBL	Submission Queue y Tail Doorbell
1000+((2y+1) * (4 << CAP.DSTRD))	4	CQyHDBL	Completion Queue y Head Doorbell

Figure 23: Controller Registers for Admin Queues

Procedure to build Admin Queues

To build the Admin queues, execute the following steps. Figure 24 below shows the major steps.

Step 1. PCIe discovers device, reads Config Header and writes BARs.

Step 2. NVMe S/W reads PCIe Config Header offset 09h (class code).

Step 3. NVMe S/W reads BARs to find NVMe registers.

Step 4. NVMe S/W waits for any previous reset to complete by monitoring CSTS.RDY for “0b”.

Step 5. (Figure 24, NVM2) NVMe S/W writes to offset 14h to disable NVMe Command set. See “: Controller Configuration Register” on page 43. This step is not given in NVMe specification 1.x, but is captured in the Initialization trace in Figure 24.

Step 6. (Figure 24, NVM4) NVMe S/W writes to offset 24h (Admin Queue Attributes). See “: Admin Queue Attributes Register” on page 47.

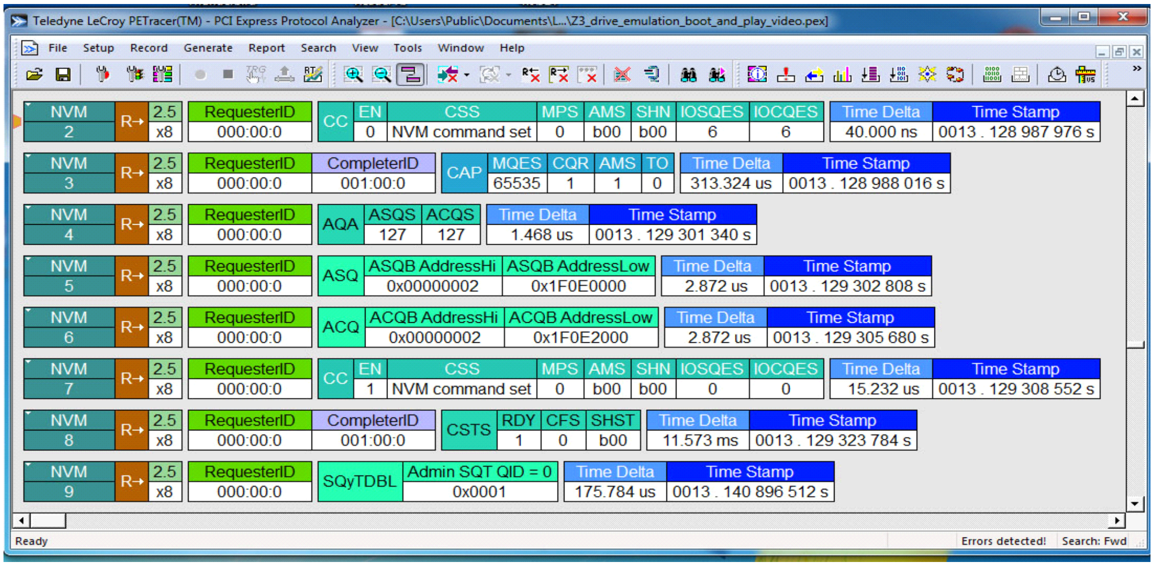
Step 7. (Figure 24, NVM5) NVMe S/W writes to offset 28 (Admin SQ base address).

Step 8. (Figure 24, NVM 6) NVMe S/W writes to offset 30 (Admin CQ base address).

Step 9. (Figure 24, NVM 7) NMVe S/W enables the controller by setting CC.EN = 1b.

Step 10. (Figure 24, NVM 8) NVMe S/W polls CSTS.RDY = 1b to indicate controller is ready.

Figure 24, NVM 9 shows the host has placed a command on the Admin SQ and has written to the doorbell to let the controller know that the SQ tail is now set to 1.


Teledyne LeCroy PETracer(TM) - PCI Express Protocol Analyzer - [C:\Users\Public\Documents\L...\Z3_drive_emulation_boot_and_play_video.pex]
File
Setup
Record
Generate
Report
Search
View
Tools
Window
Help
Ready
Errors detected!
Search: Fwd

```
NVM 2 | R→ | 2.5 x8 | RequesterID 000:00:0 | CC | EN 0 | CSS NVM command set | MPS 0 | AMS b00 | SHN b00 | IOSQES 6 | IOCQES 6 | Time Delta 40.000 ns | Time Stamp 0013 . 128 987 976 s
NVM 3 | R→ | 2.5 x8 | RequesterID 000:00:0 | CompleterID 001:00:0 | CAP | MQES 65535 | CQR 1 | AMS 1 | TO 0 | Time Delta 313.324 us | Time Stamp 0013 . 128 988 016 s
NVM 4 | R→ | 2.5 x8 | RequesterID 000:00:0 | AQA | ASQS 127 | ACQS 127 | Time Delta 1.468 us | Time Stamp 0013 . 129 301 340 s
NVM 5 | R→ | 2.5 x8 | RequesterID 000:00:0 | ASQ | ASQB AddressHi 0x00000002 | ASQB AddressLow 0x1F0E0000 | Time Delta 2.872 us | Time Stamp 0013 . 129 302 808 s
NVM 6 | R→ | 2.5 x8 | RequesterID 000:00:0 | ACQ | ACQB AddressHi 0x00000002 | ACQB AddressLow 0x1F0E2000 | Time Delta 2.872 us | Time Stamp 0013 . 129 305 680 s
NVM 7 | R→ | 2.5 x8 | RequesterID 000:00:0 | CC | EN 1 | CSS NVM command set | MPS 0 | AMS b00 | SHN b00 | IOSQES 0 | IOCQES 0 | Time Delta 15.232 us | Time Stamp 0013 . 129 308 552 s
NVM 8 | R→ | 2.5 x8 | RequesterID 000:00:0 | CompleterID 001:00:0 | CSTS | RDY 1 | CFS 0 | SHST b00 | Time Delta 11.573 ms | Time Stamp 0013 . 129 323 784 s
NVM 9 | R→ | 2.5 x8 | RequesterID 000:00:0 | SQyTDBL | Admin SQT QID = 0 0x0001 | Time Delta 175.784 us | Time Stamp 0013 . 140 896 512 s
```

Figure 24: Protocol Trace of Creating Admin Queues

Chapter Summary

NVMe Controller Registers
- Find the Registers
- Controller Registers List
- Controller Capabilities Register
- Controller Configuration Register
- Controller Status Register
- Interrupt Mask Set and Clear Registers
- Controller Reset Register
- Creating the Admin Queues

Procedure to build Admin Queues

Common Command Fields

Chapter Contents

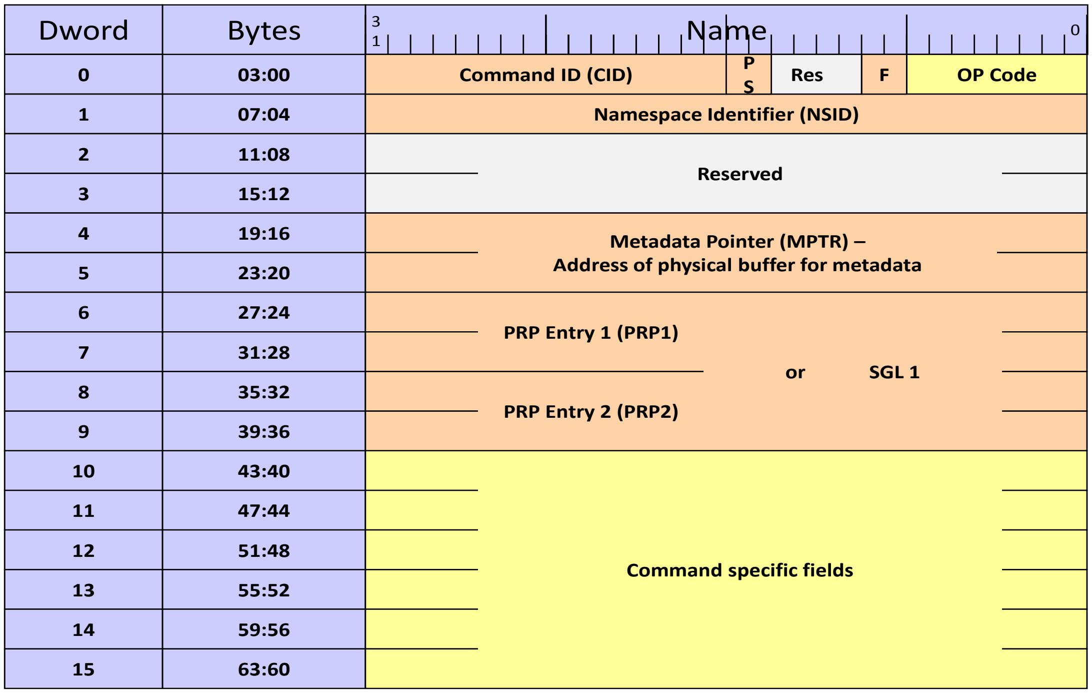
General Command Format

There are two types of commands:

- Admin - Administration of NVMe environment, and
- I/O - Management of data transfer between host and storage device.

A third type of command is Fabric Commands used in NVMe over Fabrics. They are defined in the editions of this book that include NVMe over Fabrics.

The format for both types of commands is the same. In this chapter, we cover the fields that are used the same or similar in both the Admin and I/O command sets. At the time of this book being published, the only I/O command set defined is NVMe. In this text, there are statements about I/O Command sets and the NVMe command set. Where the term I/O Command set is used, the intent it to include the NVMe command set and potentially any other command set defined in the future for this transport.



Dword	Bytes	31 Name				0
0	03:00	Command ID (CID)	PS	Res	F	OP Code
1	07:04	Namespace Identifier (NSID)				
2	11:08	Reserved				
3	15:12					
4	19:16	Metadata Pointer (MPTR) – Address of physical buffer for metadata				
5	23:20					
6	27:24	PRP Entry 1 (PRP1)				or SGL 1
7	31:28					
8	35:32	PRP Entry 2 (PRP2)				
9	39:36					
10	43:40	Command specific fields				
11	47:44					
12	51:48					
13	55:52					
14	59:56					
15	63:60					

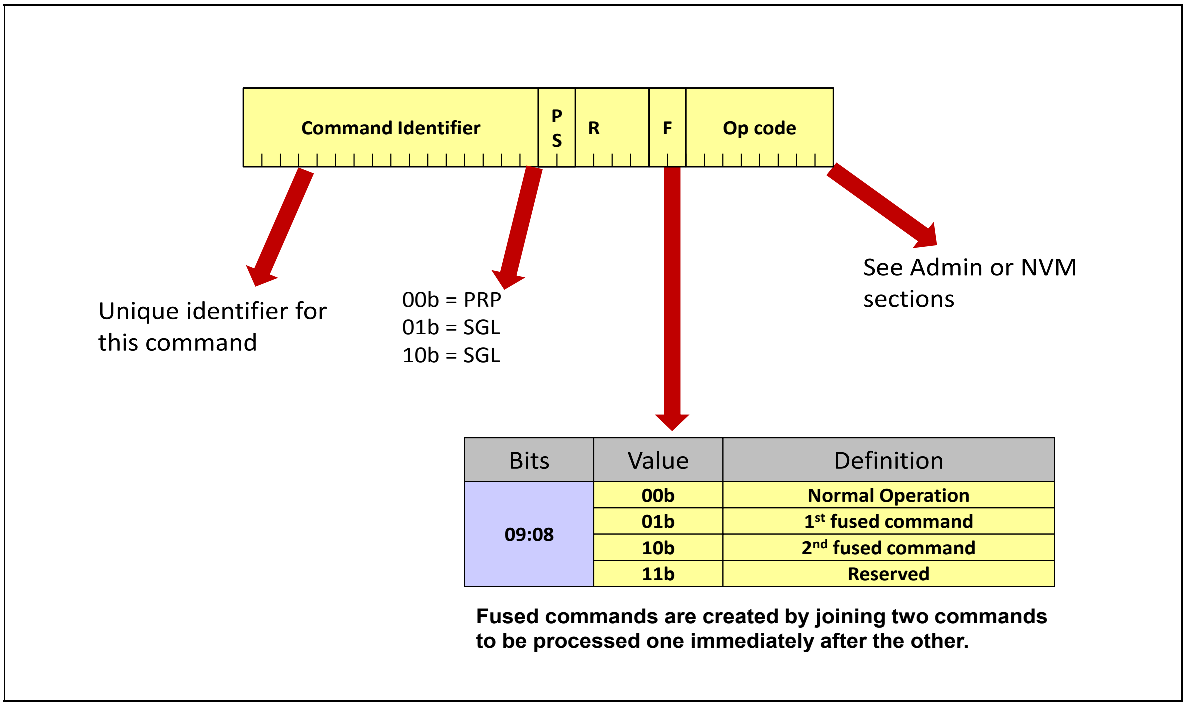
Figure 25: Admin and NVMe Command Format

Note on Figure 25 that the left column is DWords numbered 0 through 15 for a total of 16 DWords. The next column is Bytes numbers 0 through 63 for a total of 64 bytes. Both columns give the same information, one counts in DWords, the other in Bytes.

Each of the rows represents 32 bits, labeled 31 (on the left) to 0 (on the right).

In this figure, yellow represents fields that are Opcode (command) specific so will be covered in the next two chapters. Tan represents fields that are common to both command sets and will be covered in this chapter. Gray represents fields or bits that are reserved.

Command DWord 0



Bits	Value	Definition
09:08	00b	Normal Operation
	01b	1st fused command
	10b	2nd fused command
	11b	Reserved

Fused commands are created by joining two commands to be processed one immediately after the other.

Figure 26: Command DWord 0

Command Identifier

The command identifier is a 16-bit identifier unique for each command per SQ, therefore each SQ can have up to an architecture limit of 64k outstanding commands. When presenting status, the command identifier is concatenated with the SQ identifier to uniquely identify the command to which the status applies.

PRP/SGL

The PS bits defines if a PRP or SGL description is used for the data buffer. If the field contains 00b, then the data buffer, if any, uses the PRP description; See "PRP" on page 57. If the field contains 01b or 10b, then the data buffer, if any, uses the SGL description; See "SGL" on page 57.

Originally, this field was a single bit, but was expanded in Version 1.2. For the data buffer, there is no difference between 01b and 10b. However, a PS of 01b means the Metadata pointer (bytes 23:16 of the command) points to a physically contiguous buffer, and must be byte aligned. A PS of 10b means the Metadata pointer points to a SGL segment containing one SGL Descriptor and shall be QWord aligned.

If there is no data buffer for a command this field is ignored. Admin commands use only the PRP, I/O commands may use either the PRP or SGL. The PRP and SGL are defined later in this chapter.

Fused Operations

The F bit allows the application program to “fuse” two commands together. Normally, each command is processed independently of all other commands with no relationship between commands within a queue, or between queues. The fuse bits ensure that two commands are processed in order and the second command is processed immediately after the first (as though no other process took place between their execution).

If the first command fails, the second command will be aborted. If the second command fails, the completion status of the first command is sequence specific.

If the commands use a data buffer, the LBA range of both commands must be the same.
The bits indicate the order:

01b indicates the 1st command of the fused set,
10b indicates the 2nd command of the fused set, and
00b indicates normal (non-fused) command operations.

OpCode

The Opcode identifies operation the command is to perform. Opcodes values for Admin commands and I/O commands may overlap without causing a problem because Admin commands go to the Admin queues only and the I/O commands go to the I/O queues only.

Namespace Identifier (NSID)

A namespace[6] is a group of consecutive logical blocks. Which namespace the command addresses, if any, is indicated with the 32-bit NSID. Each namespace also has a configuration data structure of 4k-bytes that can be read with the Identify Namespace Admin command.

A Namespace and a Namespace ID are two different entities. The Namespace ID is an identifier for a Namespace, but it is not the Namespace itself.

A valid Namespace ID is any number from 1 to the value of the Number of Namespaces field in the Identify Controller data structure.

An inactive Namespace ID is a valid NSID that does not map to a namespace.

Private namespaces can be accessed by only one controller. Shared namespaces can be accessed by two or more controllers in an NVM subsystem.

[6] Namespaces is a word that has been used in programming languages for a while. Perhaps these definitions will help understand why NVMe defined Namespaces.
C++ - A namespace is a declarative region that provides a scope to the identifier (the names of types, functions, variables, etc) inside it. Namespaces are used to organize code into logical groups and to prevent name collisions that can occur.
Python - A namespace is a naming system for making names unique to avoid ambiguity.
XML - Namespaces provide a method to avoid element name conflicts. In XML, element names are defined by the developer. This often results in a conflict when trying to mix XML documents from different XML applications.
Mediawiki - Namespaces are a mechanism for grouping/separating wiki pages.

NVMe 1.2 provides the facility to create, attach, detach, and delete Namespaces. The commands allow the host to provide the size of the namespace and some characteristics for it. The Namespace ID and location of the Namespace are assigned by the controller.

Metadata

Metadata is information associated with each logical block that gives contextual information about that block. Example types of information are:

Creation date and time,
Expiration date and time,
Creating application program,
End-to-end data protection,
Search keys for search engines, and
Any other information or identifying data the client desires to use to define the data.

Metadata may use a separate memory buffer or be part of the user data as specified in the Identify Namespace Structure.

If the Metadata is in a separate buffer, the Metadata pointer of the command points to the memory buffer holding the Metadata. When the Namespace is formatted to use separate Metadata storage, the controller allocates separate Metadata area that is associated with each logical block. Therefore the command specifies the memory buffer and logical block for the data (both ends of the data transfer), and the command specifies the memory buffer of the Metadata. The Metadata area in the namespace is referenced using the associated logical block address.

If the Metadata is interleaved with the user data, the Metadata pointer is not used.

If the PS bits of DWord 0 of the command are 00b then bytes 23:16 of the command contain the address of the physically contiguous buffer of Metadata and must be DWord aligned.

If the PS bits are 01b then bytes 23:16 contain the address of the physically contiguous buffer of Metadata and must be byte aligned.

If the PS bits are 10b then bytes 23:16 contain the address of one SGL Descriptor and must be QWord aligned.

Parameters of the Format NVMe Admin command specify if the Metadata will be stored with the user data or in a separate storage location and also the size of the Metadata in bytes per logical block.

If the Metadata while in the host is interleaved with the user data, it will be transferred with the user data and stored on the NVMe media with the user data. If the Metadata while in the host is in a separate buffer pointed to with the Metadata field of the command, it will be transferred as a separate transfer and written separately but atomically when the user data is written.

Metadata was originally created to support Data Integrity Extensions (DIX), but is not limited to that function. This additional protection information, if present, is either the first eight bytes of Metadata or the last eight bytes of Metadata, based on the format of the namespace. For Metadata formats with more than eight bytes, if the protection information is contained within the first eight bytes of Metadata, then the 16-bit CRC called the Guard field does not cover any Metadata bytes. For Metadata formats with more than eight bytes, if the protection information is contained within the last eight bytes of Metadata, then the Guard fields covers all Metadata bytes up to but excluding these last eight bytes

Data Buffer Pointers

A large dataset or file may be fragmented in the computer memory. To make that dataset or file appear as a contiguous set of memory addresses to application programs, or for transfers between memory and storage devices, the host memory manager will create a table of pointers. Each pointer has at least the starting memory address of the next segment of data, plus the length of that segment. See Figure 27.

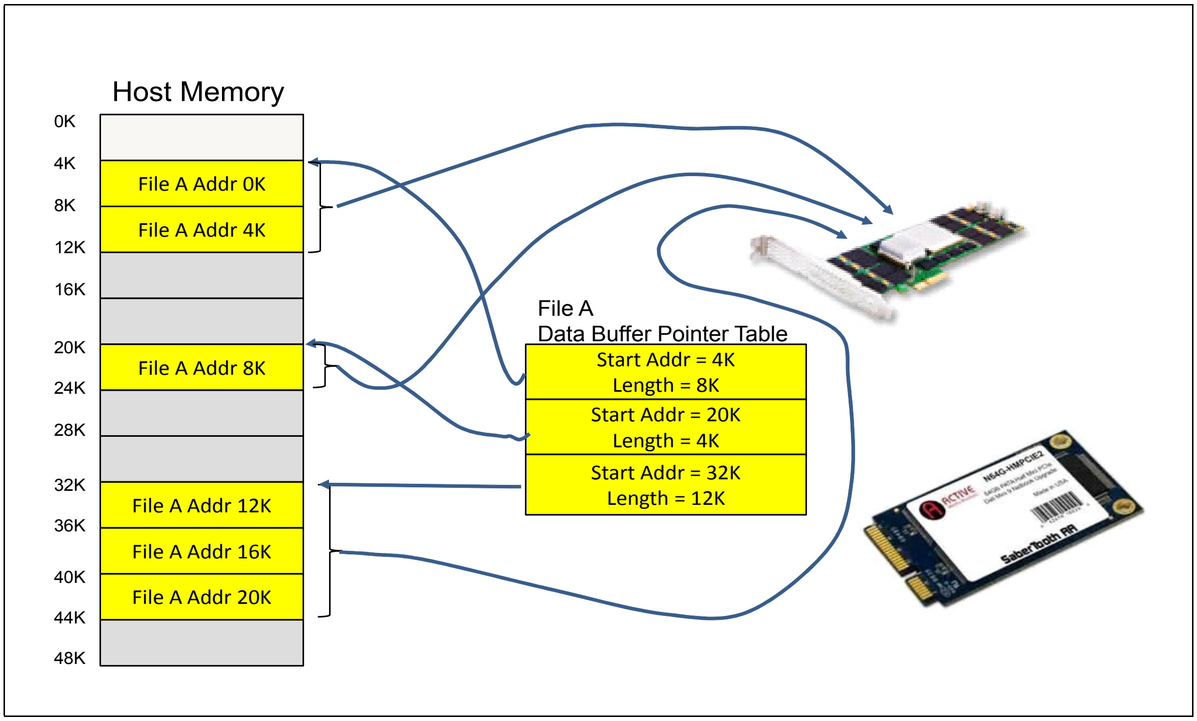
NVMe defines two structures. The simpler is called the Physical Region Page (PRP). The more flexible structure is called Scatter-Gather List (SGL).

Generic Example - Data Buffer Pointer Table

In Figure 27, on the left is the system memory containing File A plus other, non-related information. When File A was loaded into memory, there were already some pages with valid information that could not be overwritten, and other free pages interspersed. To write all of File A, the memory manager indicated three segments of free memory and built the Data Buffer Pointer Table to point to the three segments. Note that each entry in the table has a beginning memory address (here 4K, 20K and 32K) plus the length of the segment (here 8K, 4K and 12K).

To write this file to the storage device on the upper right of the picture, the host will issue one Write command giving the LBA address on the storage device. The memory manager will transfer the data as follows:

8K from memory address 4K,
4K from memory address 20K, then
12K from memory address 32K.



Start Addr = 4K Length = 8K
Start Addr = 20K Length = 4K
Start Addr = 32K Length = 12K

Figure 27: Generic Data Buffer Pointer Table

PRP

Figure 28 shows the 64-bit PRP data structure. The command allows for two PRP entries: the first located in DWords 6 and 7, and the second located in DWords 8 and 9. If there will be only one PRP entry for a command, it will be in DWords 6 and 7. If there will be exactly two PRP entries, the first entry will be in DWords 6 and 7, the second entry will be in DWords 8 and 9. If there will be more than two entries, then the DWords 6 and 7 will contain a normal PRP entry and DWords 8 and 9 will contain a pointer to a PRP table in memory[7].

PRP lists may be chained. There is no "End of List" indication. The end of the list is calculated by comparing the transfer length in the command and the PRP entries.

Admin commands use only the PRP for describing data buffers (never the SGL). In many cases, this text shows the PS bits as reserved for the Admin commands.

Within the Admin command set and the I/O command set some commands allow only one PRP entry, some commands allow a maximum of two entries and some commands allow the second entry to be a pointer to a PRP table.

All PRP entries are eight bytes long. They define a single memory page of size CC.MPS. The Page Base Address is the lowest address of this page.

The Offset is the number of bytes into this page for the beginning of the data buffer. Offset must be 0h on all pages except the first.

N indicates the page size, $2^{(n+1)}$. For 4096-byte pages, n = 11.

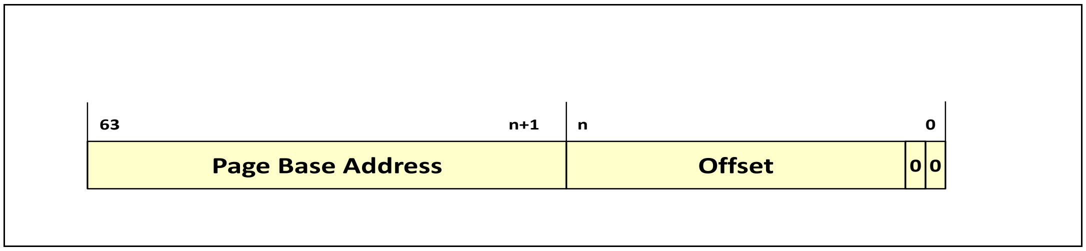
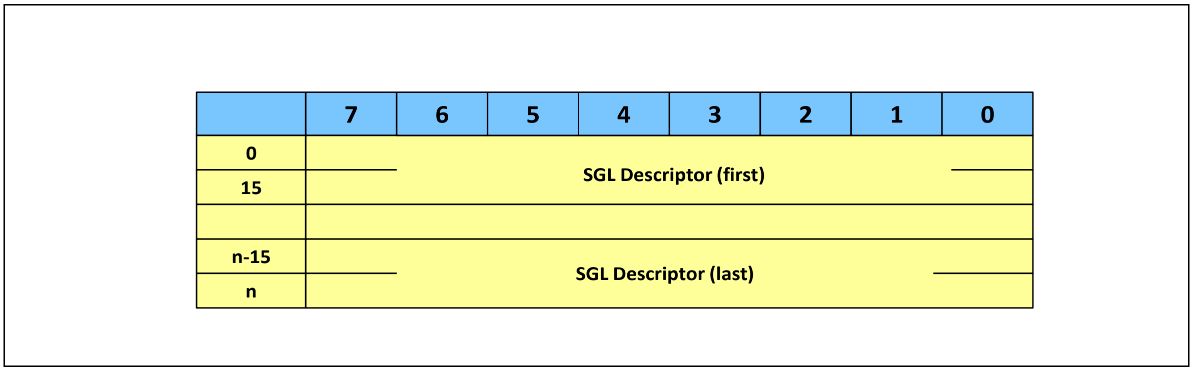


Figure 28: PRP Entry Format

SGL

Scatter-Gather Lists are more flexible (and more complex) than PRP. Each entry is 16 bytes long and is called a descriptor. One or more descriptors make up a segment. See Figure 29.

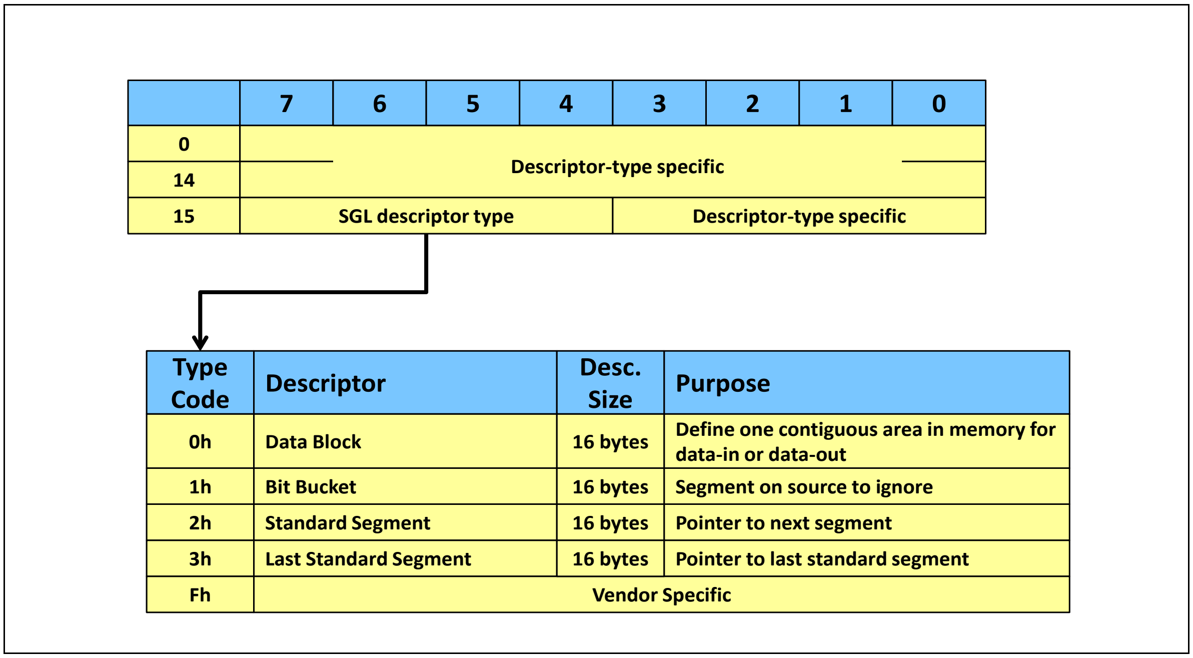
[7] An exception to this rule is for Create I/O SQ and Create I/O CQ Admin commands where if the queue is not contiguous in memory, PRP1 points to a PRP list.



	7	6	5	4	3	2	1	0
0	SGL Descriptor (first)							
15								
n-15	SGL Descriptor (last)							
n								

Figure 29: SGL Segment

Descriptors may be pointers to a memory address for data, pointers to the next segment, or define a length of read data to ignore and not place in memory. See Figure 30. Byte 15, bits 7:4 define the type of descriptor. Byte 15, bits 3:0 are listed as Descriptor Type Specific, but as of the date of publishing this book, no use has been defined for this field.

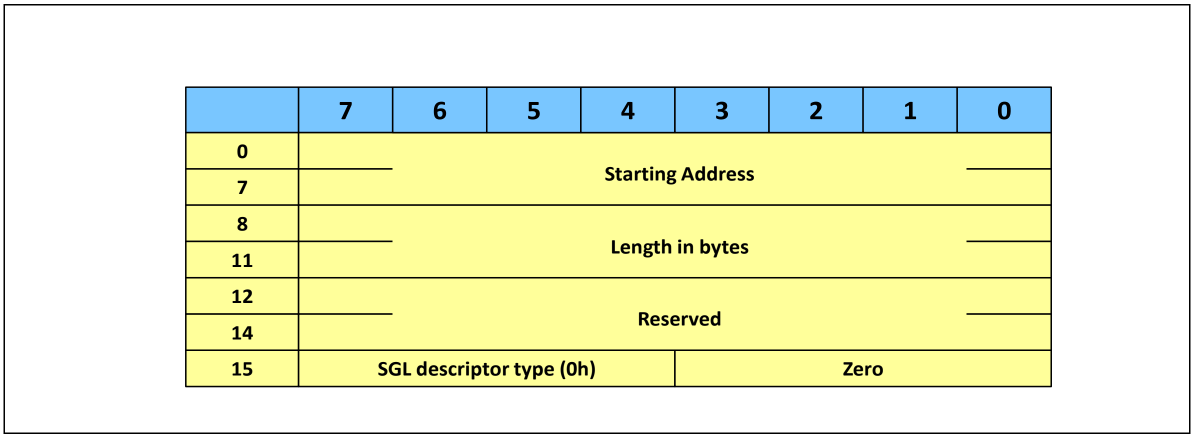


	7	6	5	4	3	2	1	0
0	Descriptor-type specific							
14								
15	SGL descriptor type				Descriptor-type specific			

Type Code	Descriptor	Desc. Size	Purpose
0h	Data Block	16 bytes	Define one contiguous area in memory for data-in or data-out
1h	Bit Bucket	16 bytes	Segment on source to ignore
2h	Standard Segment	16 bytes	Pointer to next segment
3h	Last Standard Segment	16 bytes	Pointer to last standard segment
Fh	Vendor Specific		

Figure 30: SGL Descriptors

Data Descriptors, Figure 31, contain a 64-bit starting memory address and 32-bit length field for the next data buffer location to transfer.



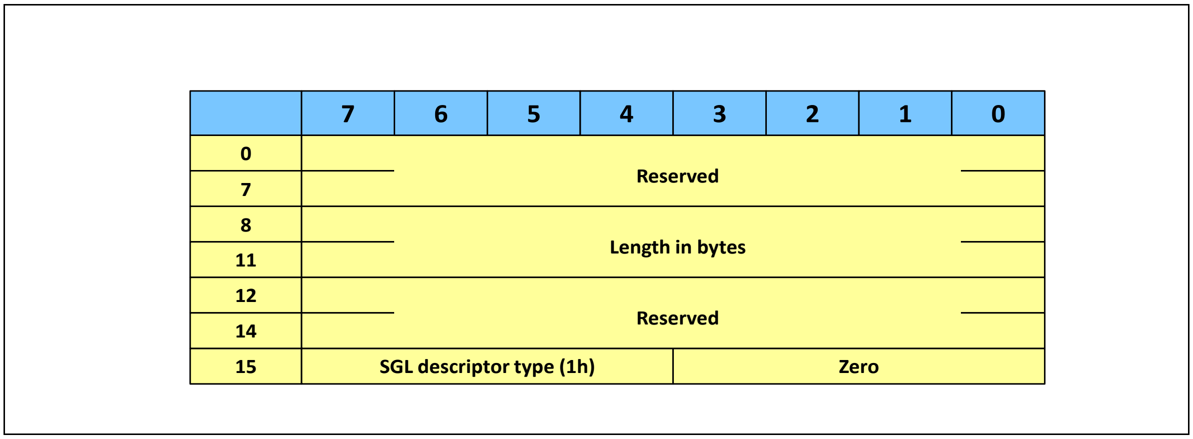
	7	6	5	4	3	2	1	0
0	Starting Address							
7								
8	Length in bytes							
11								
12	Reserved							
14								
15	SGL descriptor type (0h)				Zero			

Figure 31: SGL Data Descriptor

Bit Bucket Descriptor, Figure 32, is used on read commands only. It allows the read command to specify a contiguous range of LBA blocks on the storage device, but to ignore placing some of the data into memory. An example usage is reading a data base of employee records, each record is 4K bytes. You only want the first 72 bytes (name is 64 bytes, employee number 8 bytes). You issue a read command that covers many records and as the data comes to the host, it finds the following descriptors:

the first Data Descriptor specifies the starting memory address (DB-A) and a length of 72 bytes,
the next descriptor is a Bit Bucket Descriptor specifying 4022 bytes,
the next descriptor is a Data Descriptor with a starting memory address of DB-A + 72,
the next descriptor is a Bit Bucket Descriptor specifying 4022 bytes,
the next descriptor is a Data Descriptor with a starting memory address of DB-A + 144,
etc.

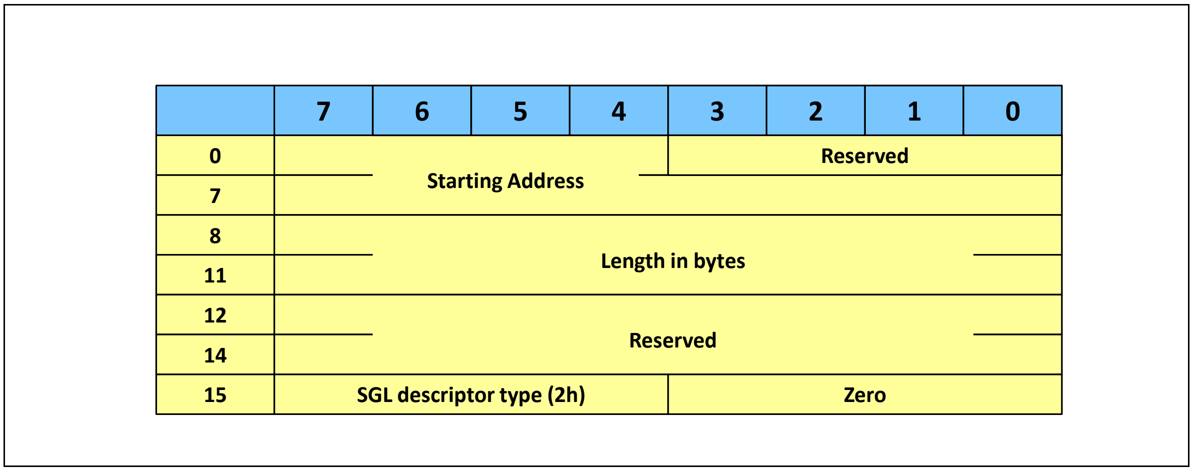
Note: The numbers 72, 144 and 4022 are decimal numbers. The descriptors' fields would contain the hexadecimal equivalents.



	7	6	5	4	3	2	1	0
0 7	Reserved							
8 11	Length in bytes							
12 14	Reserved							
15	SGL descriptor type (1h)				Zero			

Figure 32: SGL Bit Bucket Descriptor

Figure 33 shows a Segment Descriptor. It points to the next segment which has descriptors in it. The Starting Address is the beginning of the next segment; the Length in bytes is how long the next segment is, always a multiple of 16 bytes.

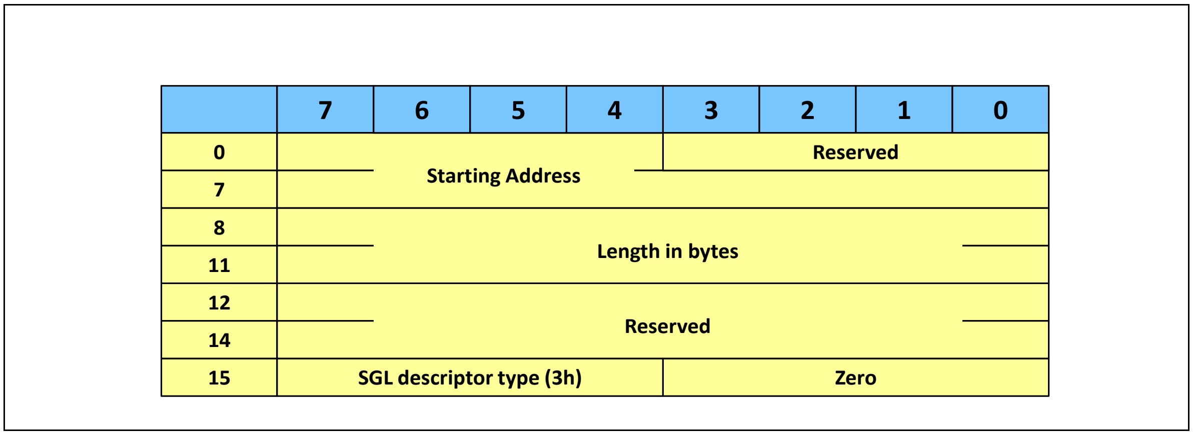


	7	6	5	4	3	2	1	0
0	Starting Address				Reserved			
7								
8	Length in bytes							
11								
12	Reserved							
14								
15	SGL descriptor type (2h)				Zero			

Figure 33: SGL Segment Descriptor

Figure 34 shows the Last Segment Descriptor which the pointer to the last segment of the SGL Segment Chain. Its format is the same as the standard Segment Descriptor, except for the Type field. The purpose of the Last Segment Descriptor is an earlier notification to the host and controller that this is the end of the data chain; if this segment descriptor type did not exist, the host and controller would not know the data transmission was completed until the end of the last data or bit bucket descriptor.

The Last Segment cannot contain a Segment Descriptor or Last Segment Descriptor.



	7	6	5	4	3	2	1	0
0	Starting Address				Reserved			
7								
8	Length in bytes							
11								
12	Reserved							
14								
15	SGL descriptor type (3h)				Zero			

Figure 34: SGL Last Segment Descriptor

Figure 35 shows a data transfer using all four types of descriptors.

On top of Figure 35 is the data of interest in a contiguous range of logical blocks (LBs) containing 18K of data (assume 512- byte blocks). The sizes marked on the NVMe Device are of no interest for the stored data, but will be of interest during the transfer. They are marked on the device to simplify the explanation.

The PCIe Memory Space shows a series of 4K memory pages. The ones marked Busy are holding information for another process and are not available for the transfer we will discuss. The tan colored memory pages are available to receive the data the host will receive from the storage device.

On the right of the figure are three SGL segments: Segment 1 contains two descriptors, Segment 2 contains three descriptors, and Segment 3 contains one descriptor. Each descriptor has its type (0, 1, 2 and 3) and other relevant information. The numbers identifying the segments are for use in this description; segments are not numbered in NVMe operations.

The host will build the SGL chain (in this case with three segments) and place a segment in DWords 6-9 that contains a pointer to Segment 1. The host will build the rest of the command, place it in the SQ at the tail pointer, update the tail pointer, and write the tail pointer in the doorbell.

The NVMe controller will fetch the command(s). When it executes this read command, it will see the LBA and length fields in the Command Specific DWords of the command and know where and how much data to read from the storage media (or cache).

This is a read command, so the NVMe controller will use PCIe Memory Write Transactions to transfer the data. The controller will read the SGL segment/descriptor in the command and see that it is a Type 2 pointing to another segment. It will read that segment from host memory space, find the Type 0 Data Descriptor and write 4K bytes of data to Data Block Address A.

The controller has read and transfered 4K of the 18K transfer length; there is still more data to transfer, so the controller will read the next descriptor in Segment 1 and discover it is a Type 2 - a pointer to another Segment. The controller will fetch that segment from host memory space and read the first descriptor. That descriptor is a Type 0 so the host will transfer the next 8K bytes of data to Data Block Address B.

The controller has read and transfered 12K of the 18K transfer length; still more data so the controller will read the second descriptor in Segment 2 and see that it is a Type 1 - Bit bucket. The controller will not transfer the next 2K bytes of data.

The controller has read 14K bytes of the 18K transfer length, so it reads the third descriptor in Segment 2 and discovers that it is a Type 3 - Last Segment Descriptor. The controller fetches that segment, reads the descriptor and reads and transfers the final 4K of data to Data Block Address C.

With all the data read and the appropriate data transferred, the NVMe controller will build the ending status, write it to the CQ at the tail pointer, and interrupt the host with legacy pin-based interrupt, MSI, or MSI-X. The host will process the ending status and write the new head pointer of the CQ to the doorbell.

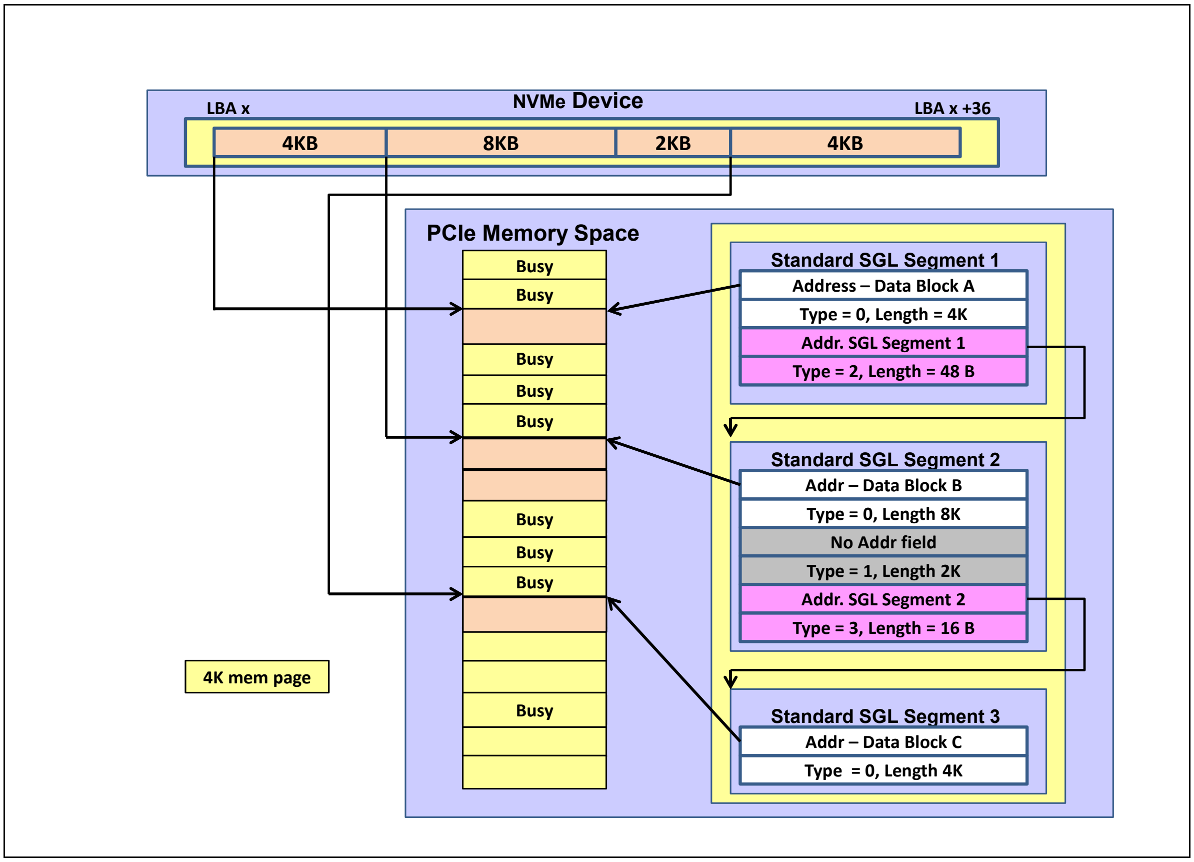


Figure 35: Data Transfer to a Data Buffer Using SGL

Figure 36 shows the use of SGL with a write command. The differences are:
The Bit Bucket type of descriptor is not available with write commands,
This uses a single segment plus the segment/descriptor in the command, and
The controller uses PCIe memory read transactions to fetch the data to write to the storage media.

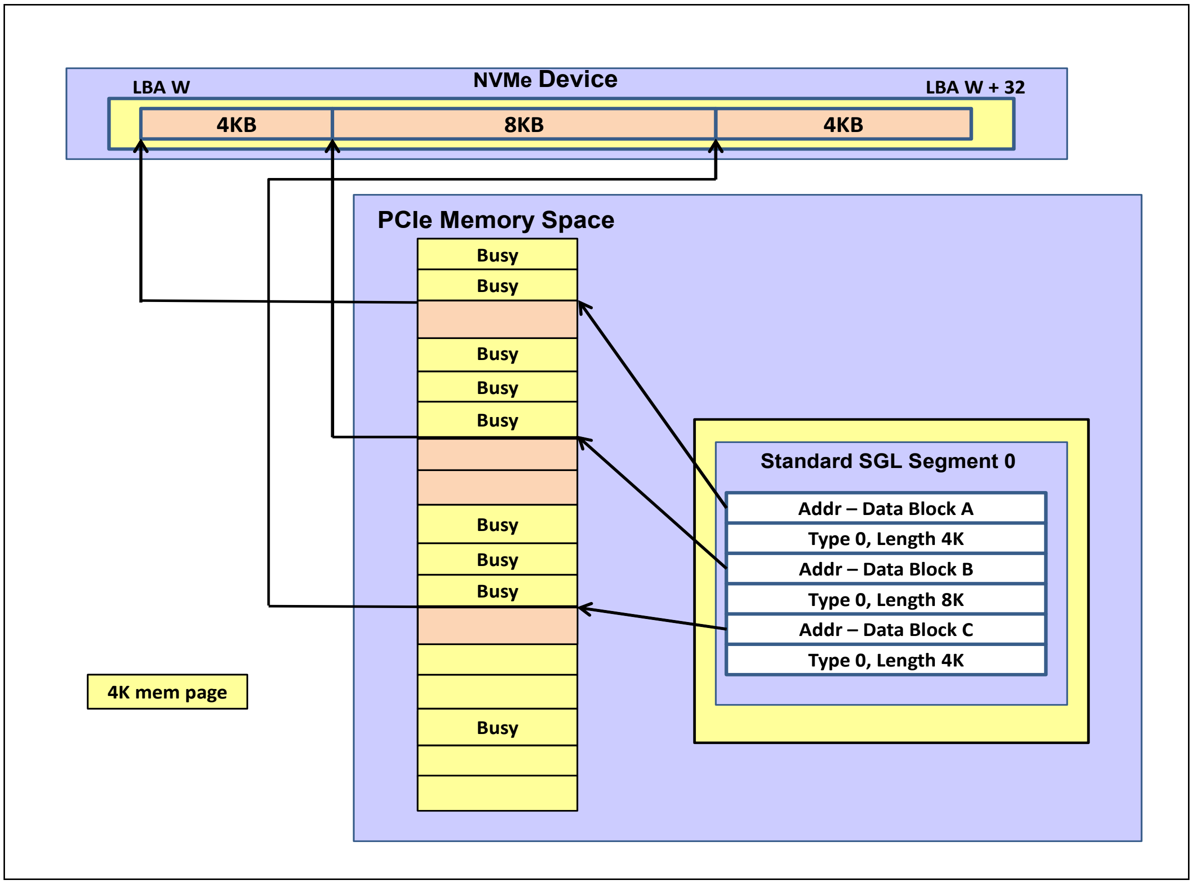
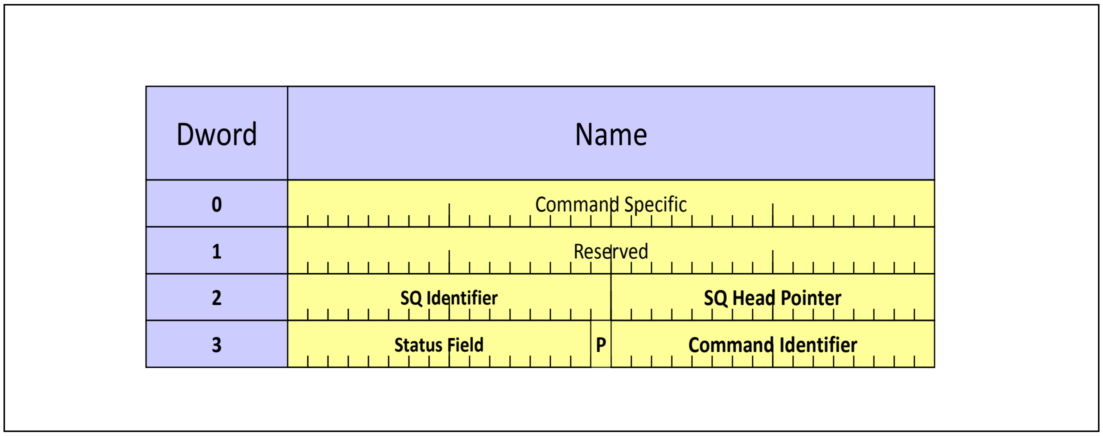


Figure 36: Data Transfer from a Data Buffer Using SGL

Status

Caution: The charts of status codes in this book are for the intention of showing how status is reported, and to describe a few status codes. For a complete list at the time of publication, see the companion Reference Manual. For the most up-to-date and accurate, see the NVMe Specification.

The controller sends 16 bytes of ending status to the host for every command that the host issued. The status is placed in the CQ associated with the SQ from which the controller fetched the command.



Dword	Name		
0	Command Specific		
1	Reserved		
2	SQ Identifier		SQ Head Pointer
3	Status Field	P	Command Identifier

Figure 37: Completion Status

The SQ Identifier matches this command completion with the SQ that submitted the command.

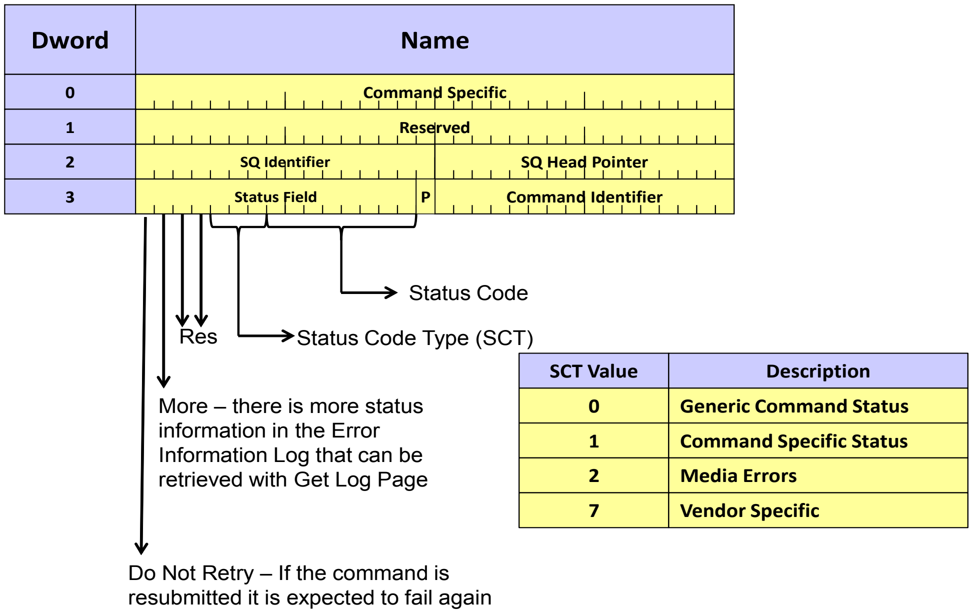
The Command Identifier with the SQ identifier uniquely identifies the command associated with this status. Because each SQ can use the full range of Command Identifiers, to uniquely identify a command requires the concatenation of the SQ ID and the Command Identifier.

The SQ Head Pointer tells the host the current head pointer so the host can determine if there are available slots in the SQ. Remember that the host places the commands on the SQ using the tail pointer and the controller takes the commands off the SQ using the head pointer. Here the controller is telling the host what is the next command it will pull off the SQ. (All commands in slots before this slot have been fetched and are being processed by the controller.) By matching the SQ tail (which the host knows because it controls the tail) and the SQ head (which the controller just told the host) the host can discover if there are any empty slots in the SQ.

To be symmetrical with the controller passing the head pointer of the SQ, it should also pass the tail pointer of the CQ. However, the NVMe committee implemented another way for the host to discover the end of the CQ, without requiring the passing of that information. It is the “P” or phase bit.

On initialization, the P bit is set to 0b for all entries in the CQ (they are all empty on initialization). Each time a slot is written, the P bit is inverted by the controller. On the first time through the CQ entries, the host will be reading the bit expecting it to be 1b. When it sees a 0b, the host knows it just read one entry beyond the end. On the second time through the CQ, the controller will invert the P bit to 0b for each status it submits; the host will expect a 0b and when it sees a 1b, it will realize it just came to the end of the queue.

The status field gives interim or completion information about the command. For details see either the latest NVMe specification or KnowledgeTek’s NVMe Reference book.



Dword	Name		
0	Command Specific		
1	Reserved		
2	SQ Identifier		SQ Head Pointer
3	Status Field	P	Command Identifier

More – there is more status information in the Error Information Log that can be retrieved with Get Log Page

Do Not Retry – If the command is resubmitted it is expected to fail again

SCT Value	Description
0	Generic Command Status
1	Command Specific Status
2	Media Errors
7	Vendor Specific

Figure 38: Status Field

See Figure 38. Status Field bit 14, Do Not Retry, indicates a type of error that will not be fixed in normal operations. If the controller was busy, trying later may work, therefore this bit would not be set. If the request referenced an LBA that was out of the Namespace range, trying later will not work, so bit 14 would be set.

Bit 13 means there is more information about this status in the Error Information Log and can be retrieved with the Get Log Page Admin command.

Bits 12 and 11 are reserved.

Bits 10, 9 and 8 identify the Status Code Type

000b – Generic Command Status (these indicate the command has completed)
001b – Command Specific Status (these indicate the current status, more processing may be required)
010b – Media Errors
111b – Vendor Specific

Bits 7-0 provide the status code.

00 – 7Fh are applicable to Admin Command Set or across multiple command sets.
80 – BFh are I/O Command set Specific Status Codes
C0 – FFh are Vendor Specific Status Codes.

Good ending status is indicated by SCT = 000b and Status Code = 00h.

If there are multiple statuses to be reported, the controller will report the status with the lowest numerical value.

Chapter 4 – Common Command Fields

Chapter Summary

- General Command Format
    - Command DWord 0
        - Command Identifier
        - Fused Operations
        - OpCode
    - Namespace Identifier (NSID)
    - Metadata
    - Data Buffer Pointers
        - Generic Example - Data Buffer Pointer Table
        - PRP
        - SGL
- Status

Admin Commands

Chapter Contents

Admin Commands

Admin Commands are commands used to configure and operate the NVMe controller, device and namespace. Admin commands exist to:

create and delete I/O queues and namespaces,

query the controller and namespace regarding capabilities or settings,

set and read features, and

download and commit (activate) firmware.

Admin commands are placed in the Admin Submission Queue (SQ ID = 00h) and the completion status is returned in the Admin Completion Queue (CQ ID = 00h).

For Admin Commands, the SGL is never used, therefore the PS bits are always 00b. It is frequently shown as reserved in this text.

The Command ID is a unique 16-bit identifier for each outstanding command in the Admin SQ. Upon completion of the command, the controller will send ending status and identify the command with the Command ID and SQ ID = 00 00h.

OpCode	M/O	NS ID used?	Command
00h	M	No	Delete I/O SQ
01h	M	No	Create I/O SQ
02h	M	Yes	Get Log Page
04h	M	No	Delete I/O CQ
05h	M	No	Create I/O CQ
06h	M	No/Yes	Identify
08h	M	No	Abort
09h	M	Yes	Set Features
0Ah	M	Yes	Get Features
0Ch	M	No	Asynchronous Event Request
0Dh	O	Yes	Namespace Management
10h	O	No	Firmware Commit
11h	O	No	Firmware Image Download
15h	O	Yes	Namespace Attachment
18h	Note 1	No	Keep Alive
7Fh			Fabric Commands
I/O Command Set Specific Commands			
80-BFh	O		I/O Command Set Specific
NVMe Specific Admin Commands			
80h	O	Yes	Format NVM
81h	O	Yes	Security Send
82h	O	Yes	Security Receive
Vendor Specific			
C0-FFh	O		Vendor Specific

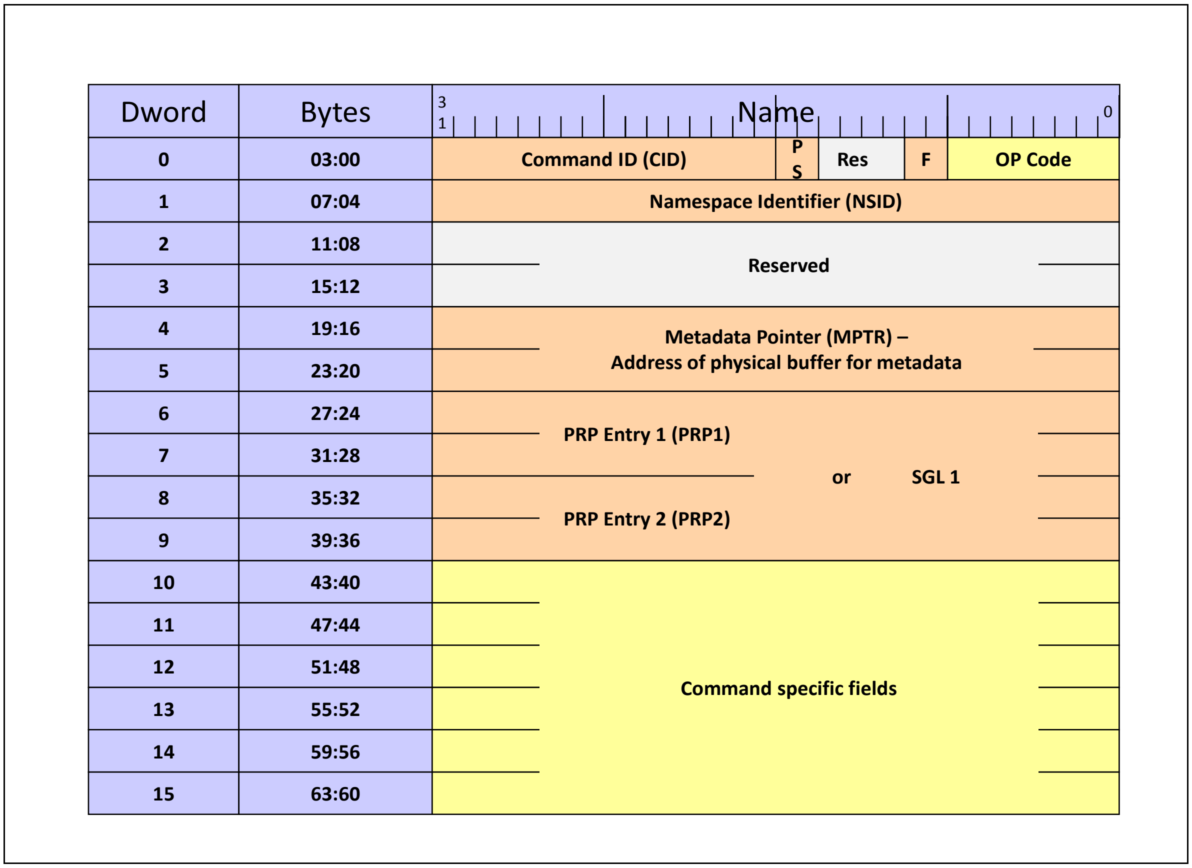
Table 14: Admin Commands

A note about OPCodes. The two least significant bits indicate data transfer action of controller.

00b - no data transfer,
01b - write to controller,
10b - read from controller,
11b - reserved.

Note 1 - Keep Alive is optional in NVMe over PCIe. See transport documents for information about requirements in NVMe over Fabrics.

Figure 39 shows the common fields (salmon color) vs. the command specific fields (yellow color). The common fields were defined in the chapter, "Common Command Fields" on page 51; command specific fields will be defined for admin commands in this chapter and for the NVMe commands in the chapter, "NVMe Commands" on page 125.



Dword	Bytes	31 Name				0
0	03:00	Command ID (CID)	PS	Res	F	OP Code
1	07:04	Namespace Identifier (NSID)				
2	11:08	Reserved				
3	15:12					
4	19:16	Metadata Pointer (MPTR) – Address of physical buffer for metadata				
5	23:20					
6	27:24	PRP Entry 1 (PRP1)	or	SGL 1		
7	31:28					
8	35:32	PRP Entry 2 (PRP2)				
9	39:36					
10	43:40	Command specific fields				
11	47:44					
12	51:48					
13	55:52					
14	59:56					
15	63:60					

Figure 39: Admin Command Format

Creation and Deletion of I/O Queues

Before any I/O commands can be sent to the controller, the host will use the proper Admin commands to create the I/O SQ and CQ. The relationship between SQ and CQ can be 1 to 1 (one SQ per CQ) or many to one (more than one SQ per CQ). However, each SQ can have only one CQ. When creating each SQ, the associated CQ is identified, therefore the CQ must exist before any associated SQs are created.

When deleting, the SQ should be empty and then must be deleted before the associated CQ can be deleted.

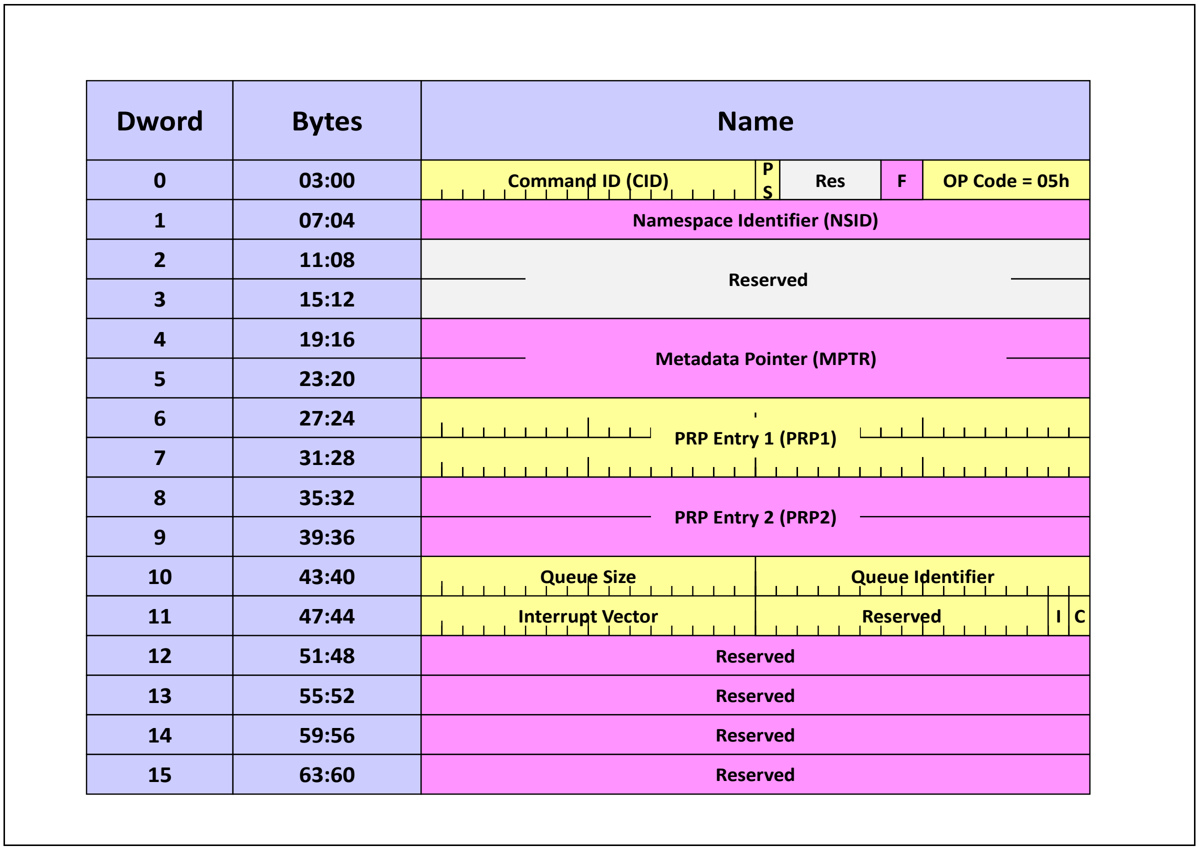
Creating an queue does not attach it to a controller. That function is accomplished with the Namespace Attachment command. Deleting a queue does detach it from the controller because it no longer exists in the NVM system.

On the following figures, the pink color designates fields not used for the specific command; yellow designates fields used by the command.

Create I/O Queues Commands

Create I/O Completion Queue

Before creating a CQ or SQ, the host may issue a Set Features command (See "Feature ID 07h - Number of Queues" on page 97) to request a certain number of CQ and SQ. The host may then issue a Get Features 07h to discover the number of queues allocated by the controller. The number allocated may be larger, equal to, or smaller than the number requested.



Dword	Bytes	Name
0	03:00	Command ID (CID) PS Res F OP Code = 05h
1	07:04	Namespace Identifier (NSID)
2	11:08	Reserved
3	15:12	
4	19:16	Metadata Pointer (MPTR)
5	23:20	
6	27:24	PRP Entry 1 (PRP1)
7	31:28	
8	35:32	PRP Entry 2 (PRP2)
9	39:36	
10	43:40	Queue Size Queue Identifier
11	47:44	Interrupt Vector Reserved I C
12	51:48	Reserved
13	55:52	Reserved
14	59:56	Reserved
15	63:60	Reserved

Figure 40: Create I/O Completion Queue

Figure 40 shows the Admin command to create an I/O Completion Queue.

Queue Identifier is a 16-bit number between 1 and x inclusive, where x is the Number of Queues reported in the Get Features 07h.

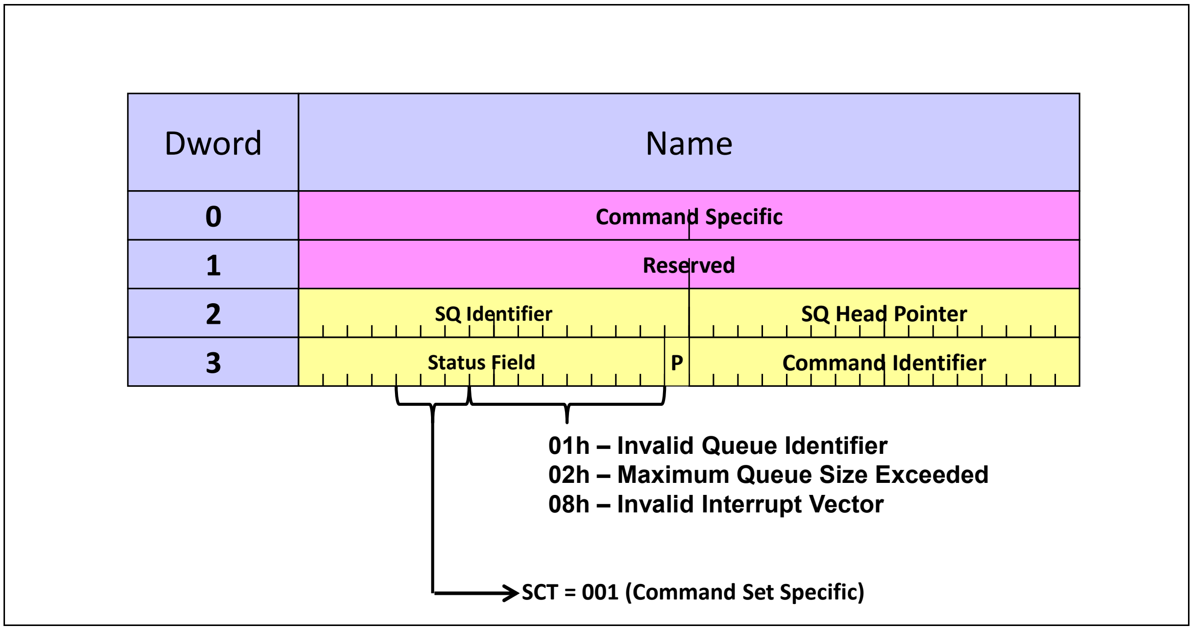
The Queue Size specifies the number of entries (slots) in the CQ. This number is 0's based. Each entry in the I/O CQ queue is 16-bytes long.

The Interrupt Vector field specifies the Interrupt Vector if using MSI-X or multiple message MSI. If using single message MSI or pin-based interrupts, this field is set to 00h. Refer to PCI Capability structure MSICAP.MC.MME[8] or MSIXCAP.MXC.TS to discovery the maximum number of interrupt vectors supported by the controller.

I bit specifies if interrupts are to be enabled. 1 means "yes."

C bit specifies if the queue must be physically contiguous in memory. 1 means "yes." If C = 1, then the PRP1 is the address in 64-bit PCIe memory address space for the queue. If C = 0, then the PRP1 is a pointer to the PRP list. PRP2 entry is not defined.

PS bits are 00b because SGL is not supported in Admin Commands.



Dword	Name		
0	Command Specific		
1	Reserved		
2	SQ Identifier		SQ Head Pointer
3	Status Field	P	Command Identifier

01h – Invalid Queue Identifier
02h – Maximum Queue Size Exceeded
08h – Invalid Interrupt Vector

SCT = 001 (Command Set Specific)

Figure 41: Create I/O Completion Queue Status

Figure 41 shows the completion status. For successful completion, SCT = 000b and Status = 00h. The other values shown in Figure 41 are for errors.

[8] The MSICAP.MC.MME and MSIXCAP.MXC.TS identifiers reference bits or fields in the PCI Capability defined in PCI Local Bus 3.0 specification. However, the identifier used by NVMe are not used in the PCI or PCIe specifications. The translation can be found in "MSI and MSI-X interrupts" on page 21 and "MSI-X Capability Structure" on page 23 where the figures are as in the PCI specification and the names follow NVMe specifications.

This is an Admin Command so the SQ Identifier will be 0000h. The Command Identifier will be the Identifier submitted in the DWord 0 of the Command to which this is the response.

The SQ Head Pointer is the current Head of Queue for the SQ. This pointer is used by the controller to track where it will begin pulling commands. The controller needs to communicate this pointer to the host so the host can discover if the queue is full or if more commands may be added by the host.

The P bit is Phase. Each time the controller writes a completion (or group of completions) it inverts the P bit. The host uses this information to find the end of queue, meaning it has taken the last valid completion plus one from the CQ.

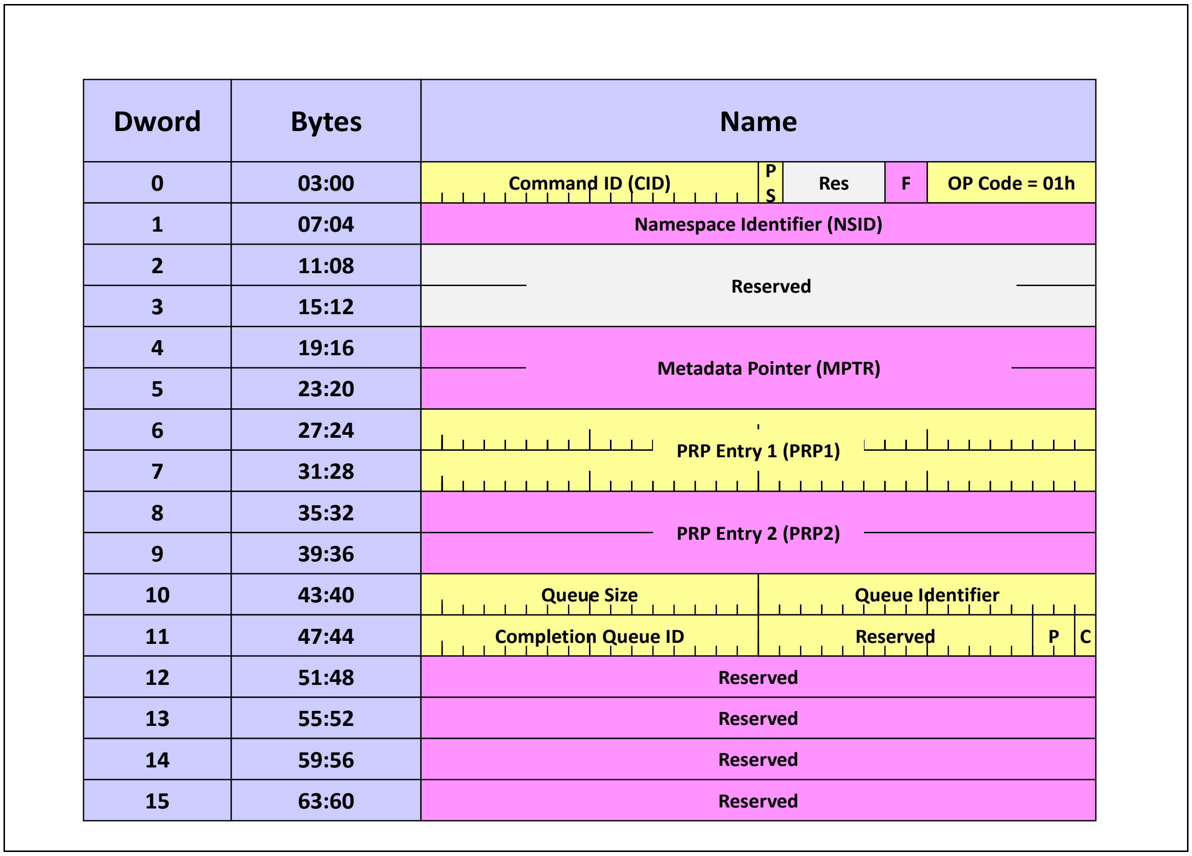
An Invalid Queue Identifier is one that is outside the range supported by the controller or one that is already in use.

Maximum Queue Size Exceeded means the number of entries (slots) requested in the Create I/O Completion Queue command is greater than the largest queue size supported by the controller as reported in the NVMe Capability register, bits 15:00 (MQES).

Invalid Interrupt Vector means that the host software requested an interrupt number outside the range supported by the controller for MSI or MSI-X. Supported interrupt range is available from the MSI capability or MSI-X capability structures of PCI/PCIe.

Create I/O Submission Queue

Figure 42 shows the Admin command to create an I/O Submission Queue.



Dword	Bytes	Name				
0	03:00	Command ID (CID)	PS	Res	F	OP Code = 01h
1	07:04	Namespace Identifier (NSID)				
2	11:08	Reserved				
3	15:12					
4	19:16	Metadata Pointer (MPTR)				
5	23:20					
6	27:24	PRP Entry 1 (PRP1)				
7	31:28					
8	35:32	PRP Entry 2 (PRP2)				
9	39:36					
10	43:40	Queue Size	Queue Identifier			
11	47:44	Completion Queue ID	Reserved	P	C	
12	51:48	Reserved				
13	55:52	Reserved				
14	59:56	Reserved				
15	63:60	Reserved				

Figure 42: Create I/O Submission Queue

Queue Identifier is a 16-bit number between 1 and x inclusive, where x is the Number of Queues reported in the Get Features 07h.

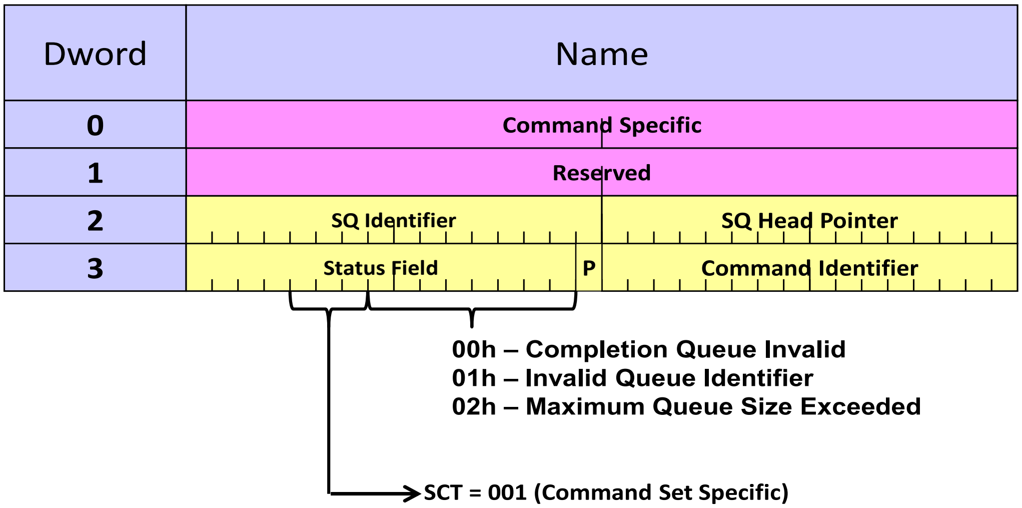
The Queue Size is specifies the number of entries in the SQ. This number is 0's based. Each entry in the I/O queue is 64-bytes long.

Completion Queue ID is the completion queue (already in existence) that will receive the Completion Status for any commands submitted on this SQ.

C bit specifies if the queue must be physically contiguous in memory. 1 means "yes." If C = 1, then the PRP1 is the address in 64-bit PCIe memory address space for the queue. If C = 0, then the PRP1 is a pointer to the PRP list. PS bits are 00b because SGL is not supported in Admin Commands.

The P bits define the priority of commands submitted on this queue.

00b - Urgent
01b - High
01b - Medium
11b - Low



Dword	Name		
0	Command Specific		
1	Reserved		
2	SQ Identifier		SQ Head Pointer
3	Status Field	P	Command Identifier

00h – Completion Queue Invalid
01h – Invalid Queue Identifier
02h – Maximum Queue Size Exceeded
SCT = 001 (Command Set Specific)

Figure 43: Create I/O Submission Queue Status

Figure 43 shows the completion status. For successful completion, SCT = 000b and Status = 00h. The other values shown in Figure 43 are for errors.

This is an Admin Command so the SQ Identifier will be 0000h. The Command Identifier will be the Identifier submitted in the DWord 0 of the Command to which this is the response.

The SQ Head Pointer is the current Head of Queue for the SQ. This pointer is used by the controller to track where it will begin pulling commands. The controller needs to communicate this pointer to the host so it can discover if the queue is full or if more commands may be added by the host.

The P bit is Phase. Each time the controller writes a completion (or group of completions) it inverts the P bit. The host uses this information to find the end of queue, meaning it has taken the last valid completion from the CQ.

Completion Queue Invalid error means that the completion queue has not yet been created, or the completion queue in the command was entered improperly.

An Invalid Queue Identifier is one that is outside the range supported by the controller or one that is already in use.

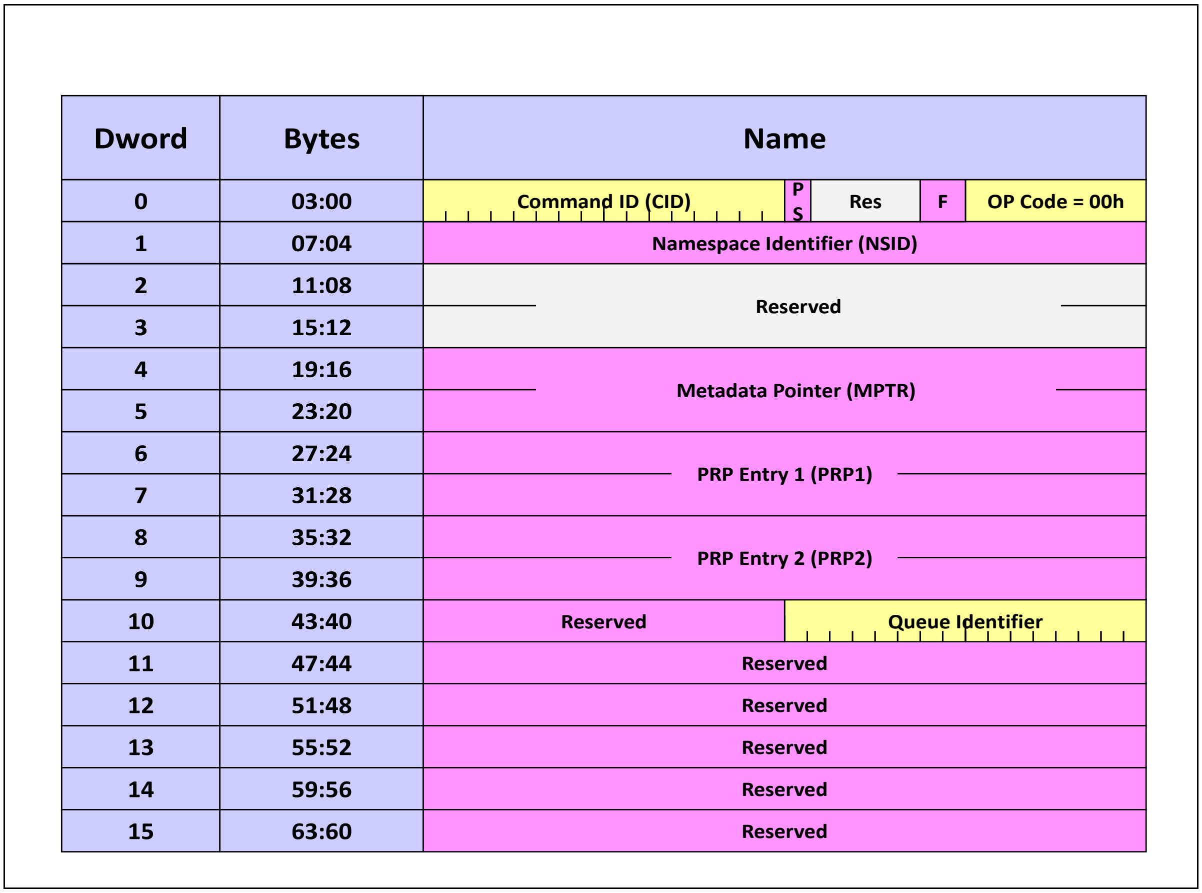
Maximum Queue Size Exceeded means the number of entries requested in the Create I/O Completion Queue command is greater than the largest queue size supported by the controller as reported in the NVMe Capability register, bits 15:00 (MQES).

Delete I/O Queues Commands

The SQ should be empty before it is deleted. If there are commands in the SQ when the delete is issued, those commands will be aborted.

All SQs associated with a CQ must be deleted before the CQ can be deleted.
The Admin SQ and Admin CQ cannot be deleted with these Delete commands.

Delete I/O Submission Queue

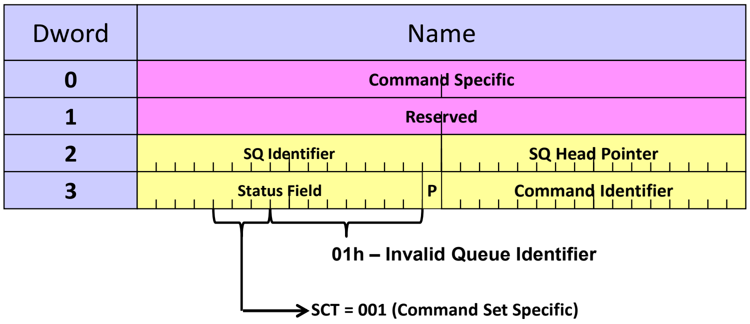


Dword	Bytes	Name
0	03:00	Command ID (CID) PS Res F OP Code = 00h
1	07:04	Namespace Identifier (NSID)
2	11:08	Reserved
3	15:12	
4	19:16	Metadata Pointer (MPTR)
5	23:20	
6	27:24	PRP Entry 1 (PRP1)
7	31:28	
8	35:32	PRP Entry 2 (PRP2)
9	39:36	
10	43:40	Reserved Queue Identifier
11	47:44	Reserved
12	51:48	Reserved
13	55:52	Reserved
14	59:56	Reserved
15	63:60	Reserved

Figure 44: Delete I/O Submission Queue

Figure 44 shows the command to delete the I/O SQ. The Queue Identifier specifies which queue to delete. The Command ID is provided to match the Completion Status with the proper command.

Figure 45, below, shows the completion status. Values and fields are described in the Create I/O SQ and CQ commands in the Common Command Fields chapter.



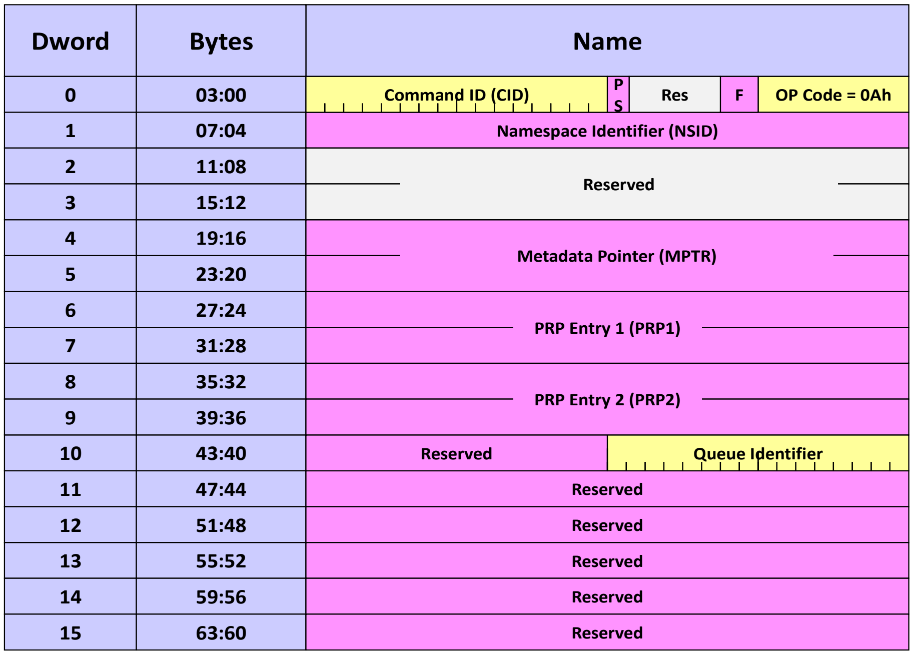
Dword	Name		
0	Command Specific		
1	Reserved		
2	SQ Identifier		SQ Head Pointer
3	Status Field	P	Command Identifier

01h – Invalid Queue Identifier
SCT = 001 (Command Set Specific)

Figure 45: Delete I/O Submission Queue Status

Delete the I/O Completion Queue

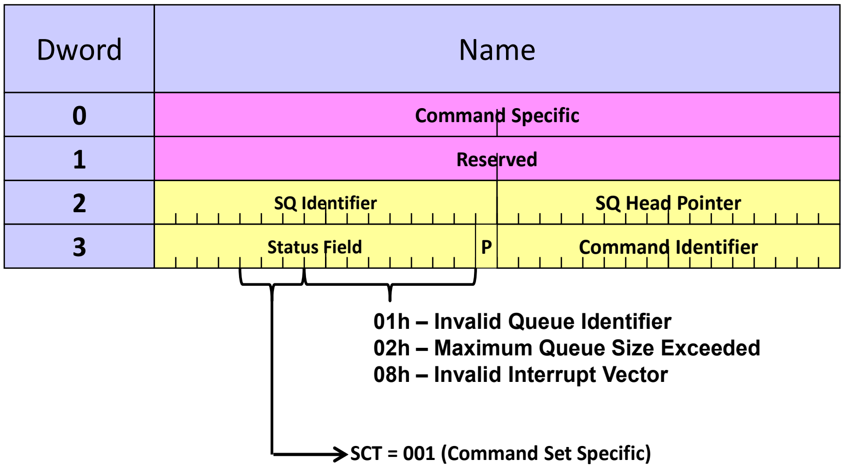
Figure 46 shows the command to delete the I/O CQ. Note that the only information for the command (besides OPCode and Command ID) is the Queue Identifier to delete.



Dword	Bytes	Name				
0	03:00	Command ID (CID)	PS	Res	F	OP Code = 0Ah
1	07:04	Namespace Identifier (NSID)				
2	11:08	Reserved				
3	15:12					
4	19:16	Metadata Pointer (MPTR)				
5	23:20					
6	27:24	PRP Entry 1 (PRP1)				
7	31:28					
8	35:32	PRP Entry 2 (PRP2)				
9	39:36					
10	43:40	Reserved	Queue Identifier			
11	47:44	Reserved				
12	51:48	Reserved				
13	55:52	Reserved				
14	59:56	Reserved				
15	63:60	Reserved				

Figure 46: Delete I/O Completion Queue

Figure 47 shows the completion status. Values and fields are described in the Create I/O SQ and CQ commands in the Common Command Fields chapter.



Dword	Name		
0	Command Specific		
1	Reserved		
2	SQ Identifier		SQ Head Pointer
3	Status Field	P	Command Identifier

01h – Invalid Queue Identifier
02h – Maximum Queue Size Exceeded
08h – Invalid Interrupt Vector

SCT = 001 (Command Set Specific)

Figure 47: Delete I/O Completion Queue Status

Identify Command

The Identify Command is the "Send me your resume" or "Tell me about yourself" command. See Figure 48 on page 82. The host software needs to know the characteristics and capability of the controller and namespace. The controller returns 4096 bytes of data describing one of the data structures specified in the CNS field:

00h - Namespace specified in the NSID if the namespace is attached to this controller,
01h - Controller,
02h - Up to 1024 active namespaces attached to this controller greater than NSID,
10h - Up to 1024 namespaces present in the NVM subsystem, whether or not attached to controllers,
11h - Identify namespace structure for NSID whether or not attached to his controller,
12h - Controller list of up to 2047 controller identifiers ≥ to CNTID and attached to NSID, or
13h - Controller list of up to 2047 controller identifiers ≥ to CNTID whether or not attached to NSID.

Controller

If the host specifies CNS = 01h, the controller will respond by writing the 4096 bytes of information defining the controller into the host memory buffer specified in DWords 6 - 9 of the command. If PRP1 specifies a buffer large enough for the entire data transfer, then PRP 2 is reserved. If PRP 1 is not large enough, then the remaining data will be placed in the buffer beginning at PRP 2. In no case will PRP 2 be a pointer to a PRP list.

Namespace

If the host specifies CNS = 00h or 11h, the controller will respond by writing the 4096 bytes of information defining the namespace specified in NSID into the host memory buffer specified in DWords 6 - 9 of the command. If PRP1 specifies a buffer large enough for the entire data transfer, then PRP 2 is reserved. If PRP 1 is not large enough, then the remaining data will be placed in the buffer beginning at PRP 2. In no case will PRP 2 be a pointer to a PRP list.

Active Namespace ID List

An Active Namespace ID is a Valid Namespace ID that maps to a Namespace. A Valid Namespace ID is one whose value is greater than 0 and less than or equal to the Number of Namespaces (NN) reported by the Identify Controller command.

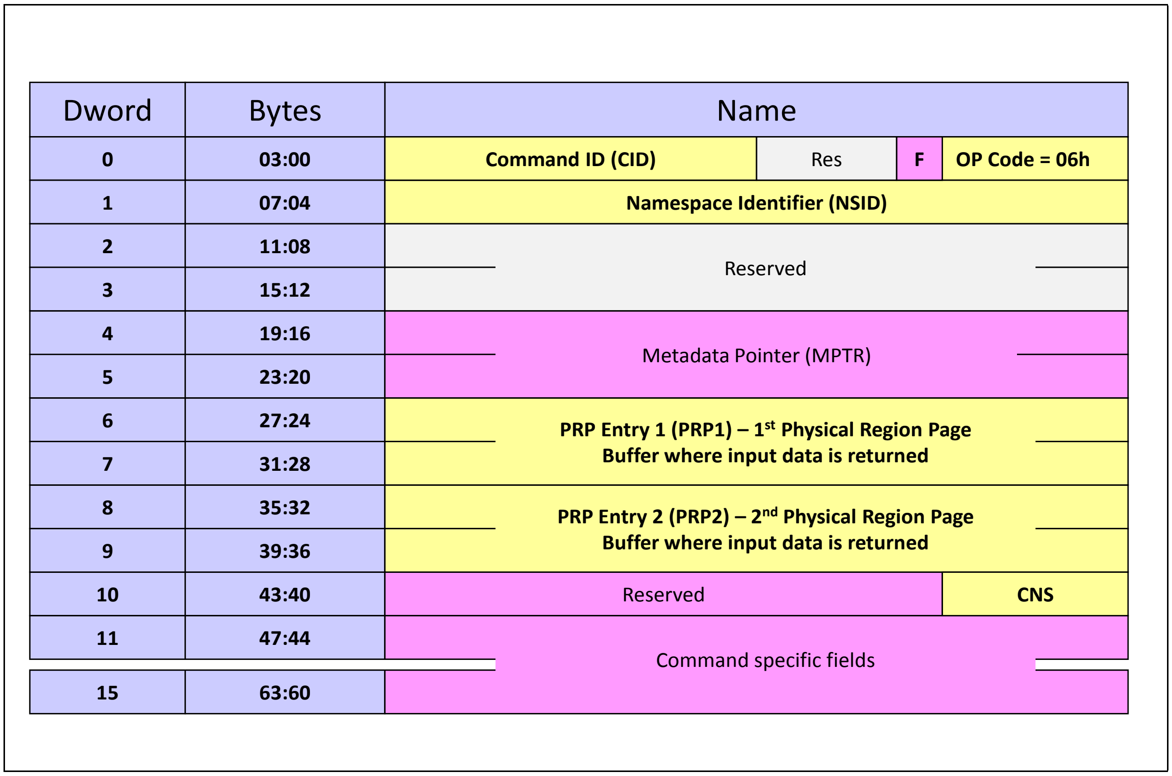
If the host specifies CNS = 02h, the controller will respond by writing a Namespace ID List of the active Namespace IDs attached to this controller and starting with the next active NS ID greater than the one specified in Dword 1 of the command, and continuing in numerical order for the next 1024 NS IDs. The list will be written into the host memory buffer specified in DWords 6 - 9 of the command. If PRP1 specifies a buffer large enough for the entire data transfer, then PRP 2 is reserved. If PRP 1 is not large enough, then the remaining data will be placed in the buffer beginning at PRP 2. In no case will PRP 2 be a pointer to a PRP list. If there are fewer than 1024 active namespaces, the field will be zero filled by the controller. If there are more than 1024 ND IDs to report, the host may issue another Identify Command with CNS equal to 02h and the NDID equal to the last ND ID just reported.

Initialization

Step 1. During initialization, the host will probably first issue the Identify command with the CNS = 01h to get basic information on the controller, including how many namespaces there are.

Step 2. The host will probably next issue the Identify command with the CNS = 02h to get a list of active namespaces accessible through this controller.

Steps 3 and on. The host will then issue the Identify command with the CNS = 00h and bytes 7:4 set to one of the active namespaces. It will do this for each possible namespace listed in step 2.



Dword	Bytes	Name			
0	03:00	Command ID (CID)	Res	F	OP Code = 06h
1	07:04	Namespace Identifier (NSID)			
2	11:08	Reserved			
3	15:12				
4	19:16	Metadata Pointer (MPTR)			
5	23:20				
6	27:24	PRP Entry 1 (PRP1) – 1st Physical Region Page Buffer where input data is returned			
7	31:28				
8	35:32	PRP Entry 2 (PRP2) – 2nd Physical Region Page Buffer where input data is returned			
9	39:36				
10	43:40	Reserved			CNS
11	47:44	Command specific fields			
15	63:60				

Figure 48: Identify Command

Controller Return Data

Bytes	Description		
Controller Capabilities and Features			
01:00	PCI Vendor ID		
03:02	PCI Subsystem Vendor ID		
23:04	Serial Number in ASCII		
63:24	Model Number in ASCII		
71:64	Firmware Revision in ASCII		
72	Recommended Arbitration Burst		
75:73	IEEE OUI (Organizational Unique Identifier (24 bits))		
76	Multi-path and Namespace Sharing		
	Bit 0	0	Single PCIe port
		1	May have more than one port
	Bit 1	0	Single controller
		1	May have more than one controller
	Bit 2	0	PCI Function
		1	SR-IOV Virtual Function
77	Maximum Data Transfer Size (2^n)		
79:78	Controller ID		
83:80	NVMe Version		
87:84	RTD3 Resume Latency in microseconds		
91:88	RTD3 Entry Latency in microseconds		
95:92	Optional Asynchronous Events Supported		
	Bit 8	1	Controller supports sending the Namespace Attribute Changed
239:96	Reserved		
255:240	Reserved for NVMe Management Interface Specification		
Admin Command Set Attributes & Optional Controller Capabilities			
257:256	Optional Admin Command Support (OACS)		
	Bit 0	1	Controller supports Security Send and Receive commands
	Bit 1	1	Controller supports Format NVM command
	Bit 2	1	Controller supports F/W Download and F/W Activate commands
	Bit 3	1	Controller supports Namespace Management and Namespace Attachment commands
258	Abort Command Limit (ACL)		
259	Synchronous Event Request Limit (AERL)		
260	Firmware Updates (FRMW)		
	Bit 0	0	1st F/W slot is Read/Write
		1	1st F/W slot (slot 1) is Read Only
	Bits 3:1		Number of F/W slots the controller supports
	Bit 4		Controllers supports F/W activation without a reset
261	Log Page Attributes (LPA)		
	Bit 0	1	Controller supports SMART/Health Log Info on a namespace basis
	Bit 1	1	Controller supports the Command Effects log page
262	Error Log Page Entries (ELPE) 0’s based		
263	Number of Power States Supported (NPSS)		

Figure 49: Identify Controller Return Data

264	Admin Vendor Specific Command Configuration (AVSCC)		
	Bit 0	0	Admin Vendor Specific Commands are Vendor Specific
		1	Admin Vendor Specific Commands use format from NVMe Specification
265	Autonomous Power State Transition Attributes (APSTA)		
	Bit 0	1	Controller supports Autonomous Power State Transitions
267:266	Warning Composite Temperature Threshold (default 0157h)		
269:268	Critical Composite Temperature Threshold		
271:270	Maximum Time for Firmware Activation		
275:272	Host Memory Buffer Preferred Size		
279:276	Host Memory Buffer Minimum Size		
295:280	Total NVM Capacity (in bytes)		
311:296	Unallocated NVM Capacity (in bytes)		
315:312	Replay Protected Memory Block Support (See 8.10 in NVMe Spec 1.2)		
511:316	Reserved		
NVM Command Set Attributes			
512	Submission Queue Entry Size (SQES)		
	Bits 3:0	Minimum SQ Entry Size for NVMe, 2^n, n shall be 6	
	Bits 7:4	Maximum SQ Entry Size for NVMe, 2^n, recommended n is 6	
513	Completion Queue Entry Size (SQES)		
	Bits 3:0	Minimum CQ Entry Size for NVMe, 2^n, n shall be 4	
	Bits 7:4	Maximum CQ Entry Size for NVMe, 2^n, recommended n is 4	
515:514	Reserved		
519:516	Number of Namespaces (NN)		
521:520	Optional NVM Command Support (ONCS)		
	Bit 0	1	Controller supports Compare Command
	Bit 1	1	Controller supports Write Uncorrectable Command
	Bit 2	1	Controller Supports Data Set Management Command
	Bit 3	1	Controller supports Write Zeros Command
	Bit 4	1	Controller supports the Save field in the Set Features Command
	Bit 5	1	Controller supports Reservations
523:522	Bit 0	1	Controller supports Compare and Write fused operation
524	Format NVM Attributes (FNA)		
	Bit 0	0	Controller supports a format per namespace
		1	All namespaces are formated the same
	Bit 1	0	Crypto or User Data erase is per namespace
		1	Crypto or User Data erase applies to all namespaces
	Bit 2	0	Crypto erase is not supported
		1	Crypto erase is supported
525	Bit 0	1	Volatile Write Cache is present
527:526			Atomic Write Unit Normal (AWUN) (in Logical Blocks, 0 based)
529:528			Atomic Write Unit Power Fail (AWUPF)
530			NVM Vendor Specific Command Configuration (NVSCC)
531			Reserved
533:532			Atomic Compare and Write Unit (ACWU)
535:534			Reserved

Figure 49: Identify Controller Return Data

539:536	SGL Support (SGLS)		
	Bits 31:19		Reserved
	18	1	Commands taht contain a data or Metadata SGL length larger that the amount of data to be transferred
	17	1	Byte aligned contiguous physical buffer of Metadata is supported
	16	1	SGL Bit Bucket is supported
	15:01		Reserved
	00	1	Controller supports SGLs for NVM Command Set
767:540[a]			Reserved
1023:768			NVM Subsytem NVMe Qualified Name (SUBNQN)
1791:1024			Reserved
2047:1792			Refer to NVMe Over Fabrics specification
Power State Descriptors			
2079:2048			Power State 0 Descriptor
. . .			
3071:3040			Power State 31 Descriptor
Vendor Specific			
4095:3072			Vendor Specific

Figure 49: Identify Controller Return Data

[a] Bytes 540 through 2017 changed when going from NVMe Base specification 1.2 to 1.2.1. This chart shows the 1.2.1 version assignments.

NVMe Qualified Name (NQN)

The 223 byte NQN is a unique name given to each host or NVM subsystem to uniquely define it for identification or authentication. The name, after being assigned is permanent for the lifetime of the host or NVM subsystem.

There are two formats defined in NVMe Base Specification 1.2.1. An organization may use any that fits its needs.

First format: Organization owns a domain name and desires a human readable string to describe the host or NVM subsystem.

“nqn.yyyy-mm.” <reverse domain name> “:” <owner assigned string>

The date must be a date that the organization owned the domain name.

A reverse domain name is to simply specify the identifier in reverse order. “KnowledgeTek.com” would become “com.KnowledgeTek”.

The owner assigned string must be unique within the date and domain name. The naming authority is responsible to ensure that the NQN is worldwide unique.

Second format: For an organization who does not own a domain name or does not desire a human readable string.

“nqn.2014-08.org.nvmexpress:uuid:” < plus a 128 bit UUID based on RFC 4122>.

All controllers in a NVM subsystem share the same NVM subsystem NQN that is reported in Identify Controller, bytes 1023:768. The Controller ID returned in the Identify Controller data structure bytes 79:78 may be used to uniquely address one of the controllers.

Power State Descriptors

The 32-byte Power State Descriptors values are built in by the device manufacturer and read by the host. Power State 0 (maximum power) is mandatory, other states define less power consumed and are optional. The higher the number for the power state, the lower the power consumption.

Bits	Description
255:184	Reserved
183:182	Active Power Scale 00b = not reported 01b = 0.0001 Watts 10b = 0.01 Watts
181:179	Reserved
178:176	Active Power Workload
175:160	Active Average Power over 10 seconds of workload
159:152	Reserved
151:150	Idle Power Scale 00b = not reported 01b = 0.0001 Watts 10b = 0.01 Watts
149:144	Reserved
143:128	Idle Average Power over 30 seconds when idle
127:125	Reserved
124:120	Relative Write Latency (RWL)
119:117	Reserved
116:112	Relative Write Throughput (RWT)
111:109	Reserved
108:104	Relative Read Latency (RRL)
103:101	Reserved
100:96	Relative Read Throughput (RRT)
95:64	Exit Latency (EXLAT) in µs
63:32	Entry Latency (ENLAT) in µs
31:26	Reserved
25	Non-operational State (NOPS)
24	Max Power Scale (MPS) 0b = 0.01 Watts 1b = 0.0001 Watts
23;16	Reserved
15:00	Maximum Power (times Max Power Scale)

Table 15: Power State Descriptors

The "Active Power" fields (cyan) give the average power consumed under workload by multiplying the value in bytes 175:160 times the scale in bytes 183:182. NVMe Specification 1.2, Section 8.4.3 defines different workloads to be used for power measurements, bytes 178:176 tell which workload was used for this power state.

The "Idle Power" fields (magenta) give the power consumed in this power state when no commands, register accesses or background processes are executing. The time for measurement starts after 10 seconds of idle and measures for 30 seconds.

The four "Relative" fields (yellow) compare processing in different power states. Lower values indicate lower latency or higher throughput (better performance). Valid values are from 0 through one less than the number of supported power states. If two power states have the same performance, they would have the same value for latency or throughput.

Exit and Entry latency (salmon) give how long in µs it takes to leave one start (exit latency) and enter another state (entry latency). To transition from one state to another, add the exit latency for the current state and the entry latency for the destination state.

Non-operational State (white) set to 1b means that in the state described in this descriptor the controller cannot process commands. If this bit is set to 0b, the controller can process commands.

The "Maximum Power" fields (olive) provide the maximum power used in this state. Multiply the Maximum Power field times the Max Power Scale field.

Namespace Return Data

Figure 50 shows the data returned if the host specifies CNS = 00h. The NSID field of the command will indicate to the controller which namespace information to return to the host.

Bytes	Description		
7:0	Name Space Size (NSZE) Total size in LBs, 0's based		
15:8	Namespace Capacity (NCAP) Max LB's that may be allocated, 1'based		
23:16	Namespace Utilization (NUSE) Current number of LB's allocated		
24	Namespace Features		
	Bit 2	1	Controller supports Deallocated or Unwritten LB for this namespace
	Bit 1	1	NAWUN, NAWUPF and NACWU fields are supported
	Bit 0	1	Namespace supports thin provisioning[a]
25	Number of LBA Formats (NLBAF) See bytes 191:128		
26	Formatted LB Size (FLBAS)		
	Bits 3:0		LB and Metadata this NS has been formatted with. 0's based
	Bit 4	0	Metadata is transferred as a separate contiguous buffer
		1	Metadata is transferred as part of data
27	Bit 0	1	NS supports metadata as part of an extended data LB
	Bit 1	1	NS supports metadata as part of a separate buffer
28	End-to-End Data Protection (DPC)		
	Bit 0	1	NS supports type 1 data protection
	Bit 1	1	NS supports type 2 data protection
	Bit 2	1	NS supports type 3 data protection
	Bit 3	1	NS supports transfer of PI as the 1st 8 bytes of metadata
	Bit 4	1	NS supports transfer of PI as the last 8 bytes of metadata
29	End-to-End Data Protection Type Setting (DPS)		
	Bits 2:0	0	Protection Information is not enabled
		1	Type 1 protection is enabled
		2	Type 2 protection is enabled
		3	Type 3 protection is enabled
	Bit 3	0	Protection Information is transferred as last 8 bytes of metadata
		1	Protection information is transferred as 1st 8 bytes of metadata

Figure 50: Identify Namespace Return Data

30	Namespace Multi-path I/O and Sharing Capabilities (NMIC)		
	Bit 0	0	Namespace is private, only accessible by one controller
		1	Namespace may be accessible from more than one controller
31	Reservation Capabilities (RESCAP)		
	Bit 0	1	NS supports Persist Through Power Loss
	Bit 1	1	NS supports Write Exclusive reservation type
	Bit 2	1	NS supports Exclusive Access Reservation type
	Bit 3	1	NS supports Write Exclusive - Registrants Only Reservation type
	Bit 4	1	NS supports Exclusive Access - Registrants Only Reservation type
	Bit 5	1	NS supports Write Exclusive - All Registrants Reservation type
	Bit 6	1	NS supports Exclusive Access - All Registrants Reservation type
32	Format Progress Indicator		
33	Reserved		
35:34	Namespace Atomic Write Unit Normal (in logical blocks)		
37:36	Namespace Atomic Write Unit Power Fail (in logical blocks)		
39:38	Namespace Atomic Compare & Write Unit (in logical blocks)		
41:40	Namespace Atomic boundary Size Normal (in logical blocks)		
43:42	Namespace Atomic Boundary Offset		
45:44	Namespace Atomic Boundary Size Power Failure		
47:46	Reserved		
63:48	NVM Capacity allocated to this namespace		
103:64	Reserved		
119:104	Namespace Globally Unique Identifier		
127:120	IEEE Extended Unique Identifier (EUI64)		
131:128	LBA Format 0 (LBAF0)		
191:188	LBA Format 15 (LBAF15)		
383:192	Reserved		
4095:384	Vendor Specific		

Figure 50: Identify Namespace Return Data

a. Byte 24, bit 0 - If the device supports thin provisioning, then Namespace capacity reported may be less than Namespace Size. If thin provisioning and Dataset Management command are both supported, then deallocating LBs shall be reflected in the Namespace Utilization field. If byte 24, bit 0 = 0b then Namespace Size and Namespace Capacity fields report the same value.

Bits	Description
15:00	Metadata Size (bytes)
23:16	LB Data Size (2^n)
25:24	Relative Performance 00b - Best Performance 01b - Better Performance 10b - Good Performance 11b - Degraded Performance
31:26	Reserved

Table 16: LB Format

Namespace Attachment

Attaches or detaches controllers to/from the Namespace. The list of controllers is pointed to by the PRP pointers. Namespaces that are "shared" can have multiple controllers attached to them; Namespaces that are "private" can have only one controller attached to them. To attach or detach the controllers is specified in Command Dword 10.

Bits	Description
31:04	Reserved
03:00	0h - Controller Attach 1h - Controller Detach

Table 17: Namespace Attachment Command Dword 10

Specific return status values are:

18h - Namespace is already attached to namespace specified
19h - Namespace is private and already attached to one controller
1Ah - Namespace is not attached (request to detach could not be completed)
1Ch - Controller List is invalid

Namespace Management

Before NVMe Rev 1.2, namespaces were created by the controller manufacturer. The Namespace Management Admin Command in Rev 1.2 provides the means for users to create or delete namespaces of their own preferences on the device. The Namespace Management command will transfer a data structure similar to that returned from the controller with the Identify Namespace command and shown in Table 19. Command Dword 10 specifies to create or delete a namespace as shown in Table 18.

Bits	Value	Description
31:04	00h	Reserved
03:00	0h	Create
	1h	Delete

Table 18: NS Management Command Dword 10

When doing a create, the Namespace ID in Command Dword 1 is left blank, the controller will assign an available NS ID to the newly created one.
When indicating a delete NS, the NSID to be deleted is specified in Command Dword 1.

Bytes	Host Specified	Description
7:0	Yes	Namespace Size (NSZE) in LB, 0 based
15:8	Yes	Namespace Capacity (NCAP) in LB, 1 based
25:16		Reserved
26	Yes	Formatted LBA Size (FLBAS) points to a supported LBA format
28:27		Reserved
29	Yes	End to End Data Protection Type Settings Bits 2:0 000b - Protection is not enabled 001b - Type 1 protection is enabled 010b - Type 2 protection is enabled 011b - Type 3 protection is enabled Bit 3 0b - protection information is last 8 bytes of metadata 1b - protection information is 1st 8 bytes of metadata
30	Yes	NS Multi-path I/0 and NS sharing Capabilities (NMIC) Bit 1 - namespace may be accessible by 2 or more controllers
383:31		Reserved
1023:384		Reserved
4095: 1024		Vendor Specific

Table 19: Namespace Management Data Structure

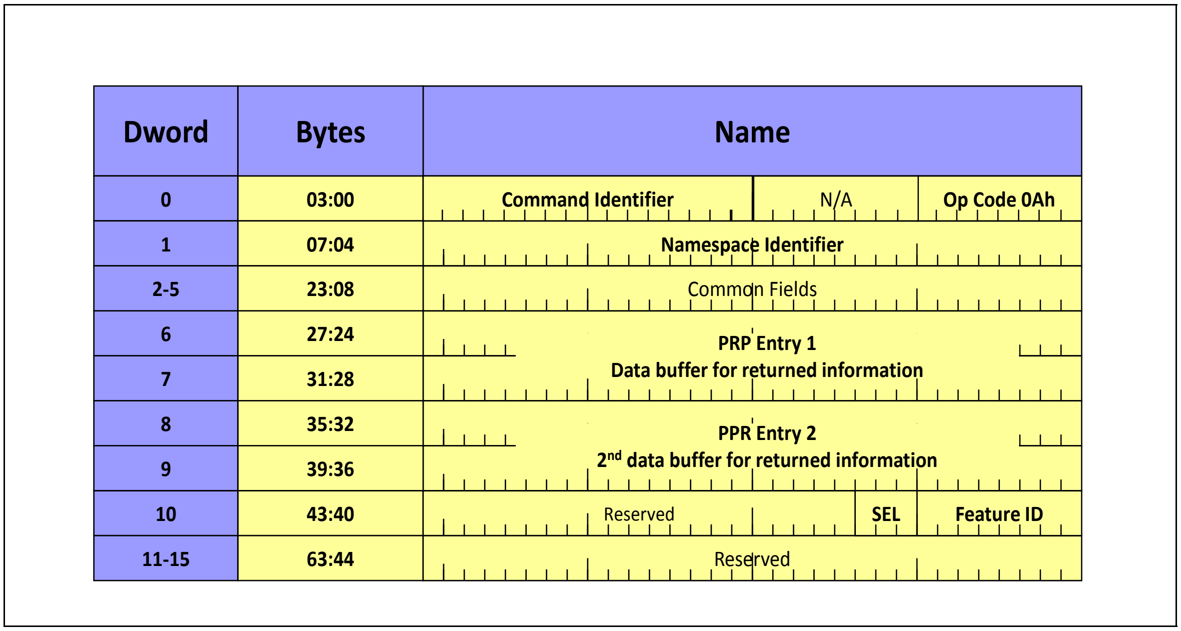
Features

Many operational parameters can be read or set using the Get Features Admin command or the Set Features Admin command. With each command, there is a Feature ID that will identify which feature the host software wants to read or write.

Features defined in the Specification are listed in Table 20 on page 93.

Get Features

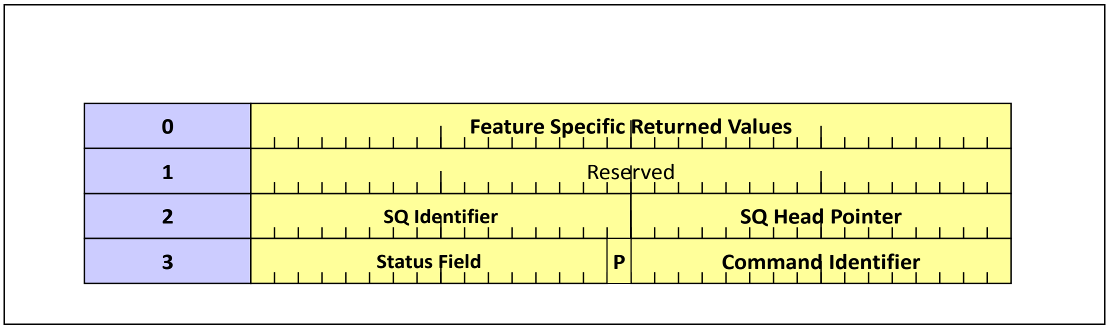
The Get Features command is shown in Figure 51. Some features are specific to Namespace and thus would use the NS ID DWord. Some features use a data buffer so the PRP entries are available. The Feature ID tells the controller which features information to return to the host. The SEL field indicates which copy of the features.
SEL - Select:
000b - Return the CURRENT settings of the feature,
001b - Return the manufacturer's DEFAULT setting of the feature,
010b - Return the SAVED value of the feature, and
011b - Return the SUPPORTED CAPABILITIES of the feature.



Dword	Bytes	Name
0	03:00	Command Identifier N/A Op Code 0Ah
1	07:04	Namespace Identifier
2-5	23:08	Common Fields
6	27:24	PRP Entry 1 Data buffer for returned information
7	31:28	
8	35:32	PPR Entry 2 2^{nd} data buffer for returned information
9	39:36	
10	43:40	Reserved SEL Feature ID
11-15	63:44	Reserved

Figure 51: Get Features Command

The returned value(s) will be in DWord 0 of the returned status for most features or in the Data Buffer pointed to by PRP. Each specification defined feature is listed with its DWord 0 values shown.

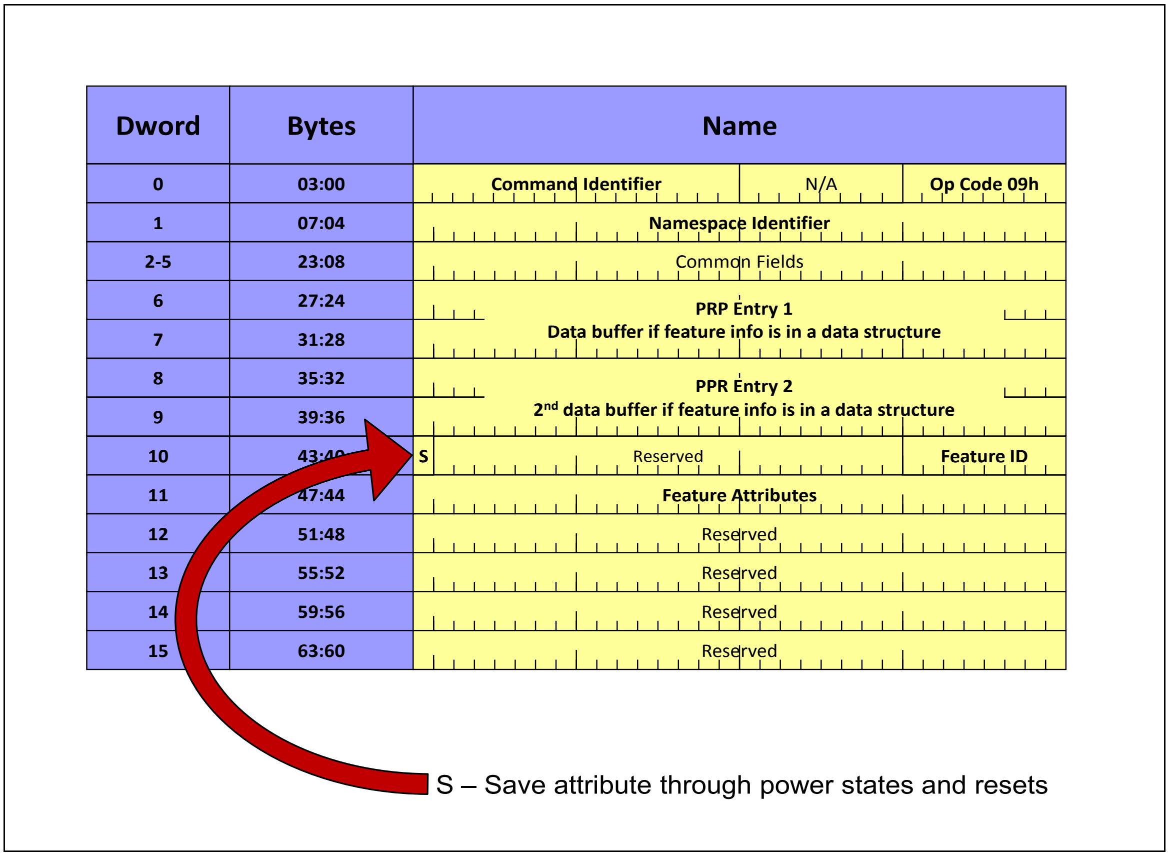


0	Feature Specific Returned Values		
1	Reserved		
2	SQ Identifier		SQ Head Pointer
3	Status Field	P	Command Identifier

Figure 52: Get Features Command Completion Status

Set Features

The Set Features command is how the host software configures the operating parameters of the NVMe controller. The feature identifier and the Save attribute are shown in DWord 10. The feature attributes are in DWord 11.



Dword	Bytes	Name		
0	03:00	Command Identifier	N/A	Op Code 09h
1	07:04	Namespace Identifier		
2-5	23:08	Common Fields		
6	27:24	PRP Entry 1 Data buffer if feature info is in a data structure		
7	31:28			
8	35:32	PPR Entry 2 2nd data buffer if feature info is in a data structure		
9	39:36			
10	43:40	S	Reserved	Feature ID
11	47:44	Feature Attributes		
12	51:48	Reserved		
13	55:52	Reserved		
14	59:56	Reserved		
15	63:60	Reserved		

S – Save attribute through power states and resets

Figure 53: Set Features Command

Table 20 lists the Feature ID that can be used in the Get Features command or the Set Features command. Following the list is an explanation for each feature.

Feature ID	Persistent	Memory Buffer	M/O	Description
01h	No	No	M	Arbitration
02h	No	No	M	Power Management
03h	Yes	Yes	O	LBA Range Type
04h	No	No	M	Temperature Threshold
05h	No	No	M	Error Recovery
06h	No	No	O	Volatile Write Cache
07h	No	No	M	Number of Queues
08h	No	No	M	Interrupt Coalescing
09h	No	No	M	Interrupt Vector Configuration
0Ah	No	No	M	Write Atomicity
0Bh	No	No	M	Asynchronous Event Configuration
0Ch	No	Yes	O	Autonomous Power State Transition
0Dh	No	No	O	Host Memory Buffer
0Fh	No	No	O	Keep Alive Timer
80h	Yes	No	O	Software Progress Marker
81h	No	Yes	O	Host Identifier
82h	No	No	O	Reservation Notification Mask
83h	Yes	No	O	Reservation Persistence

Table 20: Feature List

Feature ID 01h - Arbitration

When the SQs were created, the user assigned a priority to each queue (Urgent, High, Medium, or Low). If using Weighted Round Robin Arbitration, the Admin Queue has the highest priority and is strict priority 1. All commands from the Admin Queue are pulled before any commands from any other queues. The queues set to Urgent are strict priority 2, so all commands from the Urgent SQs are pulled before any from the other three queues are pulled. If there are no commands on the Admin SQ, nor on the Urgent SQ, the controller will pull commands from the queues of the High, Medium and Low priorities. Assuming the queues have adequate commands, the controller will pull more commands per arbitration round from the High Priority queue, fewer from the Medium Priority queue and still fewer from the Low Priority queue. Then the controller will go back to the high priority queue for another arbitration round.

The Arbitration feature provides the user the ability to program their own priority weights for High Priority, Medium Priority, and Low Priority and also Arbitration Burst.

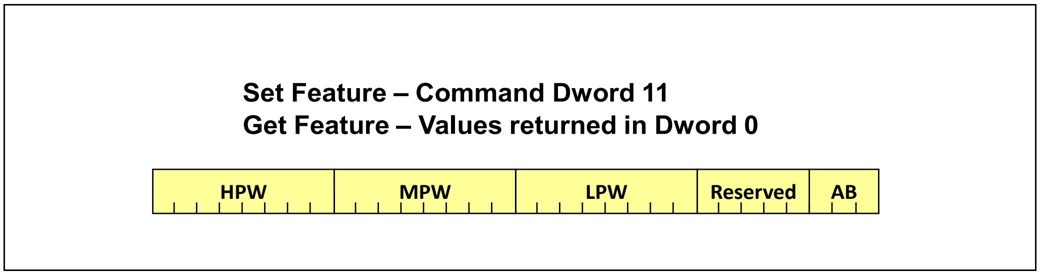

Set Feature – Command Dword 11
Get Feature – Values returned in Dword 0
HPW
MPW
LPW
Reserved
AB

Figure 54: Arbitration Feature Values

AB is the Arbitration Burst which is the maximum number of commands that the controller may launch at one time from a SQ. 2^{AB} is the number of commands; 2^7 means no limit.

HPW is High Priority Weight, the maximum number of commands that may be pulled from the high priority queues per arbitration round.

MPW is the Medium Priority Weight, the maximum number of commands that may be pulled from the medium priority queue per arbitration round.

LPW is Low Priority Weight, the maximum number of commands that may be pulled from the low priority queue per arbitration round.

HPW, MPW, and LPW are all zero based values.

An example of setting for this feature could be:
HPW = 40h
MPW = 20h
LPW = 10h
AB = 40h

Feature ID 02h - Power Management

The NVMe specification defines that the controller may be in any one of up to 32 power states. The controller defines how many Power States it supports, the transition time in and out of each state, and the power consumption while in each supported state. With the Get Feature, the host can read which power state the controller is currently in and with the Set Feature command can tell the controller to go to another state.

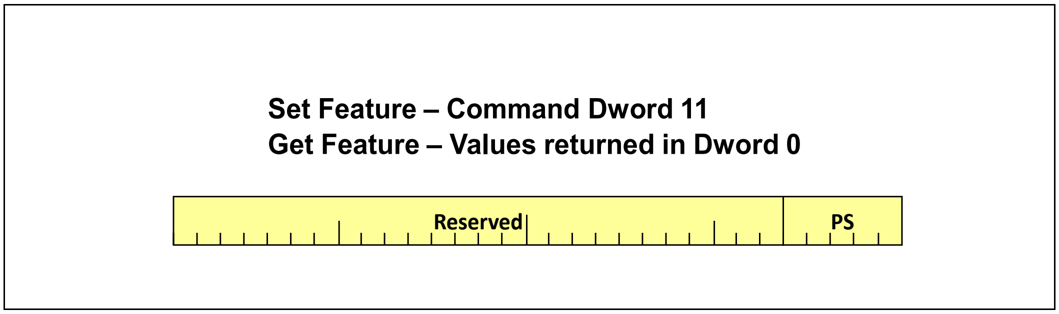

Set Feature – Command Dword 11
Get Feature – Values returned in Dword 0
Reserved
PS

Figure 55: Power Management Feature Values

The PS field is indicates the new Power State into which the controller should transition. If PS is a Power State not supported by the controller, the result of the transition is undefined.

Feature ID 03h - LBA Range Type

NVMe provides a means for the host to communicate type and attributes of LBA ranges to the controller that the controller may use to make operations more efficient. These LBA Ranges are for a specified namespace. DWord 11 on Set Features command and DWord 0 on the Get Features command status provide the number of LBA ranges defined (Figure 56). NUM is a 0's based value.

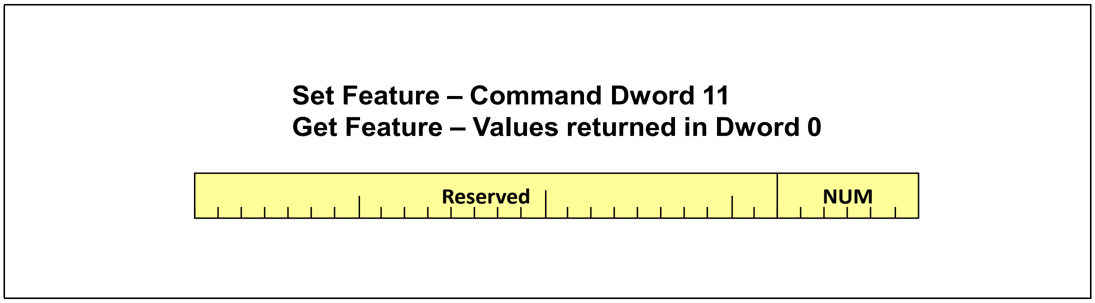

Set Feature – Command Dword 11
Get Feature – Values returned in Dword 0
Reserved
NUM

Figure 56: LBA Range Type Feature Value

The data structure for this Get Features or Set Features is pointed to by PRP1 and must be physically contiguous.

Byte	Byte Description	
00	Value	Value Description
	00h	Reserved
	01h	File System
	02h	RAID
	03h	Cache
	04h	Page / Swap file
	05 - 7Fh	Reserved
	80-FFh	Vendor Specific
01	Bits	Bit Description
	0	LBA range may be overwritten
	1	LBA range should be hidden from OS, EFI and BIOS
	2-7	Reserved
15:02	Reserved	
23:16	Starting LBA	
31:24	Number of Logical Blocks	
47:32	Unique Identifier (GUID)	
63:48	Reserved	

Table 21: LBA Range and Type Data Structure

Byte 0 indicates the use for these logical blocks.

All LBA ranges that follow a hidden range must also be hidden from the OS, EFI and BIOS.

Bytes 31:16 gives the information about where (Starting LBA) and how much (Number of Logical Blocks). The Number of Logical Blocks is a 0's based value.

Bytes 47:32 provide a global unique identifier for this LBA range.

Feature ID 04h - Temperature Threshold

If the temperature of the overall device (controller and NVM) exceeds the value in TMPTH field, in Kelvin, the controller may send an Asynchronous Event. If the value of TMPTH is 0 or FFFFh, then an Asynchronous Event should not be sent.

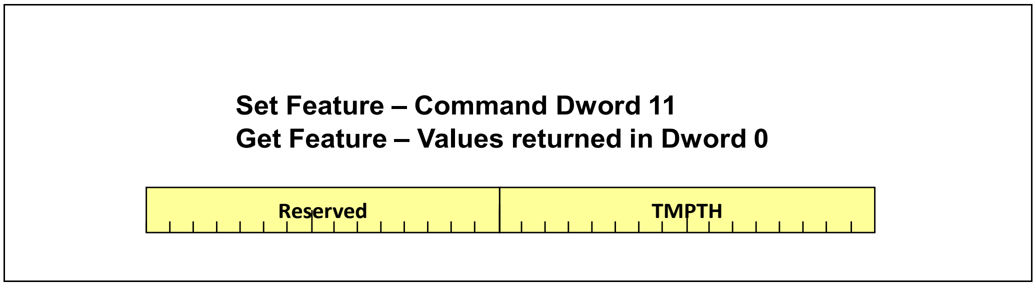

Set Feature – Command Dword 11
Get Feature – Values returned in Dword 0
Reserved
TMPTH

Figure 57: Temperature Threshold Feature Value

The use of Kelvin means there are no negative numbers for temperature.

Conversion:
Celsius^{o} = Kelvin^{o} - 273.15
Fahrenheit^{o} = ((Kelvin^{o} *9/5) - 459.67)

TMPTH is in Hexadecimal, the numbers 273.15 and 459.67 are decimal.

Feature ID 05h - Error Recovery

This feature defines the maximum time in 100 ms increments for error recovery on I/O commands that indicate a limited error recovery time is required (DWord 12, bit 31 of Read and Write commands) See "Read and Write User Data" on page 127. TLER of 00 00h means no time limit is set. The commands: Compare, Read, Write, Write Zeros, will have the Limited Retry (LR) bit set i.e. DWord 12, bit 31.

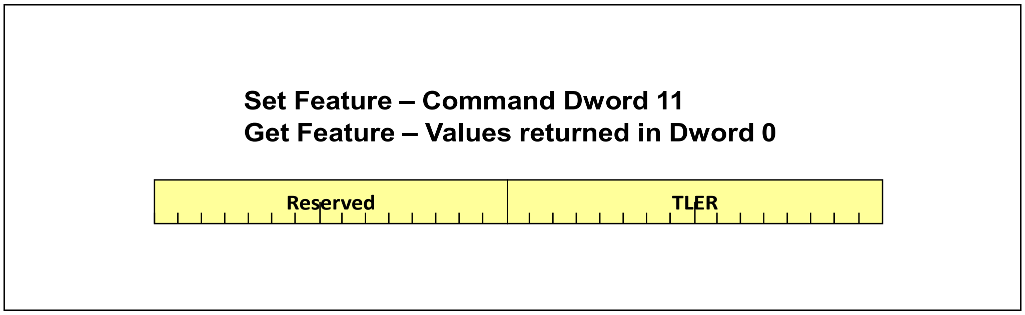

Set Feature – Command Dword 11
Get Feature – Values returned in Dword 0
Reserved
TLER

Figure 58: Error Recovery Feature Value

This feature is provided for host software that may have alternate means of recovery data or for time sensitive information or systems.

Feature ID 06h - Volatile Write Cache

This feature allows enabling and disabling volatile write cache on the device. If the device supports volatile write cache, the feature is required.

Note: if the controller can guarantee that data in the Write Cache will be saved on power failure, then the Write Cache is considered non-volatile and this bit has no effect.

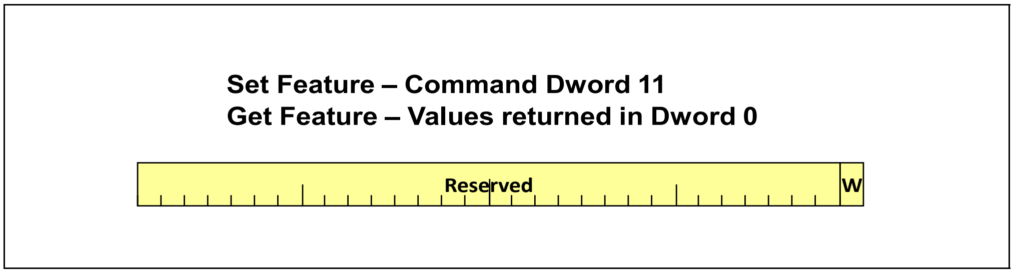

Set Feature – Command Dword 11
Get Feature – Values returned in Dword 0
Reserved
W

Figure 59: Write Volatile Cache Feature Enable

The W bit is Volatile Write Cache Enable. If set, the write cache is enabled.

Feature ID 07h - Number of Queues

The host requests that the controller make available a certain number of SQs and CQs by placing that value in DWord 11 and issuing a Set Features with Feature ID 07h.

The hosts then issues a Get Features command with Feature ID 07h. The controller returns in DWord 0 how many SQs and CQs the controller allocated (made available). Note: the number of queues allocated may be larger, equal to, or smaller than the number requested.

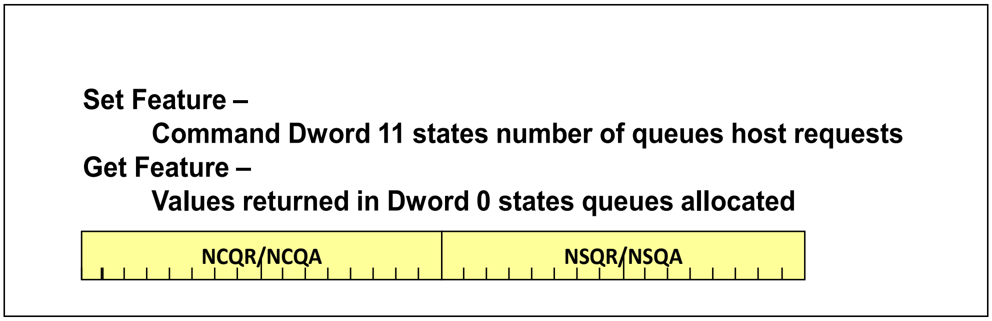

Set Feature –
Command Dword 11 states number of queues host requests
Get Feature –
Values returned in Dword 0 states queues allocated
NCQR/NCQA
NSQR/NSQA

Figure 60: Number of Queues Features

NCQR - Number of CQs requested.
NCQA - Number of CQs allocated.
NSQR - Number of SQs requested.
NSQA - Number of SQs allocated.
All four of the above numbers are 0's based. Minimum number allowed is one queue.

Feature ID 08h - Interrupt Coalescing

This feature allows the grouping of CQ interrupts to present to the host. Under normal processing, when the controller has something to tell the host, it presents an interrupt, the host stops what it is processing, saves its current conditions, handles the interrupt, restores the stored conditions, then returns to what it was processing before it was interrupted. By grouping (Coalescing) the interrupts, the host does not get interrupted as frequently and, in theory, makes processing more efficient[9].

If Interrupt Coalescing is defined, when an interrupt event occurs, the interrupt is delayed until the aggregation time expires or the aggregation threshold is reached, whichever occurs sooner. When one of those two limits occurs, all of the interrupts that have accumulated are presented to the host.

This feature applies to pin-based interrupts, MSI or MSI-X. If the interrupt mode is changed the host should re-issue this feature to ensure the values are set as desired.

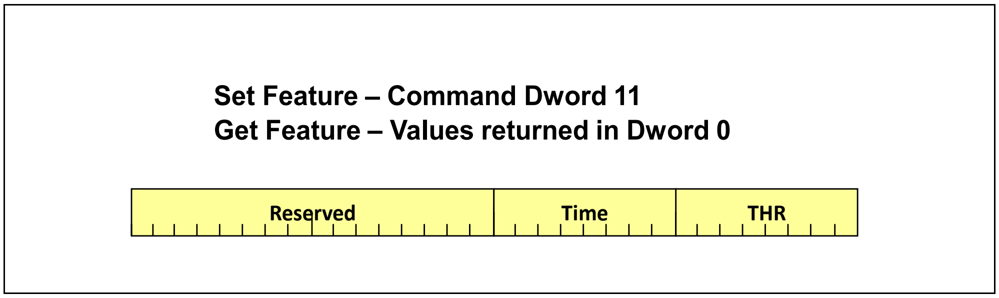

Set Feature – Command Dword 11
Get Feature – Values returned in Dword 0

Figure 61: Interrupt Coalescing Feature Values

The Time field (Aggregation Time) specifies in 100 ms the maximum time the controller may delay reporting (aggregate) interrupts. When the controller has an interrupt to report, it will start the timer and report the interrupt(s) when or before the time expires. The controller may apply this time per interrupt vector or across all interrupt vectors. This value is set to 0h on reset.

The THR (Aggregation Threshold) field specifies the maximum number of CQ interrupts may be aggregated (accumulated) per interrupt vector before interrupting the host. This is a 0’s based value and is reset to 0h on reset.

Feature ID 09h - Interrupt Vector Configuration

By default, Interrupt Coalescing is enabled for each Interrupt Vector. This feature allows the host to disable the Coalescing feature on a per Interrupt Vector basis.

[9] The author read in a report that it takes about 1000 instructions per context switch. The report did not state what the computer hardware or operating system was so that number may be for a specific condition. Regardless of the true number in your case, aggregating interrupts will reduce the number of context switches, thus reducing the number of instructions required.

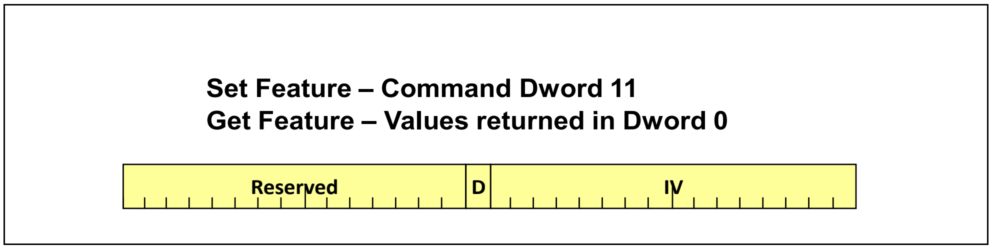

Set Feature – Command Dword 11
Get Feature – Values returned in Dword 0
Reserved
D
IV

Figure 62: Interrupt Vector Configuration Feature Values

The D field is Coalescing Disable, if set to 1b then Interrupt Coalescing is disabled for the Interrupt Vector specified in the IV (Interrupt Vector) field.

This feature must be issued with a Set Feature command for each Interrupt Vector the host wishes to disable Interrupt Coalescing.

Feature ID 0Ah - Write Atomicity Normal

This feature enables or disable Write Atomicity for normal operations (non-power fail operations).

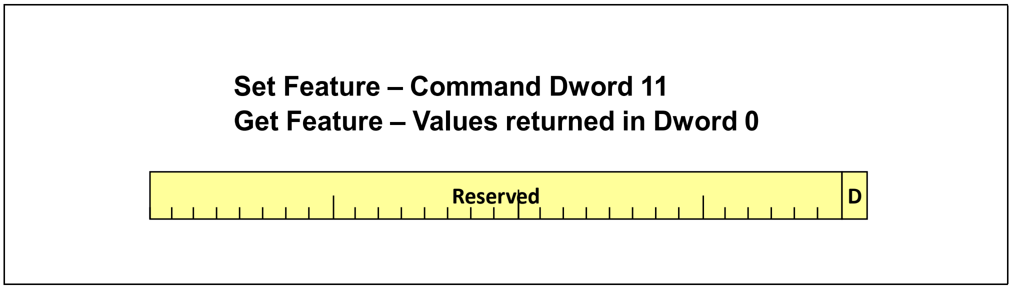

Set Feature – Command Dword 11
Get Feature – Values returned in Dword 0
Reserved
D

Figure 63: Write Atomicity Feature Value

If D (Disable Normal) is set to 1b, then the atomic write unit is required for Power Fail operations (AWUPF and NAWUPF) but not required for normal operations (AWUN and NAWUN). If D is cleared to 0b, the controller shall honor the atomic write unit for normal operations. See "Atomic Write Unit" on page 146

Feature ID 0Bh - Asynchronous Event Configuration

The NVMe specification defines an Asynchronous Event command (with no time limit) that the host can issue multiple copies to the controller. When the controller needs to provide information about an event not associated with a command (such as overtemp or device reliability issues) the controller can respond to one of the Asynchronous Event commands with ending status and the information in the SMART/Health Information Log. See "Get Log Page" on page 108.

With this Set Features command the host tells the controller what types of events the host wants the controller to complete the Asynchronous Event Command with ending status. With this Get Features command the controller tells the host what events are set in the controller to report Asynchronous Events command completion. With the Asynchronous Events command completion the controller tells the host what events did occur. See also "Asynchronous Event Request" on page 107 for the associated command and its ending status.

Set Features with Feature ID 0Bh enables (1b) or disables (0b) Asynchronous Event reporting when a listed event occurs.

Get Features with Feature ID 0Bh returns the settings of the attributes listed in Figure 64.

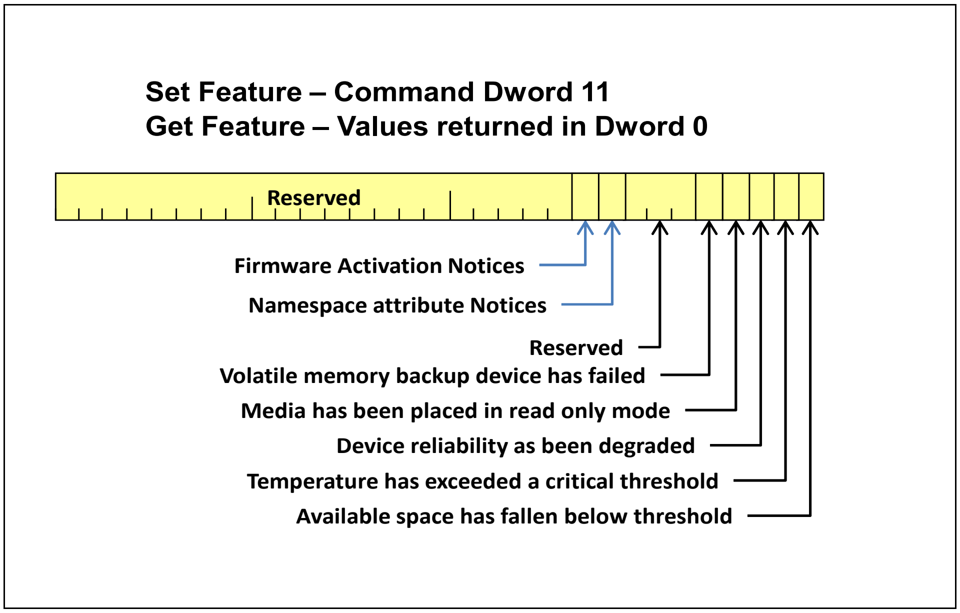

Set Feature – Command Dword 11
Get Feature – Values returned in Dword 0
Reserved
Firmware Activation Notices
Namespace attribute Notices
Reserved
Volatile memory backup device has failed
Media has been placed in read only mode
Device reliability as been degraded
Temperature has exceeded a critical threshold
Available space has fallen below threshold

Figure 64: Asynchronous Event Configuration Feature Value

If any of the bits 4:0 in Figure 64 are set, the controller can issue completion status to an outstanding Asynchronous Event Reporting command when that critical warning is written to the SMART/Health Information Log for that condition. If the bit is cleared to 0b, the controller will not issue ending status to an Asynchronous Event Reporting command.

If bit 8 is set, the controller will send an Asynchronous Event command completion status with Namespace Attribute Changed event to the host when this condition occurs. If bit 9 is set, the controller will set the Firmware Activation Starting bit in the completion status to the host when this condition occurs.

Feature ID 0Ch - Autonomous Power State Transition

If the controller supports this optional feature, the host can instruct the controller, “After you are idle for x ms, transition to power state y.” The host will use this feature to transfer a data structure containing the x and y for up to 32 Power States. Power States that are not supported or those not used will contain eight bytes of 00h in the data structure.

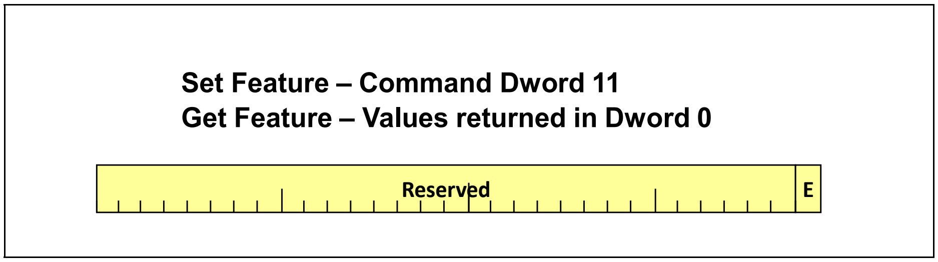
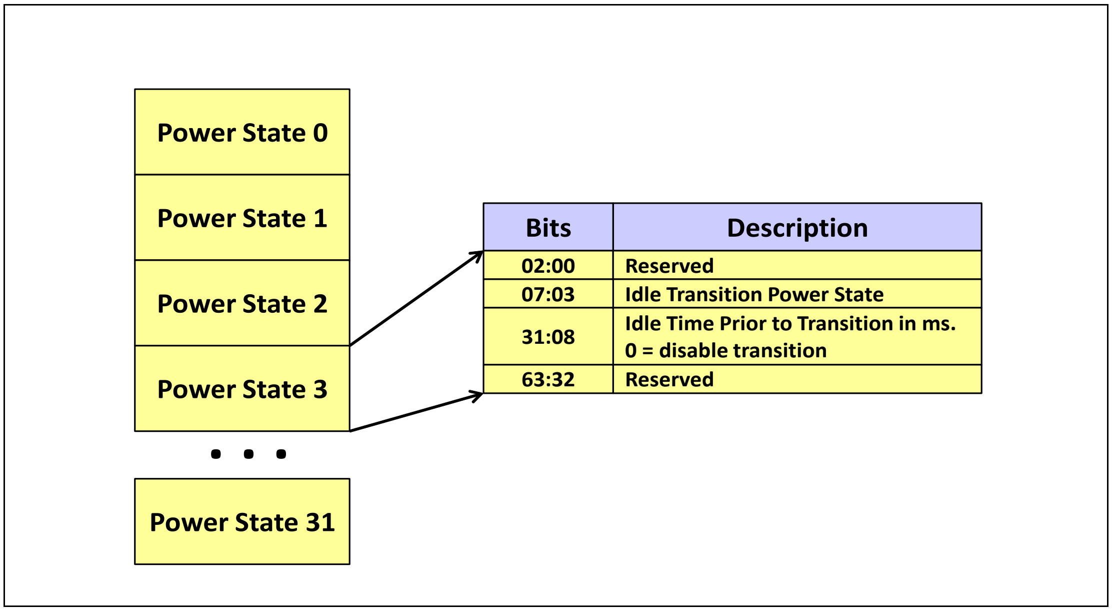

Set Feature – Command Dword 11
Get Feature – Values returned in Dword 0
Reserved
E

Figure 65: Autonomous Power State Transition Feature Enable

DWord 11 of the command for this feature contains the E bit to enable or disable this feature.

The PRP Entry points to the data structure shown in Figure 66. If the controller has been idle in its current power state for x ms (bits 31:08) then it will transfer on its own to power state y (bits 07:03).



Bits	Description
02:00	Reserved
07:03	Idle Transition Power State
31:08	Idle Time Prior to Transition in ms. 0 = disable transition
63:32	Reserved

Figure 66: Autonomous Power State Transition Description

Feature ID 0Dh - Host Memory Buffer

The Host Memory Buffer (contrast to Controller Memory Buffer) allows the host to provide its physical memory to the controller to improve the operation of the controller. The memory provided is “as available” and cannot be counted on; in other words, the controller must be able to full function properly whether it receives the host memory or not. Also, once provided to the controller, the host is prohibited from accessing that memory.

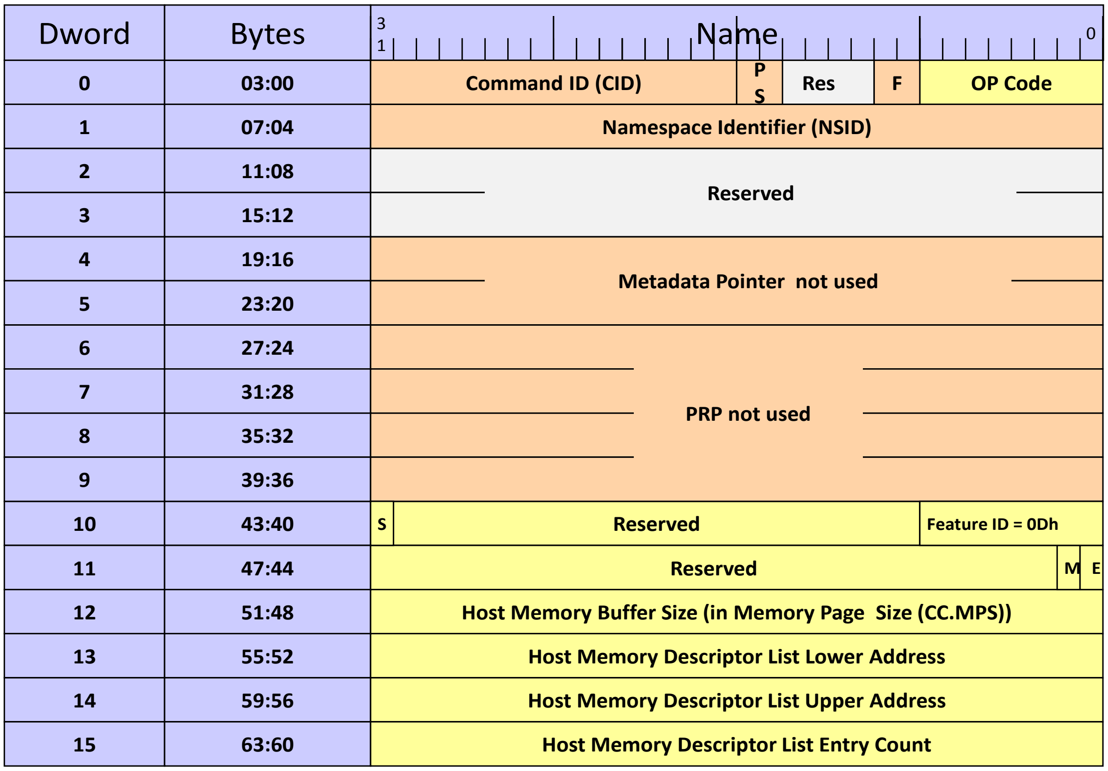
Two possible uses for this feature could be:

1) to keep the cost of the controller as low as possible, install a bare minimum of memory, and

2) install enough memory to allow normal operations, but have a method for the host to provide additional memory for high workloads. For number 2) to work, the host would have to know when the high workload existed and enable the extra memory, then disable it when the workload returned to normal.

The controller communicates its desired (preferred) Host Memory Buffers size (HMPRE) in the Identify Controller data structure, bytes 275:272, and its minimum size (HMMIN) in bytes 279:276. The numbers in those two fields are a quantity of 4KB[10] units. If the HMPRE field is greater than 0h then the feature is supported; if the HMPRE field equals 0h the feature is not supported. If the HMMIN field equal 0h (and the HMPRE is not 0h), then the host is requested to issue any amount of host memory, up to the HMPRE.

The information required to implement this feature is communicated in the command specific registers as shown in Figure 67.



Dword	Bytes	Name
0	03:00	Command ID (CID) PS Res F OP Code
1	07:04	Namespace Identifier (NSID)
2	11:08	Reserved
3	15:12	
4	19:16	Metadata Pointer not used
5	23:20	
6	27:24	PRP not used
7	31:28	
8	35:32	
9	39:36	
10	43:40	S Reserved Feature ID = 0Dh
11	47:44	Reserved M E
12	51:48	Host Memory Buffer Size (in Memory Page Size (CC.MPS))
13	55:52	Host Memory Descriptor List Lower Address
14	59:56	Host Memory Descriptor List Upper Address
15	63:60	Host Memory Descriptor List Entry Count

Figure 67: Host Memory Buffer Feature Command

The opcode field will have 09h for Set Features or 0Ah for Get Features command. Dword 10 specifies the Feature ID and the “S” bit indicating to the controller that it should save the attributes or not.

Dword 11. The E bit is Enable Host Memory.

[10] KB is defined in the NVMe Specification, under Conventions, as 1024.

Dword 11. The M bit set to 1b indicates the host is returning the exact information describing host memory that existed in the controller before a reset or entering the Runtime D3 state. The returned host memory buffer will have the exact same size, descriptor list address, descriptor list contents, and host memory buffer contents as before the E bit was cleared to 0b. If the M bit is cleared to 0b, then the host is allocation host memory resources with undefined content.

Dword 12. Please note that the controller requests an amount of host memory by specifying a number of 4KB units in the Identify Controller data structure, but the host allocates the host memory buffers as a number of CC.MPS units. Memory page size is a function of the host memory manager, and it is probable that the information has not been communicated to the Controller Configuration (CC) register when the host first reads the Identify Controller. The host must set the CC.MPS before issuing the Host Memory Buffer Set Feature.

Dwords 13 and 14 specify the 64 bit address of the host memory address of the Host Memory Descriptor List that describes the Host Memory Buffer. Dword 15 indicates the number of entires in the Descriptor List.

Bytes	Description
15:0	Host Memory Buffer Descriptor 0
31:16	Host Memory Buffer Descriptor 1
. . .	. . .
16*n+15: 16*n	Host Memory Buffer Descriptor n

Table 22: Host Memory Buffer Descriptor List

Bits	Description
127:96	Reserved
95:64	Buffer Size in MPS increments
63:00	Buffer Address

Table 23: Host Memory Buffer Descriptor Contents

For Get Features, Dword 0 of the returned information contains the same contents as Dword 11 of the Set Features (the Enable bit and the Memory Returned bit).

The data buffer contains the Attributes Data Structure shown in

Byte	Description
3:0	Host Memory Buffer size in memory page size units
7:4	Host Memory Descriptor List Address - lower
11:8	Host Memory Descriptor List Address - upper
15:12	Host Memory Descriptor List Entry count
4095:16	Reserved

Figure 68: Get Features Host Memory Buffer Attributes Data Structure

Feature ID 0Fh - Keep Alive Timer

See Keep Alive Command.

The Keep Alive Timer feature is used by the host to determine that the controller is operational, and by the controller to determine that the host is operational.

The controller communicates in Identify Controller bytes 321:320 its desired maximum value for Keep Alive commands in 100 ms increments. If the controller does not support the Keep Alive timer, it will report 0h in these two bytes. Keep Alive shall be supported in all controllers supporting NVMe over Fabrics.

NVM Command Set Specific Admin Commands

Feature ID 80h - Software Progress Marker

The Software Progress Marker feature is an indication to the Operating System (OS) of how many times the OS loaded before it successfully initialized.

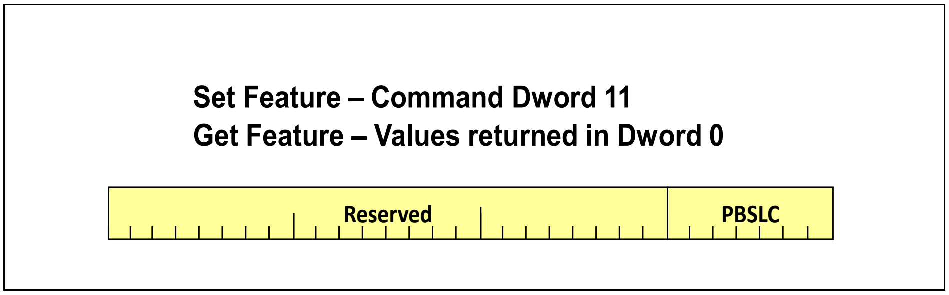

Set Feature – Command Dword 11
Get Feature – Values returned in Dword 0
Reserved
PBSLC

Figure 69: Pre-Boot Software Load Count Feature Value

PBSLC is Pre-boot Software Load Count, a counter indicating the number of times the pre-boot software loaded. After successfully loading and initializing the controller, the pre-boot software should increment this field. It is persistent across resets and power cycles, does not wrap (it is a saturation counter), and is set to 0 by the OS software after the OS has been successfully initialized.

Feature ID 81h - Host Identifier

This feature allows the host to register a 64-bit Host ID with the controller. Host ID is used by controller to determine if other controllers in the NVM subsystem are associated with the same host. Host ID, method of assigning ID, and method of ensuring uniqueness are outside scope of the specification. Host ID equal to 00h indicates that the host is not associate with any other controllers in the NVM system.

This feature is optional unless reservations are supported.

Figure 70shows the data structure pointed to by the PRP Entry.

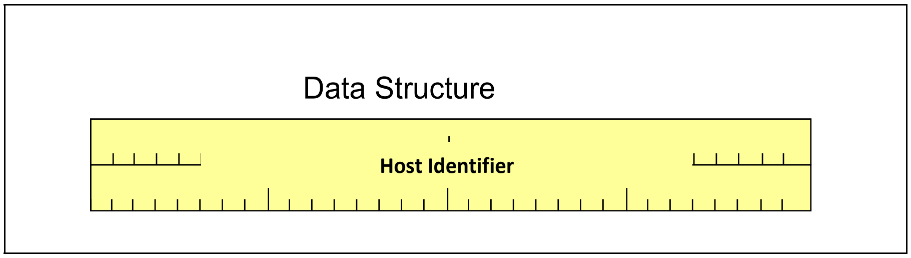


Figure 70: Host ID Feature Data Structure

Feature ID 82h - Reservation Notification Mask

This feature allows the host to mask (prohibit) three types of Reservation Notifications on a per namespace basis. If the Set Features command with Feature ID = 82h contains a Namespace ID = FF FF FF FFh this command applies to all namespaces on the controller, otherwise only to the identified namespace. If the Get Features command specifies a Namespace ID of FF FF FF FFh the command is aborted with the status of Invalid Field in Command.

Set Features turns the masks on or off. Get Features reports the mask's settings.

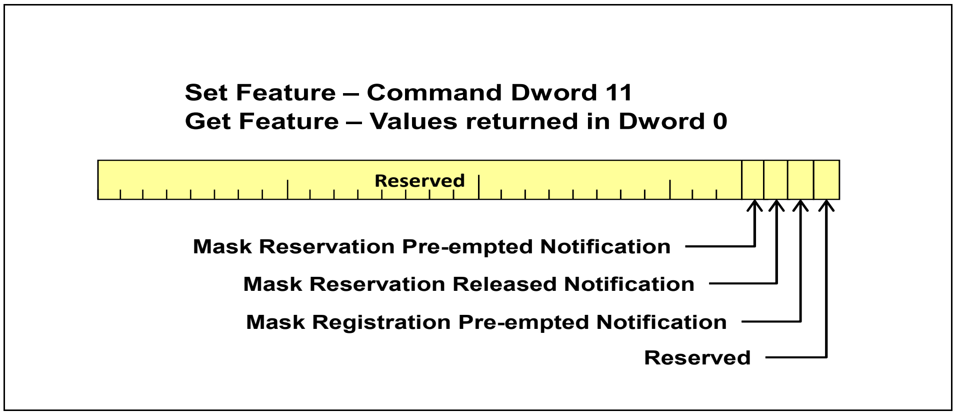
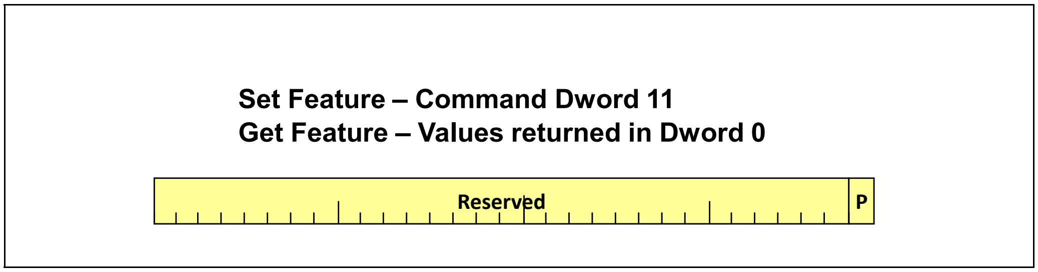

Set Feature – Command Dword 11
Get Feature – Values returned in Dword 0
Reserved
Mask Reservation Pre-empted Notification
Mask Reservation Released Notification
Mask Registration Pre-empted Notification
Reserved

Figure 71: Reservation Notification Feature Controls

Feature ID 83h - Reservation Persistence

By default, Reservations and Registrations persist across resets and are cleared by power loss. This feature can cause Registrations and Reservations to persist across power loss on a namespace basis. If the Set Features command specifies a Namespace ID of FF FF FF FFh, the command applies to all namespaces on the controller, otherwise to the indicated namespace only. If the Get Features command specifies a Namespace ID of FF FF FF FFh the command is aborted with the status of Invalid Field in Command.



Set Feature – Command Dword 11
Get Feature – Values returned in Dword 0

Figure 72: Reservation Persistence Through Power Loss

P field set to 1b indicates to the identified namespace (or all namespaces) that the Reservations and Registrations are persistent across power loss.

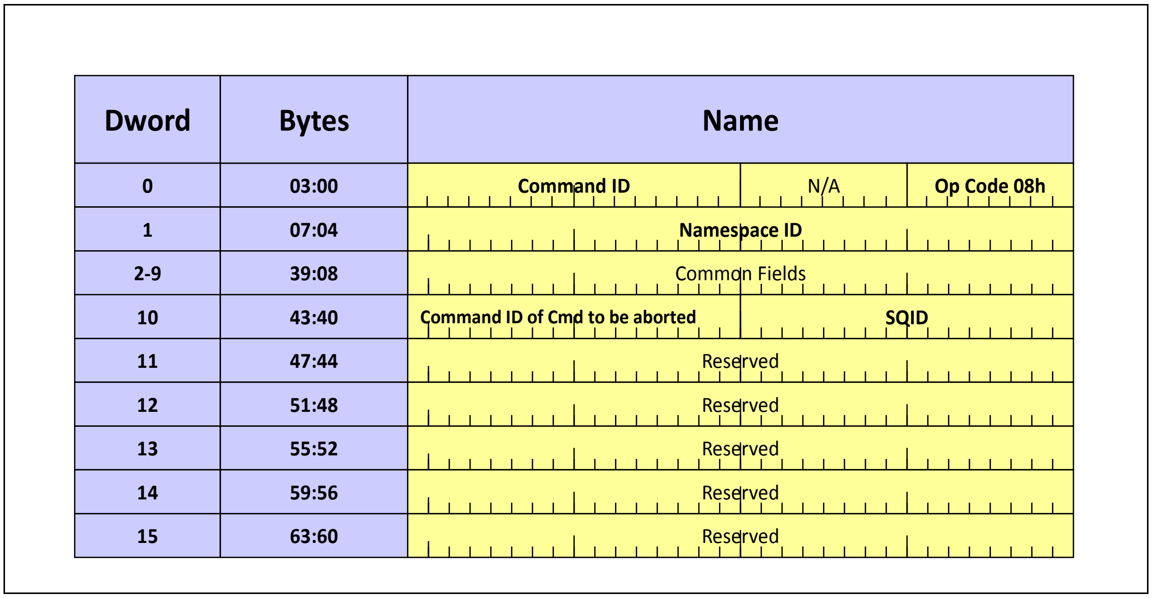
Other Commands

Abort Command

Every command in an NVMe environment can be uniquely identified by a concatenation of the SQ ID and the Command ID. By submitting that information to the controller on the Admin SQ, the controller can abort that command.

Interesting point to make: This is a command, so it has its own Command ID in DWord 0, as do all other commands.

There are no facilities to abort a group of commands with a single command. To abort a group of commands, the host must issue multiple abort commands. The abort commands will be executed on a “best effort basis.”



Dword	Bytes	Name
0	03:00	Command ID N/A Op Code 08h
1	07:04	Namespace ID
2-9	39:08	Common Fields
10	43:40	Command ID of Cmd to be aborted SQID
11	47:44	Reserved
12	51:48	Reserved
13	55:52	Reserved
14	59:56	Reserved
15	63:60	Reserved

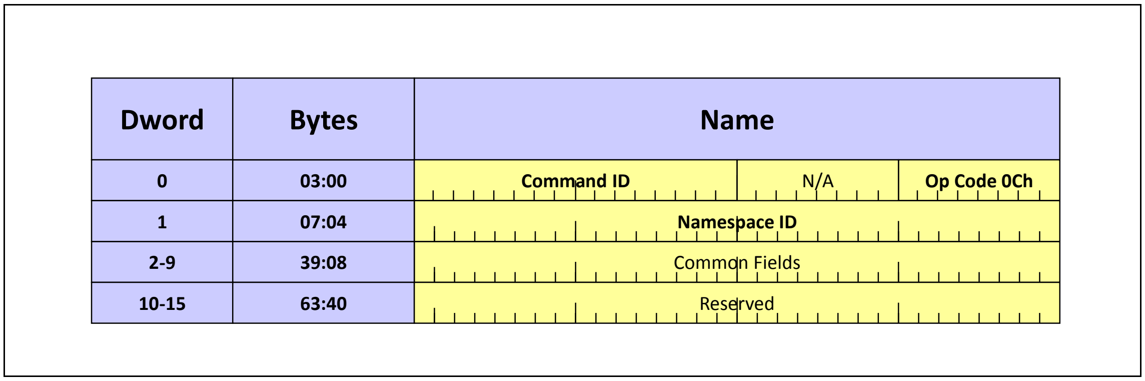
Figure 73: Abort Command

Asynchronous Event Request

The controller needs a mechanism to report events not associated with a command but that may affect operation of the controller or processing of other commands. Many other interfaces provide a similar mechanism in a number of different manners.[11]

NVMe specification defines the Asynchronous Event Request command that the host can send to the controller. These commands remain pending in the controller (have no timeout) until some event occurs that the controller needs to notify the host. The controller then returns ending status for one of the pending Asynchronous Event Request commands with event type in DWord 0, as shown in Figure 75. The host can then perform a Get Log Page to find more information about the event.

For information on how to configure this feature, see "Feature ID 0Bh - Asynchronous Event Configuration" on page 99.

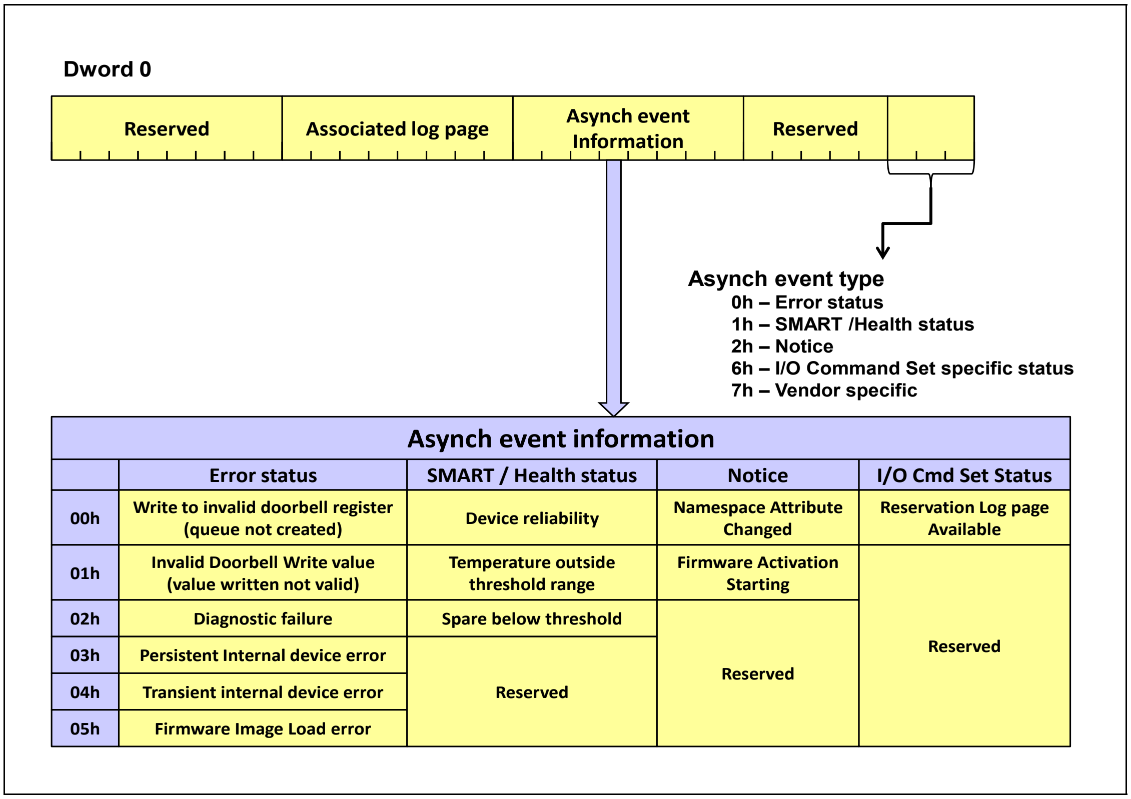


Dword	Bytes	Name		
0	03:00	Command ID	N/A	Op Code 0Ch
1	07:04	Namespace ID		
2-9	39:08	Common Fields		
10-15	63:40	Reserved		

Figure 74: Asynchronous Event Request

Figure 75 shows DWord 0 of the completion status. Read bits 2:0 first to get the Asynchronous Event type, then bits 15:8 to discover the specific status, then 23:16 to get the log page that describes this event.

[11] SCSI uses Asynchronous Event Reporting (AER) and Unit Attention (Sense Key = 6)



Asynch event type
0h – Error status
1h – SMART /Health status
2h – Notice
6h – I/O Command Set specific status
7h – Vendor specific

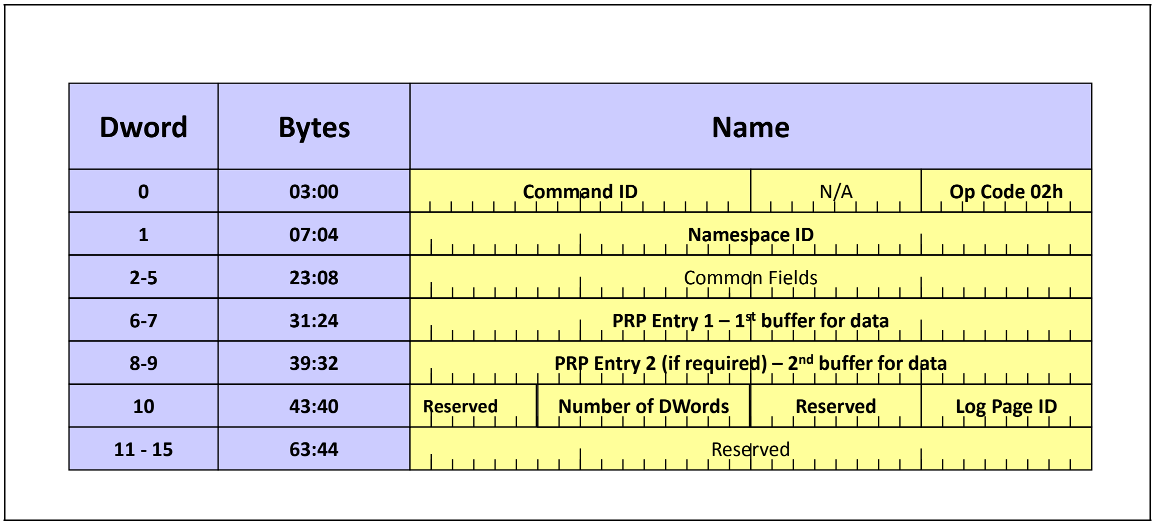
Asynch event information				
	Error status	SMART / Health status	Notice	I/O Cmd Set Status
00h	Write to invalid doorbell register (queue not created)	Device reliability	Namespace Attribute Changed	Reservation Log page Available
01h	Invalid Doorbell Write value (value written not valid)	Temperature outside threshold range	Firmware Activation Starting	Reserved
02h	Diagnostic failure	Spare below threshold	Reserved	
03h	Persistent Internal device error	Reserved		
04h	Transient internal device error			
05h	Firmware Image Load error			

Figure 75: Asynchronous Event Request Completion DWord 0

Get Log Page

Compared to other transports that have been around for a long time, NVMe Logs are very simple. There are only five different log pages for Admin Commands and one for NVMe commands, no sub-pages and no list of parameters to define the pages differently. To get a log page, build a command with:

OpCode = 02h,
give it a Command ID,
determine and communicate the Namespace ID,
provide a buffer with one or two PRP entries (no PRP pointers or SGL),
state a maximum number of DWords to return, and
tell the ID of desired log page.



Dword	Bytes	Name			
0	03:00	Command ID		N/A	Op Code 02h
1	07:04	Namespace ID			
2-5	23:08	Common Fields			
6-7	31:24	PRP Entry 1 – 1st buffer for data			
8-9	39:32	PRP Entry 2 (if required) – 2nd buffer for data			
10	43:40	Reserved	Number of DWords	Reserved	Log Page ID
11 - 15	63:44	Reserved			

Figure 76: Get Log Page Command

The PRP entries point to one or two buffers in memory to return the requested log page.

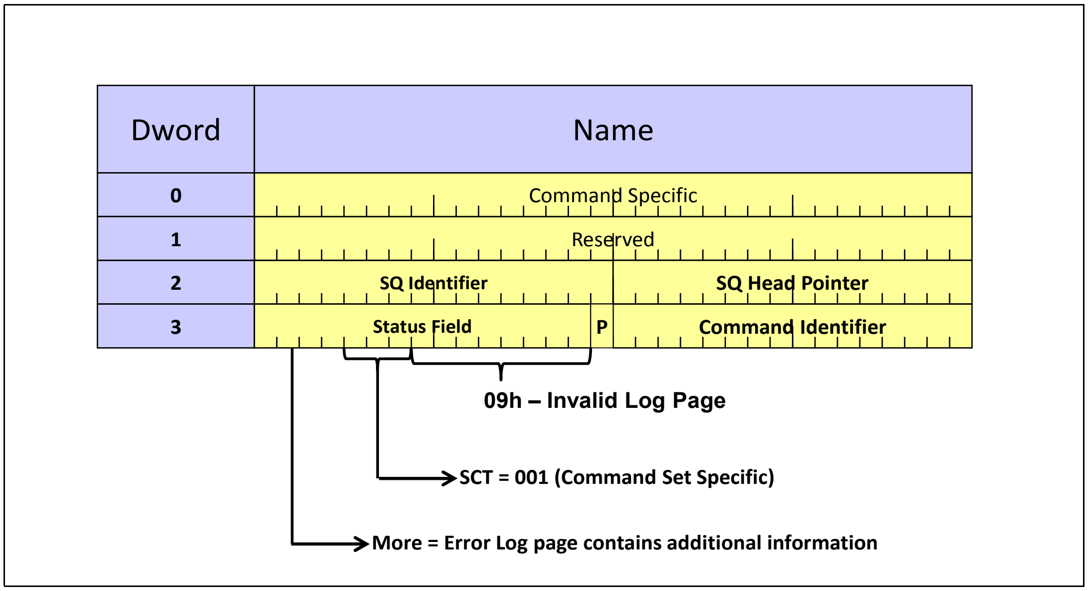
The Number of DWords tells the controller the how many DWords to return (0's based). If the host specifies a size larger than the log page requested, the results are undefined.

The Log Page ID specifies which log page to return. See Table 24 below for the page number identification.

Log Identifier	Description
00h	Reserved
01h	Error Information
02h	SMART/Health Information
03h	Firmware Slot Information
04h	Changed Namespace List
05h	Command Effects Log
80-BFh	I/O Command set specific
80h	Reservation Notification
C0-FFh	Vendor Specific

Table 24: Log Page ID Description

Two types of errors are defined in the NVMe specification for the Get Log Page commands: Invalid Log Page and "More" which is bit 30 of DWord 3. "More" indicates that an Error Log Page was created for this failure.



Dword	Name		
0	Command Specific		
1	Reserved		
2	SQ Identifier		SQ Head Pointer
3	Status Field	P	Command Identifier

09h – Invalid Log Page

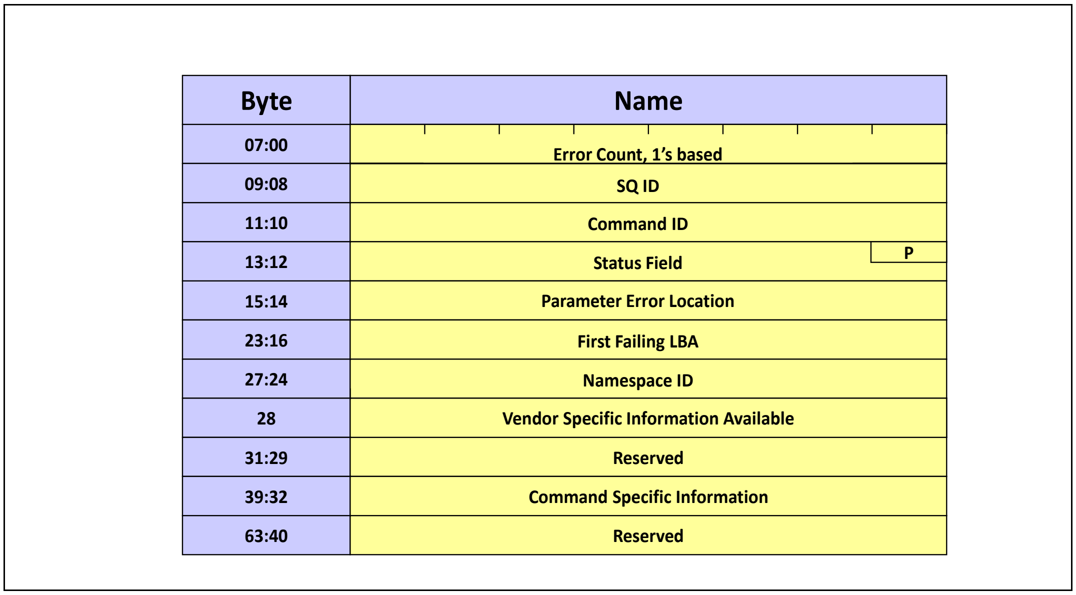
SCT = 001 (Command Set Specific)

More = Error Log page contains additional information

Figure 77: Get Log Page Command Completion

Log Identifier 1

The Error Log Page gives identifying information of where the error occurred.



Byte	Name
07:00	Error Count, 1's based
09:08	SQ ID
11:10	Command ID
13:12	Status Field P
15:14	Parameter Error Location
23:16	First Failing LBA
27:24	Namespace ID
28	Vendor Specific Information Available
31:29	Reserved
39:32	Command Specific Information
63:40	Reserved

Figure 78: Log Page 1 - Error Information

The Error Count is a 1's based incrementing counter that is persistent across power cycles. 0h means an invalid entry such as lost entries.

The SQ ID and Command ID uniquely identify the failing command. If the failure is not specific to a particular command these fields are set to FFFFh.

The Status Field is the 15 bit status field of the Completion Status; the P bit is the phase bit.

The Parameter Error Location (Table 25) indicates the byte and bit of the command parameter that is associated with the error. If the error spans multiple bits or bytes, then this field identifies the first failure. If the error is not specific to a bit or byte, then this field is set to FFFFh.

Bits	Description
7:0	Byte location
10:8	Bit location
15:11	Reserved

Table 25: Parameter Error Location

The Failing LBA field indicates the first failing LBA, if applicable.

The Namespace ID indicates the failing Namespace, if applicable.

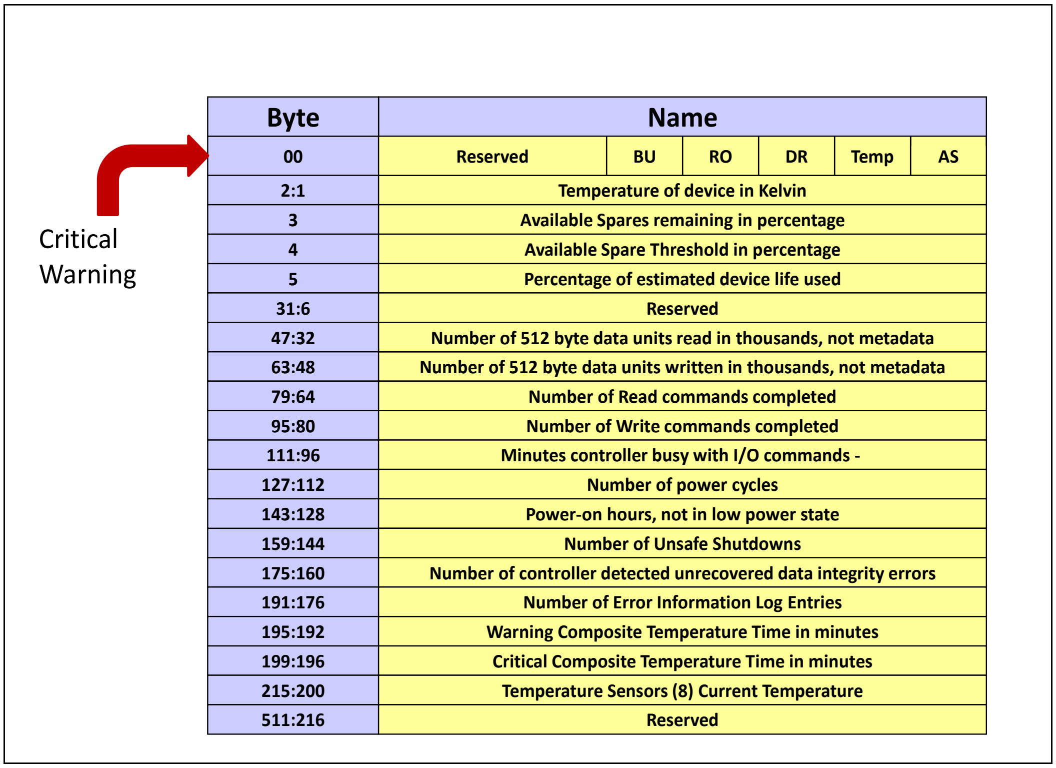
Log Identifier 2

The SMART or Health information is usually given on a device (controller and NVM) basis. The log page may also be supported on a namespace basis if indicated in Identify Controller data structure byte 261, bit 0 (1b = namespace basis). If not supported on a namespace basis, specifying any Namespace ID other than FFFFFFFFh should give Command Aborted with status of Invalid Field in Command.

Log Page 2 gives two types of information:
Critical Warnings possibly with descriptions and
Information for predictive failure analysis.

SMART stands for "Self-Monitoring, Analysis, and Reporting Technology." It was designed for predictive failure analysis. The information given on this log page provides information on usage:
Number of blocks read and written,
Number of Power-on hours,
Number of read and write commands processed, and
Number of Un-safe shutdowns.
In addition, Health information on the device detecting errors is given:
Temperature of device (controller and NVM),
Available spares remaining and threshold,
Number of unrecovered data integrity errors, and
Number of Error Information Log Entries.

T


Critical
Warning

Byte	Name					
00	Reserved	BU	RO	DR	Temp	AS
2:1	Temperature of device in Kelvin					
3	Available Spares remaining in percentage					
4	Available Spare Threshold in percentage					
5	Percentage of estimated device life used					
31:6	Reserved					
47:32	Number of 512 byte data units read in thousands, not metadata					
63:48	Number of 512 byte data units written in thousands, not metadata					
79:64	Number of Read commands completed					
95:80	Number of Write commands completed					
111:96	Minutes controller busy with I/O commands -					
127:112	Number of power cycles					
143:128	Power-on hours, not in low power state					
159:144	Number of Unsafe Shutdowns					
175:160	Number of controller detected unrecovered data integrity errors					
191:176	Number of Error Information Log Entries					
195:192	Warning Composite Temperature Time in minutes					
199:196	Critical Composite Temperature Time in minutes					
215:200	Temperature Sensors (8) Current Temperature					
511:216	Reserved					

Figure 79: Log Page 2 - SMART/Health Information

Bits	Description
AS	Available Spares has fallen below threshold
Temp	Temperature has exceeded critical threshold
DR	Device Reliability has been degraded
RO	Medial has been placed in Read Only Mode
BU	Backup Device for volatile memory has failed
Reserved	Reserved

Figure 80: Critical Warning Field bits

The Available Spare Space Remaining is a percentage of the capacity of the NVM that is available. An Asynchronous Event ending status may be reported when the available capacity falls below the percentage value in Available Spare Threshold field. Range for both of these fields is from 0 to 100%.

Percentage of estimated device life used may report values from 0 to 254. If the used time exceeds the estimated life by more than 254%, then this field reports 255.

The Number of 512 byte data units read or written is returned in thousands of data units and rounded up to the next thousand. Metadata is not included. Compare and Read commands are included in the Read data units. Write commands are included in the Write data units but Write Uncorrectable commands are not. Write Zeros commands are not mentioned in the specification in this regard.

Unsafe Shutdown is defined as when a shutdown notification is not received prior to loss of power.

Number of controller unrecovered data integrity errors includes errors such as:

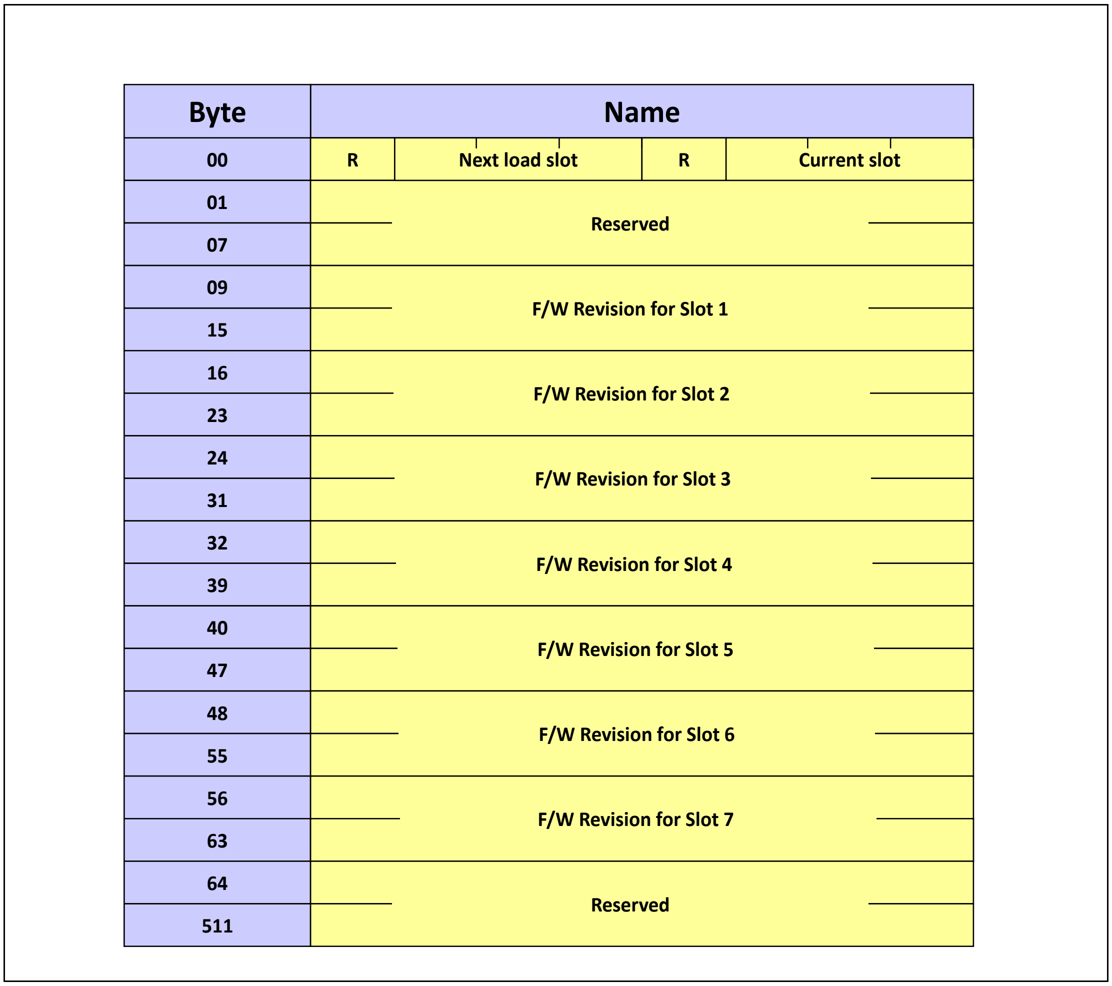
Uncorrectable ECC,
CRC checksum failure, and
LBA tag mismatch.

Log Identifier 3

Log page 3 provides an easy way to track the firmware revisions in each of the firmware slots, plus from which slot the currently running firmware was loaded and which slot contains the firmware to be activated next time the controller is reset.

If no firmware is in a particular slot or if that slot is not supported, the controller will return all zeros for that slot.

This log page is global to the device.



Byte	Name			
00	R	Next load slot	R	Current slot
01	Reserved			
07				
09	F/W Revision for Slot 1			
15				
16	F/W Revision for Slot 2			
23				
24	F/W Revision for Slot 3			
31				
32	F/W Revision for Slot 4			
39				
40	F/W Revision for Slot 5			
47				
48	F/W Revision for Slot 6			
55				
56	F/W Revision for Slot 7			
63				
64	Reserved			
511				

Figure 81: Log Page 3 - Firmware Slot Information

Log Identifier 04h

This is a list of up to 1024 NSIDs that that have changed since the last time this page was read. If more than 1024 namespaces have been changed since this page was last read, the first entry is set to FF FF FF FFh and the rest are zero filled.

Log Identifier 05h

Log page 5 is used to describe the commands that the controller supports and the effects of those commands on the state of the NVM subsystem. There is one 4-byte Command Effect data structure per possible Admin command and one 4-byte Command Effect data structure per possible I/O command.

This is of interest primarily when the submitted command may change the controller or namespace. If the command may change the namespace capability, the host should wait until all commands previously submitted complete, issue the command that may change the capability and wait for it to complete, then re-issue the Identify command and/or perform necessary initialization.

Bytes	Description
03:00	Admin Command Supported 0
07:04	Admin Command Supported 1
11:08	Admin Command Supported 2
...	...
1019:1016	Admin Command Supported 254
1023:1020	Admin Command Supported 255
1027:1024	I/O Command Supported 0
1031:1028	I/O Command Supported 1
...	...
2043:2040	I/O Command Supported 254
2047:2044	I/O Command Supported 255
4095:2048	Reserved

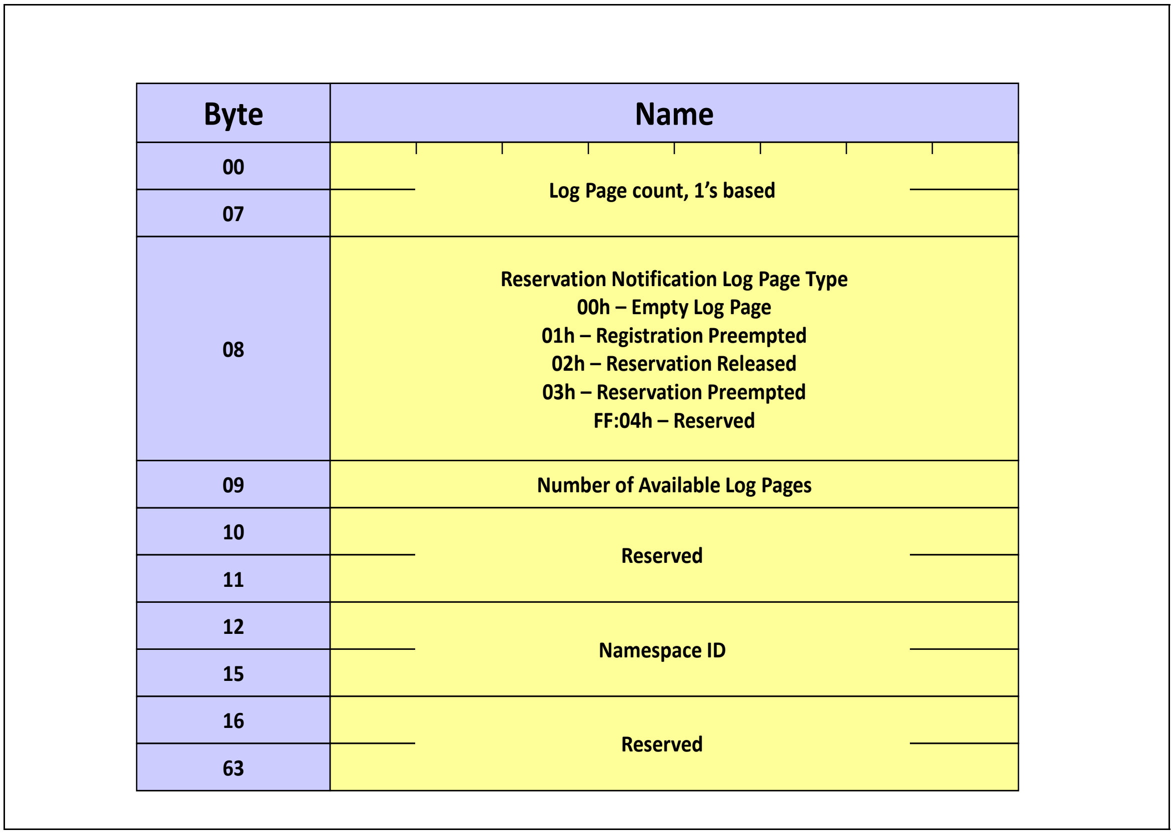
Table 26: Command Effect Log Page Contents

Bits	Description	
31:19	Reserved	
18:16	Value	Definition
	000b	No command submission or execution restriction
	001b	Command may be submitted when there is no other outstanding command to the same namespace and another command should not be submitted to the same namespace until this command is complete.
	010b	Command may be submitted when there is no other outstanding command to any namespace and another command should not be submitted to any namespace until this command is complete.
15:05	Reserved	
04	Controller Capability Change	
03	Namespace Inventory Change	
02	Namespace Capability Change	
01	Logical Block Content Change	
00	Command Supported	

Table 27: Command Effect Data Structure

Log Identifier 80h - Reservation Notification

A Reservation Notification log page is created when an unmasked reservation notification occurs on any namespace that may be assessed by the controller. A Get Log Page command returns a single reservation notification page to the PRP defined data buffer.



Byte	Name
00	Log Page count, 1's based
07	
08	Reservation Notification Log Page Type 00h – Empty Log Page 01h – Registration Preempted 02h – Reservation Released 03h – Reservation Preempted FF:04h – Reserved
09	Number of Available Log Pages
10	Reserved
11	
12	Namespace ID
15	
16	Reserved
63	

Figure 82: NVMe Log Page 80 - Reservation Notification Log

The Log Page Count field is a wrapping, incrementing counter of log pages; it is reset to 0h on controller reset and is incremented with every unmasked Reservation Notification (see "Feature ID 82h - Reservation Notification Mask" on page 105). This creates a unique identifier for this notification.

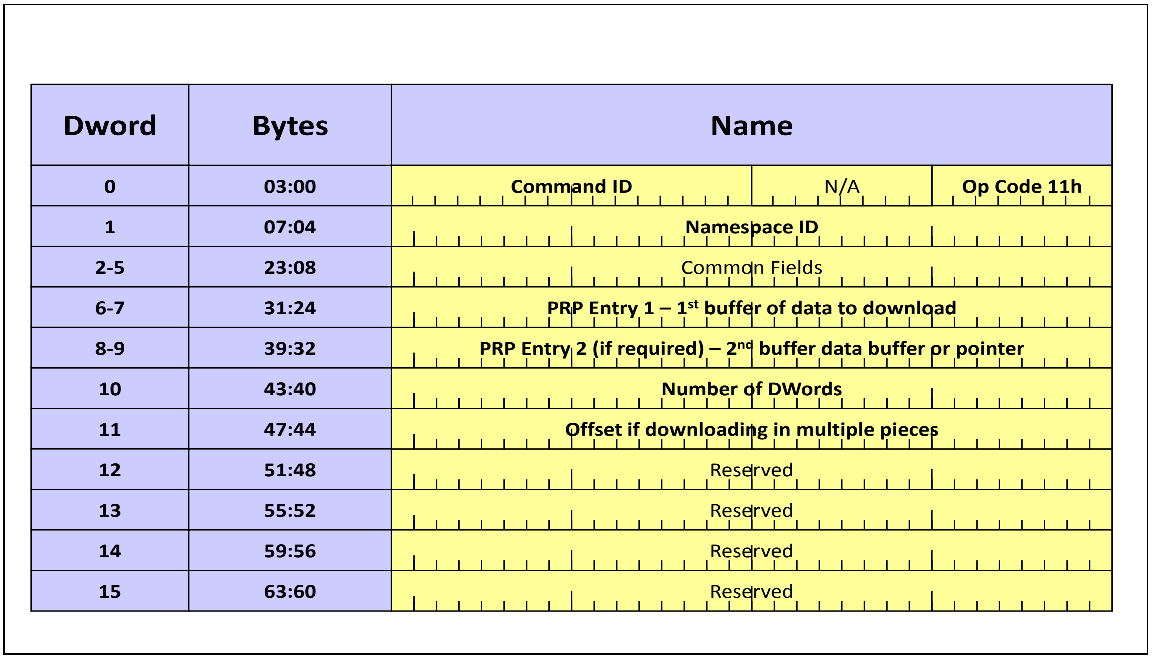
The Reservation Notification Log Page type indicates the Reservation Notification for the returned log page. A value of 00h means there are no available logs to return.

Number of Available Log Pages is a number of additional unread log pages. If there are more than 255 additional available log pages, the value of 255 will be returned.

Namespace ID is the namespace that is associated with the Reservation Notification described on this log page.

Firmware Download

Firmware Download command downloads into a controller buffer parts or all of a firmware revision. The parts do not have to be downloaded in order, but the controller must verify the parts have no DWord range overlaps. The Firmware Commit command will verify the download is valid, place it in a slot (1 - 7) and indicate the slot to use for firmware on the next controller reset.



Dword	Bytes	Name		
0	03:00	Command ID	N/A	Op Code 11h
1	07:04	Namespace ID		
2-5	23:08	Common Fields		
6-7	31:24	PRP Entry 1 – 1st buffer of data to download		
8-9	39:32	PRP Entry 2 (if required) – 2nd buffer data buffer or pointer		
10	43:40	Number of DWords		
11	47:44	Offset if downloading in multiple pieces		
12	51:48	Reserved		
13	55:52	Reserved		
14	59:56	Reserved		
15	63:60	Reserved		

Figure 83: Firmware Download Command

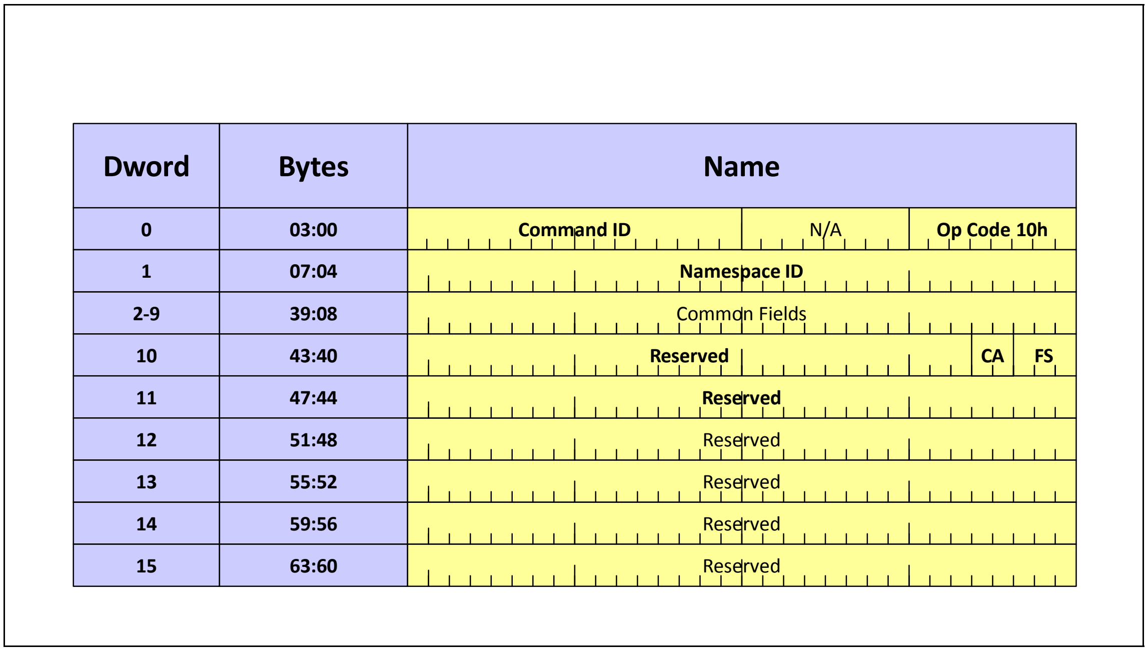
The PRP entries point to the host memory address from which to fetch the firmware.
The Offset is the starting point in that buffer.
The Number of DWords is how much to download on this command. This number is 0's based.

Firmware Commit

Firmware download fetches the firmware from host memory to a buffer in the controller. Firmware commit[12]:

validates the complete download,
places it in a specified slot, and
indicates the next slot for firmware load on controller reset.

[12] Firmware Commit was called Firmware Activate before NVMe Rev 1.2.



Dword	Bytes	Name		
0	03:00	Command ID	N/A	Op Code 10h
1	07:04	Namespace ID		
2-9	39:08	Common Fields		
10	43:40	Reserved	CA	FS
11	47:44	Reserved		
12	51:48	Reserved		
13	55:52	Reserved		
14	59:56	Reserved		
15	63:60	Reserved		

Figure 84: Firmware Commit Command

CA	Commit Action
00b	Place image in slot FS, image not activated
01h	Place image in slot FS, activate image at next reset
10b	Image in slot FS is activated at next reset

Figure 85: Firmware Activate Action field

FS is the Firmware slot. 000b means the controller will choose the next slot.

Keep Alive

See Keep Alive Feature.

The Keep Alive is a command used with the Keep Alive Timer Feature. It is a method to let the host know that the controller is active, and to let the controller know the host is still active. The host sends the Keep Alive command to the controller within the specified time limit, and the controller returns ending status for that command.

Operation:

1. Host will read bytes 321:320 of Identify Controller to get the controller’s desired timeout time. It is specified in 100 ms increments.
2. Host will issue the Keep Alive Admin command within that time limit.
3. Controller will process the Keep Alive Command as soon as possible and return ending status. Processing the command includes resetting the timeout timer.

If the Keep Alive Timer expires,

The controller records an Error Information Log Entry and sets the Controller Fatal Status (CSTS.SFS) bit.

The host assumes all outstanding commands are not completed and will reissue them if appropriate.

Other commands have no effect on the timer.

NMVe transports that cannot detect a connection loss in a timely manner must require that the Keep Alive timer is enabled.

Fabric Commands

Refer to Part Two of this book.

NVMe Command Set Specific Admin Commands

The NVMe specification allows for each command set to define commands that will be sent to the Admin queue. These commands have OpCodes in the range of 80-BFh. For the NVMe command set, there are three commands, shown in Table 28.

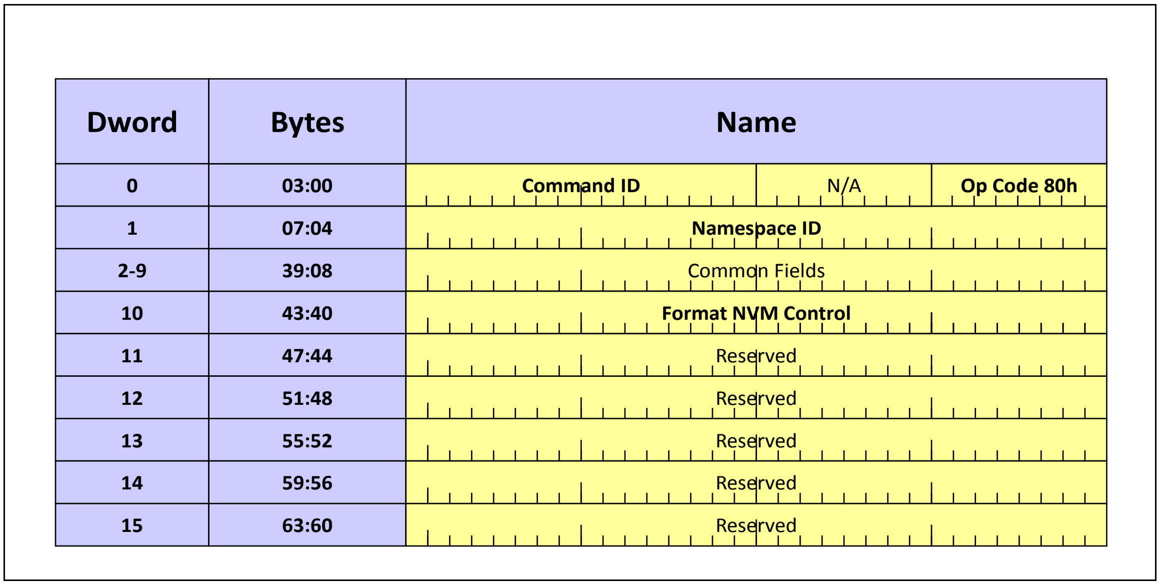
OpCode	Command	Namespace Used?
80h	Format NVM	Yes
81h	Security Send	Yes
82h	Security Received	Yes

Table 28: NVMe Command Set Admin Commands

Format NVM

This command provides capability to perform a low-level format on the NVM media to:

change LB data size,
change metadata size,
change or set protection information, or
perform Secure Erase.



Dword	Bytes	Name		
0	03:00	**Command ID**	N/A	**Op Code 80h**
1	07:04	**Namespace ID**		
2-9	39:08	Common Fields		
10	43:40	**Format NVM Control**		
11	47:44	Reserved		
12	51:48	Reserved		
13	55:52	Reserved		
14	59:56	Reserved		
15	63:60	Reserved		

Figure 86: Format NVM Command

See Figure 87 for the fields of DWord 10.

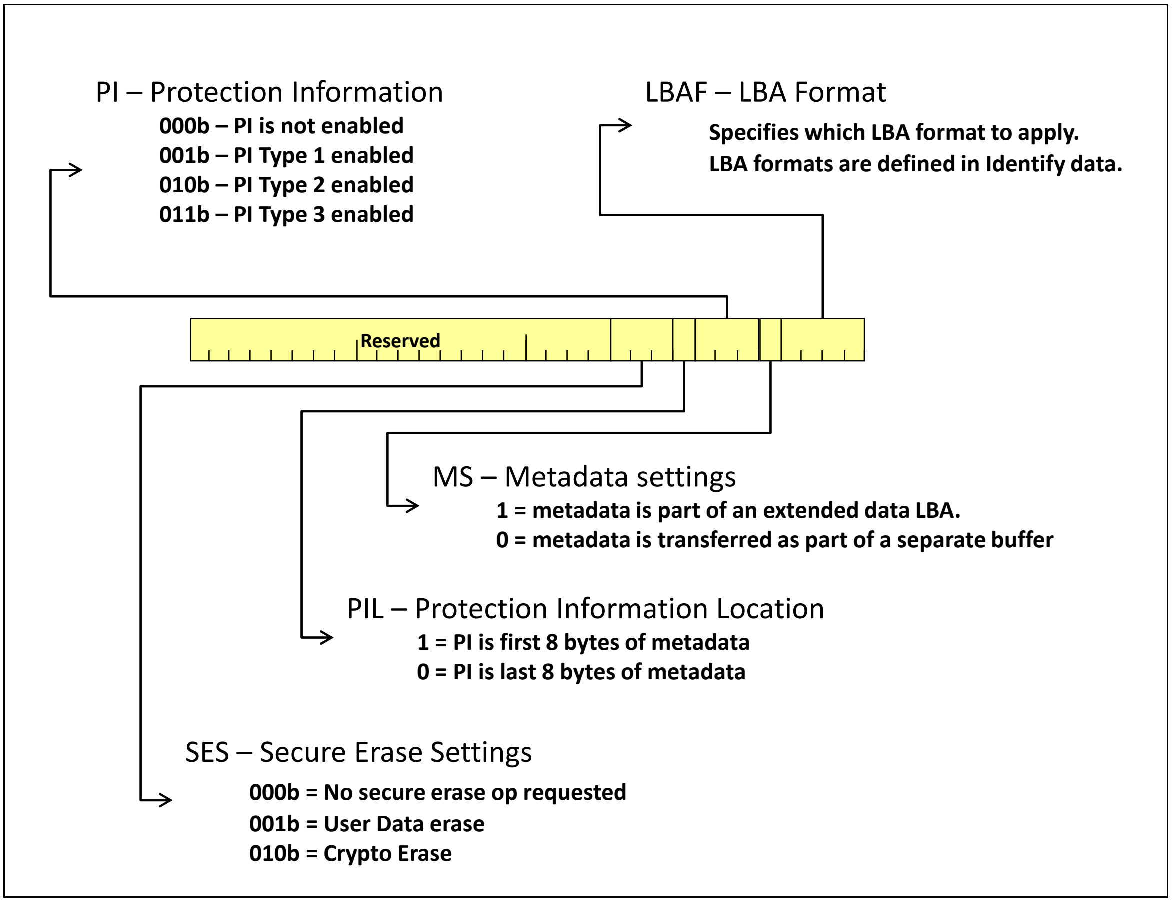

PI – Protection Information
000b – PI is not enabled
001b – PI Type 1 enabled
010b – PI Type 2 enabled
011b – PI Type 3 enabled
LBAF – LBA Format
Specifies which LBA format to apply.
LBA formats are defined in Identify data.
Reserved
MS – Metadata settings
1 = metadata is part of an extended data LBA.
0 = metadata is transferred as part of a separate buffer
PIL – Protection Information Location
1 = PI is first 8 bytes of metadata
0 = PI is last 8 bytes of metadata
SES – Secure Erase Settings
000b = No secure erase op requested
001b = User Data erase
010b = Crypto Erase

Figure 87: Format NVM Control - DWord 10

For information on PI field see “DIF” on page 9.
For information on MS and PIL fields see “Metadata” on page 55.
For information on LBAF field see Table 16 on page 88.

Under Secure Erase Settings, 001b erases all user content present in the NVM subsystem. 010b erases all user content present in the NVM subsystem by deleting the encryption key with which the user data was encrypted. Secure Erase may apply to all namespaces or a particular namespace. See Identify Controller byte 524, bit 1.

Security

The goals of security are listed below.

Communication security (protecting communications) is divided into:

Confidentially
Cryptographic data integrity
Peer Entity Authentication
Data Origin Authentication
Authorization
Access Controls

System Security (protecting systems from inappropriate usage) is divided into:

Denial of Service

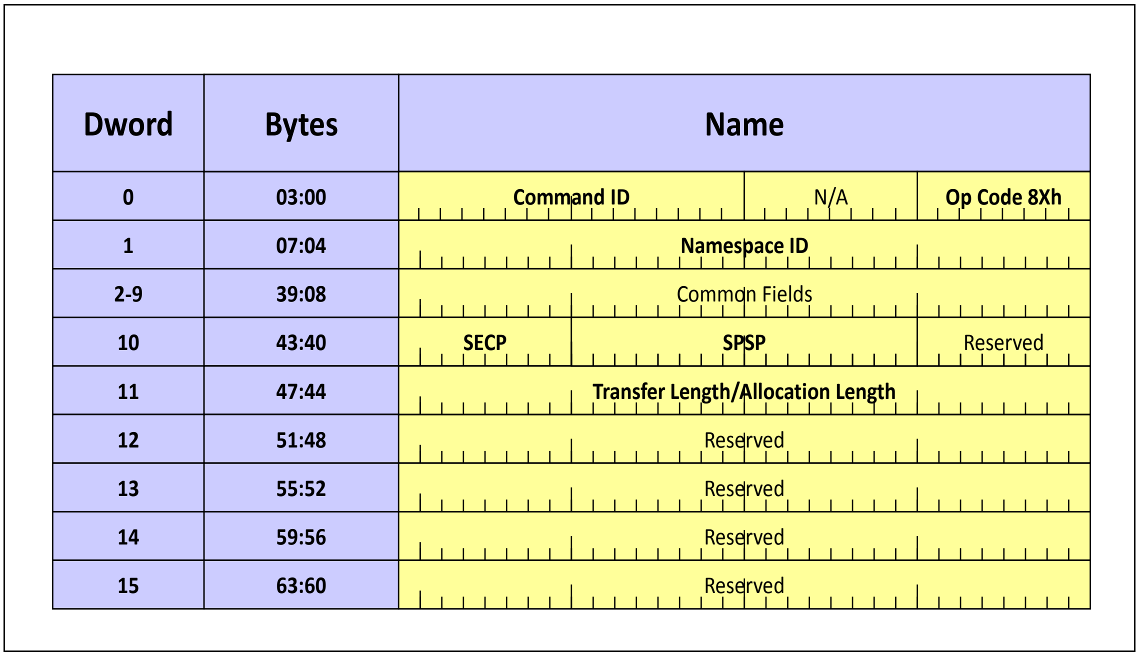
Unauthorized usage

It is recognized that given sufficient time, equipment and talent, any security system can be breached. Each security protocol has a cost in terms of design, testing and built prices, and additional overhead in performance. Each of the security protocols can protect against some threats to some level of protection; and cannot protect against other threats or a greater threat than the security was designed to cover. Therefore security protocols balance the cost vs. the benefit.

The NVMe specification references the SCSI Primary Commands - 4 (SPC4) standard for security definition. That standard is available in draft form from www.t10.org. It defines two security commands, Security Send and Security Receive. Please refer to SPC-4 for information on Security.

Security Send and Security Receive

Security Send command (OpCode 82h) sends security information from the host to the NVMe controller. It describes the number of bytes of data for transfer in the Transfer Length field.
Security Receive command (OpCode 81h) asks the controller to return the requested security information. It defines the size of the hosts buffer with the Allocation Length field.



<table>
<tr><th>Dword</th><th>Bytes</th><th colspan="3">Name</th></tr>
<tr><td>0</td><td>03:00</td><td>Command ID</td><td>N/A</td><td>Op Code 8Xh</td></tr>
<tr><td>1</td><td>07:04</td><td colspan="3">Namespace ID</td></tr>
<tr><td>2-9</td><td>39:08</td><td colspan="3">Common Fields</td></tr>
<tr><td>10</td><td>43:40</td><td>SECP</td><td>SPSP</td><td>Reserved</td></tr>
<tr><td>11</td><td>47:44</td><td colspan="3">Transfer Length/Allocation Length</td></tr>
<tr><td>12</td><td>51:48</td><td colspan="3">Reserved</td></tr>
<tr><td>13</td><td>55:52</td><td colspan="3">Reserved</td></tr>
<tr><td>14</td><td>59:56</td><td colspan="3">Reserved</td></tr>
<tr><td>15</td><td>63:60</td><td colspan="3">Reserved</td></tr>
</table>

Figure 88: Security Send Command

SECP is an NVMe specification abbreviation for Security Protocol. Protocols are those defined in SPC-4 plus protocol EAh defined in ACS-4.

SPSP is an NVMe specification abbreviation for Security Protocol Specific as defined in SPC-4.

Chapter Summary

- Admin Commands
  - Creation and Deletion of I/O Queues
    - Create I/O Queues Commands
    - Delete I/O Queues Commands
  - Identify Command
    - Controller
    - Namespace
  - Namespace Attachment
  - Namespace Management
  - Features
    - Get Features
    - Set Features
    - Feature ID 01h - Arbitration
    - Feature ID 02h - Power Management
    - Feature ID 03h - LBA Range Type
    - Feature ID 04h - Temperature Threshold
    - Feature ID 05h - Error Recovery
    - Feature ID 06h - Volatile Write Cache
    - Feature ID 07h - Number of Queues
    - Feature ID 08h - Interrupt Coalescing
    - Feature ID 09h - Interrupt Vector Configuration
    - Feature ID 0Ah - Write Atomicity Normal
    - Feature ID 0Bh - Asynchronous Event Configuration
    - Feature ID 0Ch - Autonomous Power State Transition
    - Feature ID 0Dh - Host Memory Buffer
    - Feature ID 0Fh - Keep Alive Timer
    - Feature ID 80h - Software Progress Marker
    - Feature ID 81h - Host Identifier
    - Feature ID 82h - Reservation Notification Mask
    - Feature ID 83h - Reservation Persistence
  - Other Commands
    - Abort Command
    - Asynchronous Event Request
    - Get Log Page
    - Firmware Download
    - Firmware Commit
- NVM Command Set Specific Admin Commands
  - Format NVM
  - Security

Chapter Summary

NVMe Commands

Chapter Contents

Why a New Command Set

Efficiency! The SCSI and ATA command sets have many different variations of the basic read and write commands. Decoding them and all of the parameters does not take much time compared to the cylinder seek or rotational latency times of the hard disk drives, but SSDs do not have those mechanical motions so the decode time can add a significant amount to the command processing time.

Relevance! The SCSI and ATA command sets have commands (Prevent Allow Media Removal, Start Stop Unit, Extended Copy, Read Media Serial Number) that have no use in current SSDs with a PCIe interface. A new command set can contain only the commands that are used on this device type and interface.

Current! Many SCSI and ATA commands were added because they seemed like a good idea, at least to a few people, but are not widely supported (ORWrite) and others were useful but now are less so (Read/Write (6), (10) and (12)).

The writers of the specification can start with a clean slate and write an efficient, relevant and current command set using knowledge gained from past experiences. SCSI and ATA command sets have served us well, and will continue to serve us in the future, but it is time to slowly start to migrate to the command set of the future.

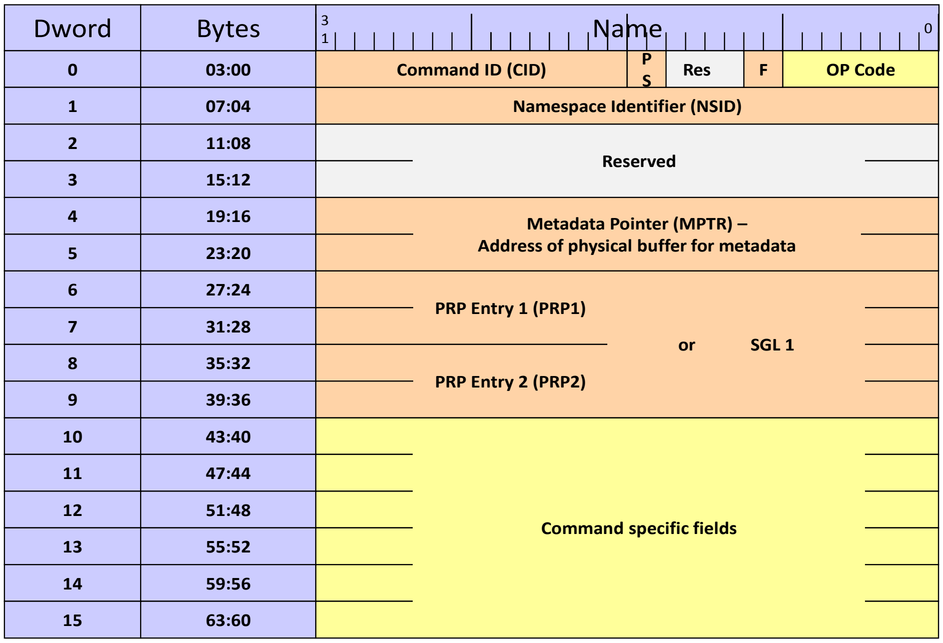
NVMe Commands

The NVMe Command set has only 11, four of which are not used unless there are two or more hosts accessing a controller. All commands have exactly the same format, providing for an efficient and quick decode mechanism.

OpCode	M/O	Command
00h	M	Flush
01h	M	Write
02h	M	Read
04h	O	Write Uncorrectable
05h	O	Compare
08h	O	Write zeros
09h	O	Dataset Management
0Dh	M/O	Reservation Register
0Eh	M/O	Reservation Report
11h	M/O	Reservation Acquire
15h	M/O	Reservation Release
80-FFh	O	Vendor Specific

Figure 89: NVMe Commands

Please reference “Common Command Fields” on page 51 to review the tan fields in Figure 90. This chapter will cover the yellow fields in Figure 90.

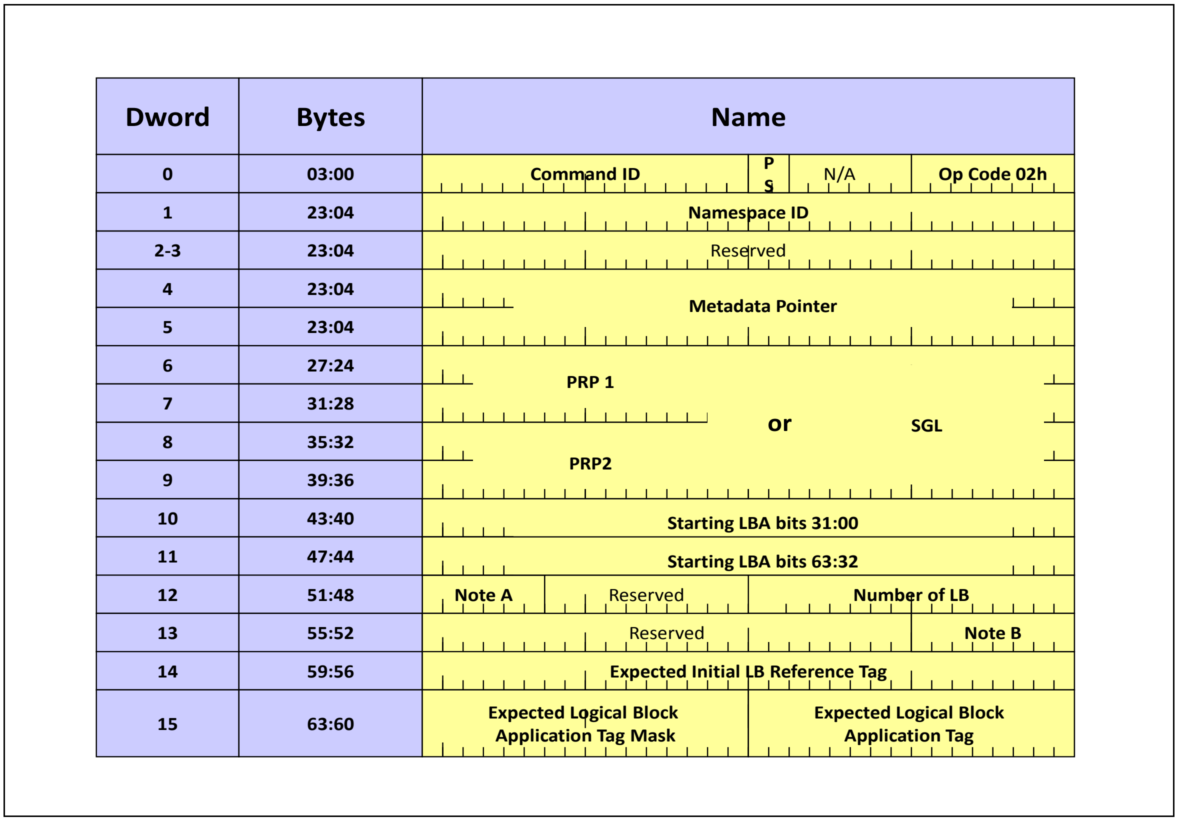


Dword	Bytes	Name (31 … 0)				
0	03:00	Command ID (CID)	PS	Res	F	OP Code
1	07:04	Namespace Identifier (NSID)				
2	11:08	Reserved				
3	15:12					
4	19:16	Metadata Pointer (MPTR) – Address of physical buffer for metadata				
5	23:20					
6	27:24	PRP Entry 1 (PRP1)	or SGL 1			
7	31:28					
8	35:32	PRP Entry 2 (PRP2)				
9	39:36					
10	43:40	Command specific fields				
11	47:44					
12	51:48					
13	55:52					
14	59:56					
15	63:60					

Figure 90: NVMe Command Format

Read and Write User Data

The Read (Figure 91) and Write (Figure 92) commands transfer user data from the host memory to the controller media (Write command), or from the controller media to the host memory (Read command). These commands identify the applicable namespace, and within the namespace the starting LBA, and the Number of Blocks. If the host sends a read command, the controller will get the data from its storage (volatile or non-volatile) and issue a PCIe Write Transaction to place the data in the PCIe memory address specified by the PRP entries or the SGL entries. If the host sends a write command, the controller will use PCIe memory address specified by the PRP entries or the SGL entries to fetch the data and place it in its storage (volatile or non-volatile).

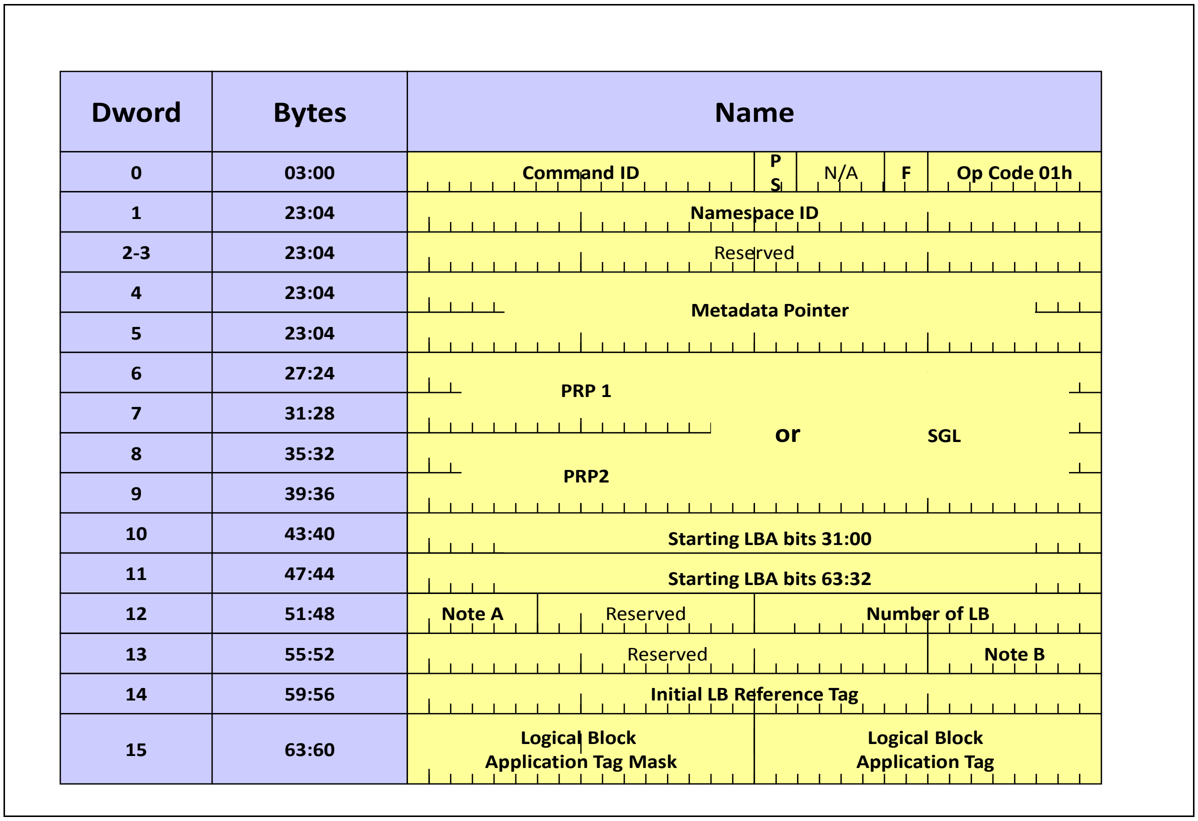


Dword	Bytes	Name
0	03:00	Command ID PS N/A Op Code 02h
1	23:04	Namespace ID
2-3	23:04	Reserved
4	23:04	Metadata Pointer
5	23:04	
6	27:24	PRP 1 or SGL
7	31:28	
8	35:32	PRP2
9	39:36	
10	43:40	Starting LBA bits 31:00
11	47:44	Starting LBA bits 63:32
12	51:48	Note A Reserved Number of LB
13	55:52	Reserved Note B
14	59:56	Expected Initial LB Reference Tag
15	63:60	Expected Logical Block Application Tag Mask Expected Logical Block Application Tag

Figure 91: Read Command

If the namespace has been formatted for end-to-end data protection, the write command specifies the Initial Reference Tag, the Logical Block Application Mask and the Logical Block Application Tag. The read command gives the expected values for those fields so the controller can recognize a possible error.

Note A (DWord 12) and Note B (DWord 13) are defined in Figure 93 on page 130.



<table>
<tr><th>Dword</th><th>Bytes</th><th colspan="2">Name</th></tr>
<tr><td>0</td><td>03:00</td><td>Command ID</td><td>P S
N/A
F
Op Code 01h</td></tr>
<tr><td>1</td><td>23:04</td><td colspan="2">Namespace ID</td></tr>
<tr><td>2-3</td><td>23:04</td><td colspan="2">Reserved</td></tr>
<tr><td>4</td><td>23:04</td><td colspan="2" rowspan="2">Metadata Pointer</td></tr>
<tr><td>5</td><td>23:04</td></tr>
<tr><td>6</td><td>27:24</td><td rowspan="2">PRP 1</td><td rowspan="4">or
SGL</td></tr>
<tr><td>7</td><td>31:28</td></tr>
<tr><td>8</td><td>35:32</td><td rowspan="2">PRP2</td></tr>
<tr><td>9</td><td>39:36</td></tr>
<tr><td>10</td><td>43:40</td><td colspan="2">Starting LBA bits 31:00</td></tr>
<tr><td>11</td><td>47:44</td><td colspan="2">Starting LBA bits 63:32</td></tr>
<tr><td>12</td><td>51:48</td><td>Note A
Reserved</td><td>Number of LB</td></tr>
<tr><td>13</td><td>55:52</td><td>Reserved</td><td>Note B</td></tr>
<tr><td>14</td><td>59:56</td><td colspan="2">Initial LB Reference Tag</td></tr>
<tr><td>15</td><td>63:60</td><td>Logical Block Application Tag Mask</td><td>Logical Block Application Tag</td></tr>
</table>

Figure 92: Write Command

DWord	Bit	Description
12 Note A	31	Time Limited Retry (LR)
	30	Force Unit Access (FUA)
		Protection Information Field
	29	If Read command, strip protection information If Write command, insert protection information
	28	Enable checking the 16-bit CRC (Guard) field
	27	Enable checking the Application Tag Field
	26	Enable checking the Reference Tag field
13 Note B	7	Incompressible
	6	Sequential Request
		(Bits 5:4) Access Latency
	00b	No latency information provided
	01b	Longer latency acceptable
	10b	Typical latency
	11b	Smallest possible Latency
		(Bits 3:0) Access Frequency
	0h	No frequency information provided
	1h	Typical number of reads and writes for this LBA range
	2h	Infrequent reads and writes for this LBA range
	3h	Infrequent writes, frequent reads
	4h	Frequent writes, infrequent reads
	5h	Frequent writes and reads
	6h	one time read
	7h	Speculative read (prefetch op) (read only)
	8h	LBA range to be overwritten soon (read only)

Figure 93: Note A and Note B for Read and Write Commands

Time Limited Retry (bit 31) works with “Feature ID 05h - Error Recovery” on page 96 to tell the controller that on this command, limit the time for error recovery or use full error recovery capabilities. Some reasons for setting this bit may be: audio or video data (on-time is more important than being correct), host has other error recovery mechanisms, or program requiring data has lower priority.

Force Unit Access (bit 30) tells the controller that this data must be placed in non-volatile storage (write command) or retrieved from non-volatile storage (read command) even if volatile cache is available. This function is used for data verification, error recovery, or device testing. It would not be used for normal operations.

Bits 29:26 are used for End-to-End Data Protection. See “End to End Data Protection” on page 7. Protection Information Action (PRACT) (bit 29) allows the host to give the controller permission to remove or add protection information. This bit may be set if the namespace was not formatted for protection information, yet the host sent the information. It may do this because other devices in the data path may be able to verify the information, especially the Guard field. If the namespace is formatted for protection information, the bit will probably be set to 0b.

Protection Information Check (PRCHK) (bits 28, 27 and 26) give the host the ability to tell the controller to check or not check the three fields of the protection information.
Bit 28 enables or disables checking of the Guard field.

Bit 27 enables or disables checking of the Application Tag field.
Bit 26 enables or disables checking of the Logical Block Reference Tag field.

NVMe has added a facility to disable checking regardless of the setting of PRCHK. If:

the namespace is formated for Type 1 or Type 2 protection, and
the Application Tag has a value of FFFFh, then
all checks are disabled regardless of the PRCHK.

If:

the namespace is formated for Type 3 protection,
the Application Tag has a value of FFFFh, and
the Reference Tag has a value of FFFFh FFFFh, then
all checks are disabled regardless of the PRCHK.

DWord 13 (Note B) is for Dataset Management.

Incompressible (bit 7) communicates to the controller that the data is not compressible (1b), or that no information on compression is provided (0b).

Sequential Request (bit 6) if set to 1b communicates that this command is one of two or more commands that form a sequential access. If set to 0b, no information is provided about sequential access.

Access Latency (bits 5 and 4) provide information from the host to the controller on how soon this data is needed. It could be used by the controller to prioritize commands in its queue.

Access Frequency (bits 3:0) provide information to the controller about how the data will be access. If the controller can use the information, it may be able to provide better service to the host.

Reservations

Overview

Reservations are a mechanism that one host can use to reserve a namespace for that host's exclusive use. Reservations are only meaningful if there are two or more hosts accessing a namespace. A reservation on a namespace by one host restricts other hosts' access to that namespace.

Support for reservations is optional. Controllers in a NVMe subsystem must have the same support for reservations; namespaces are not required to have the same support for reservations. Controllers indicate support in the "Optional NVM Command Support" field (bit 5) in the Identify Controller data structure. Namespaces indicate support in the Reservations Capabilities (RESCAP) field in Identify Namespace data structure.

By default, registrations and reservations persist across all resets except power loss. Reservations may be set to persist across power loss by setting the Persist Through Power Loss State bit in the RESCAP of the Identify Namespace.

See Figure 94. Reservations are between Host ID and Namespace ID. In other words if Host A reserves NS ID 1 through controller 1, Host A can also get to NS ID 1 through controller 2. Commands issued by Host B and Host C when they do not have sufficient permissions will be aborted with Reservation Conflict status (SCT = 000b, Status code = 83h). See Figure 38 on page 66.

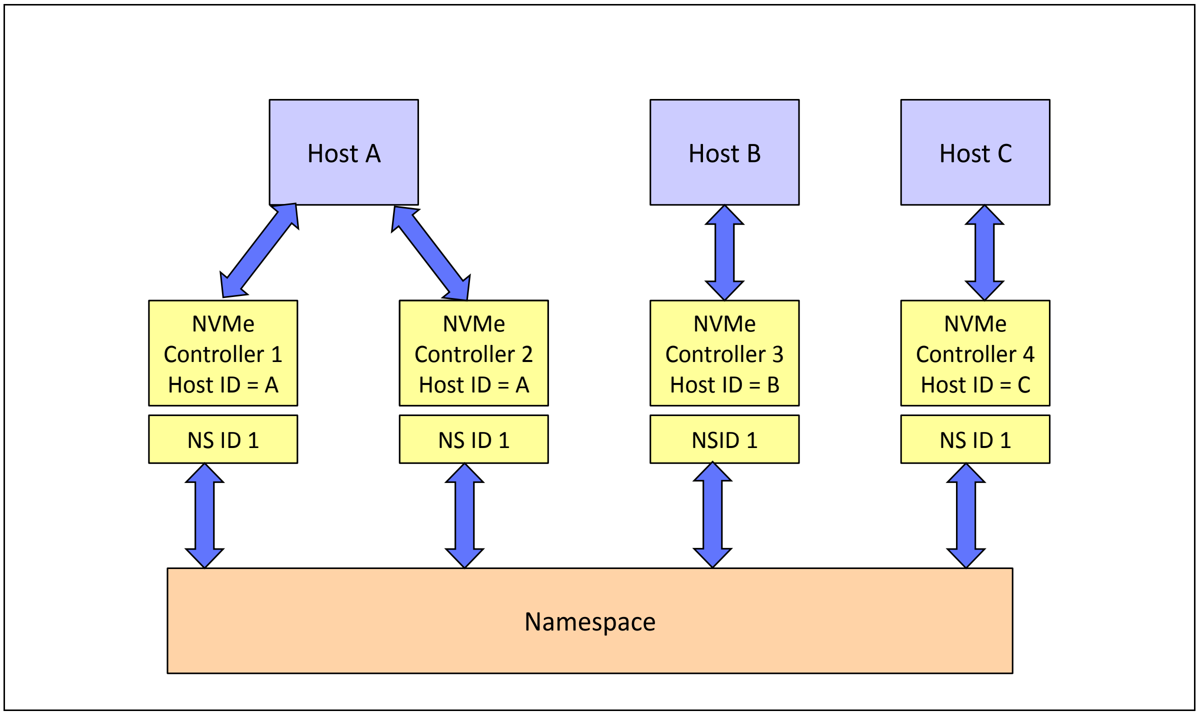

Host A
Host B
Host C
NVMe Controller 1 Host ID = A
NVMe Controller 2 Host ID = A
NVMe Controller 3 Host ID = B
NVMe Controller 4 Host ID = C
NS ID 1
NS ID 1
NSID 1
NS ID 1
Namespace

Figure 94: Reservations with multiple controllers

Not all commands are aborted with Reservation Conflict when sent to a NS that is reserved by another host.

Table 29 shows if read or write commands can be executed in face of different reservation types by different categories of hosts.

The first two columns give the Reservation Type. Write Exclusive (1h) means the reservation holder is the only one who can write, others can read the media. Exclusive Access (2h) means the reservation holder is the only one who can access the media. Registrants only (3h and 4h) modifies the above two statements to indicate that hosts that are registered can access the media but there is still only one reservation holder. All Registrants (5h and 6h) means that all hosts that are registered are reservation holders.

Across the top are the categories of host: Reservation holder, Registrant, and Non-Registrant. Under each of those three are listed read type of commands and write type of commands. In the columns under Read and Write, “Y” means the command type (read or write) issued from the category of host (Reservation Holder, Registrant, or Non-Registrant) with the given type of reservation (1, 2, 3, 4, 5, or 6h) can access the media, “N” means the command may not access the media.

Write Exclusive means that only write commands are restricted by the reservation, Exclusive Access means that all media access commands are restricted by the reservations.

The host that holds the reservation has unrestricted access to the reserved namespace.

A non-registered host can read the media if the reservation type is Write Exclusive, but can do no media access at all if the reservation type is exclusive access.

Only the Reservation Holder can release a reservation. Other hosts may be able to preempt a reservation.

If the current reservation is Registrants Only, and if a Registrant preempts the reservation, then the existing holder is unregistered, the existing reservation is released and a new reservation is created with the new registrant. This is intended for use in clustered systems and the system holding the reservation goes down, another host can take over the workload of the down host.

Value	Reservation type	Reservation Holder		Registrant		Non-Registrant		Reservation Holders
		Read	Write	Read	Write	Read	Write	
1h	Write Exclusive	Y	Y	Y	N	Y	N	1
2h	Exclusive Access	Y	Y	N	N	N	N	1
3h	Write Exclusive - Registrants only	Y	Y	Y	Y	Y	N	1
4h	Exclusive Access - Registrants only	Y	Y	Y	Y	N	N	1
5h	Write Exclusive - all Registrants	Y	Y	Y	Y	Y	N	All registrants are Reservation Holders
6h	Exclusive Access - all Registrants	Y	Y	Y	Y	N	N	All registrants are Reservation Holders

Table 29: Reservation Types vs. Access

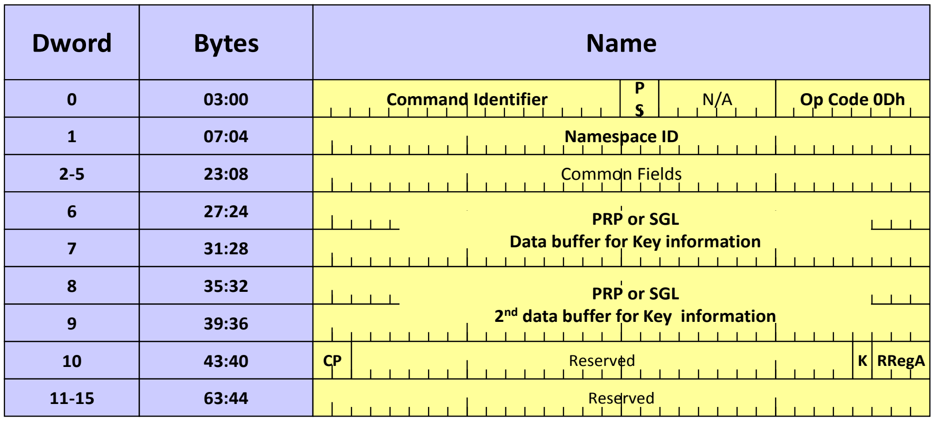
Registration Commands

There are four reservation commands that one host can use to reserve a namespace for that host’s exclusive use. The are:
Reservation Register - host must register a key with that namespace before acquiring a reservation,
Reservation Acquire - getting that namespace for the host’s exclusive use,
Reservation Report - discovering what other hosts have reservations and registrations on a namespace, and
Reservation Release - releasing a reservation the host holds on a namespace

Reservation Register

Before registering a key with a namespace, the host must use the Set Features command to register a host identifier with each controller.
Then the host can use the Reservation Register command to register a key with the namespace. The host will create a key and include the location of the buffer where it stored the key in its memory. The value and method of creating the key is outside the scope of the specification. The key is to identify the host, however, multiple hosts may register with the same key.

Set Features --> Host Identifier --> Controllers
Reservation Register --> Registration Key --> Namespace



Dword	Bytes	Name			
0	03:00	Command Identifier	PS	N/A	Op Code 0Dh
1	07:04	Namespace ID			
2-5	23:08	Common Fields			
6	27:24	PRP or SGL Data buffer for Key information			
7	31:28				
8	35:32	PRP or SGL 2nd data buffer for Key information			
9	39:36				
10	43:40	CP	Reserved	K	RRegA
11-15	63:44	Reserved			

Figure 95: Reservation Register Command

The K field, if set, means to ignore the existing key. Fields that use the Current Reservation Key will not be validated if this bit is set.

The CP field is for Persist Through Power Loss

00b - no change to PTPL state
01b - Reserved
10b - Set PTPL state to 0
11b - Set PTPL state to 1

The RRegA field is Reservation Register Action:

000b - Register Reservation Key
001b - Unregister Reservation Key
010b - Replace Reservation Key

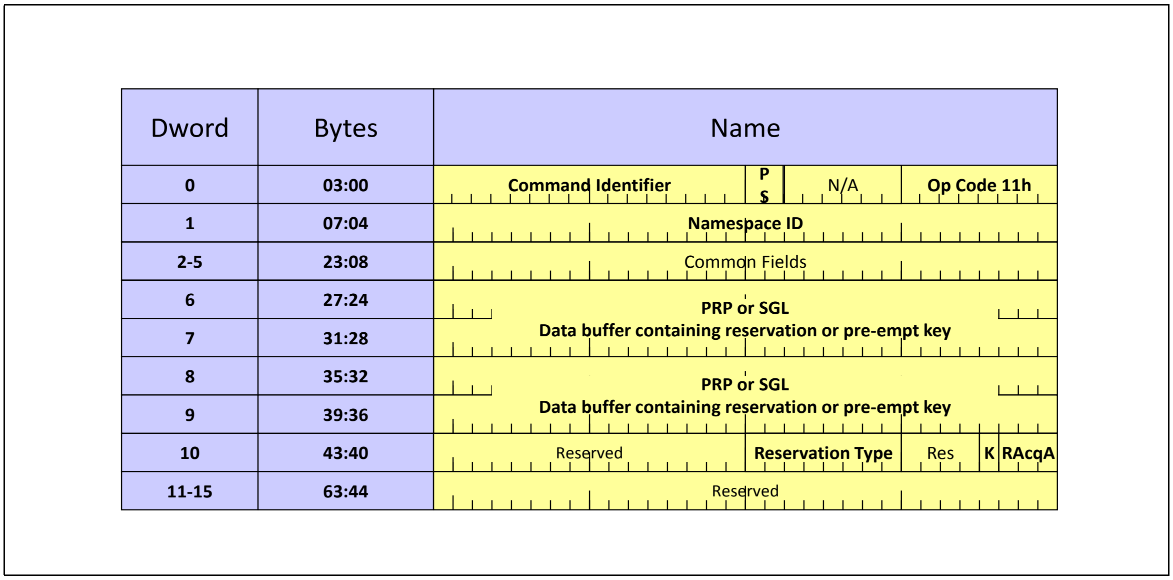
See the table below for the format and usage of the current and new reservation keys.

Bytes	M/O	Name	Applicable RRA
7:0	M	Current Reservation Key Ignore this field is K = 1	If RRA = 1 or 2
15:8	M	New Reservation Key	If RRA = 0 or 2

Table 30: Reservation Register Data Structure

Reservation Acquire

When necessary a host that has registered a key with the controller, can issue a Reservation Acquire command as shown in Figure 96. The PRP or SGL points to the data structure in host memory that contains the key, if necessary.



Dword	Bytes	Name				
0	03:00	Command Identifier	PS	N/A	Op Code 11h	
1	07:04	Namespace ID				
2-5	23:08	Common Fields				
6	27:24	PRP or SGL Data buffer containing reservation or pre-empt key				
7	31:28					
8	35:32	PRP or SGL Data buffer containing reservation or pre-empt key				
9	39:36					
10	43:40	Reserved	Reservation Type	Res	K	RAcqA
11-15	63:44	Reserved				

Figure 96: Reservation Acquire Command

The K field, if set, means to ignore the existing key. Fields that use the Current Reservation Key will not be validated if this bit is set.

The RAcqA field is Reservation Acquire Action:

000b - Acquire (get the requested reservation).

001b - Preempt (Override the existing reservation and make me the reservation holder),

010b - Preempt and Abort (Override the existing reservation, make me the reservation holder, and Abort all commands the previous reservation holder has outstanding).

Bytes	M/O	Name	Applicable RAA
7:0	M	Current Reservation Key Ignore this field is K = 1	All
15:8	M	Reservation Key to be preempted	If RAA = 1 or 2

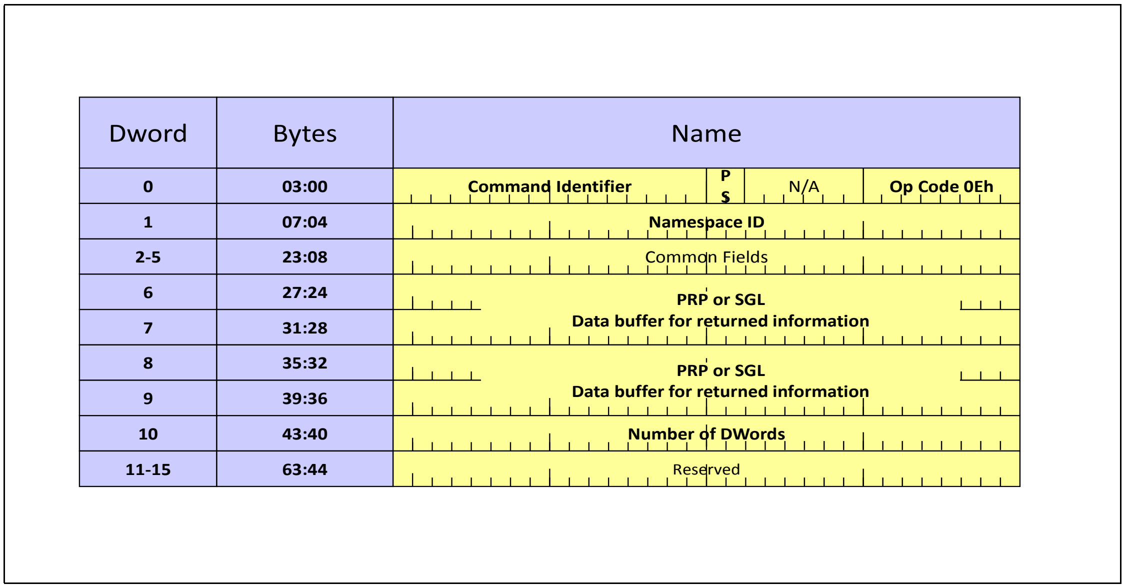
Table 31: Reservation Acquire Data Structure

The Reservation types are shown in Table 29 on page 133.

Reservation Report

The Reservation Report command returns to the host memory a list describing the registration and reservation status for the namespace. The PRP1/PRP2 or SGL points to the memory buffer. DWord 10 specifies the size of the buffer in DWords and is 0 based.

The command is shown in Figure 97 and the returned data structure is in Table 32.



Dword	Bytes	Name
0	03:00	Command Identifier P S N/A Op Code 0Eh
1	07:04	Namespace ID
2-5	23:08	Common Fields
6	27:24	PRP or SGL Data buffer for returned information
7	31:28	
8	35:32	PRP or SGL Data buffer for returned information
9	39:36	
10	43:40	Number of DWords
11-15	63:44	Reserved

Figure 97: Reservation Report Command

Bytes	Name
3:0	Generation
4	Reservation Type
6:5	Number of Registered Controllers
8:7	Reserved
9	Persist Through Power Loss State 00h =Reservations are releases and registrants are cleared 01h = Reservations and registrants persist
23:10	Reserved
47:24	Registered Controller Data Structure 0
24*n+47: 24*(n+1)	Registered Controller Data Structure n

Table 32: Reservation Report Data Structure

Bytes 3:0 in Table 32 show the generation of this information. This 32-bit wrapping counter increments any time one of the following occurs:

Reservation Register,
Reservation Release w/Clear, or
Reservation Acquire w/Preempt or Preempt and Abort.

The value of the Generation field is not meaningful, but if it is different from the last time it was read, one of those events occurred. If the Generation field is the same, those events did not occur since last read.

Byte 4 is the Reservation type (RTYPE) from the left two columns of Table 29 on page 133.

Bytes 6:5 tell the number of registered controllers (REGCTL) that are associated with hosts that are registrants of the namespace. It is also the number of Registered Controller Data Structures (Table 33) in this data structure.

Byte 9 is the Persist Through Power Loss State (PTPLS) associated with the namespace.

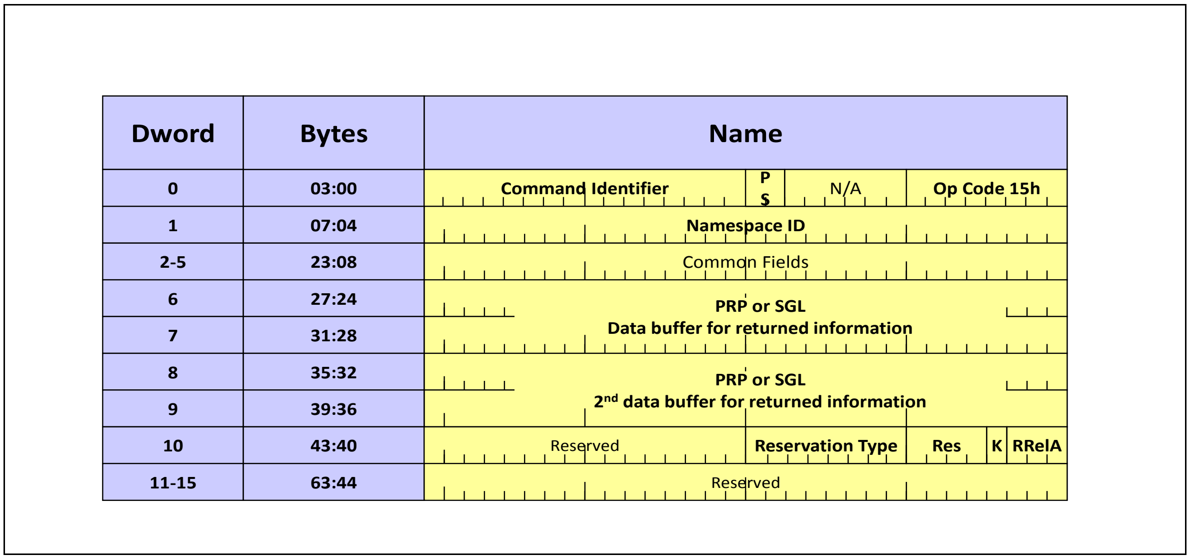
Bytes 24 and above contain the 24 byte Registered Controller Data Structures as shown in Table 33.

Bytes	Description
1:0	Controller ID (CNTLID)
2	Reservation Status (RCSTS) Bit 0 = 1 if the controller is associated with a host that holds a reservation on the namespace
7:3	Reserved
15:8	Host ID (HOSTID)
23:16	Reservation Key (RKEY)

Table 33: Registered Controller Data Structure

Reservation Release

Reservation Release releases the reservation held on the namespace. Normally, it will be released by the host that holds the reservation. If a key will be used, it will be in the host memory buffer pointed to by the PRP entires or the SGL entry.



Dword	Bytes	Name
0	03:00	Command Identifier PS N/A Op Code 15h
1	07:04	Namespace ID
2-5	23:08	Common Fields
6	27:24	PRP or SGL Data buffer for returned information
7	31:28	
8	35:32	PRP or SGL 2nd data buffer for returned information
9	39:36	
10	43:40	Reserved Reservation Type Res K RRelA
11-15	63:44	Reserved

Figure 98: Reservation Release Command

Reservation Type if from the left two columns of Table 29 on page 133.

K field tells the controller to ignore the current reservation key.

The RRelA (Reservation Release Action specifies the registration action:

000b - Release the reservation, or
001b - Clear.

Clear a reservation:

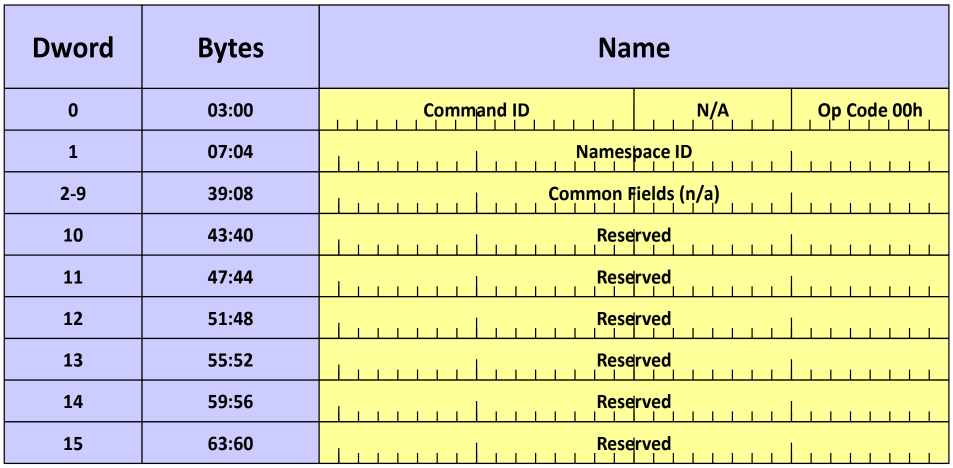
May be requested by any host that is a registrant,
Forces the release of a reservation held on the namespace, and
Unregisters all registrants.

Forcing the release of a reservation and unregistering all registrants is performed as an Atomic Op

Other Commands

Flush

The flush command causes the controller to commit all data and metadata associated with the specified namespace to non-volatile media. The flush command applies to all commands completed prior to the submission of the Flush command. The controller may flush data and metadata associated with other namespaces also.



Dword	Bytes	Name		
0	03:00	Command ID	N/A	Op Code 00h
1	07:04	Namespace ID		
2-9	39:08	Common Fields (n/a)		
10	43:40	Reserved		
11	47:44	Reserved		
12	51:48	Reserved		
13	55:52	Reserved		
14	59:56	Reserved		
15	63:60	Reserved		

Figure 99: Flush Command

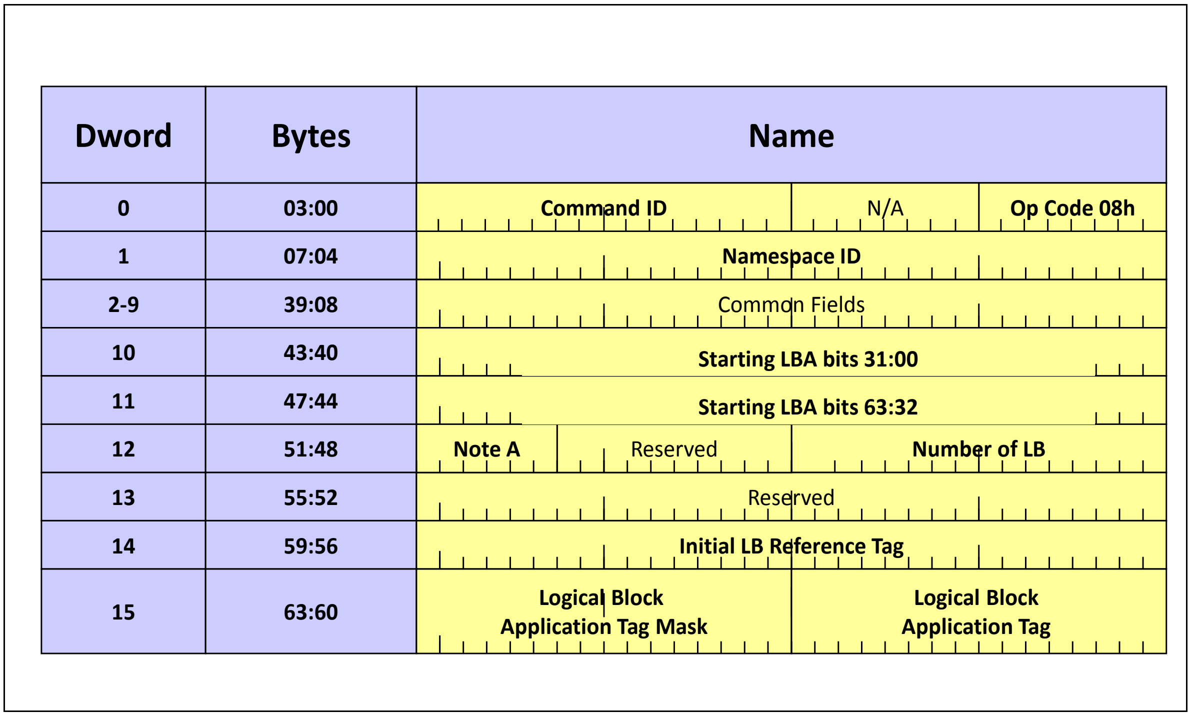
Write Zeros

The Write Zeros command sets the contents of a range of Logical Blocks to zero. The range begins with the Starting LBA and continues for the Number of Blocks. Following successful completion of this command, a read to logical blocks in the range written to zeros will return zeros, until a write occurs to the logical block range.

The left most six bits of DWord 12 mean as follows:
Bit 31 - Limited Retry (LR). If set to “1b”, perform a limited amount of retries to write the data to the media. If set to “0b”, the controller should use all available error recovery means to write the data to the NVM.

Bit 30 - Force Unit Access (FUA). If set, this bit indicates that the controller is to place the data in non-volatile storage (media) rather than a volatile cache.

Bits 29:26 - Protection Information Field (PRINFO). These bits are used only if the namespace if formatted for end-to-end data protection.
Bit 29 - Protection Information Action (PRACT). If set to 1b, the protection information is to be inserted.
Bit 28 - indicates to check the 16-bit CRC
Bit 27 - indicates to check the Application Tag
Bit 26 - indicates to check the Reference Tag (sequential counter).



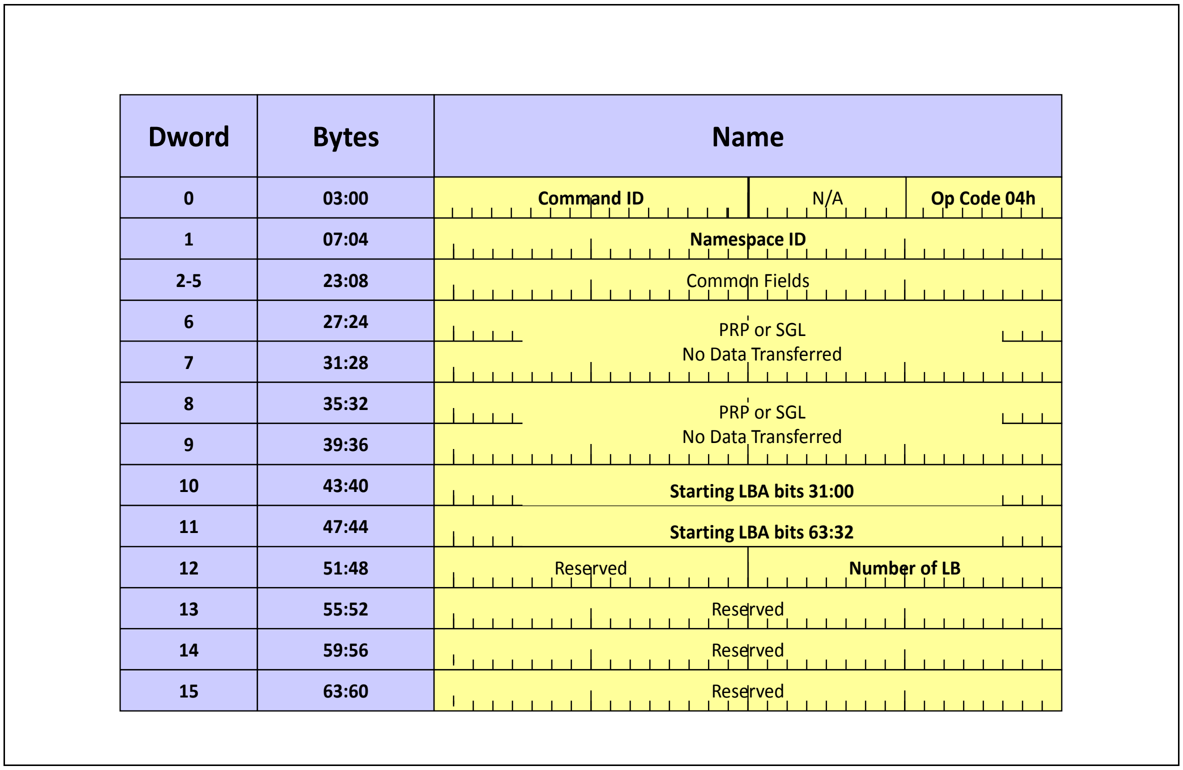
<table>
<tr><th>Dword</th><th>Bytes</th><th colspan="4">Name</th></tr>
<tr><td>0</td><td>03:00</td><td colspan="2">Command ID</td><td>N/A</td><td>Op Code 08h</td></tr>
<tr><td>1</td><td>07:04</td><td colspan="4">Namespace ID</td></tr>
<tr><td>2-9</td><td>39:08</td><td colspan="4">Common Fields</td></tr>
<tr><td>10</td><td>43:40</td><td colspan="4">Starting LBA bits 31:00</td></tr>
<tr><td>11</td><td>47:44</td><td colspan="4">Starting LBA bits 63:32</td></tr>
<tr><td>12</td><td>51:48</td><td>Note A</td><td>Reserved</td><td colspan="2">Number of LB</td></tr>
<tr><td>13</td><td>55:52</td><td colspan="4">Reserved</td></tr>
<tr><td>14</td><td>59:56</td><td colspan="4">Initial LB Reference Tag</td></tr>
<tr><td>15</td><td>63:60</td><td colspan="2">Logical Block
Application Tag Mask</td><td colspan="2">Logical Block
Application Tag</td></tr>
</table>

Figure 100: Write Zeros Command

Write Uncorrectable

The Write Uncorrectable command marks a group of Logical Blocks as invalid. That group starts with the Starting LBA and extends for the Number of Logical Blocks. Reads to those blocks will return “Uncorrectable Read Error” status (SCT = 2h, Status = 81h).

To clear the “invalid” status of the blocks, perform a write operation on them.



Dword	Bytes	Name		
0	03:00	**Command ID**	N/A	**Op Code 04h**
1	07:04	**Namespace ID**		
2-5	23:08	Common Fields		
6	27:24	PRP or SGL No Data Transferred		
7	31:28			
8	35:32	PRP or SGL No Data Transferred		
9	39:36			
10	43:40	**Starting LBA bits 31:00**		
11	47:44	**Starting LBA bits 63:32**		
12	51:48	Reserved	**Number of LB**	
13	55:52	Reserved		
14	59:56	Reserved		
15	63:60	Reserved		

Figure 101: Write Uncorrectable Command

Upon completion of the command, the controller will post ending status in the appropriate CQ.

Compare

Upon receipt of this command, the controller will read from the media the data specified by the Starting LBA and Number of Blocks, and will compare that to the data specified by the PRP or SGL. If metadata is provided, the comparison will compare the metadata.

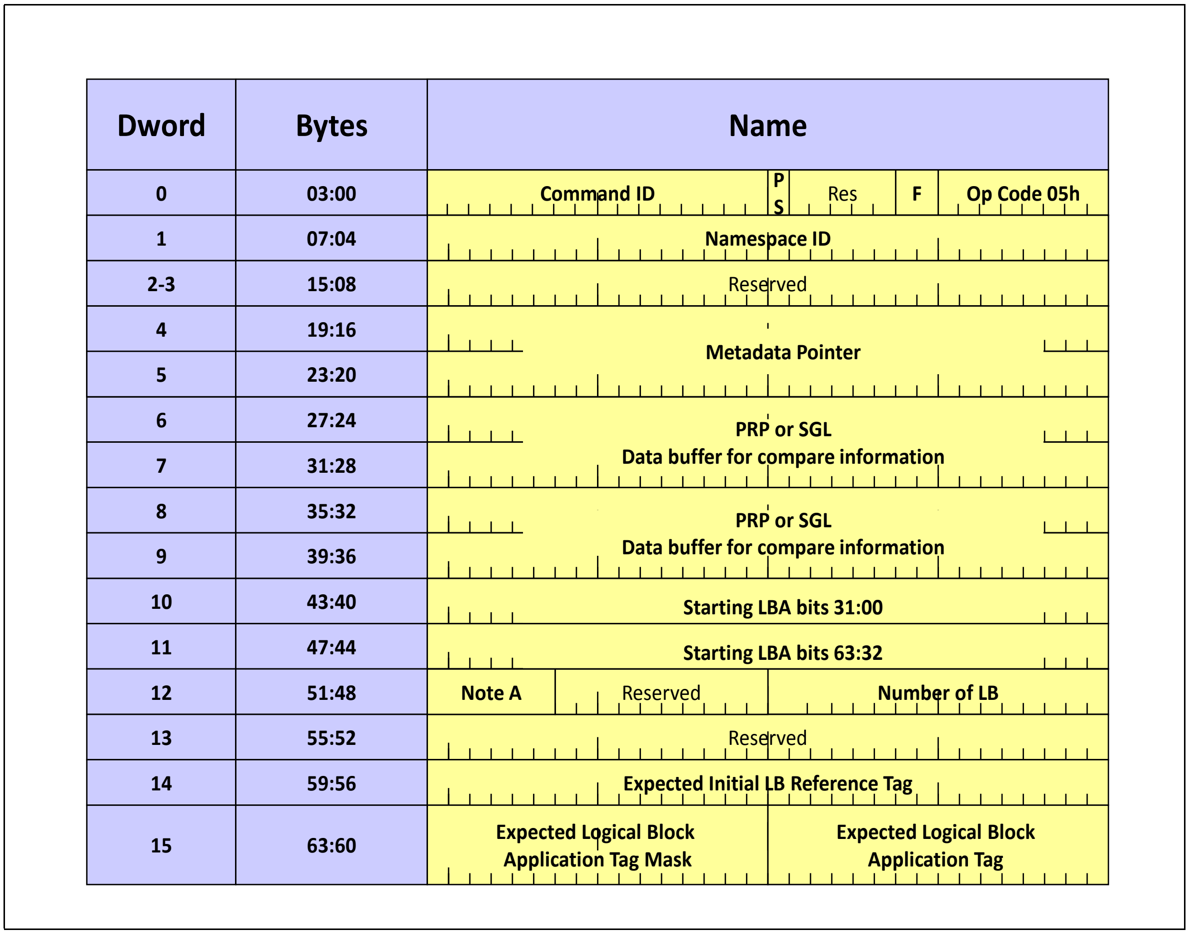
The six bits of Note A in DWord 12 are decoded as follows:

Bit 31 - Limited retires. If “1b” controller should apply limited retry efforts[13]. If set to “0b” controller should apply all available error recovery means to retrieve the data.

Bit 30 - Force Unit Access (FUA). if “1b” specifies the data must be read from non-volatile media instead of volatile cache. This bit may be set for error recovery, performance analysis, configuration, or testing but will seldom be set for normal operations.

Bits 29:26 - Protection Information Field (PRINFO). Specifies the protection checks and actions to be performed. These four bits are used only if the namespace is formated for end-to-end data protection.
Bit 29 - Protection Information Action (PRACT). For reads, 1b indicates the protection information is stripped; 0b indicates the protection information is passed.
Bit 28 - If 1b, check the Guard field, 16-bit CRC.
Bit 27 - If 1b, check the Application tag.
Bit 26 - If 1b, check the Logical Block Reference Tag (sequential counter).

[13] The word “should” is advisory but not mandatory. The specification defines it as “flexibility of choice with a strongly preferred alternative.” “Limited retry efforts” is not defined except by its normal English language meaning. Therefore this sentence probably means that if bit 31 is set to a “1”, the controller may retry as much as it wants, ranging from using all available means to no retries.



<table>
<tr><th>Dword</th><th>Bytes</th><th colspan="6">Name</th></tr>
<tr><td>0</td><td>03:00</td><td colspan="2">Command ID</td><td>PS</td><td>Res</td><td>F</td><td>Op Code 05h</td></tr>
<tr><td>1</td><td>07:04</td><td colspan="6">Namespace ID</td></tr>
<tr><td>2-3</td><td>15:08</td><td colspan="6">Reserved</td></tr>
<tr><td>4</td><td>19:16</td><td colspan="6" rowspan="2">Metadata Pointer</td></tr>
<tr><td>5</td><td>23:20</td></tr>
<tr><td>6</td><td>27:24</td><td colspan="6" rowspan="2">PRP or SGL
Data buffer for compare information</td></tr>
<tr><td>7</td><td>31:28</td></tr>
<tr><td>8</td><td>35:32</td><td colspan="6" rowspan="2">PRP or SGL
Data buffer for compare information</td></tr>
<tr><td>9</td><td>39:36</td></tr>
<tr><td>10</td><td>43:40</td><td colspan="6">Starting LBA bits 31:00</td></tr>
<tr><td>11</td><td>47:44</td><td colspan="6">Starting LBA bits 63:32</td></tr>
<tr><td>12</td><td>51:48</td><td>Note A</td><td>Reserved</td><td colspan="4">Number of LB</td></tr>
<tr><td>13</td><td>55:52</td><td colspan="6">Reserved</td></tr>
<tr><td>14</td><td>59:56</td><td colspan="6">Expected Initial LB Reference Tag</td></tr>
<tr><td>15</td><td>63:60</td><td colspan="2">Expected Logical Block Application Tag Mask</td><td colspan="4">Expected Logical Block Application Tag</td></tr>
</table>

Figure 102: Compare Command

If the Protection Information settings in the command are invalid, the controller will post failing SCT = 1h, Status Code 81h - Invalid Protection Information.

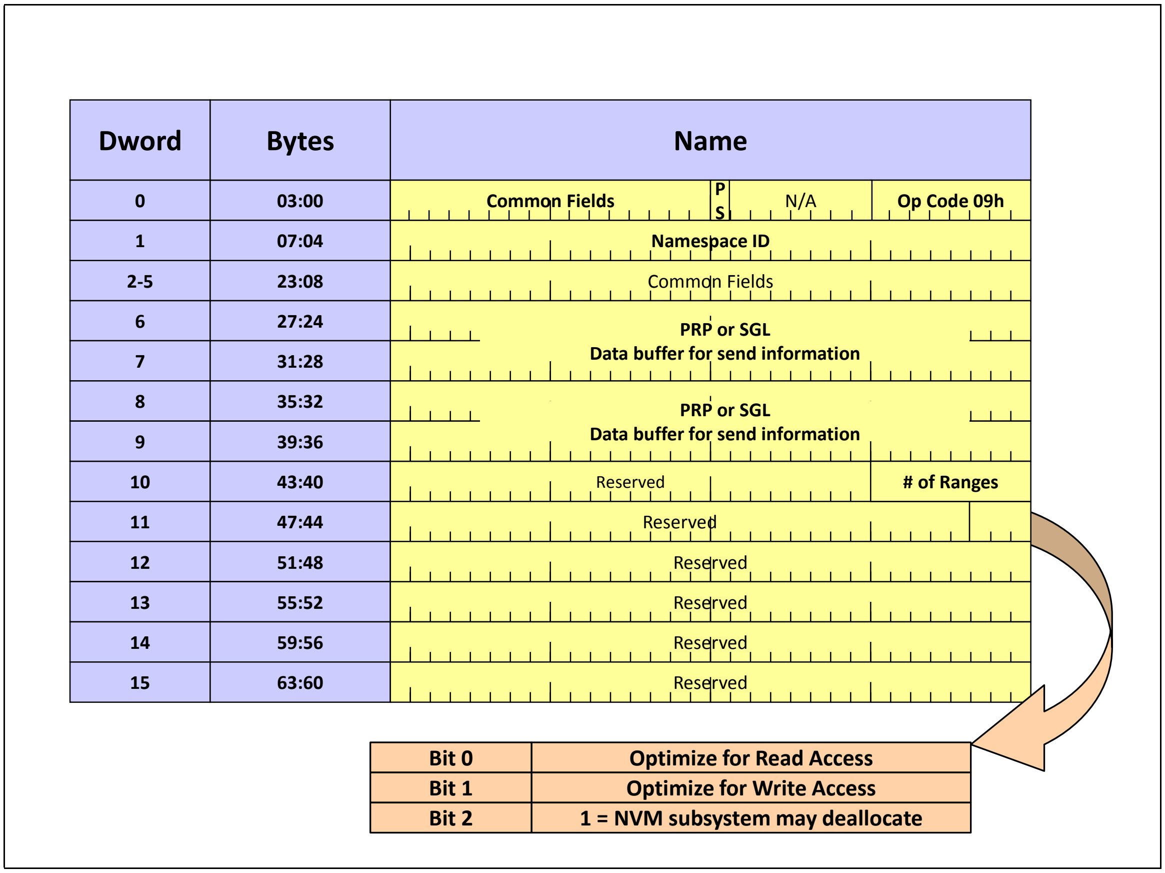
If the media data is the same as the buffer data, the compare is successful and the controller will post ending status to the appropriate CQ with SCT = 000b and Status code = 00h. It there is one or more mis-compares, the controller will post ending status as Compare Failure (SCT=2h, Status Code 85h).

Dataset Management (DSM)

DSM is an advisory command that provides information to the controller about expected accesses to ranges of Logical Blocks. Context information such as access size, frequency of reads and writes, tolerance for latency and other information that the controller may be able to use.

The specification does not state it, but certain information is written in a manner that indicates the intended use of this command is to have one DSM Command issued for each dataset to be managed.

The host places up to 255 LBA ranges in memory and indicates to the controller the Number of Ranges in DSM Command DWord 10, and the location of the ranges in the PRPs or SGL addressed by DWords 6-9. If using PRP, neither PRP Entry 1 nor PRP Entry 2 shall be a pointer to a PRP List.



Dword	Bytes	Name
0	03:00	Common Fields · PS · N/A · Op Code 09h
1	07:04	Namespace ID
2-5	23:08	Common Fields
6	27:24	PRP or SGL Data buffer for send information
7	31:28	
8	35:32	PRP or SGL Data buffer for send information
9	39:36	
10	43:40	Reserved · # of Ranges
11	47:44	Reserved
12	51:48	Reserved
13	55:52	Reserved
14	59:56	Reserved
15	63:60	Reserved

Bit 0	Optimize for Read Access
Bit 1	Optimize for Write Access
Bit 2	1 = NVM subsystem may deallocate

Figure 103: Data Set Management Command

DWord 11 passes some information that applies to all ranges addresses by this command.

Bit 2 set to 1b states that the controller may deallocate all LBAs within the given ranges. If a read access occurs to a deallocated LBA[14], the controller may return all zeros, all ones, or the last data written to the associated LBA.

[14] This appears to be similar to the ATA DataSet Management Command in ACS=4 with the "Trim" sub-command or the SCSI "Unmap" command. When the LBAs are deallocated, they become candidates for background erase (garbage collection) to prepare them for future writes.

Bit 1 set to 1b states that the controller should optimize the dataset for write access as an integral unit. The host expects to perform operation son all ranges provided as an integral unit for write, indicating that if a portion of the dataset is written it is expected that all the ranges in the dataset are going to be written.

Bit 0 set to 1b states that the controller should optimize the dataset for read access as an integral unit. The host expects to perform operations on all ranges provided as an integral unit for read, indicating that if a portion of the dataset is read it is expected that all the ranges in the dataset are going to be read.

The range format is given in Table 34 on page 145. The Context Attributes are given in Table 35 on page 145.

Range	Byte	Field
Range 0	03:00	Context Attributes
	07:04	Length in logical blocks
	15:08	Starting LBA
Range 1	19:16	Context Attributes
	23:20	Length in logical blocks
	31:24	Starting LBA
Range 255	4083:4080	Context Attributes
	4087:4084	Length in logical blocks
	4095:4088	Starting LBA

Table 34: DSM Range Definition

Bits	Attribute	Description
31:24	Command Access Size	Number of Logical Blocks expected to be transferred in a single Read or Write Command
23:11	Reserved	
10	WP: Write Prepare	1 = Listed range is to be written in the near future
09	SW: Sequential Write Range	1 = DS should be optimized for sequential writes
08	SR: Sequential Read Range	1 = DS should be optimized for sequential read
07:06	Reserved	
05:04	AL: Access Latency	00b - No latency information provided
		01b - Longer latency acceptable
		10b - Typical latency
		11b - Smallest possible latency
03:00	AF: Access Frequency	0h - No frequency information provided
		1h - Typical number of Read and Writes expected to this LBA range
		2h - Infrequent writes and infrequent reads to this LBA range
		3h - Infrequent writ4es and frequent reads to this LBA range
		4h - Frequent writes and infrequent reads to this LBA range
		5h - Frequent writes and frequent reads to this LBA range

Table 35: DSM Context Attributes

Atomic Operations

Overview

"Atomic" in computers usually means to do several operations as a single unit.

In PCIe and NVMe, the term "Atomic Operation" includes three very different features. The first is treating multiple commands or operations as a single command or operation. This is the usage of the term in PCIe. The second, called Atomic Write Units, is guaranteeing that a write operation either works in its entirety or does not work at all. This is the usage of the term in NVMe. A third meaning of the term atomic operation is to preform multiple functions as a single unit. For example in preempting a reservation, the controller will:

1) unregistering a current reservation holder,
2) releasing the reservation, and
3) creating a new reservation for the new holder, all as a single operation.

In NVMe, treating two commands as a single operation is called "Fused Operations."

Multiple Commands/Operations

PCI originally had a feature called Lock which reserved the entire path from the requester to the completer, blocking out other users, until the series of commands had been completed. This was used primarily for error recovery, configuration, or when a group of commands must be executed without any intervening commands. However, it also kept other traffic from using the path, even if that traffic was not destined for the same device.

PCIe added the Atomic Operation which put the responsibility of completing the commands (or operations) as a single entity on the PCIe function, freeing up the path between the devices for other traffic. Currently, PCIe defines three Atomic Operations: FetchAdd, Swap, and Compare and Swap. Please check the PCIe Specification 2.1 or later for information on these Atomic Operations.

Atomic Write Unit

The Atomic Write Unit feature places the responsibility on the controller to ensure the entire "unit" of data is written or not written. It is to prevent the possibility of a future read getting some new data and some old data if the controller had a failure during the previous write. In the past, on hard disk drives where a record overwrote the previous data in the assigned logical blocks, the host was responsible for handling errors or failure on writes. Today, using SSDs, the new data is written in a completely new area and the old data is not touched. If a failure occurs during the write, the SSD can simply return the old data on future reads.

Atomic Write Unit guarantees that a read command following a failure during a write operation, the controller will return only new data or only old data and never a mixture of some new and some old.

Of course, the controller will return the appropriate failure status if a write operation does fail.

An atomic Write Unit is the number of logical blocks that can be guaranteed to be written atomically to the media. The value may be different for normal operations (AWUN) vs. power fail conditions (AWUPF). Originally defined for the entire controller, NVMe reversion 1.2 added sizes on a per namespace basis (NAWUN and NAWUPF). There are also parameters for Atomic Compare and Write[15] (ACWU and NACWU). The power fail values must be equal to or small than the normal values. The namespace values musts be equal to or greater

[15] Atomic Compare and Write is two fused commands in NVMe; the compare command and the write command.

than the controller values. The transfer length (number of blocks) of the write command may be less than, equal to, or greater than the AWUN.

Limit	Parameter Name	Value
Controller	Atomic Write Unit Normal (AWUN)	
	Atomic Write Unit Power Fail (AWUPF)	≤AWUN
	Atomic Compare and Write Unit (ACWU)	
Namespace	Namespace Atomic Write Unit Normal (NAWUN)	≥AWUN
	Namespace Atomic Write Unit Power Fail (NAWUPF)	≥AWUPF ≤NAWUN
	Namespace Atomic Compare and Write Unit (NACWU)	≥ACWU
Namespace Boundary	Namespace Atomic Boundary Size Normal (NABSN)	≥NAWUN
	Namespace Atomic Boundary Offset (NABO)	≤NABSN ≥NABSPF
	Namespace Atomic Boundary Size Power Fail (NAVSPF)	≥NAWUPF

Table 36: Atomic Write Unit Definitions

Boundary Size is the number of logical blocks between two boundaries (0 based). Boundaries are defined by the device based on its characteristics such as pages, a NAND plane, or other characteristic. Boundaries are enforced on a namespace only, not on the controller write units. If the controller does not guarantee atomicity across atomic boundaries, it shall set AWUN, AWUPF, and ACWU to 0h (one LB).

Figure 104 below, shows a storage device with logical blocks 0 through max. Each of the yellow or blue groups of logical blocks is an atomic unit. For any write to a single yellow or a single blue group of logical blocks in Figure 104, the controller will guarantee that all the data was written or none of the data was written.

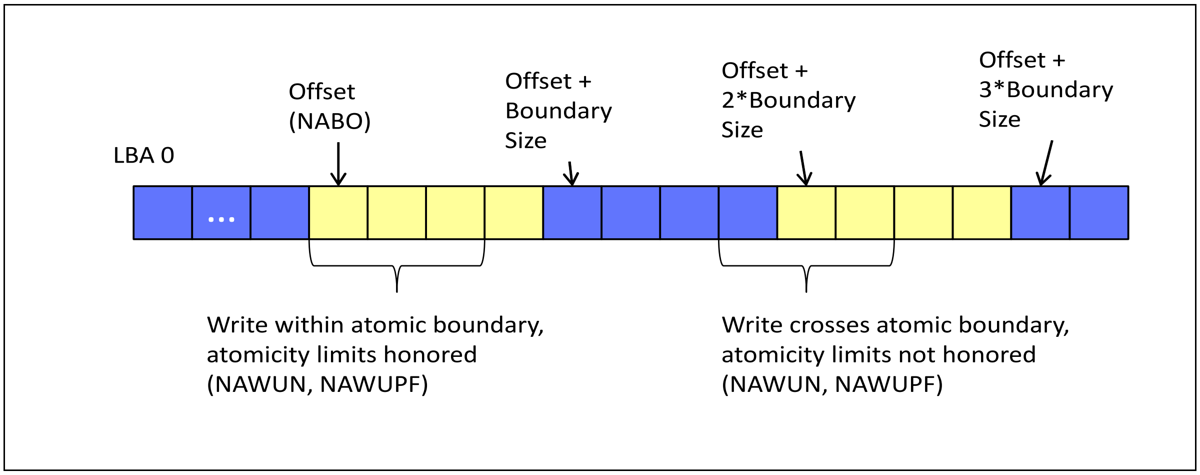

LBA 0
Offset (NABO)
Offset + Boundary Size
Offset + 2*Boundary Size
Offset + 3*Boundary Size
...
Write within atomic boundary, atomicity limits honored (NAWUN, NAWUPF)
Write crosses atomic boundary, atomicity limits not honored (NAWUN, NAWUPF)

Figure 104: Atomic Boundaries

Figure 104 shows two write commands. The left most command is whole within one yellow group (between two boundaries) and therefore the controller will guarantee write atomicity. The right most write crosses an atomic boundary so write atomicity is guaranteed within each atomic unit, not across the entire write command. Specifically, write atomicity is guaranteed within the single blue block and it is guaranteed within the two yellow blocks, but not guaranteed across all three blocks.

Chapter Summary

- Why a New Command Set
- NVMe Commands
  - Read and Write User Data
  - Reservations
    - Reservation Register
    - Reservation Acquire
    - Reservation Report
    - Reservation Release
  - Other Commands
    - Flush
    - Write Zeros
    - Write Uncorrectable
    - Compare
    - Dataset Management (DSM)
- Atomic Operations
- Chapter Summary

Queuing

Chapter Contents

What is NVMe Queuing

The host will create commands on the host's schedule, and the controller will execute the commands on the controller's schedule. Chances are the two schedules are not in sync. To provide a mechanism of synchronization, the host will place commands on a queue as it creates the commands, and the controller will fetch the commands as it is able to handle them. At times the host could be producing commands at a rate of 10's, 100's or even thousands of times faster than the controller can execute them. At other times, the controller may be able to keep up with the host.

NVMe defines two types of queues: 1) for the host to submit commands to the controller (Submission Queues (SQ)), and 2) for the controller to send completion status to the host (Completion Queues (CQ)). To allow for expansion, the NVMe specification provides the capability to support up to 64K queues per controller, with up to 64K commands per queue. Facilities are provided for the controller and for the host to limit the number of queues and also the number of entries or slots per queue.

Generic Queues

Queue characteristics

Drive-side Command Queue vs. Host-side Submission Queue

Many people in the disk drive industry (either HDD or SSD) including those writing host-side software for the drives or designing host bus adapters supporting SCSI, ATA/IDE, SATA, SAS, Fibre Channel, iSCSI or USB are familiar with tag command queuing. When the drive receives a command, it decides where to place it in its queue for most efficient execution. There may be additional information from the host limiting the placement in the queue.

NVMe queuing does not work that way. It is purely a first-in-first-out queue. The host places commands in the next available entry slot on the appropriate queue and the drive pulls the commands off in the order they exist on the queue. There is no concept of some commands having a "head of queue", "ordered", "simple", "queue priority", or "maintain sequential data integrity" attributes.

Host-side submission queues and drive-side command queues work completely independently of one another. After the drive pulls a command off the submission queue, it will place the command in a command queue slot based on command attributes and drive algorithms.

Figure 105 attempts to emphasis the differences in general concepts. Details of operation and definition of terms are later in this chapter.

As an application program creates a command for a storage device, the command is placed on the circular queue in the host based on the tail pointer. That pointer will then increment and point to the next slot. When the drive is ready for another command, it will fetch the command based on the head pointer of the host-side queue. The head pointer will later be updated to show the slot is available. The pointers eventually go all the way around the circular queue with the last address "n". "N+1" points back to the first address, "base."

The command that has been selected/fetched from the host will go to the drive. The drive will analyze the command and place it into its queue based on various algorithms defined in the storage interface standards or in a manner for most efficient operation based on the device design. The commands will always leave the queue from slot 0 - the head of queue.

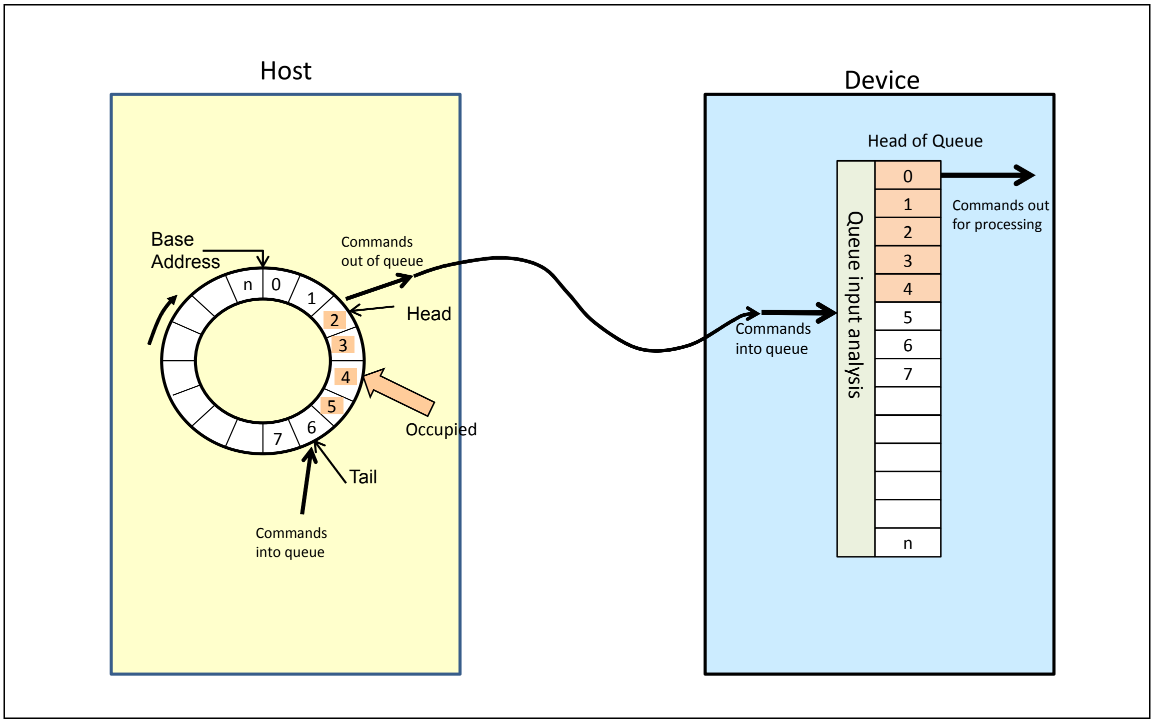


Figure 105: Host-Side Queue vs. Drive-Side Queue

Linear vs. Circular queues

Some queues such as at the theater or bus stops are linear. They begin at the box office or bus stop sign and extend away as far as necessary for the number of people in line. When the people at the head of the queue (those waiting the longest) are serviced and removed from the queue, the other behind them all move up toward the head of the queue. If 10 people get on the bus, everyone else moves up 10 slots.

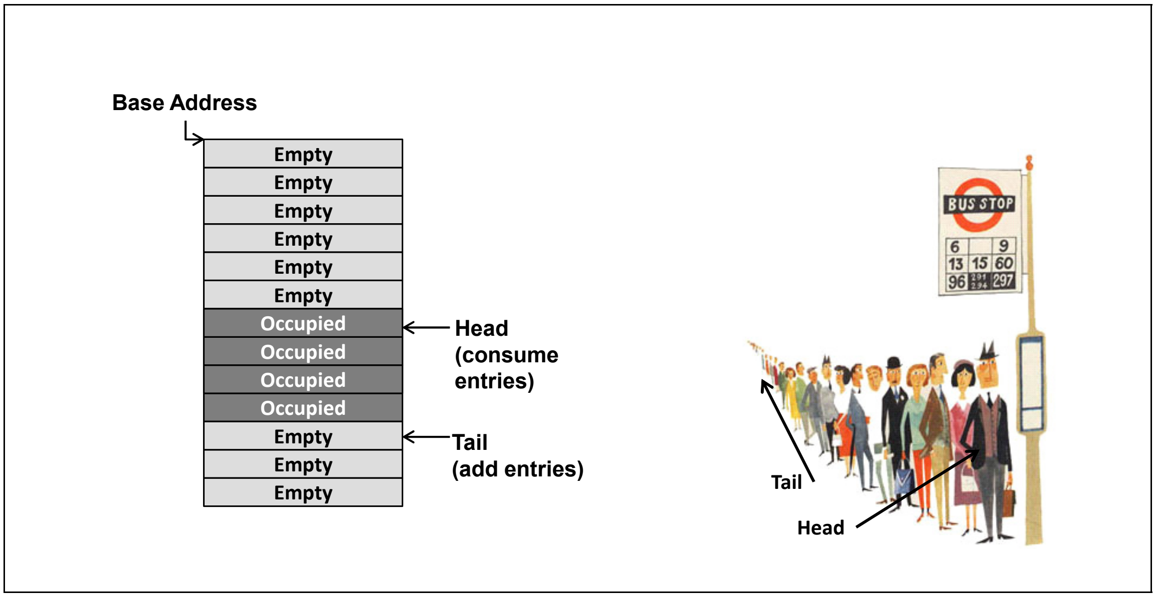


Figure 106: Linear Queue Examples

Other queues work on the same concept, except the end of the queue connects back to the beginning making it circular. In memory, the physical addresses begin with a base address and continue to the end of the queue with a pointer at the last address pointing to the first address as the next entry. For this type of queue to work, pointers are required to identify the head and tail of the queue. Conveniently, they are called the head pointer (beginning of valid queue entries) and tail pointer (end of valid queue entries). Entries are added at the tail pointer and the tail pointer is updated to the next empty slot. Entries are removed from the head pointer and the head pointer is updated to the next valid slot. See Figure 107.

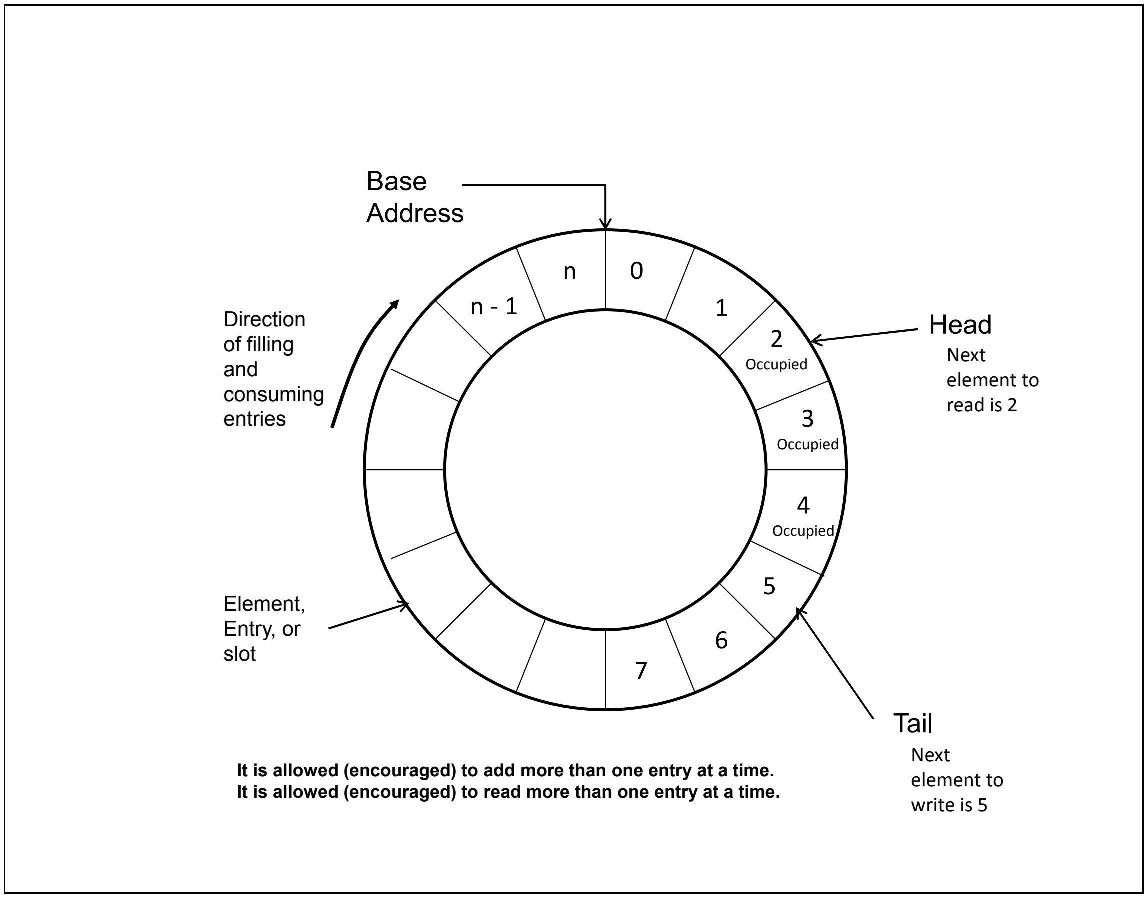

Base Address
n
0
n - 1
1
2
Occupied
3
Occupied
4
Occupied
5
6
7
Head
Next element to read is 2
Direction of filling and consuming entries
Element, Entry, or slot
Tail
Next element to write is 5
It is allowed (encouraged) to add more than one entry at a time.
It is allowed (encouraged) to read more than one entry at a time.

Figure 107: Circular Queue

Queue base address

Figure 106 and Figure 107. A queue has to have a starting point. The queue for the movie theater starts at the box office and extends away from the sign. In computer memory, the queue is given a base address as a starting point and includes the memory addresses incrementing from there to the maximum amount of memory used by the queue.

Queue size (number of entries)

Perhaps there is no practical limit to how many people can be in the queue for a new release of a movie, but in computers it is very desirable to limit the number of entries in the queue. With no limit on number of entries, it is entirely possible to overwrite information in memory that follows the location of the queue.

Queue entry size

Not all people in the queue for the movie are the same size (and some have different “personal space” zones they want to maintain) but in computers it is most efficient for all entries to be the same size. If each entry is a fixed size (e.g. 64 bytes) when one entry is removed from the queue, the queue manager can find the next entry by multiplying by that size (in this case 40h).

Because there are a fixed number of entries, with each entry the same size, the memory manager can determine the size of the queue in bytes by multiplying one number times the other. By adding that to the base address, the memory manager knows the beginning and ending addresses of that queue.

Head pointer

The head pointer points to the oldest entry in the queue. In the theater queue, it points to the person who arrived first of all the people waiting to get in. In the case of a linear queue, the head pointer always points to the beginning of address of the queue. In a circular queue, the head pointer moves around the queue, always pointing to the next entry to be pulled off the queue.

Tail pointer

The tail pointer points to first empty slot in the queue - the slot where the next entry will be placed. In a linear queue, this pointer will increase or decrease depending upon the number of valid entries in the queue. In a circular queue this pointer will always move in the direction of increasing memory addresses, except when wraps from the last entry to the first.

NVMe Queues

NVMe queues are circular. There can be up to 64K queues with up to 64K entries per queue. Controller capability can limit the maximum number of queues and the maximum number of entries per queue.

In NVMe 1.0 and 1.1, the queues were physically located in host memory. NVMe 1.2 defined a Controller Memory Buffer Feature which allows the queues for a controller to be physically located on the controller. The Controller Memory Buffer is optional. In either case, the queues are addressed by the PCIe 64-bit memory address. Reference “Controller Memory Buffer” on page 45.

Type by Contents

Submission Queue (SQ)

SQ are used by the host to submit commands to the controller. Every entry (slot) in the queue is 64 bytes long and all commands are defined as 64 bytes. SQs have a priority associated with them: urgent, high, medium, or low. Each SQ must be associated with a single CQ. This association is created when creating the SQ.

Completion Queue (CQ)

CQ are used by the controller to send command completion status to the host. Many SQs can be associated with a single CQ. CQ are all 16 bytes long. The CQ must be created before any associated SQs can be created.

Type by Command Set

Admin Queues

Admin queues are for the management of the controller. There is a single Admin SQ (SQ ID = 00h) and a single Admin CQ (CQ ID = 00h) per controller. The Admin queues are build by writing to Controller Configuration Registers in the Configuration Address Space. See “Creating the Admin Queues” on page 46. Admin queues can have up to 4096 entries each.

I/O Queues

The I/O Queues are for submitting commands from the supported I/O command sets. The only I/O command set defined at the time of the writing of this book is NVMe. SQ ID and CQ ID are always greater than 00h for I/O queues.

NVMe Pointer Operation

The generic head pointer, tail pointer and their usage are defined above.

In NVMe, upon initialization, the head and tail pointer point to the queue base address. Any time the head and tail pointer point to the same entry, the queue is considered empty.

When the host adds one or more commands to the SQ, it will update the tail pointer to point to the next empty slot and ring the doorbell by writing the new tail pointer slot number to the doorbell register in the controller register space. The doorbell is a memory address within the host's 64-bit memory address space, but physically located on the controller. See Table 7 on page 41. Writing to the address has the side effect of interrupting the controller.

The controller will see that the new tail pointer is different from its copy of the tail pointer so the controller will fetch the command or commands that are available and that the controller can manage. The head pointer is moved in the controller so it knows from where to fetch the next commands, but the new head pointer is not communicated to the host yet meaning that the commands are still on the SQ and may be fetched again if necessary - perhaps for error recovery.

The controller will place the commands into its drive-side command queue according to the task attributes passed with the command and the controller's own algorithm.

When ready, the controller will fetch the commands from its command queue and process the command, transferring data if necessary. Moving data between the controller and host does not use queues.

The controller will complete processing the command by writing the ending status to the appropriate CQ and update the CQ tail pointer. It will then interrupt the host. It could use the pin-based interrupt of INT A/B/C/D, MSI or MSI-X interrupts. See "MSI and MSI-X interrupts" on page 21.

The host will see the interrupt and read the CQ from the head pointer and process the ending status for the commands that have finished. It knows which command the status references because the 16 byte status contains the SQ ID and the Command ID.

If the host will not be doing any error recovery, it will then update the CQ head pointer and notify the controller by writing to the CQ head doorbell.

Queue Empty/Queue Full

The initialization process sets the tail pointer and head pointer both at the base address of the queue. The queue is empty. If the submitter (host for SQ, controller for CQ) submits an entry, the tail pointer is moved one slot forward for each command or status placed in the queue. The head and tail are no longer equal and the queue is no longer empty. If the submitter adds four more entries (total of 5), the tail is five slots greater than the head. When the consumer fetches the five entries and moves the head pointer forward five slots the tail and head pointers are again equal and the queue is empty. Rule: if the tail pointer and head pointer point to the same slot, the queue is empty.

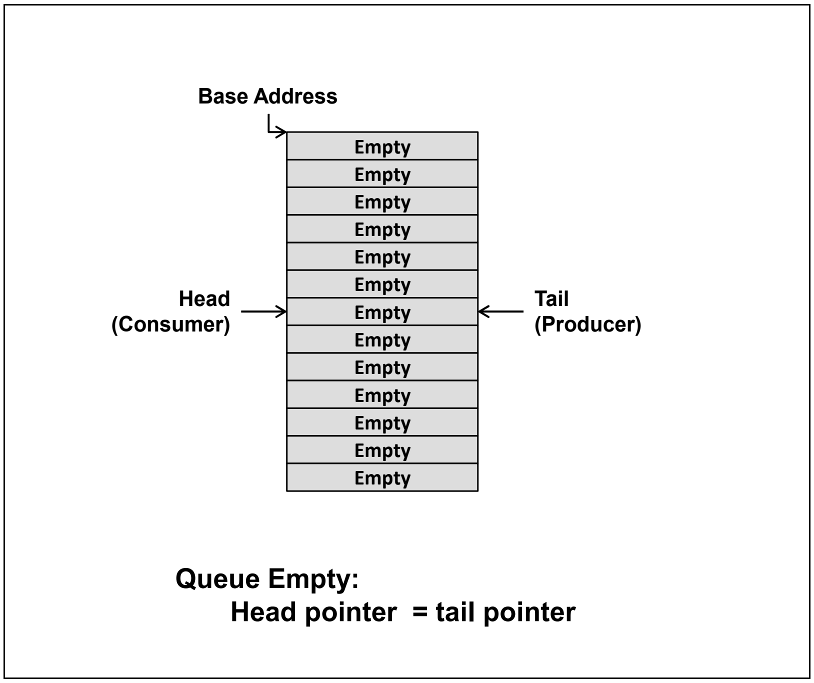


Figure 108: Queue Empty example

The other end of the spectrum is queue full. When the tail pointer continues to advance, wrapping at its highest memory address back to the base address, and then continues to advance, it will eventually reach the head pointer. If it were to continue past the head pointer, slots containing valid and un-fetched commands or status would be overwritten and lost. To prevent that from happening, the submitter may not add an entry if that would make the tail pointer equal the head pointer because tail = head means queue empty.

Remember that the head pointer points to the next entry to be fetched. In other words, it is pointing to the oldest valid entry in the queue.
The tail pointer points to the next slot to receive an entry. In other words, it is pointing to the next available empty slot.
From this we can see that there will always be at least one empty slot in the queue, even if it is completely full.

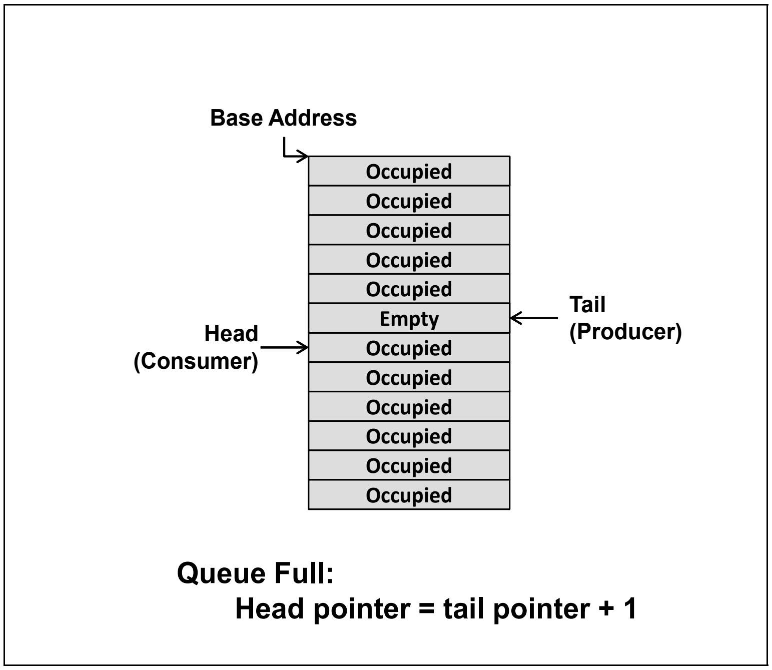


Figure 109: Queue Full Example

Table 37 shows in a single place the uses of the queue pointers. It should be a summary of the material covered about queues so far in this book.

The first four content rows of the table describe how the host uses the head and tail pointer for both the SQ and CQ (total of four pointers). The last four content rows describe how the controller uses the head and tail pointer of both the SQ and CQ (total of four pointers). Of course for a given SQ or a given CQ there are only two pointers and both the host and controller uses the same pointers.

Line	Device	Queue	Pointer	Needed to know	Information Maintained or Passed
1	Host	SQ	Head	Is Queue full?	Passed in Completion queue entry
2			Tail	Add an entry	Information maintained by host
3		CQ	Head	Fetch Status Entry	Information maintained by host
4			Tail	Are there more entries?	Information not needed by host; used P bit to discover last entry
5	Controller	SQ	Head	Fetch next command(s)	Information maintained by controller
6			Tail	There are more entries	Passed in SQ tail doorbell
7		CQ	Head	Release completion queue entries	Passed in CQ doorbell
8			Tail	Add entry	Information maintained by controller

Table 37: NVMe Queue Pointer Operation

Line 1: The host needs to know the SQ head pointer to tell if the queue is full. If the queue is full, the host cannot add any commands to it. The controller uses this pointer to pull the commands off the queue so keeps track of the pointer. The controller will tell the host in each completion status posting the current value of the SQ head pointer. The current value may have nothing to do with the command for that this status is associated with.

Line 2: The host maintains the SQ tail pointer, when it adds a command to the SQ, it will update the pointer (and pass that new pointer value to the controller in the SQ tail doorbell).

Line 3: The host maintains the CQ head pointer. When it is able to process completion status, it will use the CQ head pointer to find the next CQ entry, pull the status off the queue and update the pointer. (After processing the status, the host will pass the CQ head pointer to the controller via the CQ head doorbell.)

Line 4: The CQ tail pointer is maintained by the controller because it is the controller that adds ending status entries to the CQ tail. The host could use this information to discover where the last entry is (are there more entries for the host to process). However, the NVMe committee created a different manner to communicate that information from the controller to the host - the P or Phase bit.

Line 5: The SQ head pointer is maintained by the controller. It used the head pointer to know where to fetch the next command(s) and then updates the pointer to know from where to fetch next time.

Line 6: The controller needs to know the SQ tail to see if the host added more commands. This information is passed from the host to the controller via the SQ tail doorbell.

Line 7: The controller needs to know the CQ head pointer to know if the CQ is full. It will compare the head and tail pointers and if the tail is one less than the head, the queue if full.

Line 8: The controller maintains the CQ tail pointer and places the next completion status in the CQ entry pointed to by the tail pointer, then it updates the pointer.

Queue Priority

There are two priority mechanisms defined in NVMe: Round Robin which is mandatory for all controllers, and Weighted Round Robin with Urgent which is optional. Also, Controllers may support a vendor specific priority mechanism. The priority mechanisms supported are in the Controller Capabilities Registers, bits 18:17. The priority mechanism currently selected is in the Controller Configuration Register, bits 13:11.

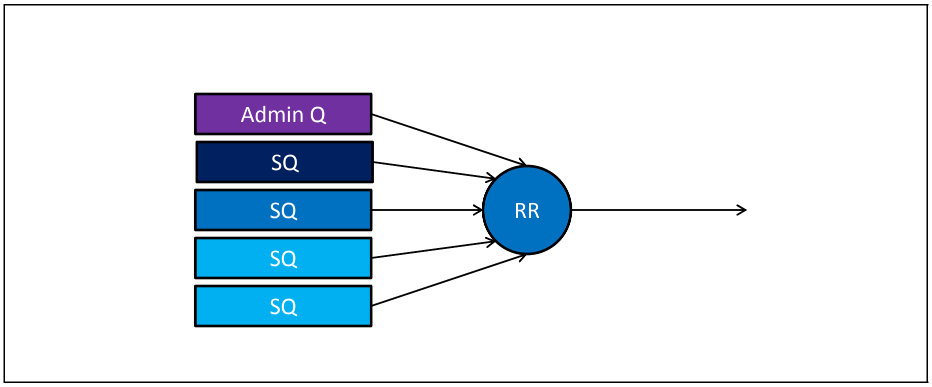


Figure 110: Round Robin Priority

Figure 110 shows the Round Robin priority mechanism. All queues have an equal priority, including the Admin Queue. The controller pulls one or more commands, up to the Arbitration Burst setting, from each queue per round.

The other mechanism defined in the NVMe specification is Weighted Round Robin w/Urgent shown in Figure 111. This introduces Strict Priority and Weighted Priority. The method of assigning priority is discussed in “Create I/O Queues Commands” on page 72

Strict Priority means that ALL of the commands in the Admin Queue will be fetched and will start processing before any commands from the other queues - Strict Priority 1. Next, ALL commands in the queues assigned Urgent Priority - Strict Priority 2 - will be fetched and start processing before any commands are fetched from a lower strict priority queue. All queues with Urgent priority assigned to them will be treated in as round robin. Notice the two SQs assigned Urgent (blue) in Figure 111. Finally all the other queues will be available for providing commands to the controller as Strict Priority 3.

Weighted Round Robin means that the queues with higher priority will be given preference over the queues with lower priority. In NVMe, there are three levels of priority for weighted round robin. An example of how it may work is that with the High Priority queues the controller may fetch 30 commands at a time. From the Medium Priority queues the controller may fetch 20 commands at a time. From the Low Priority queues the controller may fetch 5 commands at a time. All queues within each of the priorities are selected with round robin and shown in the figure.

The Weighted Round Robin numbers (Set Features ID = 01h Arbitration) control the WRR bubbles below. The Arbitration Burst applies to each SQ.

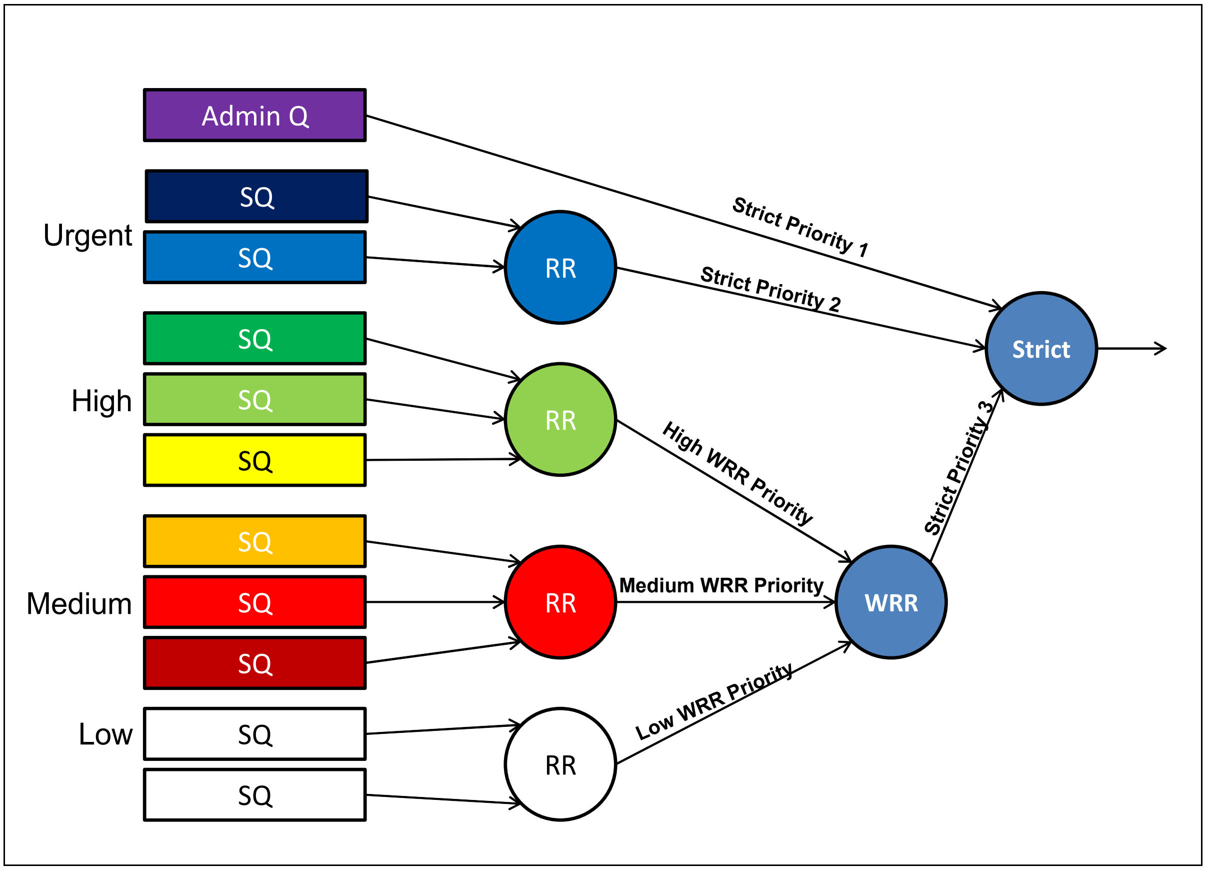


Figure 111: Weighted Round Robin with Urgent Priority

Doorbells

Doorbells are defined as Controller Registers beginning at register address 1000h. They are physically located in the controller memory which is located in PCIe memory address space specified by the device's BAR 0 and BAR 1. See also "NVMe Controller Registers" on page 40.

On each controller, each SQ has a tail doorbell written by the host when it adds commands to the queue. Writing the new queue tail to the doorbell tells the controller the queue entry for the last command in the queue. (The tail points to the first empty entry so the controller knows that the entry immediately before that is the last valid command.)

On each controller, each CQ has a head doorbell written by the host when it completes processing of returned status. The value written to the CQ head doorbell tells the controller what queue entries are now available for new status.

Each controller can have up to 64K SQs and up to 64K CQs. Each doorbell register is four bytes long.

If the doorbells are implemented in software, it may be advantageous to have each doorbell on a cache line boundary. To implement that, the Specification defines a "doorbell stride" (DSTRD) which places memory locations between each doorbell. If each cache line is 32 bytes, each doorbell can be configured to start on a 32 byte boundary. DSTRD = (2 $^{(2+\mathrm{DSTRD})}$) in bytes. A DSTRD value of 3 means a stride of 32 bytes which means each doorbell is on a cache line boundary.

For hardware implementations, there may be no benefit for the alignment. A DSTRD of 0 (DSTRD = (2 $^{(2+0)}$) means a stride of 4 bytes which means the doorbells are packed with no space between them.

In the controller registers list the location of each SQ doorbell is calculated as:
1000h + (2y * (4<< CAP.DSTRD)) to 1003h + (2y * (4 << CAP.DSTRD)).
Each CQ doorbell is calculated as:
1000h + ((2y + 1) * (4 << CAP.DSTRD)) to 1003h + ((2y + 1) * (4 << CAP.DSTRD)).

where y is the SQ or CQ ID,
CAP.DSTRD is the capabilities register, bit doorbell stride (bits 35:32), and
<< means shift left two places.

To calculate the address of the SQ 5 tail doorbell address using a DSTRD = 3:
Addr = 1000h + (2 * 5 * (4 shifted left 3 places))
Addr = 1000h + (10 * 32)
Addr = 1320h

An NVM Express controller is associated with a single PCI Function. PCIe as defined in the PCI Express 3.1 Specification has a single upstream port on each function or bridge. SR-IOV maintains this single root concept. Therefore there can be only one host writing to each controller's doorbells. Multi-Root Input Output Virtualization (MR-IOV), when implemented, may allow multiple hosts to connect to a single controller.

Chapter Summary

- What is NVMe Queuing
- Generic Queues
    - Queue characteristics
        - Drive-side Command Queue vs. Host-side Submission Queue
        - Linear vs. Circular queues
        - Queue base address
        - Queue size (number of entries)
        - Queue entry size
        - Head pointer
        - Tail pointer
- NVMe Queues
    - Type by Contents
        - Submission Queue (SQ)
        - Completion Queue (CQ)
    - Type by Command Set
        - Admin Queues
        - I/O Queues
    - NVMe Pointer Operation
        - Queue Empty/Queue Full
- Queue Priority
- Doorbells

RDMA over PCIe

Chapter Contents

Overview

Caution: RDMA has two different descriptions, one a subset of the other. This chapter describes the more restricted description which is what NVMe over PCIe uses.

InfiniBand (NVMe over Fabrics) defines the more inclusive version which includes additional features:
Information transfer controlled by the remote device (defined in this chapter),
OS Bypass.
TCP (OSI Layer 4) Offload Engine, and
Verbs for abstraction of defined functions.
Ethernet, when using either iWARP or RoCE (NVMe over Fabrics), uses the more inclusive description.

What is RDMA

Remote Direct Memory Access.

From Wikipedia: Access from the memory of one computer into the memory of another computer without involving either one's operating system. Note for NVMe: the one computer may be a storage device.

RDMA is based on two synergistic concepts: direct access to a network interface (e.g. a NIC) straight from application space, and an ability for applications to exchange data directly between their respective virtual buffers across a network, all without involving the operating system directly in the address translation and networking processes needed to do so.

Benefits:
High throughput,
Low latency,
No operations required on the part of either CPU or caches, and
Transfer continues in parallel to other system operations.

Local DMA Operation

See Figure 112 below. The processor writes a memory address for the transfer to the DMA controller in the host chip set. It then sends the write or read command, including command op code, LBA, sector count, command identifier (queue tag) and other command specific information to the remote device. The remote device transfers the data to or from a host controlled buffer and the DMA controller in the host will access memory in the same host associated with the data.

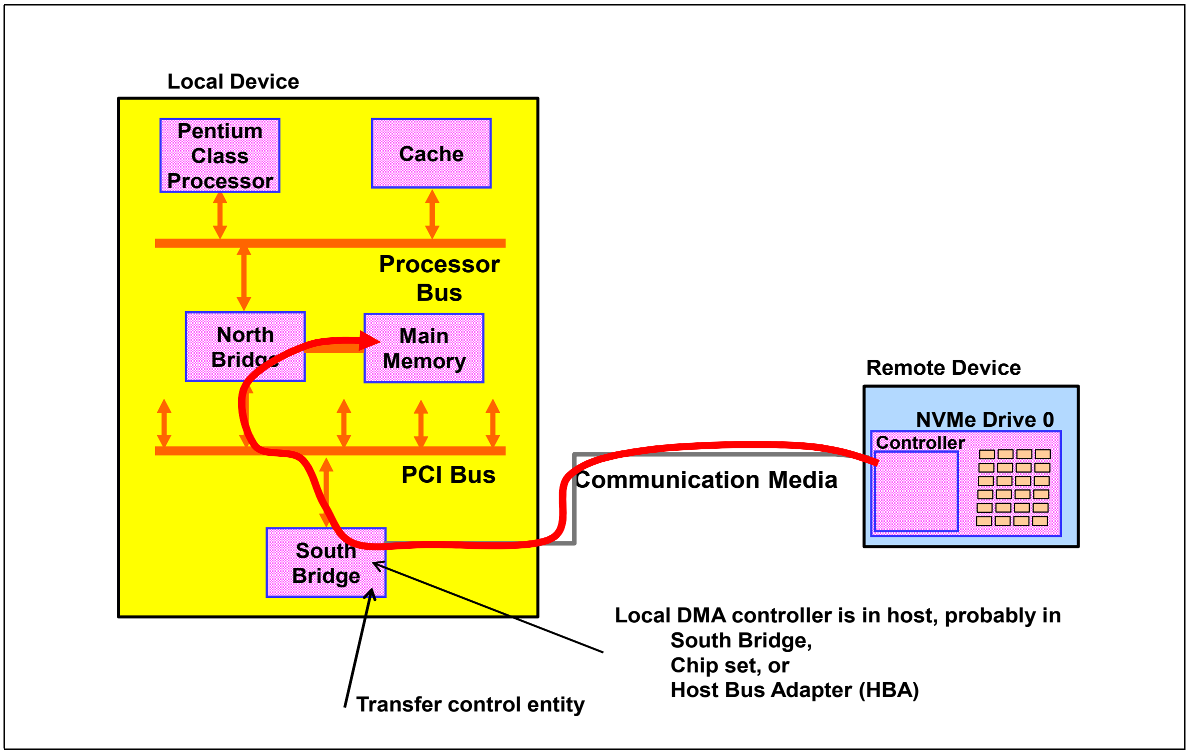


Figure 112: DMA Operation

Memory fragmentation of the file transferred is handled by the host memory manager building a Scatter-Gather List, also called a Physical Region Page List or Physical Region Descriptor Table. The contents of that list, page, or table will be the memory addresses accessed by the DMA Controller.

No memory addresses, real or virtual, are exposed outside the CPU frame.

Remote DMA Operation

See Figure 113 below. When using RDMA, the processor builds the command including the op code (what to do), LBA (where to do it), sector count (how much), command identifier (also called queue tag) and either the memory addresses or a pointer to the memory addresses. For an example in NVMe, see Figure 91 on page 128. The device will handle memory fragmentation though Scatter-Gather Lists (see “SGL” on page 57) or Physical Region Pages (see “PRP” on page 57). The NVMe storage device will then use PCIe write transactions for a read command or PCIe read transaction for a write command. If using a transport other than PCIe, that transport must supply a mechanism to provide for those transfers.

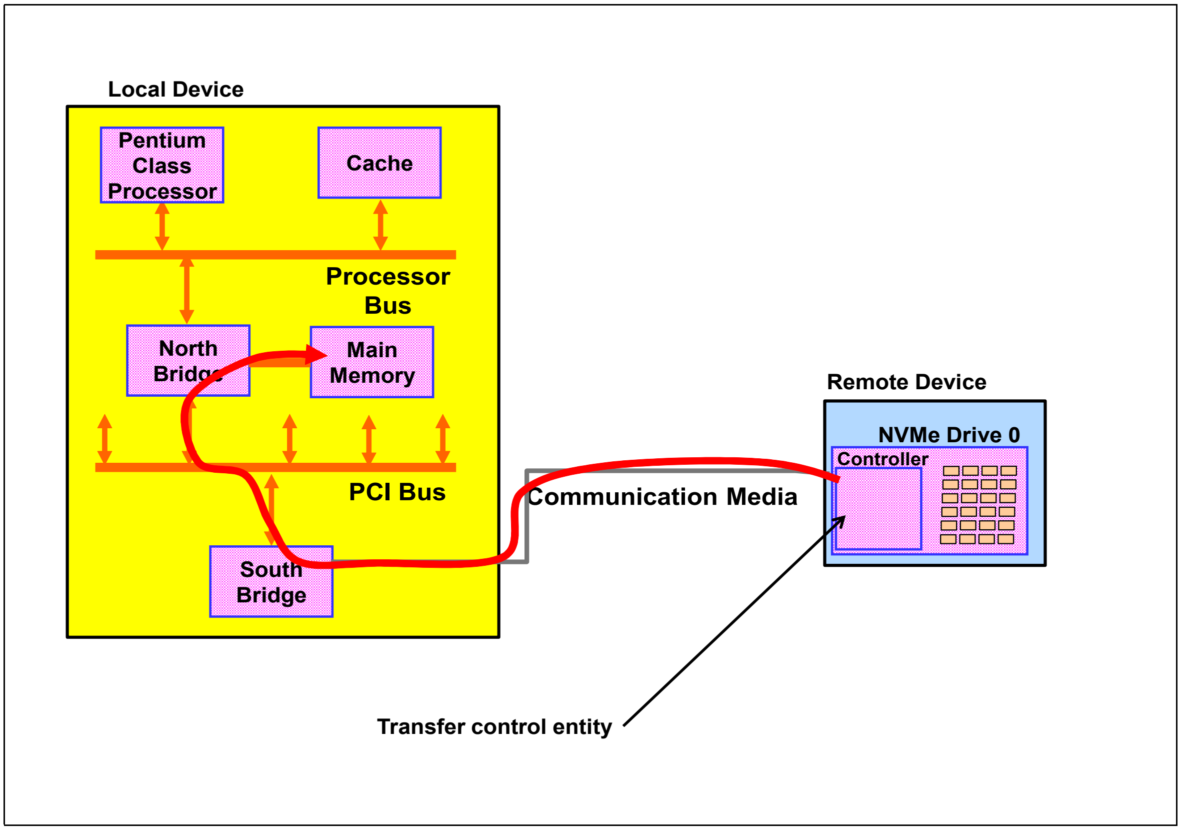


Figure 113: RDMA Operation

In the past[16], RDMA has been accused of weak security in allowing actual memory addresses to be transferred across an external cable. That is still a concern that must be addressed in some method outside the scope of PCIe or NVMe.

Benefits of RDMA

No HBA is needed to transfer the data. The memory addresses are given to and used by the remote storage device.

The remote device may directly access application memory, eliminating the necessity of the host moving data between a host supplied buffer and the application memory.

Once the CPU sets up the transfer, it can perform other CPU specific duties rather than monitor the transfer. This makes the CPU much more efficient.

[16] In 1998, IEEE 1394 (also known as FireWire (Apple Trademark) or iLink (Sony Trademark)) attempted to implement RDMA but was forced to revert to standard DMA with a queue tag.

Disadvantages of RDMA

A. The remote device must be smarter and able to handle the memory address. However, since the 1960's, computer architects have been moving the intelligence out from the CPUs to the I/O devices. This may be considered just another step in that progression.

B. Actual physical and virtual memory addresses are transfered across wires outside the CPU frame or enclosure. That is considered a security threat as someone may get access to those wires and read the data. Several methods could be used to reduce this threat, such as:

having all the equipment in a secure area,
encrypt the data before sending it across the wires, or
use optical transmission medium rather than electrical.

The relative benefits and disadvantages of RDMA are different in each case.

Chapter Summary

Overview

- What is RDMA
- Local DMA Operation
- Remote DMA Operation

Benefits of RDMA

Disadvantages of RDMA

Translating SCSI Commands

Chapter Contents

Why SCSI Commands

Many programs use SCSI commands and have been widely debugged for many years. Customers trust the programs and their I/O routines. To translate the I/O routines to NVMe commands would be a massive effort with a high risk of introducing errors. The companies that create the programs do not want to spend the millions of dollars to do the translation, and customers are not willing to make such a major change and put their business at risk.

This chapter is about The NVM Express: SCSI Translation Reference. This appears to be an interim "fix" because it does not cover all SCSI commands for block devices, nor does it attempt to translate all parameters within the commands it does cover.

Command Translation

SCSI Devices

In standard SCSI, there are generally two types of devices, Initiators and Targets.

Initiators are computers or computer-like devices that contain Application Clients. Application Clients create commands. Initiators are identified by SCSI addresses that are specific to the transport protocol.

Targets are usually peripherals that receive the command to be executed. Targets are identified by SCSI addresses that are specific to the transport protocol.

Logical Units are a subset of a target and contain Device Servers. Device Servers execute the commands. Logical Units are identified by a Logical Unit Number (LUN). Disk drives (HDD and SSD) probably have only one LUN, and that will be LUN 0.

Initiators, Targets and LU are addressable. Application Clients and Device Servers are not addressable within SCSI.

Some devices may perform both initiator and target functions. Those are referred to as operating in initiator mode or target mode. Disk drives (HDD or SSD), Tape drives, CD or DVDs usually operate in target mode only so we usually don't say "mode" for them.

Application Clients and Device Servers are not germane to our discussion of command translation.

Nexus

Nexus means connection. In SCSI it specifies the connection necessary to perform operations. Possible Nexus arguments:
IT - Initiator and Target - Established in SAS with the Open Address Frame. It is established in NVMe through the PCIe Transaction Header using ID routing. The header contains the Requester ID and the Bus and Function of the destination device.

ITL - Initiator, Target, and LUN - Established in SAS with the Command Information Unit. In NVMe there is not a direct relationship to the LUN. The NVMe controller can have multiple Namespaces, as the SCSI target may have multiple LUNs. Both the Namespace and LUN are addressable. However, the SCSI LUN contains a Device Server which processes commands, meaning that each LUN could respond to a different command set. The command set in NVMe is defined at the controller level, so all Namespaces must respond to the same command set.

ITLQ - Initiator, Target, LUN, and Queue Tag - Established in SAS with the frame header. The Command Identifier concatenated to the SQ ID forms the Queue Tag in NVMe. The ITLQ nexus structure was removed as a nexus in SAM-5.

Task

Generally a "Task" is a SCSI command.

Common Field Translation

Command parameters are of two types, those that are in many different commands (common fields) and those that are in only a few commands (command specific fields).

Logical Block Address (LBA)

In SCSI commands that read or write user data, the LBA specifies the starting block for the read or write. The Transfer Length then specifies the number of blocks to transfer.

In NVMe, this field is translated (copied) to the Starting LBA field of the Read or Write NVMe command.

Transfer Length

Transfer Length is the amount of data to be transfer with a read user data or write user data command. For block devices it is specified in the number of logical blocks.

In NVMe, this is translated (usually copied) to the Number of Blocks field of the Read or Write NVMe commands. If the SCSI Transfer Length is greater than 2^{16}, multiple NVMe commands may be needed to satisfy the request.

Allocation Length

Allocation Length is used by data-in commands, except read user data. This field in SCSI informs the Logical Unit how many bytes of buffer space the initiator has allocated (provided) for the receive data. The SCSI Logical Unit will return all of the data requested up to, but not in excess of the allocation length. Any excess buffer space is not written.

In NVMe, it is up to the initiator to provide enough buffer space for the information requested.

Auto Contingent Allegiance

Auto Contingent Allegiance is not supported in NVMe.

Parameter List Length

Parameter List Length is used by data-out commands, except write user data. This field informs the logical unit how many bytes of data the initiator has to pass to the logical unit.

In NVMe, it is up to the initiator to provide enough buffer space for passing all of the information.

Control byte

The Control byte is the last byte of fixed length Command Descriptor Block (CDB) or the second byte of the variable length CDBs. It has a bit for controlling Normal Auto Contingent Allegiance.

In NVMe the Control Byte support is unspecified.

DPO

The DPO bit tells the logical unit that the initiator does not plan on using this information again and therefore the Logical Unit may flush the information from its cache. If the Logical Unit does not flush the information based on this bit, it will flush it on its least recently algorithm. The bit is advisory only.

In NVMe such retention characteristics are not supported.

FUA

Force Unit Access causes the information to be read from or written to the non-volatile media, even if there is a copy in cache. It is used when the initiator does not trust the cache data, during error recovery, or in performance testing.

This bit is copied to the FUA bit in the NVMe command.

Immed

With the Immed bit (Immediate) the initiator tells the Logical U nit that after it verifies the command was received correctly, it may return ending status - even before it starts to execute the command. This is used on commands that take a long time to execute such as Start or Format.

In NVMe if the Immed bit is 0h, the ending status will be returned to the host after the command has finished processing. If the Immed bit is 1h, the command shall be terminated with Check Condition status, Illegal Request Sense Key and Illegal Field in CDB for Additional Sense Code.

Group Number

Group Number allow the gathering of statistics for Read and Write commands into subsets based on customer preferences.

In NVMe support for the Group Number is unspecified.

Product Identification

In SCSI, a 16-byte ASCII field assigned by the vendor.

In NVMe - this field is set to the first 16 bytes of the Model Number field within the Identify Controller Data Structure.

Product Revision Level

In SCSI, a 4-byte ASCII field assigned by the vendor.

In NVMe, this field is set to the first four bytes of the Firmware Revision field within the Identify Controller Data Structure.

T10 Vendor Id

In SCSI, this 8-byte ASCII field specifies the manufacturer of the device.

In NVMe, this field is set to "NVMe" followed by 4 spaces. (43 56 4D 65 20 20 20 20h)

Vendor Specific ID

In SCSI, the Vendor Specific Identifier is a field that identifies the Vendor. There are many formats including EUI-64, NAA, or SCSI name string. (See SPC-4, section 7.8.6).

In NVMe, this field is 36 bits long with no further description.

WRProtect

In SCSI, the WRProtect field specifies what fields in the DIF[17] or DIX[18] are checked on write commands. Using different settings of WRProtect, the host can specify that each of the fields are checked or not. The keywords for checking are: Shall (mandatory), Shall Not (prohibited), May (optional, no preference), or it Depends.

In NVMe the translation is as follows in Table 38.

WRProtect Code	PRACT	PRCHK
000b	1	000b
001b, 101b	0	111b
010b	0	011b
011b	0	000b
100	0	100b

Table 38: WRProtect Translation

RDProtect

In SCSI, the RDProtect field specifies what field in the DIF or DIX are check on read commands. Using different settings of RDProtect, the host can specify that each of the fields are checked or not. The keywords for checking are: Shall (mandatory), Shall Not (prohibited), May (optional, no preference), or it Depends.

In NVMe, the translation is as follows in

[17] DIF is Data Integrity Field, an End-to-End data protection mechanism defined by ANSI T10. Generally, the data protection fields are stored with the data in a longer physical block.

[18] DIX is Data Integrity Extensions, an End-to-End data protection mechanism defined by Storage Networking Industry Association (SNIA). The data protection information may be stored in a separate location from the primary data.

RDProtect	PRACT	PRCHK
000b	1b	111b
001b, 101b	0	111b
010b	0	011b
011b	0	000b
100b	0	100b

Table 39: RDProtect Translation

Command Translation

Read and Write Commands

All SCSI Read and Write commands used to read or write user data on the Logical Unit's media are optional. Originally, the CDB's were 6 bytes long, then transitioned to 10 bytes, 12 bytes, 16 bytes and 32 bytes. Each longer format allowed more options or larger LBA's or Transfer Lengths. A major option of the 32 byte long CDB is End-to-End Data protection (DIF or DIX).

NVMe provides only one format of Read and Write command which allows for a 64-bit LBA which is the largest allowed in any of the SCSI CDBs. The SCSI Read and Write commands that are 16 bytes or 32 bytes long allow for a 32-bit transfer length; transfer lengths of over 16 bits requires two NVMe read or write commands.

NVMe provides for DIF and DIX by allowing the protection information to be transfer as Metadata to be stored with the data or in a separate Metadata buffer, depending upon how the media was formatted.

Inquiry, Mode Select, Mode Sense

In SCSI, target and LU characteristics and mode parameters are read with the Inquiry command or Mode Sense command. Most parameters are changed with the Mode Select command. Parameters that effect the mode of operation are grouped by category in pages. The Mode Sense and Mode Select commands access the parameters by page number.

NVMe provides much more information with the Identify command and the Controller or Namespace data structures. Mode Parameters can be are simulated with the Get Features and Set Features commands. See "Get Features" on page 90

Log Pages

In SCSI log pages provide detailed description on errors or operating conditions. Over time, a lot of complexity has crept into the Log Pages with some pages having sub-pages and parameters.

In NVMe there are only six log pages, acquired by the host from the controller with a Get Log command. The command and pages are in the Admin Commands of this text. See "Get Log Page" on page 108.

VPD Pages

The host can read Vital Product Data (VPD) information in SCSI using the Inquiry command and specifying the VPD page.

In NVMe, only the following pages are defined as being supported:

Page Code	Description	Source of Information	M/O
00h	Supported VPD Pages	NVMe Specification	M
80h	Unit Serial Number	From EUI64 of Identify Namespace Data Structure. Insert "_" after 4th, 8th and 12th digit and a "." after the 16th digit	M
83h	Device Identification	From Identify Namespace Data Structure	M
86h	Extended Inquiry Data		O
B1h	Block Device Characteristics VPD Page		M
All Others	Command shall be terminated with: Check Condition status, Illegal Request Sense Key, and Illegal Field in CDB ASC		M

Figure 114: Supported VPD Pages

Status and Sense Data

SCSI has a very few Status codes but a very structured hierarchical sense data structure.

Code	Meaning
00h	Good, command completed without error
02h	Check Condition, target/LU detected a condition that the Initiator needs to check. May be an error or not.
08h	Busy
18h	Reservation Conflict
28h	Queue Full/Task Set Full
30h	Auto contingent Allegiance (ACA) Active
40h	Task Aborted

Table 40: SCSI Sense Codes

If the Status Code is 02h - Check Condition - there will usually be a Sense Data structure available to further define the condition. First check the 4-bit Sense Key, then the Additional Sense Code (ASC) and Additional Sense Code Qualifier (ASCQ). The sense keys generally will let the host driver software to decide which error routine to execute.

0h	No Sense
1h	Recovered Error
2h	Not Ready
3h	Medium Error (Hard error on the media)
4h	Hardware Error
5h	Illegal Request
6h	Unit Attention
7h	Data Protect
8h	Blank Check (Generally for Read-only or Write-Once devices)
9h	Vendor Specific
Ah	Copy Aborted
Bh	Aborted Command
Ch	Reserved
Dh	Volume Overflow (for buffered devices)
Eh	Mis-compare
Fh	Completed. (May occur on successful command)

Table 41: SCSI Sense Keys

The ASC and ASCQ are each 8 bits long. The ASCQ modifies or expands the ACS field. For example ASC of 29h means Power on or reset occurred.
ASC-ASCQ of:

29-01 means Power on occurred
29-02 means SCSI Bus Reset occurred
29-03 means Bus Device Reset Function occurred
29-04 means Device Internal Reset
29-05 means Transceiver mode changed to single-ended (parallel SCSI only)
29-06 means Transceiver mode changed to LVD (parallel SCSI only)
29-07 means IT Nexus Loss occurred

Values for each of 00h - 7Fh are defined by the SCSI committee or reserved for their use. Values of 80h - FFh are vendor specific. SPC-4[19] rev 36q has 19 pages of committee defined ASC-ASCQ conditions.

The SCSI Commands Reference 1.3 Gold.pdf available from NVMExpress.org/specifications translated some NVMe conditions to their respective SCSI Status/Sense/ASC. Table 42shows those translations.

[19] SCSI Primary Commands, Available from www.t10.org

SCSI			NVMe
Status Code	Sense Key	ASC	Status Code
Good	No Sense	N/A	Success Completion
Check Condition	Not Ready	LU Not Ready, Cause not reportable	NS Not Ready
	Medium Error		Data Transfer Error
			Capacity Exceeded
		Peripheral Device Write Fault	Write Fault
		Unrecovered Read Error	Unrecovered Read Error
		LB Guard check failed	E-to-E Guard check error
		LB Application Tag check failed	E-to-E Application Tag check error
		LB Ref Tag Check Failed	E-to-E Reference tag check error
	H/W Error	Internal Target Failure	Internal Device Error
	Illegal Request	Access Denied - Invalid LU ID	Invalid NS or Format
		LBA out of Range	LBA our of Range
		Invalid Cmd OP Code	Invalid Cmd Op Code
		Invalid field in CDB	Invalid Field in Cmd
			Completion Q Invalid
			Abort Command Limit exceeded
		Format Command Failed	Invalid Format
		Invalid Field in CDB	Conflicting Attributes
		Access Denied - Invalid LU ID (Media Error)	Access Denied
	Miscompare	Miscompare during verify Op	Compare Failure
Task Aborted	Aborted Command	Warning - Power loss expected	Cmds aborted doe to power loss notification
			Cmd Abort Requested
			Cmd Aborted due to failed fused cmd
			Cmd Aborted due to missing fused cmd
			Cmd Aborted due to SQ Deletion

Table 42: SCSI to NVMe Status and Sense Translation

Task Management Functions

In SCSI, commands are referred to as tasks. Task management functions are SCSI's method for the host to manage the commands that have been sent to the drive. NVMe has translated those SCSI Task Management Functions to NVMe function as shown in Table 43.

Function Name	Nexus	Function in SCSI	NVMe Translation
Abort Task	ITLQ	Delete a single command sent to a SCSI Target/LUN	Use Abort Admin command
Abort Task Set	ITL	Delete all commands that one SCSI initiator sent to the SCSI Target/LUN	Use separate Abort Admin command for each outstanding command to abort
Clear Task Set	ITL	Delete all commands that all initiators have sent to the SCSI Target LUN	Use separate Abort Admin command for each outstanding command to abort
Clear ACA	ITL	Clear Auto Contingent Allegiance	Unspecified, ACA not defined in NVMe
IT Nexus Reset	IT	Reset the target and all LUs on that target	Controller returns Function Succeeded if there are outstanding commands in the SQ, otherwise return Function Complete
Logical Unit Reset	ITL	Reset the identified logical unit and all dependent logical units	Shall be supported by writing a 0 to Enable field of the Controller Configuration Register
Query Task	ITLQ	Ask LUN if it has a specific command	May be supported
Query Task Set	ITL	Ask LUN if it has any command from this initiator	Unspecified
Query Asynchronous Event	ITL	Ask LUN if it has any events to report that are not associated with a command from this initiator.	Unspecified

Table 43: Task Management Functions Translation

Chapter Summary

- Why SCSI Commands
- Command Translation
    - SCSI Devices
    - Nexus
    - Task
    - Common Field Translation
        - Logical Block Address (LBA)
        - Transfer Length
        - Allocation Length
        - Auto Contingent Allegiance
        - Parameter List Length
        - Control byte
        - DPO
        - FUA
        - Immed
        - Group Number
        - Product Identification
        - Product Revision Level
        - WRProtect
        - RDProtect
    - Command Translation
        - Read and Write Commands
        - Inquiry, Mode Select, Mode Sense
    - Log Pages
    - VPD Pages
- Status and Sense Data
- Task Management Functions
- Chapter Summary

NVMe Management Interfaces

Chapter Contents

Overview

This chapter is called “Out-of-Band Management” meaning that the management of the devices or transport is not performed through the same mechanism as data flows. In-band management with NVMe is through the Admin commands. Benefits of out-of-band management are:

it does not (usually)[20] impact the data flow, and
it may work where in-band management would fail.

On February 24, 2015, the NVMe committee added a specification for an out-of-band management facility to provide minor maintenance functions. It is called Technical Note: NVMe Basic Management Command. It is based on SMBus – System Managed Bus – which is a two wire, low speed transport running on contacts other than the data contacts. SMBus is similar to I^2C, in that both have a clock wire and a data wire, and they use similar signaling.

On November 17, 2015, the committee released a more full function specification for out-of-band Management. It is based on a Management Component Transport Protocol (MCTP) which will allow the host to discover, modify or configure the NVMe device. This interface is called NVMe Management Interface.

The Basic Management Command specification was move to an annex of the NVMe Management Interface specification.

This chapter first covers the full Management Interface and then covers the Basic Management Command.

NVMe Management Interface

NVMe-MI Introduction

The NVMe committee wanted to make the Management Interface easy to understand and easy to develop, using as much existing standards and designs as possible and still have a working interface. To this end, they designed the Management Interface to run the already defined Management Component Transport Protocol (MCTP) which runs on either the SMBus or the PCIe Vendor Defined Messages (VDM). The upper layer protocol could be any compliant Management Application, such as Remote Console. See Figure 115. The committee desired that the Management Interface is both processor and transport agnostic.

NVMe Management Interface provide two slots for simultaneous command execution. It is envisioned that while one slot is executing a command that takes a long time, other shorter duration commands could be executing in the other slot.

[20] It is possible to design a system that uses the management described in this chapter but travels over the same path as the data. In that case, the management commands would compete with the data for the transport resources.

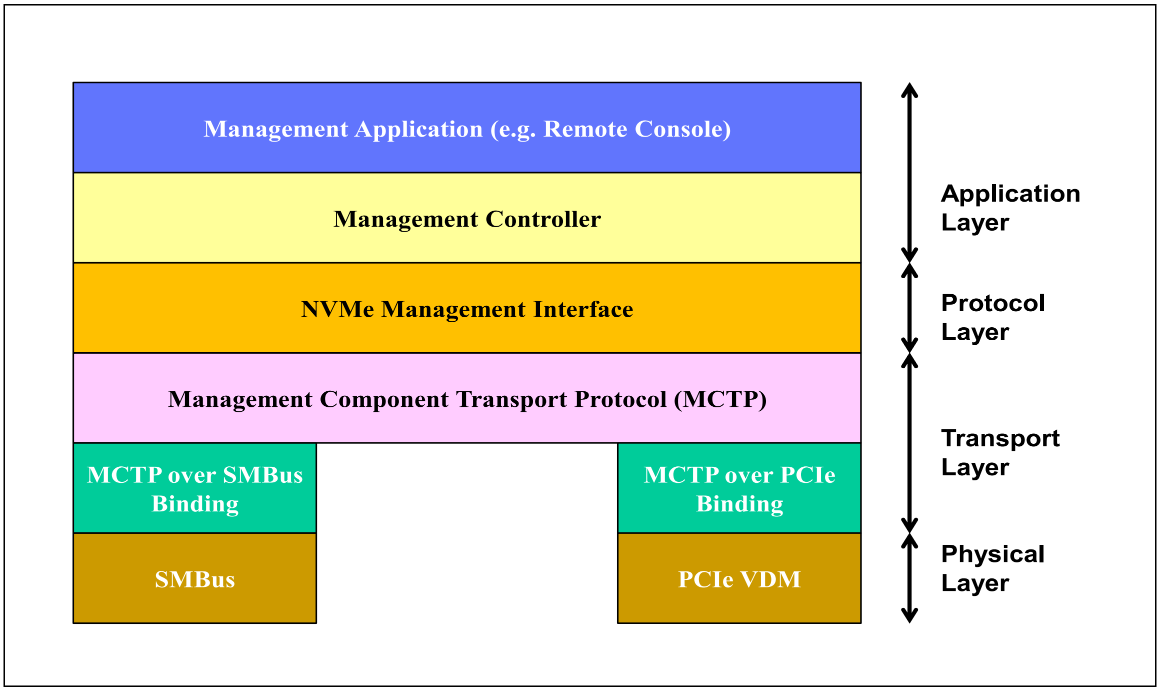


Figure 115: Management Interface Block Diagram

Documentation

The following documents are available from Distributed Management Task Force, Inc. at www.dmft.org.

Document Identifier	Document Name
DSP0235	NVMe Management over MTP Binding
DSP0236	MCTP Base Specification
DSP0237	MCTP SMBus Transport Binding
DSP0238	MCTP PCIe VDM Transport Binding
DSP0239	MCTP IDs and Codes

Figure 116: MCTP Specifications

The following documents are available from NVM Express, at www.NVMexpress.org.

NVMe Specifications
NVMe 1.2.1 - Base NVMe Specification
NVMe Management Interface 1.0 Gold - Management Interface

Figure 117: NVMe Specifications

Example Configurations

The following four sample configurations are examples only and not intended to dictate or limit your configurations.

Figure 118 with one PCIe port will transport both the data and management commands over the same PCIe hardware, thus perhaps impacting data performance. It is the low cost method, and the simplest method for users.

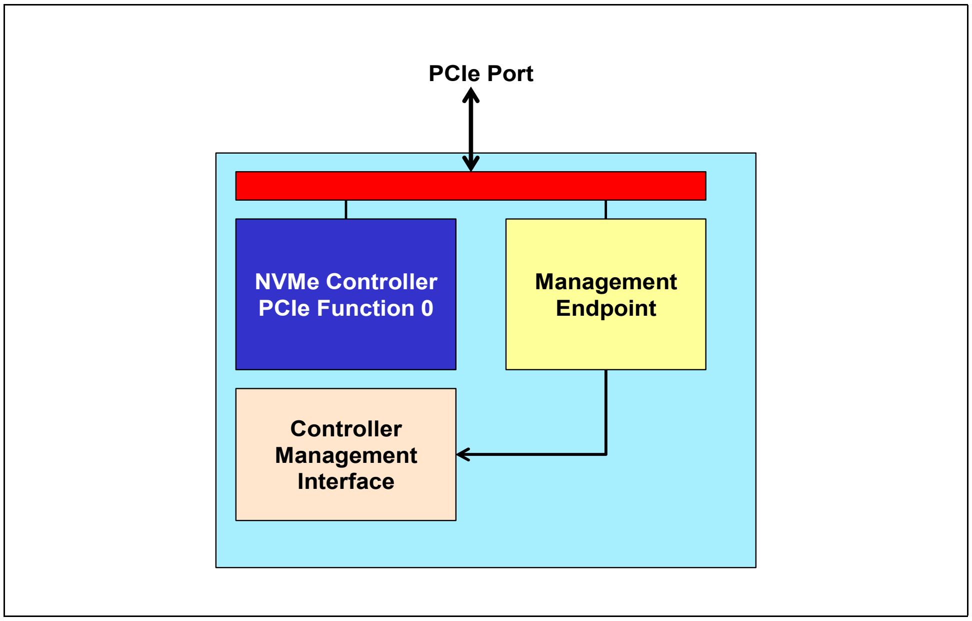

PCIe Port
NVMe Controller PCIe Function 0
Management Endpoint
Controller Management Interface

Figure 118: Management with One PCIe Port

Figure 119 adds a second PCIe port so the management does not compete with the data for transport resources. This could be configured on a PCIe board and a plug-in slot by using a X8 PCIe hardware configured into two X4 PCIe links, or by using two PCIe slots as high-end graphics cards do. If using a cabled PCIe transport, the possibilities are limited only by your imagination.

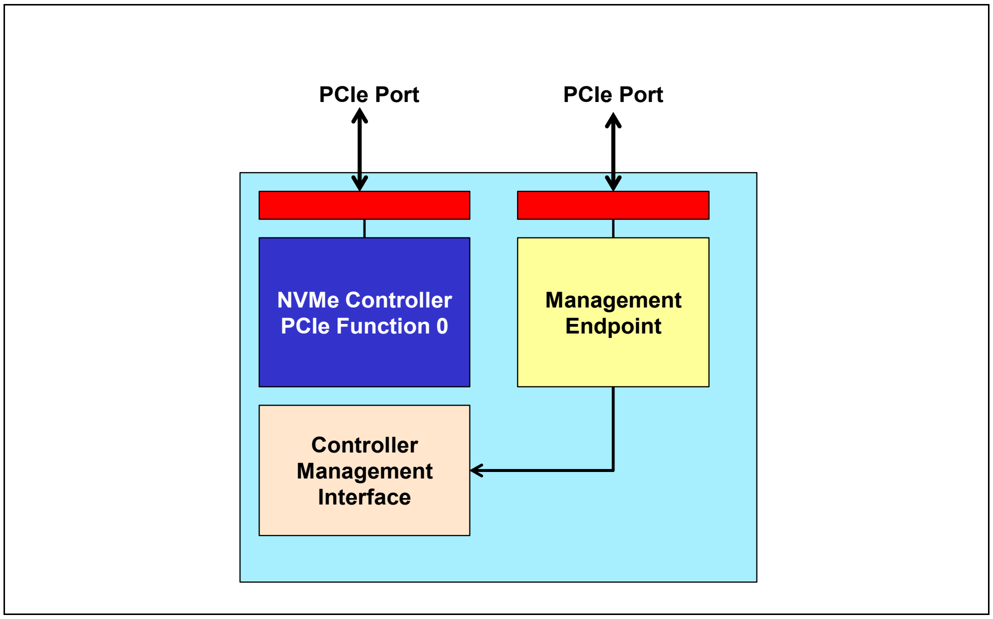

PCIe Port
PCIe Port
NVMe Controller PCIe Function 0
Management Endpoint
Controller Management Interface

Figure 119: Management with 2 Dedicated PCIe Ports

To provide adequate bandwidth, the data port in Figure 119 (above) would probably be multi-lane. The management port could probably be a single lane.

Figure 120 shows an example where the SMBus connects to a Serial EEPROM to read and write information to control the NVMe Controller.

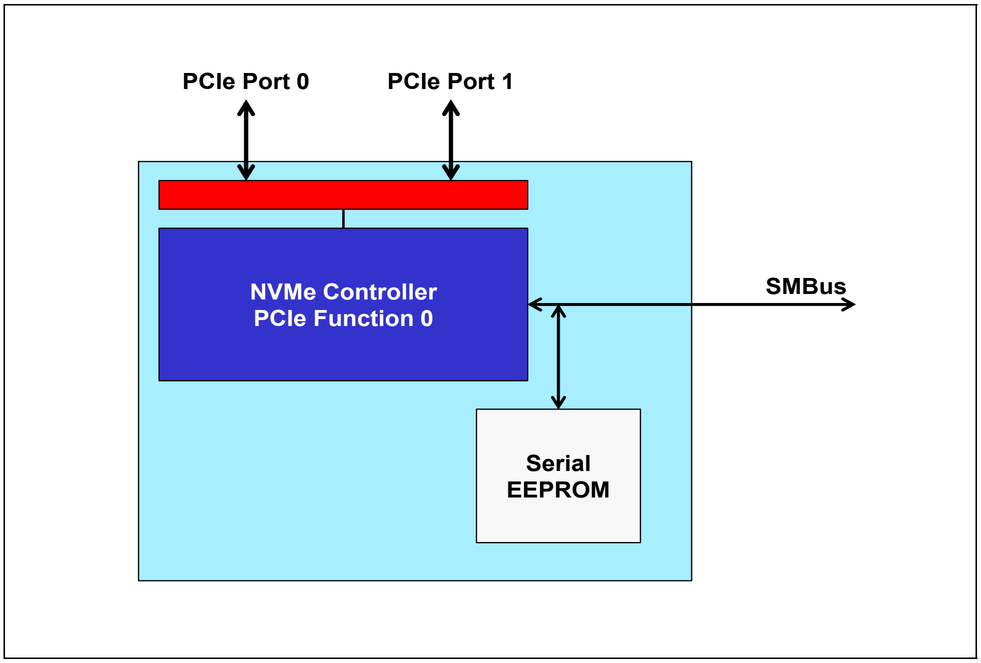


Figure 120: Management with SMBus

Figure 121 shows an SMBus management connected to a multi-function, multi-port NVMe device.

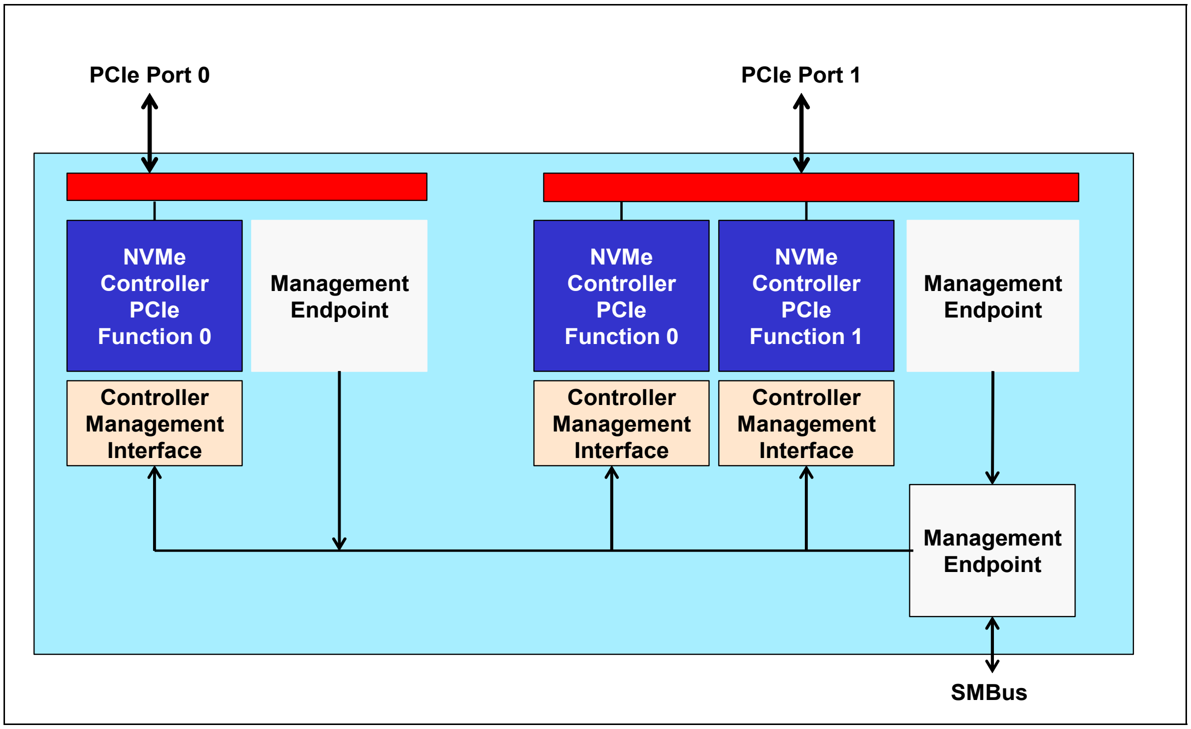

PCIe Port 0
PCIe Port 1
NVMe Controller PCIe Function 0
Management Endpoint
NVMe Controller PCIe Function 0
NVMe Controller PCIe Function 1
Management Endpoint
Controller Management Interface
Controller Management Interface
Controller Management Interface
Management Endpoint
SMBus

Figure 121: Dual Port, Multiple Function, SMBus

NVMe-MI Detail

Please reference Figure 115. Nearly all communication between two devices today, whether using electrical, optical, or wireless transmission media, flows through layers. In the case of NVMe-MI, the application layer creates the Request .

The Request is passed to the Protocol layer, defined in the NVMe-MI specification, where the header and Message Integrity Check (32-bit CRC) are added.

The Request then is passed to the Management Component Transport (MCTP) where another header is added.

It then is passed to the transport, defined in NVMe-MI as either SMBus or PCIe VDM.

The receiving device has equivalent layers which use the header information to help control or route the Request for proper handling. The receiving device uses the Message Integrity Check to verify the Request was received correctly.

Whereas Figure 115 shows the layers, Figure 122 shows the packet that is transported between devices. The application layer creates the Payload and passes it to the Protocol Layer (NVMe-MI) which adds the Protocol Header and Trailer. It then goes to the MCTP layer which adds a MCTP header and passes it to the Physical layer which adds the Physical Header and Trailer.

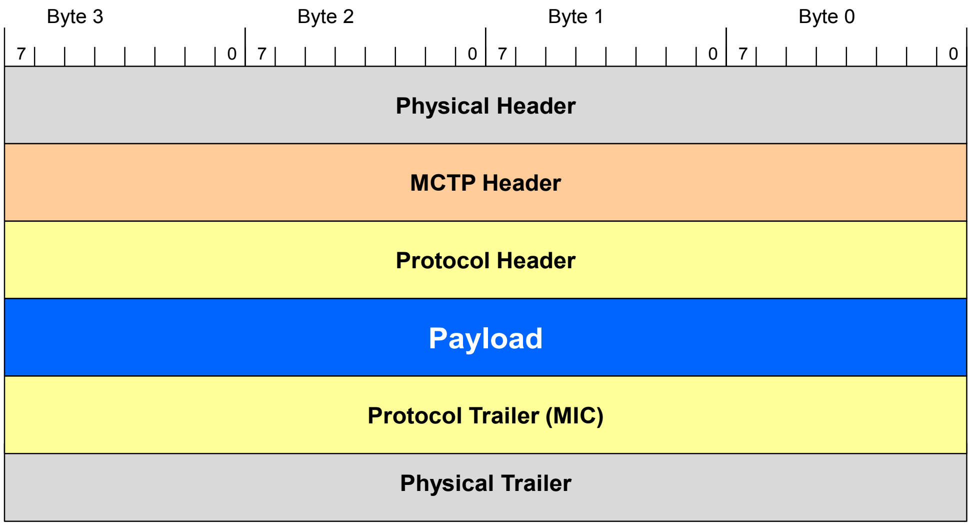


Figure 122: NVMe-MI Packet Overview

We could explorer each of these layers from the transmit side (top down) or from the receive side (bottom up). For this edition of the book, let's look at the contents as seen by the receiving device.

Important note: MCTP specifications use big endian byte ordering, but NVMe specifications use little endian byte ordering. Drawings in this chapter are shown little endian.

Physical

The physical layer:
for initialization:
detects the presence of a connecting device
discovers the communication mode of the device
negotiated speed and link configuration
for operation:
achieves bit, byte, word, and/or dword synchronization
decodes the bit stream into characters usable by the link layer

The physical layers named in the NVMe-MI specification are:
PCIe (see PCIe chapter), and
SMBus (later in this chapter)

MCTP Header

Figure 123shows the MCTP header fields. The bits are defined in the MCTP Base specification, DSP0236. They are sown in the MCTP specification as big endian, but in the NVMe-MI interface and here as little endian byte order.

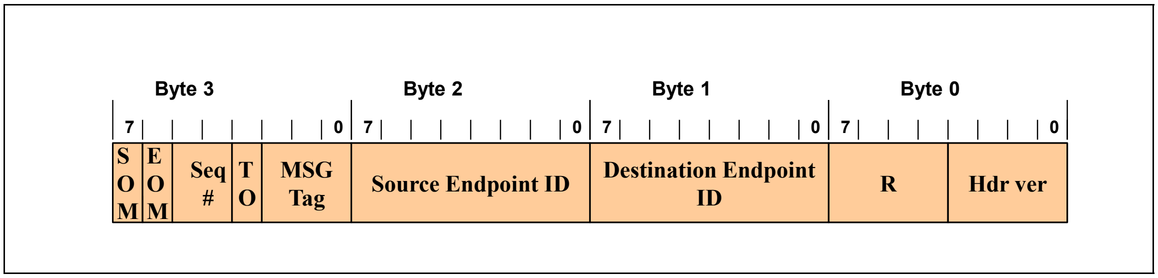


Figure 123: MCTP Header fields

The upper layer message may be divided into multiple MCTP packets. The Start of Message and End of Message bits in this packet tell if this portion of the message is the start of the message, the end of the message, or an intermediate portion of that message. Control Primitive messages must take no more than one MCTP packet, therefore, both SOM and EOM would be set for Control Primitives.

The packet sequence number increments once for each packet for this message, modulo 4. The packet with the SOM bit set may be any value (0-3), but it is recommended that the first packet per message be an increment modulo 4 from the previous packet with an EOM bit set.

TO is tag owner. TO set to 1b means that the source of the message originated the message tag.

MSG Tag with the Source Endpoint ID identifies a unique message at the MCTP transport level. This is equivalent to the SCSI or SATA/NQC command identifier or command tag.

Source Endpoint ID and Destination Endpoint ID fields identify the originator and intended recipient for this MCTP packet.

The Hdr ver field is Header Version. It can be different for each binding. The value is 0001b for both SMBus and PCIe VDM binding (DSP0237 ver 1.0.0 and DSP0238 ver 1.0.2)

NVMe-MI Header and Trailer

As noted earlier, the ULP (NVMe) message may be split into multiple MCTP packets for transmission and then reassembled at the receiver as shown in Figure 124. When doing this, only the first packet will have the message header and only the last packet will have the Message Integrity Check (MIC).

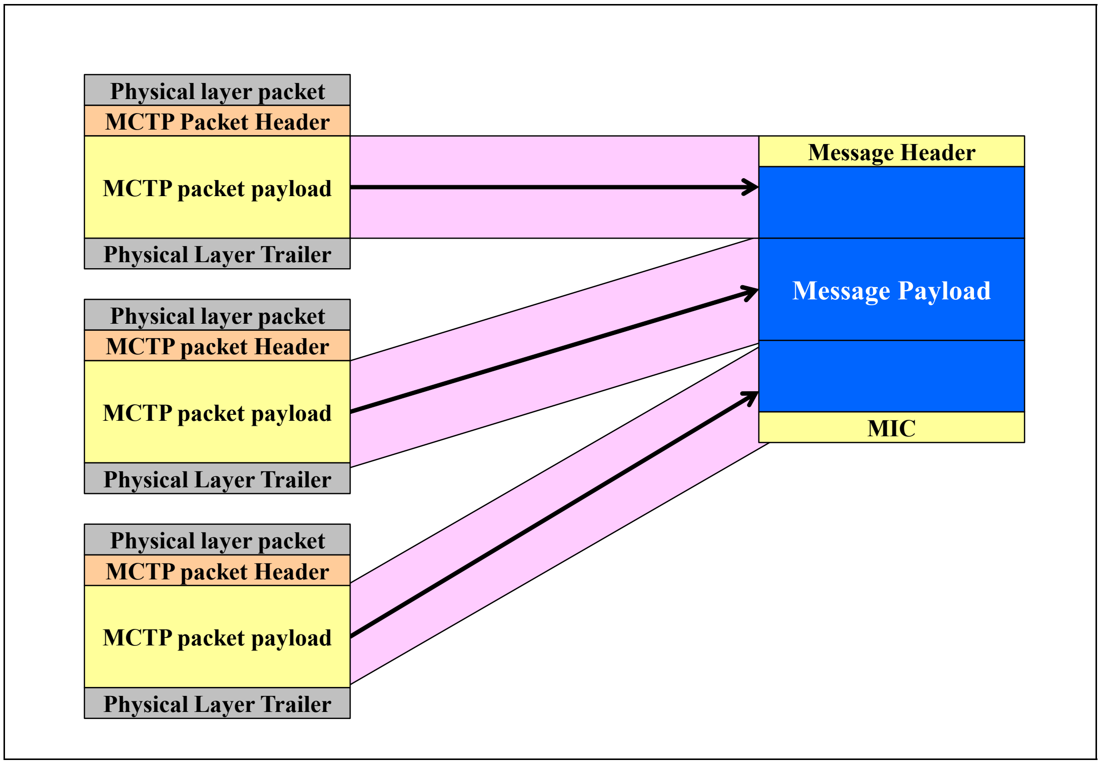


Figure 124: Payload Header defined by NVMe-MI

Messages may be 4224 bytes maximum (4K + 128 bytes). Maximum packet size is determined by the underlying Physical layer.

Figure 125 shows the header for the message. It is defined in NVMe Management Interface specification.

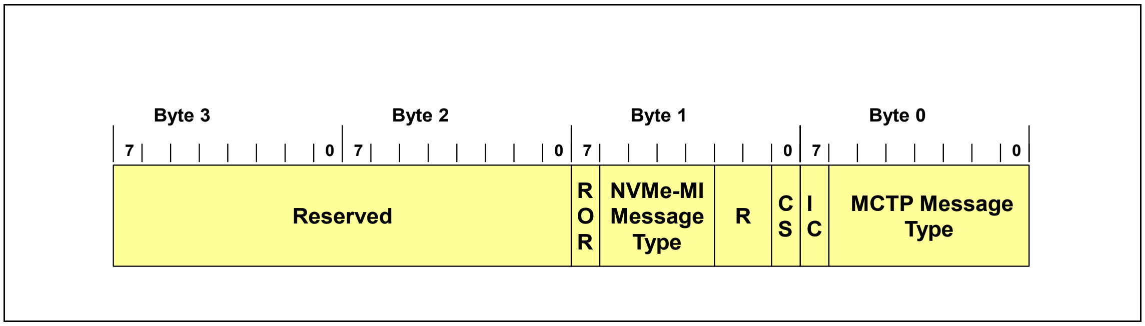


Figure 125: Message Header

The MCTP Message Type is defined by the MCTP Base Specification. It shall be set to 04h in all NVMe-MI messages. See MCTP IDs and Codes, DSP0239.

IC (Integrity Check) specifies if the Message is protected by an Message Integrity Check. This field must be set to 1b.

CS identifies which command slot to use.

See Figure 126 for the NVMe-MI Message Type (the number in the square bracket).

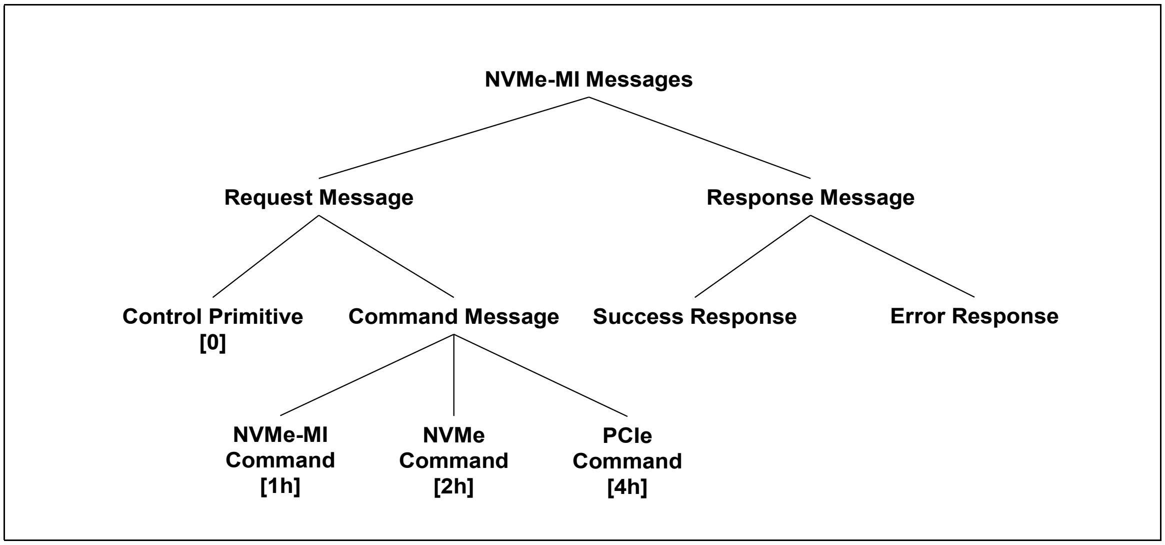


Figure 126: Chart of NVMe-MI Messages

ROR is Request or Response. Request = 0b; Response = 1b.

NVMe-MI Payload

Control Primitive

The NVMe-MI Message Type is set to 0h.

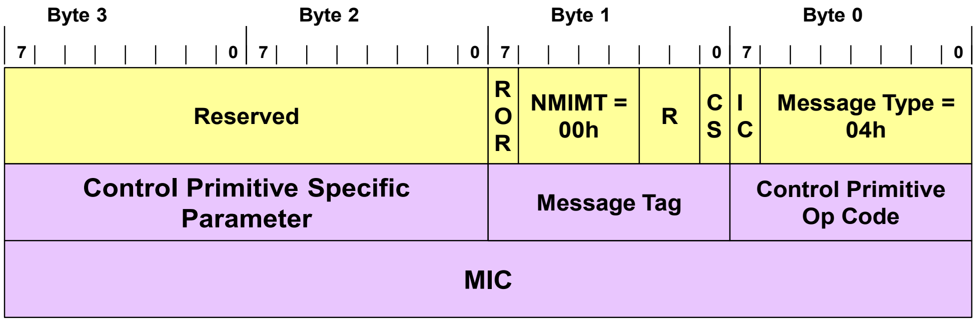


Figure 127: Control Primitive Header and Payload

OpCode	Control Primitive
00h	Pause
01h	Resume
02h	Abort
03h	Get State
04h	Replay

Figure 128: Control Primitive OpCodes

NVMe-MI Commands

The NVMe-MI Message Type is set to 1h.

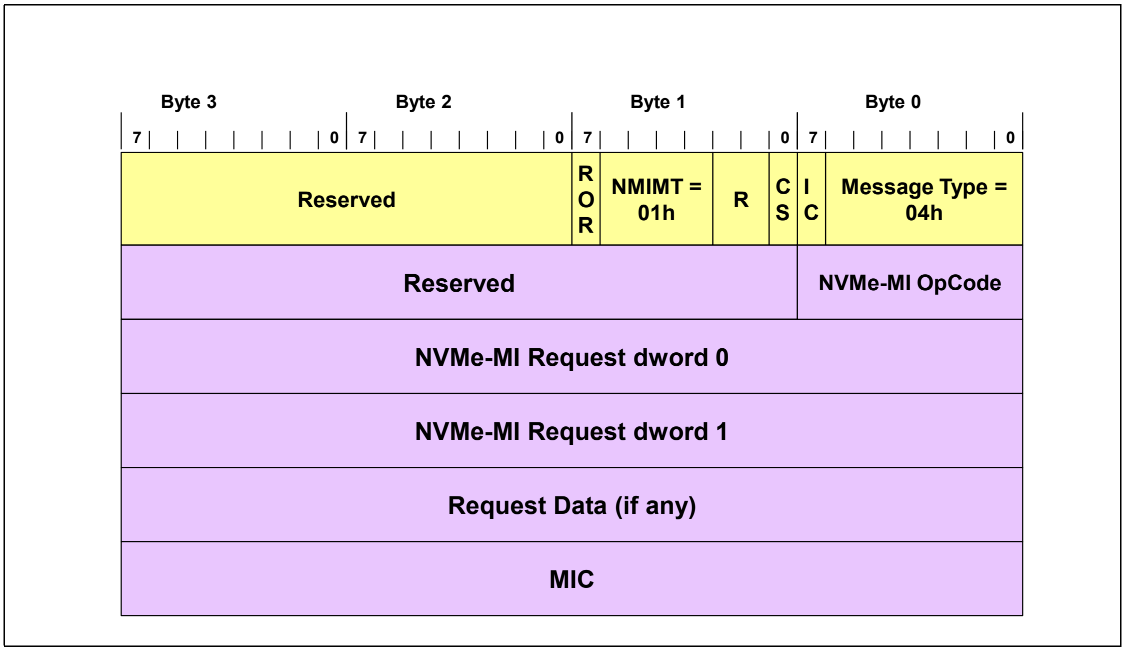


Figure 129: NVMe-MI Header and Payload

OpCode	Command
00h	Read NVMe-MI Data Structure
01h	NVM Subsystem Health Status Poll
02h	Controller Health Status Poll
03h	Configuration Set
04h	Configuration Get
05h	VPD read
06h	VPD Write
07h	Reset
C0-FFh	Vendor Specific

Figure 130: NVMe-MI Command OpCodes

NVMe Admin Commands

The NVMe-MI Message Type is set to 2h.

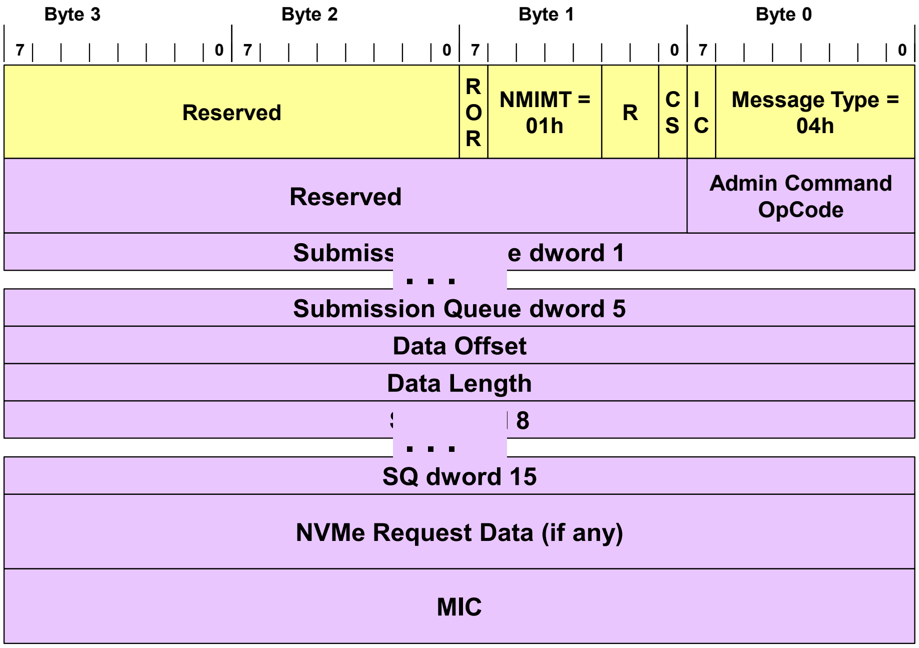


Figure 131: NVMe Admin Command Header and Payload

OpCode	Command
02h	Get Log Page
06h	Identify
09h	Set Features
0Ah	Get Features
0Dh	Namespace Management
10h	Firmware Commit
11h	Firmware Download
15h	Namespace Attach
80h	Format NVM
81h	Security Send
82h	Security Receive

Figure 132: NVMe Admin Commands for NVMe-MI OpCodes

PCIe Transactions

The supported, optional, PCIe Transactions (called PCIe Commands in the NVMe-MI specification) allow for PCIe type access to memory address space, configuration space, and I/O address space. Only addresses mapped to the specified controller may be accessed.

The NVMe-MI Message Type is 4h for PCIe Transactions.

Op Code	PCIe Transaction
00h	PCIe Configuration Read
01h	PCIe Configuration Write
02h	PCIe Memory Read
03h	PCIe Memory Write
04h	PCIe I/O Read
05h	PCIe I/O Write

Table 44: Available PCIe Transaction OpCodes

MCTP

The Management Component Transport Protocol (MCTP) created by the Distributed Management Task Force (DMTF) that defines a communication model to facilitate communication between different intelligent hardware components of today's computer systems. The communication model includes a message format, transport description, message exchange patterns, and configuration and initialization messages. The MCTP Serial Transport Binding Specification (DSP0253) defines how the MCTP base protocol and MCTP control commands are delivered over a physical serial transport type and medium.

The headers of MCTP used in NVMe-MI have been defined above in Figure 123.

NVMe Basic Management Command

The SMBus specification[21] defines the signaling. Its has chapters on:

Physical Layer,
Data Link Layer,
Network Layer
Bus Protocols
Address Resolution
Differences between SMBus and I^{2}C.

The NVMe Technical Note[22] defines the contents of the packet.
The NVMe specification: Basic Management Command has a single chapter on the three defined commands and

[21] Available from http://www.smbus.org/specs/ or www.powerSIG.org.

[22] Available from http://nvmexress.org/wp-content-uploads-NVM_Espress_Manaagment_Interface_1_0_gold.pdf.

the bit definitions for those commands. This specification has been moved into an annex of the NVMe-MI specification.

NVMe Management Functions

There are only three commands defined:

00h – Read the drive status
08h – Read a block of static data
FFh – Reset the Arbitration bit

Also defined is the ability for an I^{2}C read of status and vendor content across SMBus block boundaries.

See Figure 133 below. Each transfer/command begins with the Start indication, followed by the Destination address and then the command. On the first two examples "Read Drive Status" and "Read Static Data" the information: Start, address, and command was all outbound from the host. The return data requested by the host must travel the other direction. To reverse direction, there is another Start indication followed by the Destination address plus the direction bit = 1b. Then the data follows: Length byte, Data, Packet Error Code (PEC) and Stop indication.

The third example does not cause a turn around, so there is only one Start indication.

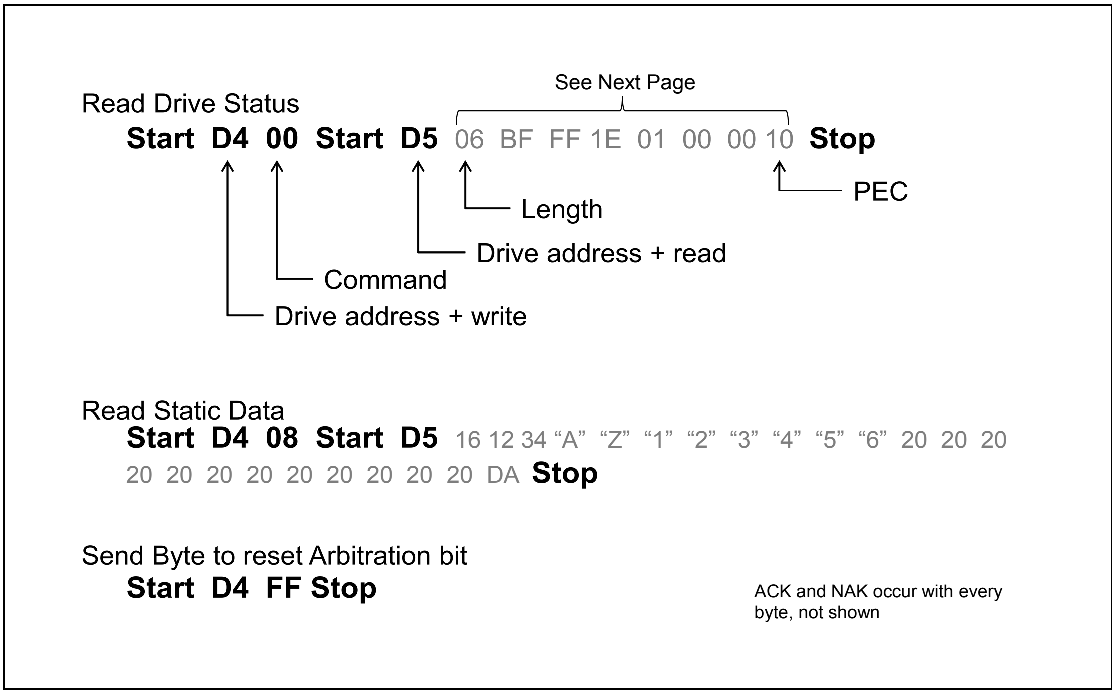
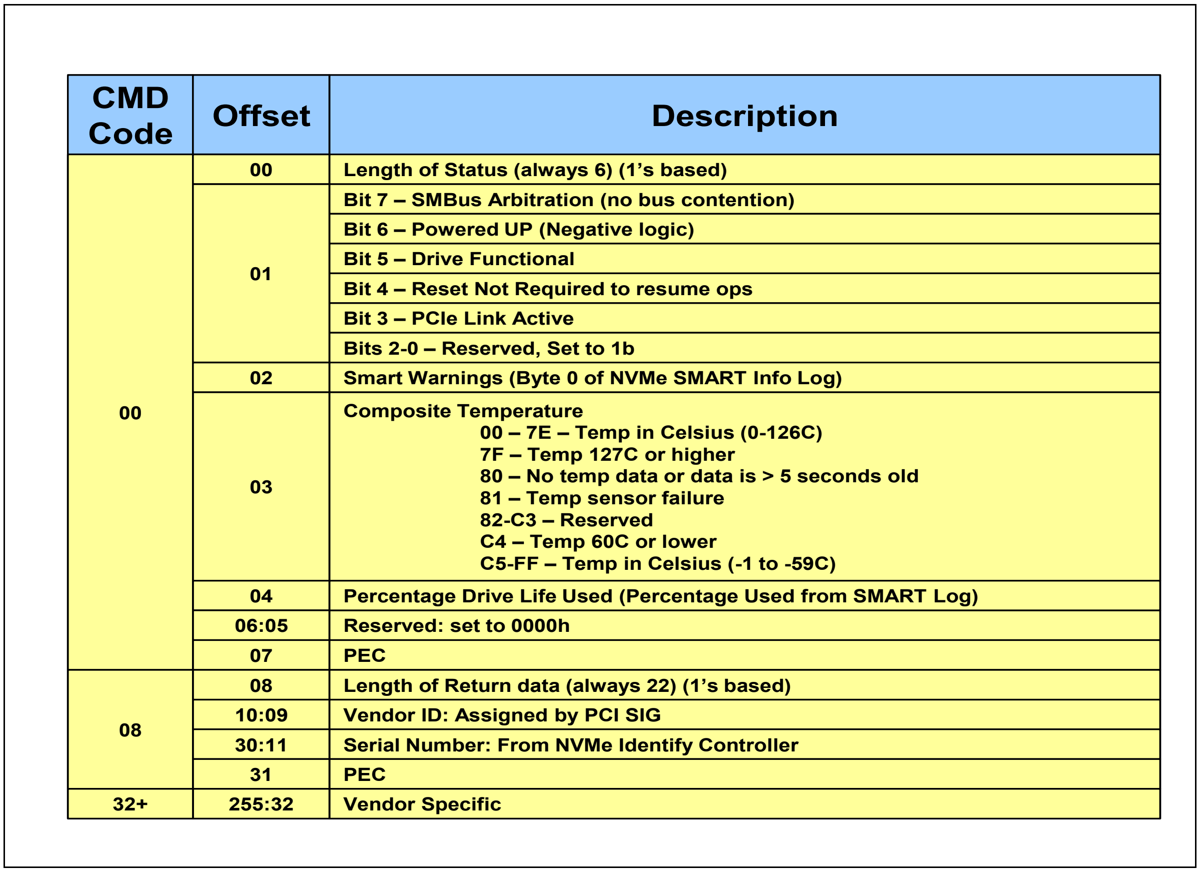

Read Drive Status
See Next Page
Start D4 00 Start D5 06 BF FF 1E 01 00 00 10 Stop
Length
PEC
Drive address + read
Command
Drive address + write
Read Static Data
Start D4 08 Start D5 16 12 34 "A" "Z" "1" "2" "3" "4" "5" "6" 20 20 20
20 20 20 20 20 20 20 20 20 DA Stop
Send Byte to reset Arbitration bit
Start D4 FF Stop
ACK and NAK occur with every byte, not shown

Figure 133: NVMe Basic Management Commands, Examples



CMD Code	Offset	Description
00	00	Length of Status (always 6) (1's based)
	01	Bit 7 – SMBus Arbitration (no bus contention)
		Bit 6 – Powered UP (Negative logic)
		Bit 5 – Drive Functional
		Bit 4 – Reset Not Required to resume ops
		Bit 3 – PCIe Link Active
		Bits 2-0 – Reserved, Set to 1b
	02	Smart Warnings (Byte 0 of NVMe SMART Info Log)
	03	Composite Temperature 00 – 7E – Temp in Celsius (0-126C) 7F – Temp 127C or higher 80 – No temp data or data is > 5 seconds old 81 – Temp sensor failure 82-C3 – Reserved C4 – Temp 60C or lower C5-FF – Temp in Celsius (-1 to -59C)
	04	Percentage Drive Life Used (Percentage Used from SMART Log)
	06:05	Reserved: set to 0000h
	07	PEC
08	08	Length of Return data (always 22) (1's based)
	10:09	Vendor ID: Assigned by PCI SIG
	30:11	Serial Number: From NVMe Identify Controller
	31	PEC
32+	255:32	Vendor Specific

Figure 134: NVMe Basic Management Command Responses

In the above example, the status is:
BFh - Drive had SMBus Arbitration, Drive is Powered Up, Drive is functional, Reset is not required, and PCIe Link is active.
FFh - SMART Warnings.
1Eh - Temperature in Celsius 30°C = 86°F
01h - Percentage of Drive Life Used.
00 00h - Reserved.

The start and stop indications, the PEC, and the clocking of the data are all defined by the SMBus specifications.

SMBus Operations

The two wires of the SMBus are SMBCLK (clock) and SMBDAT (data). The data is placed on the SMBDAT wire when the SMBCLK signal is low and read when the SMBCLK is high.

Each command is started with a Start indication by transitioning the SMBDAT line low while the clock is high. See Figure 135 and Figure 136 for an example.

Following the Start Bit is the destination address. The destination address consists of bits 7:1 which are the address and bit 0 which is the direction for data flow. For example D4 goes to device D4 with a data transfer from the Master[23] to the Slave (Write). Address D5 goes to the same device with a data transfer from the Slave to the Master (Read). As is frequently the case the direction indicated by "0" indicating out from the host and "1" indicating into the host.

The command immediately follows the destination address. If there will be read data (command codes 00h and 08h) the direction for data flow must be reversed which means a new address sent. To do this, the Master issues a new Start (clock high, data line transitions low) and sends the Destination Address with bit 0 set (read).

For write data commands (none defined at this time), the direction does not reverse and the above step is not necessary.

For non-data commands (FFh) following the command is a Stop indication; SMBDAT transitions high while the SMBCLK is high.

To transfer data, the sender (Master on write data, Slave on read data) sends an 8-bit length field (number of bytes following this byte), the data, and the PEC (Packet Error Code byte). The length field does not include itself, nor the PEC.
Following the data and the PEC, the Master signals Stop.

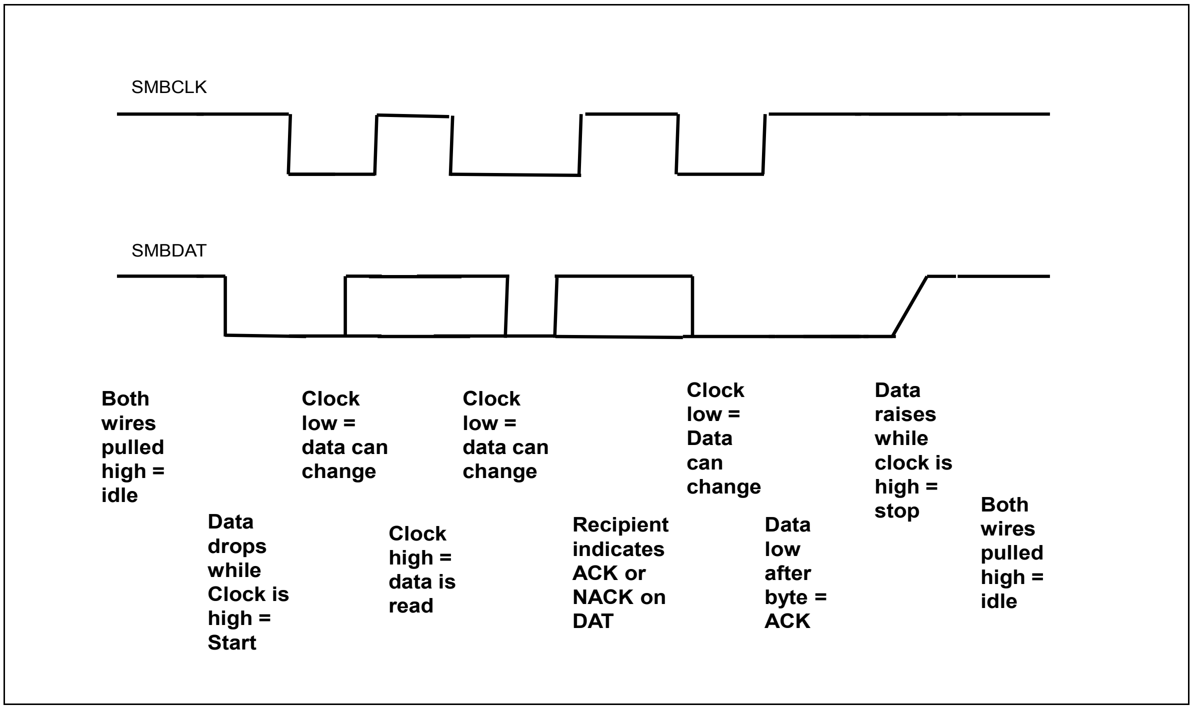

SMBCLK
SMBDAT
Both wires pulled high = idle
Data drops while Clock is high = Start
Clock low = data can change
Clock high = data is read
Clock low = data can change
Recipient indicates ACK or NACK on DAT
Clock low = Data can change
Data low after byte = ACK
Data raises while clock is high = stop
Both wires pulled high = idle

Figure 135: SMBus Signaling

[23] The terms "Master" and "Slave" are terms used in the SMBus Specification and by the NVM Express Technical Note:NVMe Basic Management Command.

ACK/NAK Signaling

See Figure 136 on page 200. Each 8-bit byte (address, command, or data) is followed with a 9th clock for Acknowledgment (ACK) from the receiver; if the receiver received the byte properly, the receiver will signal by causing the SMBDAT line to go low, if the receiver detected an error on the data line, it will signal a Negative Acknowledgment (NAK) by leaving the SMBDAT line high during the 9th clock.

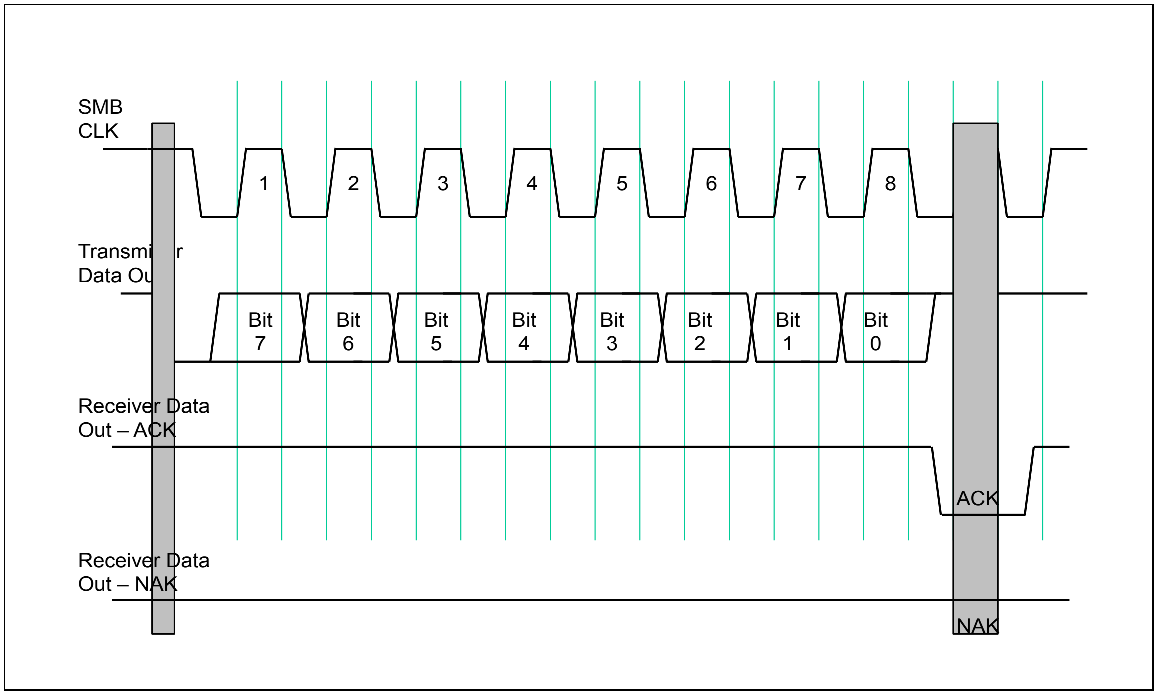


Figure 136: SMBus Ack/NAK Signaling

Chapter Summary

There are two management interfaces defined by the NVMe committee: the NVM Express Management Interface, and the Basic Management Command.

The NVM Express Management Interface allows the host to discover the device’s characteristics and configure the device through an out-of-band interface.

The Basic Management Command allows the host to get certain information from the device through an out-of-band interface.

Part Two: NVMe over Fabrics

NVMe Over Fabrics

Chapter Contents

Overview

What is NVME Over Fabrics?

NVMe over PCIe has become a success, but now customers are asking for:

A. more flexibility,

B. greater number of devices than PCIe can provide,

PCIe is point-to-point, Gen 3 allows only 256 buses/devices per root complex port,

Fibre channel allows 2^{24} devices (16 million); 20,000 being a reasonable upper limit currently,

Ethernet with IPv6 allows an almost unlimited number of devices,

C. greater reach (distance) than PCIe can provide,

PCIe Gen 3 can reach 20 inches per hop, PCIe Gen 4 requires re-timers to reach even 20 inches,

Fibre Channel can reach 50 km[24] per hop,

D. redundancy (multi-port) for path sharing and fail over, and

PCIe has a single upstream port,

FC and SAS were designed from the beginning with at least two upstream ports,

E. ability to use existing, installed fabric such as Ethernet and InfiniBand.

The NVMe committee created a specification, released June, 2016, called NVMe Over Fabrics. It takes the advantage of the efficient NVMe command set and the scalable queue pairs that were created specifically for PCIe and creates a mapping to allow them (command set and queue pairs) to be placed on nearly any transport. The committee refers to the specification as “transport agnostic.”

Part of the NVMe Over Fabrics specification removes PCIe specific operations or features, such as MSI and MSI-X Interrupts. Interrupts will be handled by a transport specific method.

A side benefit of removing the interrupts is that as SSDs move from NAND flash to newer, faster storage media (e.g. spin torque) the time for a read and/or write to complete is less than the time to process an interrupt. Systems are starting to move to polling for completion status, rather than waiting for an interrupt.[25] [26]

Target applications may include:

real-time analytics,
hyper scale,
e-commerce and,
shared Storage.

[24] McData had a director to director communication they advertised as reaching 190 km.

[25] The following two paragraphs are notes that I took during presentations. I did not investigate the specifics, nor can I verify the veracity of what was said, nor of the accuracy of what I wrote.

Some numbers thrown around by experts at the Flash Memory Summit in 2016 were that now we can complete a storage operation at about 20 µ seconds. With the new technologies device time will be under 6 µ seconds. It takes about 1,000 instructions to do a context switch to handle an interrupt. With that, polling will be more efficient that handling interrupts.

At the Flash Memory Summit in 2015, someone gave a talk on Interrupt Aggregation and pointed out that, in their lab, by aggregating interrupts they improved the performance by 90%. This means that by having one system interrupt to handle a group of interrupt events, the system became much more efficient; in other words, the handling of interrupts is becoming a major impact on system performance.

[26] It was reported at FMS 2016, that Linux 4.7 kernel will support polling for storage command completion.

What is NVMe?

NVMe originally meant Non-Volatile Memory over PCIe. With the movement to Fabrics, the acronym was changed to mean Non-Volatile Memory Express. Their web site is, and has been, NVMexpress.org.

NVMe is a very efficient command set designed for SSDs. Whereas SCSI and ATA have over 200 commands with many options each, NVMe has only eleven I/O commands to access customer data - and all eleven are formated similarly. This makes for a very efficient decode mechanism in the logical unit or controller.

NVMe also has a group of Admin commands to configure and manage to the Controller. These are formated similar to the I/O commands. They are used during initialization, configuration, error recovery, and status management.

NVMe defines a scalable series of submission queues to issue commands to the controller, and a series of completion queues for the controller to pass ending status (responses) to the host. All submission queues have 64 bytes per entry (slot) and all completion queues have 16 bytes per entry (slot)[27]. There can be from one to 2^{64} submission queues per controller (PCIe function on PCIe transports) and from one to 2^{64} completion queues per controller. Each queue can have from two to 2^{64} entries.

NVMe defines a set of registers that may be accessed (read and/or write) by the host and accessed by the controller. The method of accessing the registers is transport specific. These registers provide controller **capability** information (read-only from the host), controller **configuration** information (read-write from the host), and controller **status** information (read-only from the host).

What are Fabrics?

Fabrics are defined as, "a group of interconnected switches." Of course, the word "switches" in this case would include router, directors, and perhaps expanders and gateways. Perhaps SCSI would define Fabrics as Service Delivery Subsystems.

The following two pictures show fabrics. In both pictures there are servers on the left, storage on the right, and the fabrics in the center to connect the servers to the storage. These are very simple drawings meant only to show the concept of fabrics. The fabrics can be as simple or as complex as required by the customer and supported by the transport.

[27] The number of bytes per entry is configurable when the queues are built, a feature that may be helpful if alternate command sets are ever defined.

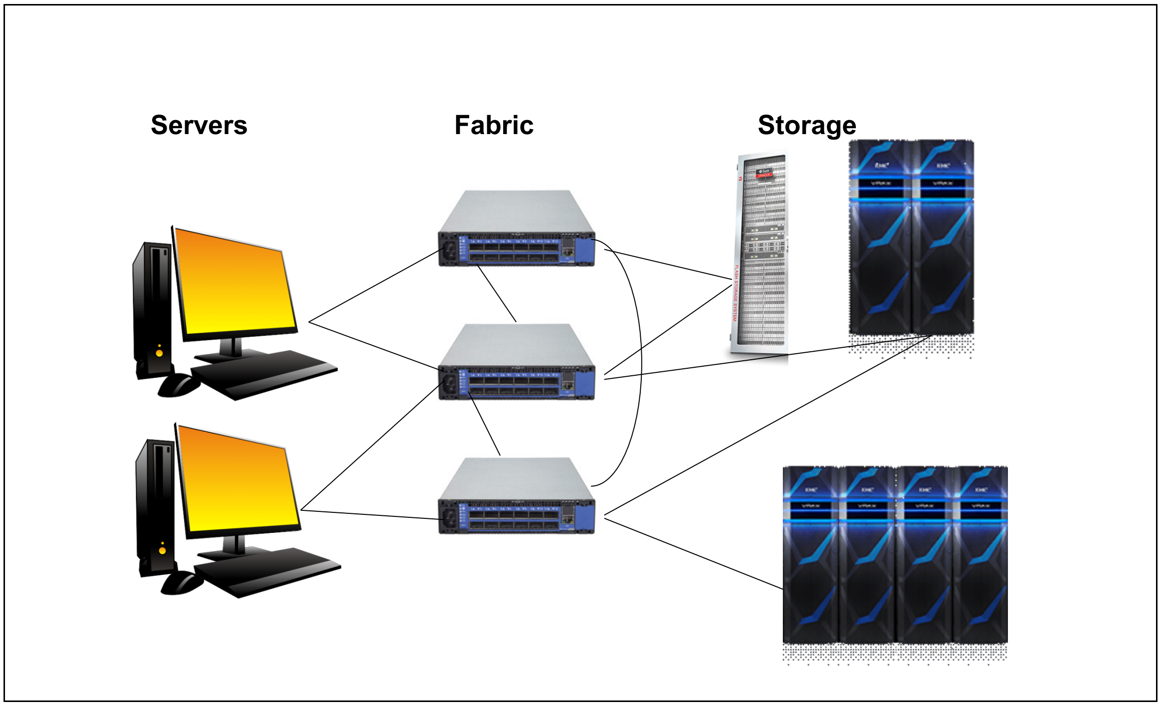


Figure 137: Simple view of Fabrics

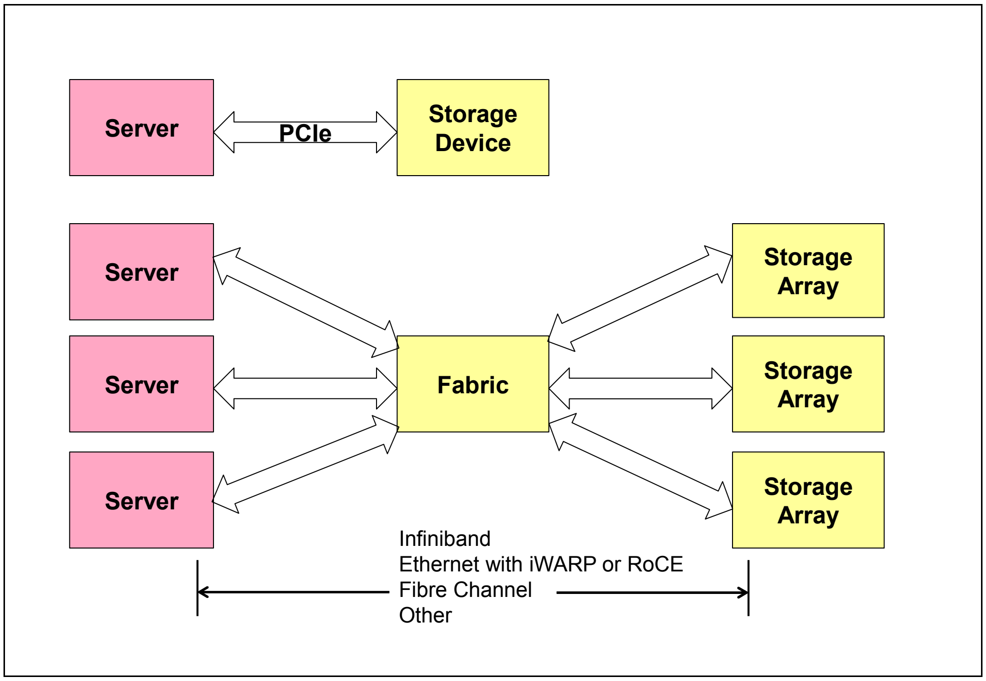


Figure 138: Drawing of Fabrics

Different Views of NVMe over Fabrics

The following drawings are showing different views of NVMe over Fabrics. Each emphasizes a different characteristic.

Figure 139 shows how to connect NVMe software to an NVMe SSD. The NVMe over Fabrics Specification abstracts the NVMe software and Controller, and allows the transport group to define how NVMe over Fabrics will be implemented on that transport. Currently, there are discussions for InfiniBand, two variations for Ethernet, and Fibre Channel. The NVMe Specification and the NVMe over Fabrics Specification should not have to change or adapt for any of the existing transports, nor for future transports.

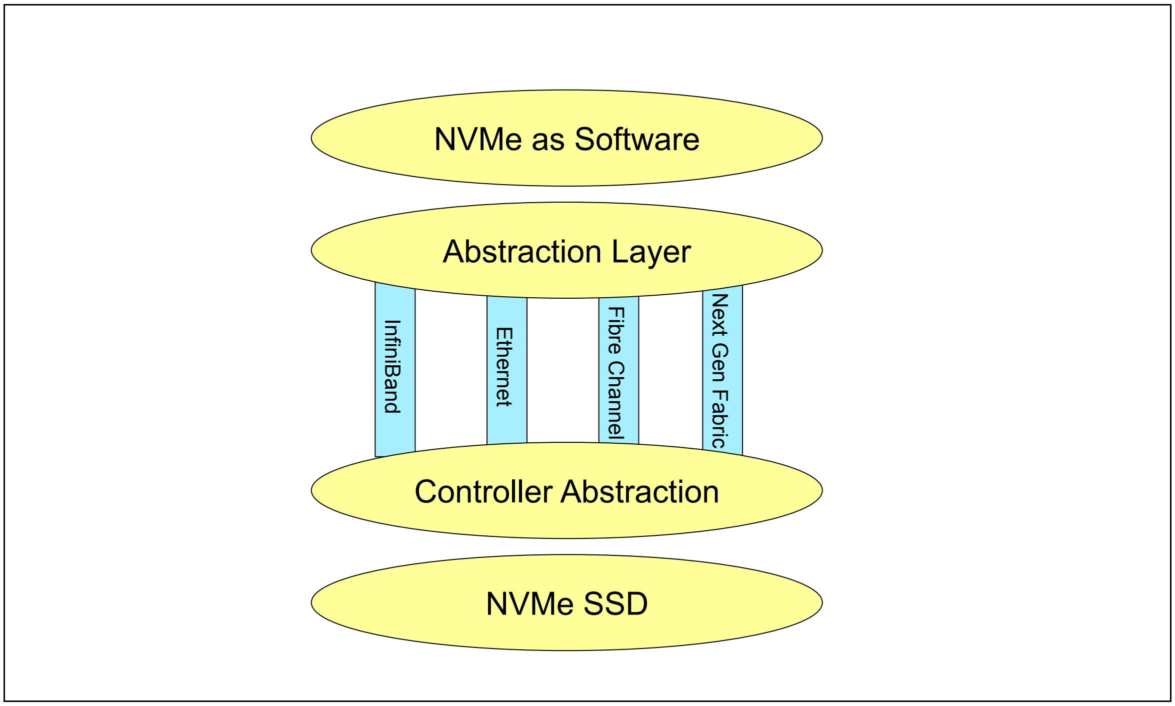


Figure 139: Connecting NVMe as software to NVMe SSD

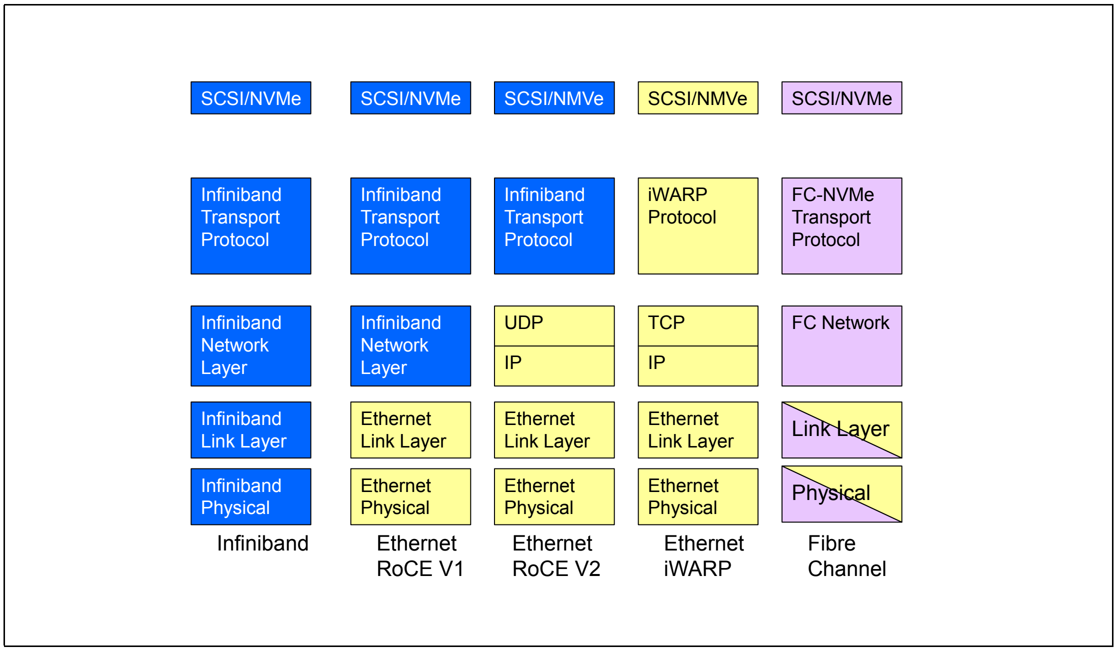
Figure 140 shows a comparison of the Fabrics currently considered. Each of these transports is considered in a separate chapter later in this book. Here is a broad-brush view of each of them.

This chart is drawn to reflect the Open Systems Initiative (OSI) model defined by the International Standards Organization (ISO). The seven layers of that model, starting at the top are:

Application - High-level APIs, including resource sharing,
Presentation - Translating data between network services and the application,
Session - Managing communication sessions such as a continuous exchange of information
Transport - Providing message services and connections
Network - Dealing with how devices on different subnets may address each other,
Data Link - Describing how this device talks to its peer at the other end of the physical link, and
Physical - Describing bits and encoding, connectors and media.

Only the lower four layers are shown on this chart. The upper three layers are outside the scope of the standards defined in this book.

InfiniBand is used primarily for High Performance Computing (HPC) where there are many computers connected as a cluster and together solve a single problem. It is necessary for them to exchange large amounts of data rapidly, hence it defines and uses a Remote Direct Memory Access (RDMA) to directly write the data from the application memory of one computer or storage device to the application memory space of another computer or storage device. InfiniBand traditionally used SCSI commands for storage operations, but now may switch to NVMe. See “Remote Direct Memory Access (RDMA)” on page 227. See “InfiniBand Basics for NVMe ” on page 235.



Infiniband	Ethernet RoCE V1	Ethernet RoCE V2	Ethernet iWARP	Fibre Channel
SCSI/NVMe	SCSI/NVMe	SCSI/NMVe	SCSI/NMVe	SCSI/NVMe
Infiniband Transport Protocol	Infiniband Transport Protocol	Infiniband Transport Protocol	iWARP Protocol	FC-NVMe Transport Protocol
Infiniband Network Layer	Infiniband Network Layer	UDP IP	TCP IP	FC Network
Infiniband Link Layer	Ethernet Link Layer	Ethernet Link Layer	Ethernet Link Layer	Link Layer
Infiniband Physical	Ethernet Physical	Ethernet Physical	Ethernet Physical	Physical

Figure 140: Comparison of Fabrics

RoCE (RDMA over Convergent Ethernet) was created by the InfiniBand Trade Association to use InfiniBand, including its RDMA over Ethernet. RoCE V1 replaces the InfiniBand Link and Physical layers with the Ethernet Link and Physical layers. This allows use of Ethernet for communicating within a single subnet (no routers or gateways). RoCE V2 also replaces the InfiniBand Network Layer with Ethernet’s Network layer and Ethernet’s UDP (User Datagram Protocol - Ethernet layer 4). This allows InfiniBand to be transported across different subnets. Please see the InfiniBand chapter for a more full discussion of RoCE.

iWARP was created by the Internet Engineering Task Force (IETF), the same group that defines Ethernet. iWARP is a group of three specifications that add RDMA to Ethernet. Please see the Ethernet chapter later in this book for a more full discussion of iWARP.

Fibre Channel (FC) is a high speed, long reach transport that is most widely used in Storage Area Networks (SAN). FC does not support RDMA and their implementation of NVMe over Fabrics over Fibre Channel does not add it. FC does have specification that define Fibre Channel over Ethernet layers 1 and 2 as indicated by the yellow and purple boxes above. Please see the Fibre Channel chapter later in this book for a more full discussion.

Figure 141 shows the transport of commands, data and status in each of the fabrics variants.

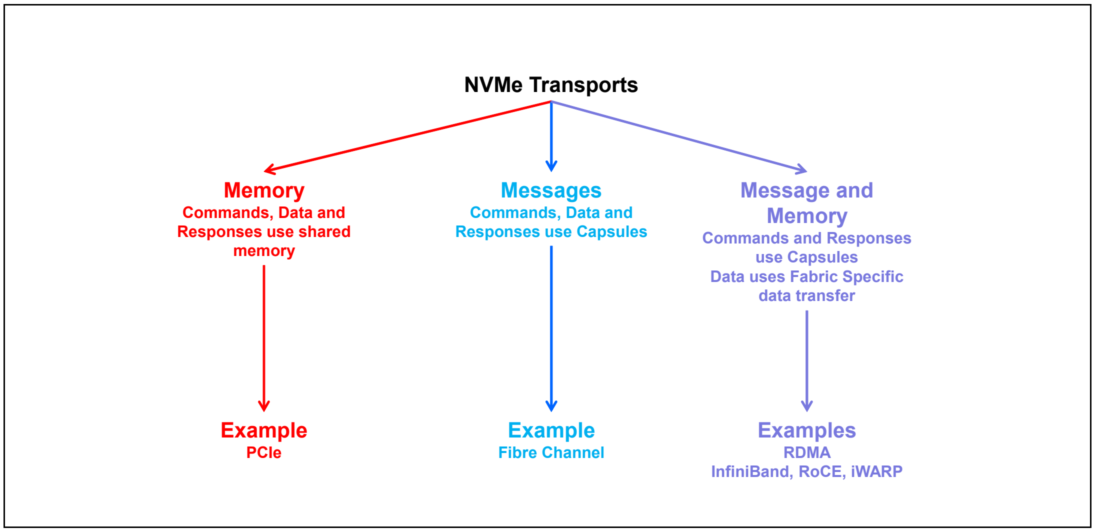

NVMe Transports
Memory
Commands, Data and Responses use shared memory
Messages
Commands, Data and Responses use Capsules
Message and Memory
Commands and Responses use Capsules
Data uses Fabric Specific data transfer
Example
PCIe
Example
Fibre Channel
Examples
RDMA
InfiniBand, RoCE, iWARP

Figure 141: Variation on Transporting Commands, Data and Status

In PCIe, commands, data, and status are transferred using memory reads and memory writes within the shared memory. This is called Memory I/O.

In Fibre Channel, commands, data, and status are transferred using Fibre Channel frames carrying NVMe Over Fabrics Capsules. This is called Channel I/O.

Using InfiniBand, RoCE, and iWARP, commands and status is carried in capsules. Data is transferred using RDMA.

Comparison of NVMe and NVMe Over Fabrics

Similarities

Same command sets
    Admin and I/O command sets
Same queuing mechanism
    Admin and I/O queues
    Up to 64K I/O queues
    Up to 64K entries in I/O queues
    Submission and Completion queues
    Priority associated with Submission queues
Support for multiple namespaces and namespace sharing.
Support for multi-path and dual port on devices.
Support for enterprise capabilities.
Robust error reporting.
Scalable host control interface.

Differences

No defined interrupt mechanism (use transport define mechanism).
Only 1-1 mapping of submission and completion queues.
Create/Delete I/O submission/completion queues commands not supported.
Metadata as a separate buffer is not supported.
PRPs not supported.
Completion Queue flow control not supported.
Doorbell stride has a fixed value of 0h.

Extensions

New Fabric Command set.
Discovery controller.

Encapsulation

Overview of Encapsulation

In the currently discussed Fabrics, the commands and responses (ending status) cannot be conveniently transferred via shared memory writes as in PCIe. For this reason, the commands are put into a capsule with additional information as necessary. The response is placed in a capsule with any additional information. NVMe over Fabrics defines the maximum length of the capsule as “n”, leaving that to the transport for definition.

Command Capsules

Figure 142 shows a command capsule. It contains the NVMe command, complete, unabridged and unaltered plus optionally the SGL or data. Examples could be:

Write command and data with or without Metadata[28], or
Write command and SGL pointers.

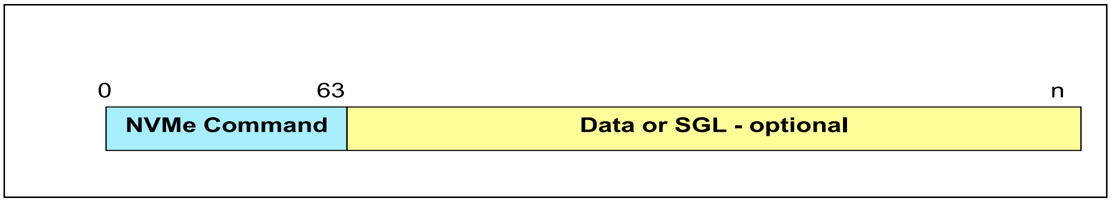


Figure 142: Command Capsule

The maximum capsule size for the Admin Queue commands is 64 bytes. The controller indicates the maximum size for I/O commands in the Identify Controller response (bytes 1795-1792).
The maximum capsule size for Fabric commands is 64 bytes regardless if the command is submitted to the Admin queue or I/O queue.

[28] Separate Metadata is not supported by NVMe over Fabrics. This Metadata must be included as a part of the data.

Status Capsules

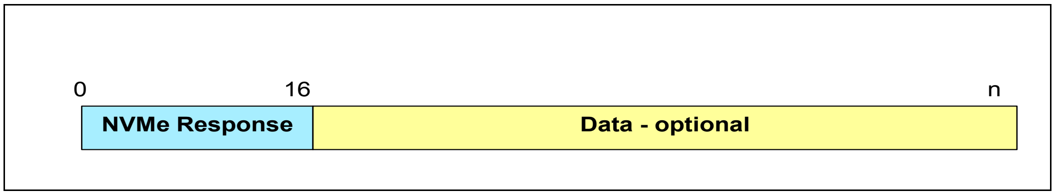


Figure 143: Response Capsule

The maximum capsule size for the Admin Queue response is 16 bytes[29]. The controller indicates the maximum size for I/O responses in the Identify Controller response (bytes 1799-1796).
The maximum capsule size for Fabric command responses is 16 bytes regardless if the command was submitted to the Admin queue or I/O queue.

Encapsulating the Capsules

Figure 144 shows how the capsules are encapsulated within the transport defined frame. This Figure is intended to be generic, each transport defines the various blocks to meet the transport's requirements.

Capsules are transport agnostic and are defined independent of the lower layer transport units, such as frames, messages, or packets. A NVMe capsule may take one or more of the lower layer units.

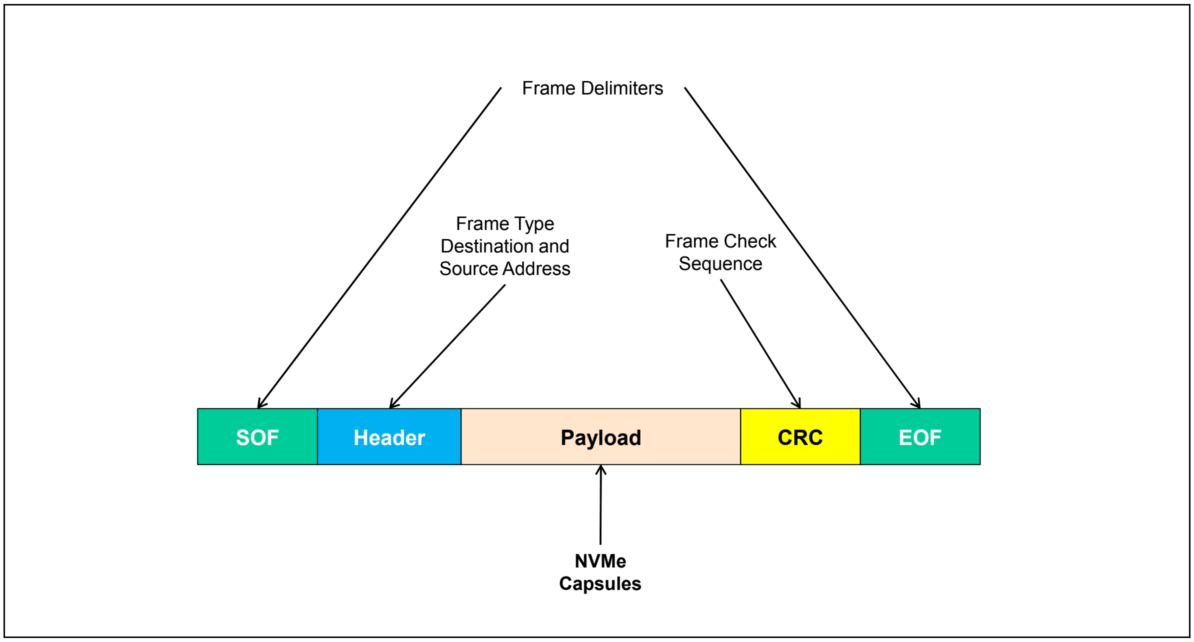


Figure 144: Frame encapsulating NVMe capsules

[29] NVMe over Fabrics 1.0 Gold, section 7.3.2. In-capsule data is not supported for the Admin Queue.

The Frame Delimiters include a Start of Frame (SOF) and an End of Frame (EOF). The SOF must be a recognizable configuration of bits or bytes. The EOF may be a recognizable configuration of bits or bytes, or may be inferred by the Beginning of Frame and a length field.

The Header contains the type of frame contents, the source and destination addresses, and other information determined to be of help to the transport or the receiving device.

The maximum length of the Payload field is defined by the transport. The contents of the Payload field are defined by an upper layer protocol; in the cases of this book, it is the NVMe capsules.

The Frame Check Sequence is usually a 32-bit CRC, but may be of any other check to allow the receiver to determine if the frame was received correctly.

Commands

Admin Command

Table 45 shows all of the defined Admin commands and which are supported by NVMe over Fabrics and which are supported by the Discovery Controller.

Op Code (h)	Name	NVMeOF	Discovery Controller
00	Delete I/O SQ	NS	NS
01	Create I/O SQ	NS	NS
02	Get Log Page	Mandatory	Mandatory
04	Delete I/O CQ	NS	NS
05	Create I/O CQ	NS	NS
06	Identify	Mandatory	Mandatory
08	Abort	Mandatory	NS
09	Set Features	Mandatory	NS
0A	Get Features	Mandatory	NS
0C	Asynch Event Request	Mandatory	NS
0D	Namespace Mgmt	Optional	NS
10	Firmware Commit[a]	Optional	NS
11	Firmware D/L	Optional	NS
15	Namespace Attachment	Optional	NS
18	Keep Alive	Optional	NS
7F	Fabric Commands	Mandatory	Mandatory

Table 45: Admin Commands supported by Fabrics and Discovery Controller

[a] Firmware Commit was named Firmware Activate in NVMe 1.0 and 1.1

NS means Not Supported.

The Fabric Command “Connect” creates the SQ and CQ. The delete of the SQ and CQ is performed by a transport specific manner.

I/O Commands

There are no changes to the I/O commands defined in the NVMe Over Fabrics specification.

Fabric Commands

There are five Fabric Commands defined in the NVMe Over Fabrics 1.0 Gold specification. These commands retain the familiar 64-byte format of the Admin and I/O commands from the base NVMe specification, except:

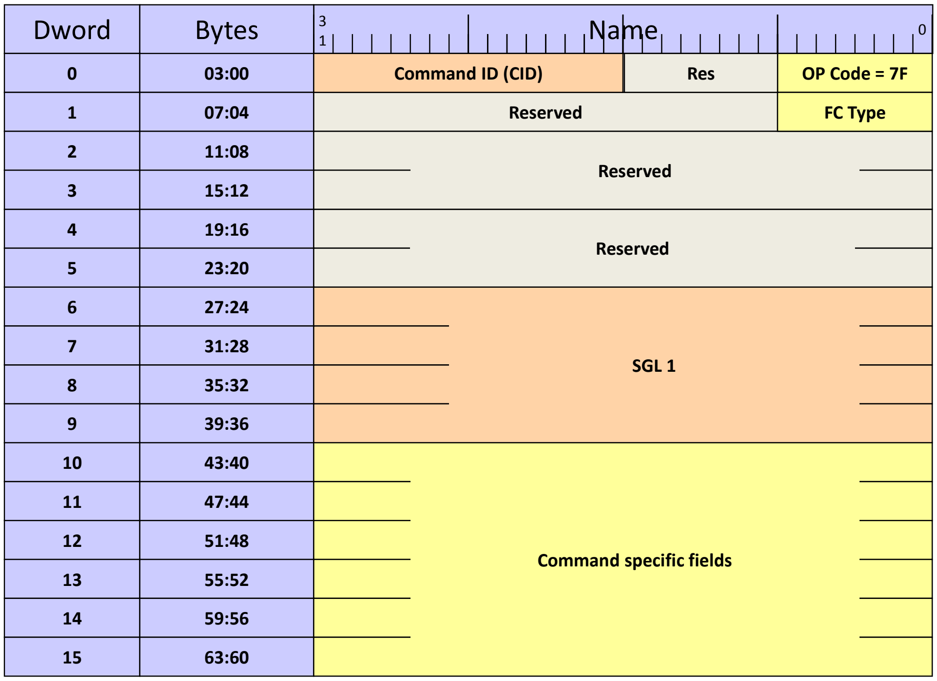
The OpCode for all Fabric commands is 7Fh,
Namespace is a storage device construct, therefore is not used for Fabric Commands
There is a Fabric Command Type (FCTYPE) code in byte 4,
PRP/SGL bits are reserved (PRP is not supported in Fabrics),
Fusing is not supported in Fabric commands, and
Metadata fields are reserved.

Op Code (h)	FC Type (h)	Name	M/O	I/O Queue
7F	04	Property Get	M	No
7F	00	Property Set	M	No
7F	01	Connect	M	Yes
7F	05	Authentication Send	O	Yes
7F	06	Authentication Receive	O	Yes

Table 46: Fabric Commands

All Fabric commands may be submitted to the Admin SQ. The commands marked “Yes” under the I/O Queue in Table 46 may be submitted to the I/O SQ.

Figure 145 shows the generic format of Fabric Commands.



Dword	Bytes	Name		
0	03:00	Command ID (CID)	Res	OP Code = 7F
1	07:04	Reserved		FC Type
2	11:08	Reserved		
3	15:12			
4	19:16	Reserved		
5	23:20			
6	27:24	SGL 1		
7	31:28			
8	35:32			
9	39:36			
10	43:40	Command specific fields		
11	47:44			
12	51:48			
13	55:52			
14	59:56			
15	63:60			

Figure 145: Fabric Command Format

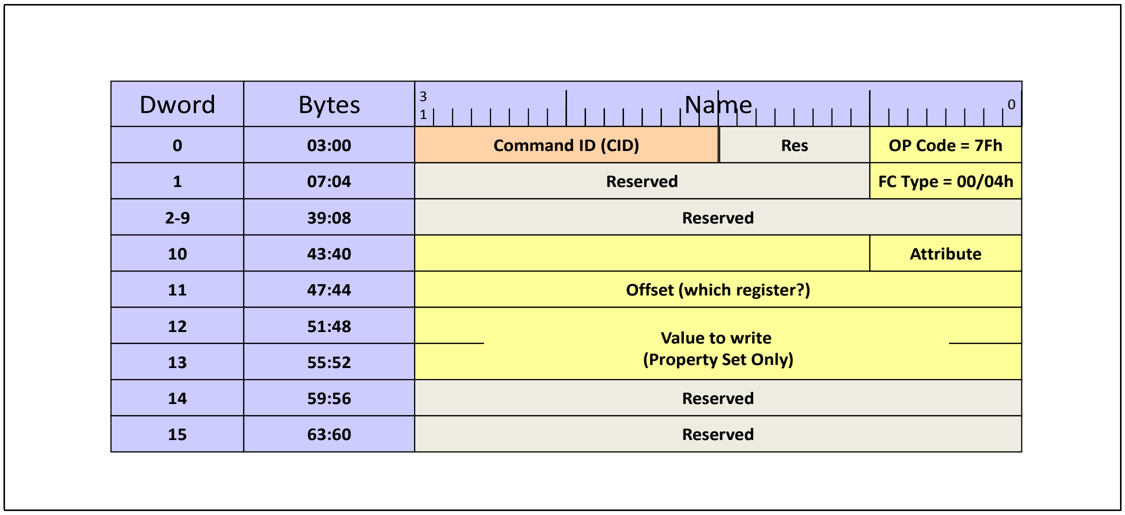
Property Get and Property Set Commands

Both commands are mandatory.

These two commands allow the host to read (get) or write (set) some of the NVMe Registers (the registers beginning at BAR 0). See ": Property Get and Set Registers" on page 218. These two commands must use the Admin Queue.

Command format:

Figure 146 shows the command format for the Property Set and Property Get Fabric Commands.



Dword	Bytes	Name		
		31		0
0	03:00	Command ID (CID)	Res	OP Code = 7Fh
1	07:04	Reserved		FC Type = 00/04h
2-9	39:08	Reserved		
10	43:40			Attribute
11	47:44	Offset (which register?)		
12	51:48	Value to write (Property Set Only)		
13	55:52			
14	59:56	Reserved		
15	63:60	Reserved		

Figure 146: Property Set and Property Get Fabric Command

Byte 40 is the Attribute byte.

Bits 7:3 are reserved

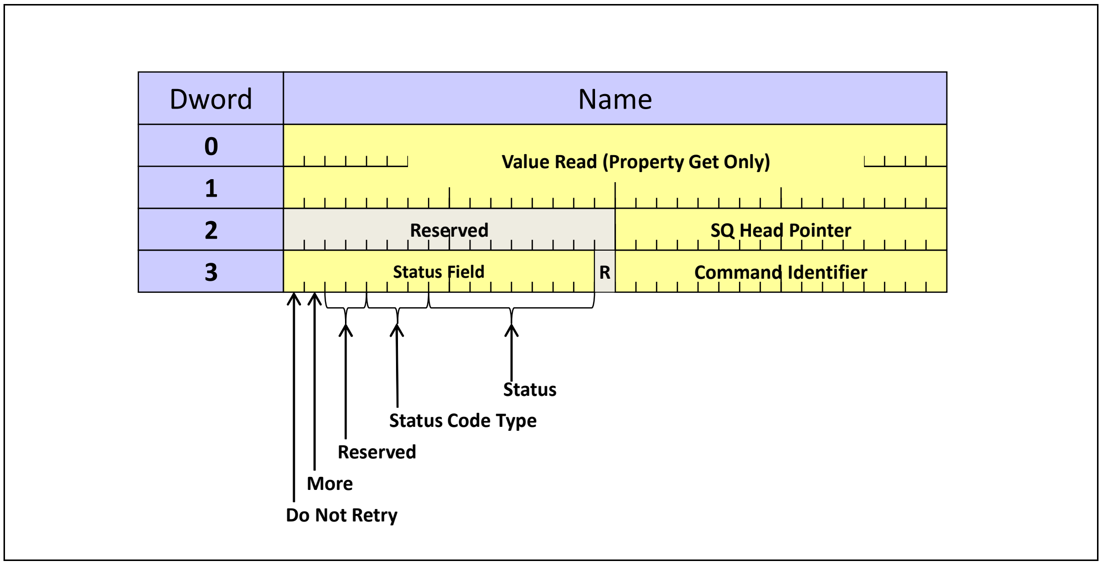
Bits 2:0

000b - 4 bytes

001b - 8 bytes

Response format:

Figure 147 shows the response for Property Get and Property Set Fabric Commands.



Dword	Name		
0	Value Read (Property Get Only)		
1			
2	Reserved		SQ Head Pointer
3	Status Field	R	Command Identifier

Figure 147: Property Set and Property Get Response

Bytes 3:0 - register value for 4 byte register (Property Get only)

Bytes 7:0 - register value for 8 byte register (Property Get only)

The Status Field is defined in NVMe 1.2.1 and later. Specific status codes for individual fabric commands are defined in NVMe over Fabrics 1.0 Gold and later, and may be defined in the transport documents. Most of the codes are listed in the PCIe & NMVe Reference Manual available from KnowledgeTek.

Property Definitions:

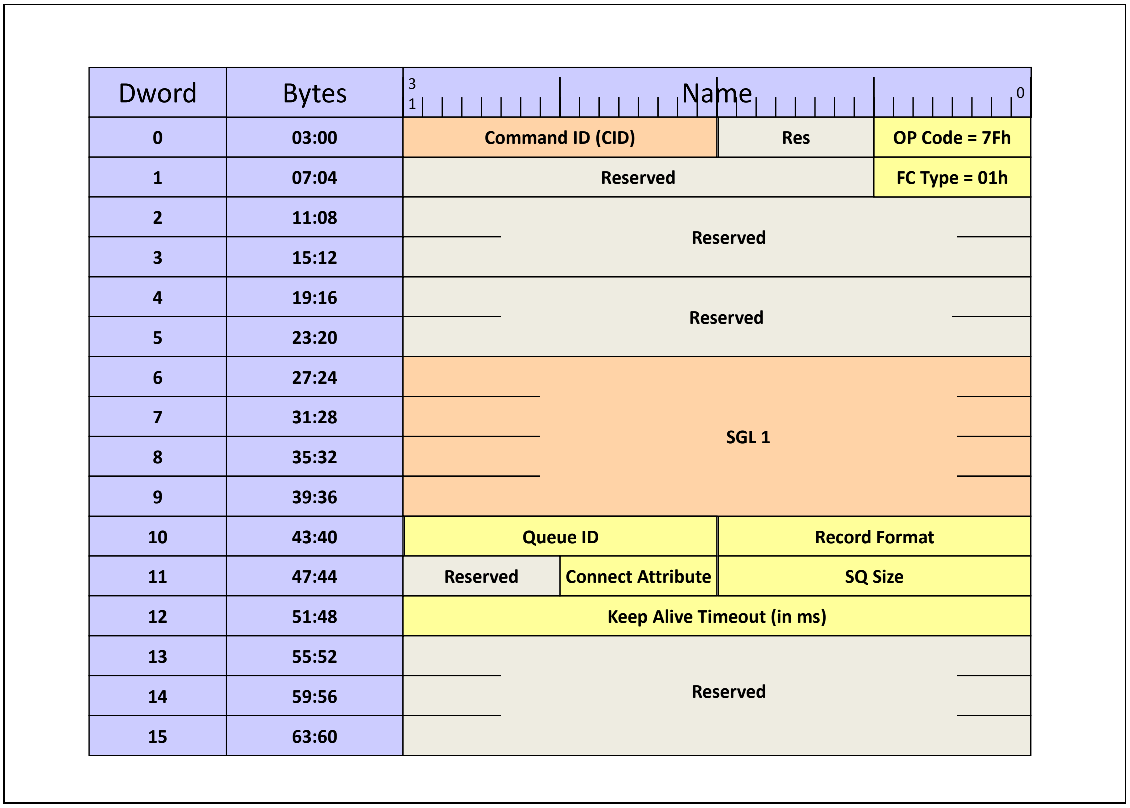
This list of Properties (Registers) accessible with the Property Set and Property Get commands are the same registers as defined in the base NVMe Specification at BAR 0/1. For a detail description of the registers, See ": Controller Registers" on page 41. The Interrupt Mask Set and Interrupt Mask Clear, the Admin Queue, the Controller Memory Buffer registers, and the Doorbells are not available with these two Fabrics commands.

Start (h)	Length	Symbol	Description
00	**8**	**CAP**	**Controller Capabilities**
08	**4**	**VS**	**Version**
0C	4	INTMS	Reserved
10	4	INTMC	Reserved
14	**4**	**CC**	**Controller Configuration**
18	4	Reserved	Reserved
1C	**4**	**CSTS**	**Controller Status**
20	**4**	**NSSR**	**NVM Subsystem Reset (optional)**
24	4	AQA	Reserved
28	8	ASQ	Reserved
30	8	ACQ	Reserved
38	4	CMBLOC	Reserved
3C	4	CMBSZ	Reserved
40		Reserved	Reserved
F00	256	Reserved	Reserved for Command Set
1000	1K	Reserved	Reserved for Fabrics definition

Table 47: Property Get and Set Registers

Connect Command

The Connect Command is used by the host to create a Submission and Completion Queue pair for use by the controller. This command may be submitted to either the Admin Queue or the I/O Queue.



Dword	Bytes	Name 31			0
0	03:00	Command ID (CID)		Res	OP Code = 7Fh
1	07:04	Reserved			FC Type = 01h
2	11:08	Reserved			
3	15:12				
4	19:16	Reserved			
5	23:20				
6	27:24	SGL 1			
7	31:28				
8	35:32				
9	39:36				
10	43:40	Queue ID		Record Format	
11	47:44	Reserved	Connect Attribute	SQ Size	
12	51:48	Keep Alive Timeout (in ms)			
13	55:52	Reserved			
14	59:56				
15	63:60				

Figure 148: Connect Command Format

Queue ID specifies the ID for both the SQ and CQ. SQ Size is the number of entries in the SQ; it is a 0's based number.

Record Format specifies the format of the Connect Command Capsule. Those compliant with NVMe over Fabrics 1.0 Gold specify a Record Format of 00 00h.

Connect attribute is:

Bits 7:2 are reserved
Bits 1:0 are defined in Table 48

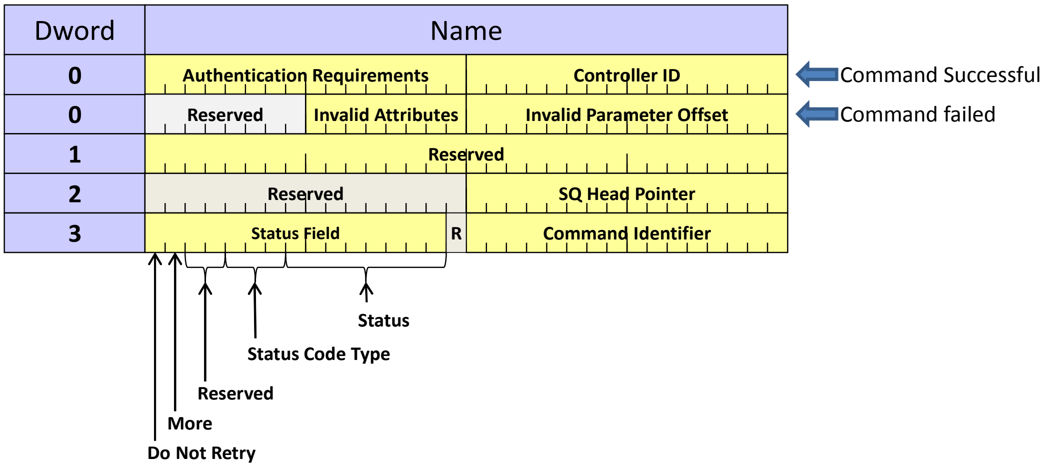
Value	Definition
00b	Urgent
01b	High
10b	Medium
11b	Low

Table 48: Connect Attributes (SQ Priority)

For information on the Keep Alive Timeout see Keep Alive Command and Keep Alive Timer feature, both in Chapter 5.

Connect Response

There are two possible formats for dword 0: one if the command was successful and a different format if the command failed. Figure 149 shows both formats



Dword	Name		
0	Authentication Requirements		Controller ID
0	Reserved	Invalid Attributes	Invalid Parameter Offset
1	Reserved		
2	Reserved		SQ Head Pointer
3	Status Field	R	Command Identifier

Figure 149: Connect Command Response

The Authentication Requirements are shown in Table 49. These values are bit-mapped meaning that each

Bits	Meaning
15:1	Reserved
0	TCG Security Protocols

Table 49: Authentication Requirements

different bit represents a different protocol. All the bits that are set are candidate protocols to be used.

The Invalid Parameter Offset specifies how many bytes the invalid parameter is from the start of SQE or the Start of data. The Invalid Attributes are shown in Table 50.

Bits	Meaning
7:1	Reserved
0	0 - invalid parameter specified from start of SQE
	1 invalid parameter specified from start of data

Table 50: Invalid Attributes

The Controller ID specifies the ID allocated to the host, as assigned by the controller. There is no need for the SQ ID, in NVMe over Fabrics, there is a 1-to-1 relationship between Completion Queue and Submission Queue.

For definition of other fields see Figure 37 on page 65.

Authentication Send and Authentication Receive Commands

The chapter on Authentication in the NVMe over Fabrics 1.0 Gold is about 1.5 pages long. It gives a quick description of the two types of Authentication and then references other documents. Either authentication type, both or neither may be required. There is no interaction between the two types.

Fabric Secure Channel

Fabrics Discovery Log Page Entry (Transport Requirements field) specify if Fabric Secure Channel must be used or not. See Figure 152. If Fabric Secure Channel is required by the controller, and it has not been established, the NVM subsystem shall not allow capsules to be transferred.

Fabric Secure Channel will be established and executed by a means specified by the indicated protocol, for instance IPsec.

NVMe In-Band Authentication

NVMe In-Band Authentication is performed after the Connect commands successfully completes. It uses the Authentication Send and Receiver commands. Both commands are optional.

The response to the Connect command indicates if NVMe In-Band Authentication is required, and if so, which protocols. See Figure 149. If any bit in the Authentication Requirements is set to 1b, the controller requires that the host authenticate on each queue via that security protocol. Authentication will probably use NQN.

The Authentication Send Fabric command (FCTPYE = 05h) and the Authentication Receive Fabric command (FCTYPE = 06h) use the Security Protocol as defined in SPC-4. Please refer to either SPC-4 at www.t10.org at http://www.t10.org/cgi-bin/ac.pl?t=f&f=spc4r37.pdf or Security Features for SCSI Commands (SFSC) at http://www.t10.org/cgi-bin/ac.pl?t=f&f=sfscr02.pdf.

Discovery Service

Overview

Discovery service is an abstraction in NVMe over Fabrics that defines how the hosts/initiators find the devices/targets they need for communication. It will run and be used during initialization and also when any device is added or removed from the fabric.

One side of this Discovery Controller is a target function that responds to queries from the NVMe host. This side of the Discovery Controller will be referred to as the “host side” in this text. The other side of the Discovery Controller will connect to and query the fabric using the protocol for that fabric. This will be referred as the “fabric side” and is defined in the protocol documents of the fabric in use. How the fabric side works is outside the scope of the NVMe specifications.

Figure 150 shows a topology with three Discovery Controllers, one in each storage cabinet. It is envisioned that each Discovery Controller will discover the drives (physical or logical) in the cabinet and report those drives, when queried, to the host.

Each host will query one of the controllers with Get Log Page. How the host finds the controller is outside the specification. It could be:

Hosts Configuration file,

OS property,
Hypervisor, or
some other mechanism.

That Discovery Controller will report the storage devices that it knows of and the address of the next controller.

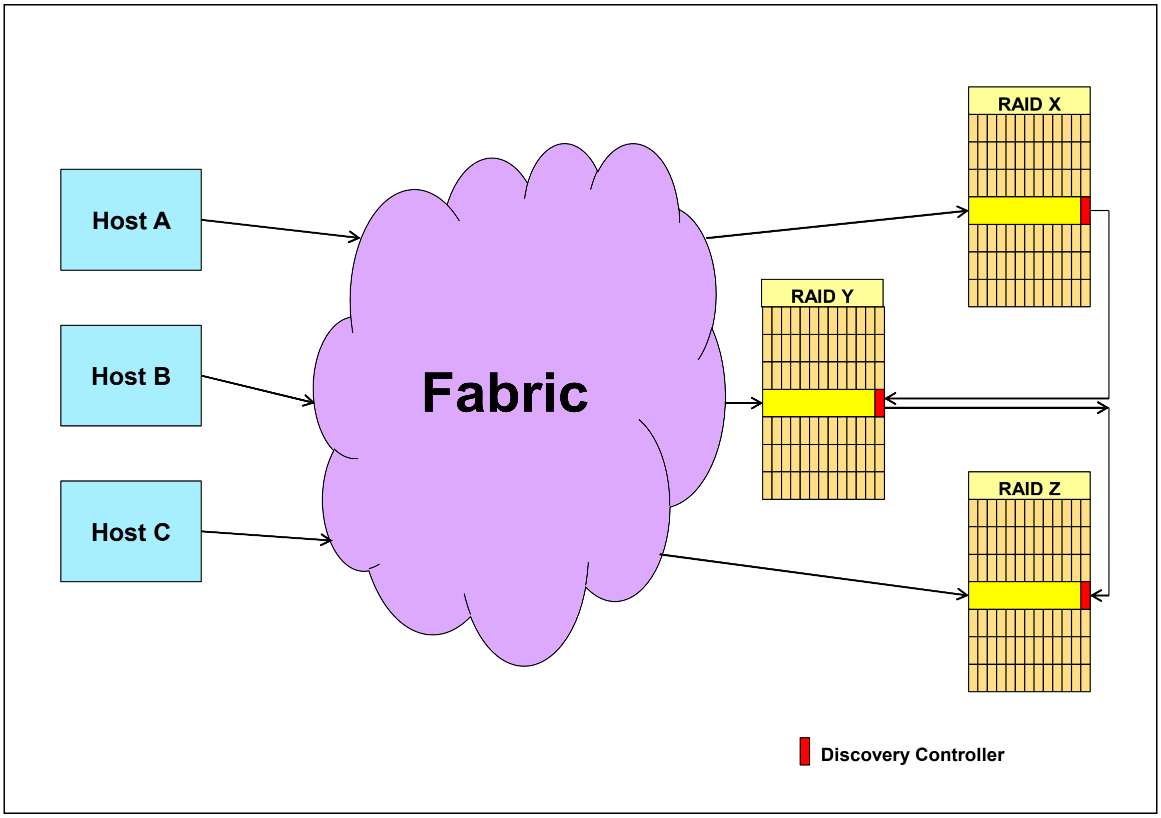


Figure 150: Discovery Controller in the Fabric

Discovery Controller Details

Discovery Procedure

Host finds initial Discovery Controller via a procedure outside the NVMe specifications. In this example, we will assume the Discovery Controller in RAID X is the initial Discovery Controller.

Host does transport connect to NVMe Subsystem containing the Discovery Controller.

Host issues Connect Fabric Command to Admin Queues in Discovery Controller. It uses the well-known Network Qualified Name (NQN) of “nqn.2014-08.org.nvmexpress.discovery.”

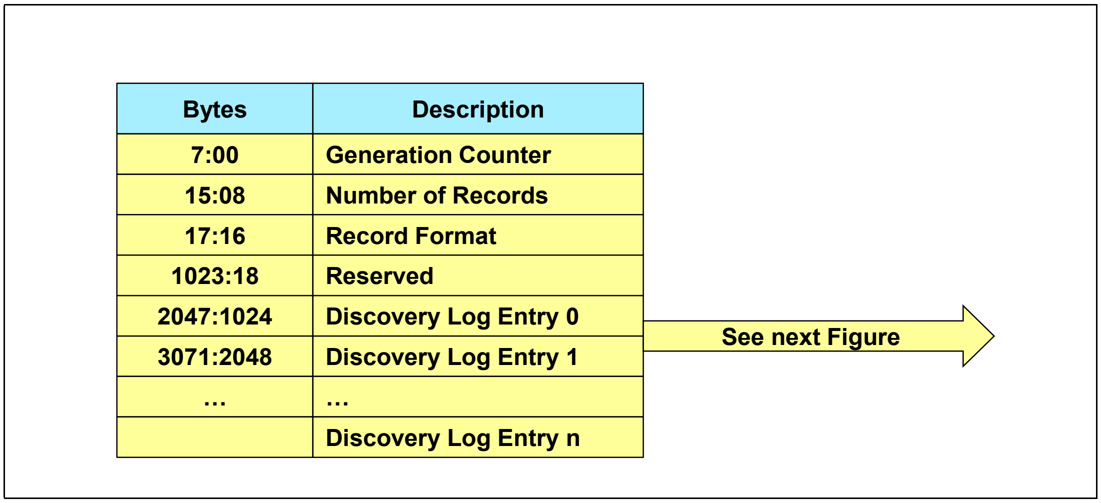
Host issues Identify Controller NVMe command.

Host issues NVMe Get Log Page 70h command.

Discovery Controller returns log pages

If a log page entry indicates a link to another Discovery Controller, perform the above with the new Discovery Controller.

Figure 151 shows the layout of the log page. The total length of this log page is transport dependent.



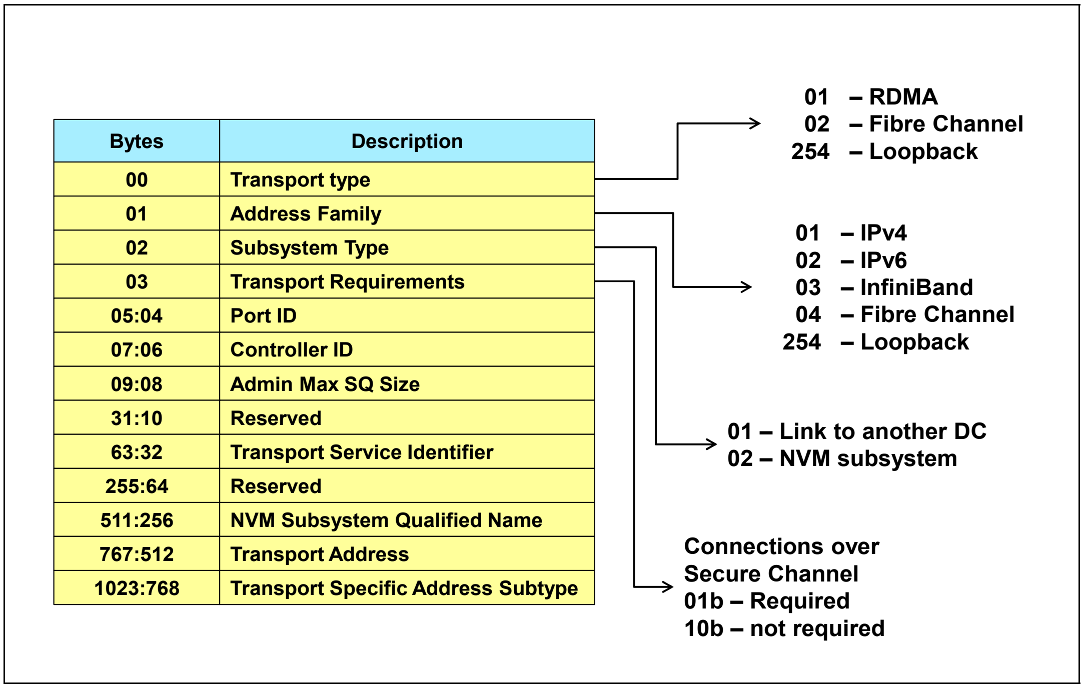
Bytes	Description
7:00	Generation Counter
15:08	Number of Records
17:16	Record Format
1023:18	Reserved
2047:1024	Discovery Log Entry 0
3071:2048	Discovery Log Entry 1
…	…
	Discovery Log Entry n

See next Figure

Figure 151: Discovery Log Page = 70h

The first eight bytes are a generation counter. It gets updated each time any of the log page entries gets changed. The host will read this generation counter on its first read, then read the log page entries and finally go back and read the generation counter. If its value is the same as the first read, the host knows that no changes were made to the information during its read(s).

Figure 152 shows the entry describing each device.



Bytes	Description
00	Transport type
01	Address Family
02	Subsystem Type
03	Transport Requirements
05:04	Port ID
07:06	Controller ID
09:08	Admin Max SQ Size
31:10	Reserved
63:32	Transport Service Identifier
255:64	Reserved
511:256	NVM Subsystem Qualified Name
767:512	Transport Address
1023:768	Transport Specific Address Subtype

01 – RDMA
02 – Fibre Channel
254 – Loopback

01 – IPv4
02 – IPv6
03 – InfiniBand
04 – Fibre Channel
254 – Loopback

01 – Link to another DC
02 – NVM subsystem

Connections over
Secure Channel
01b – Required
10b – not required

Figure 152: Discovery Log Entry

Note that there are 1K-bytes for the header of the log page and 1K-bytes for each entry. Fibre Channel has a payload length of 2K-bytes so the first FC frame could only have one log page entry and each subsequent FC frame could have two log entries.

Chapter Summary

- Chapter Contents
  - What is NVME Over Fabrics?
  - What is NVMe?
  - What are Fabrics?
  - Different Views of NVMe over Fabrics
- Comparison of NVMe and NVMe Over Fabrics
  - Similarities
  - Differences
  - Extensions
- Encapsulation
  - Overview of Encapsulation
  - Command Capsules
  - Status Capsules
  - Encapsulating the Capsules
- Commands
  - Admin Command
  - I/O Commands
  - Fabric Commands
    - Property Get and Property Set Commands
    - Connect Command
    - Authentication Send and Authentication Receive Commands
- Discovery Service
  - Overview
  - Discovery Controller Details

Remote Direct Memory Access (RDMA)

Chapter Contents

Overview of RDMA

Remote Direct Memory Access basically means to directly access the memory of one device from another device to read from or write to that memory. NVMe over PCIe was designed to do this from the beginning. The NVMe command actually transfers the memory address from the host to the controller via the PRP or SGL in the command.

This means that actual memory addresses will be transmitted outside of the device, leading to security concerns. These concerns are not addressed by the RDMA specifications, being left up to system implementers to use the proper security methods required by their operation. Certain organizations may require multiple layers of strict security (e.g. military, National Security Agency, Internal Revenue Service) while others are adequately protected by a locked door and passwords (e.g. small businesses, homeowners).

Internet Engineering Task Force (IETF) defines RDMA as including the above definition plus several other facilities to make it very functional. These are:

OS bypass - applications send commands, including memory addresses directly to the end device, bypassing the Operating System and its checks, services, and overhead.

Zero copy - data flows directly from the application memory of the sending device to the application memory of the receiving device, eliminating the copy of the data from HBA buffer to Operating System buffer to Application buffer on each end.

CPU offloading - (in the case of Internet, this is called a TOE for TCP/IP Offload Engine) OSI layers 1-4 are handled in hardware instead of software stacks making the processing much faster.

OS Bypass

Most Operating Systems (OS) provide a lot of services to application programs and users, and protection to other programs, the computer, data at rest on the devices, and itself. However, to do this, it adds substantial overhead with translate into delay[30]. It the application program can be trusted to create the necessary commands properly and not interfere with other programs devices and data, it would make the handling of data much faster. Please see Figure 153 for a normal I/O stack using OS services and then Figure 154 for an example of OS Bypass.

[30] It can take up to 1000 instructions for the context switch (switching from application mode to supervisor mode, or back. Once the context switch is made to supervisor mode, then the device driver software much create the commands and the buffers, and the HBA driver software must setup the HBA to communicate the commands, transfer the data, and receive and handle the status.

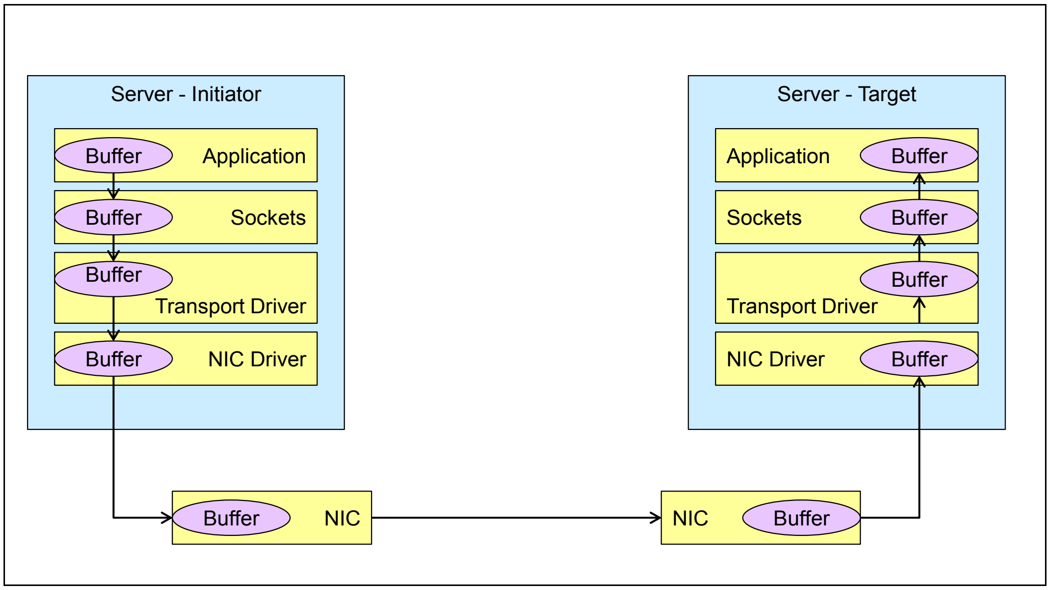


Figure 153: Without OS Bypass

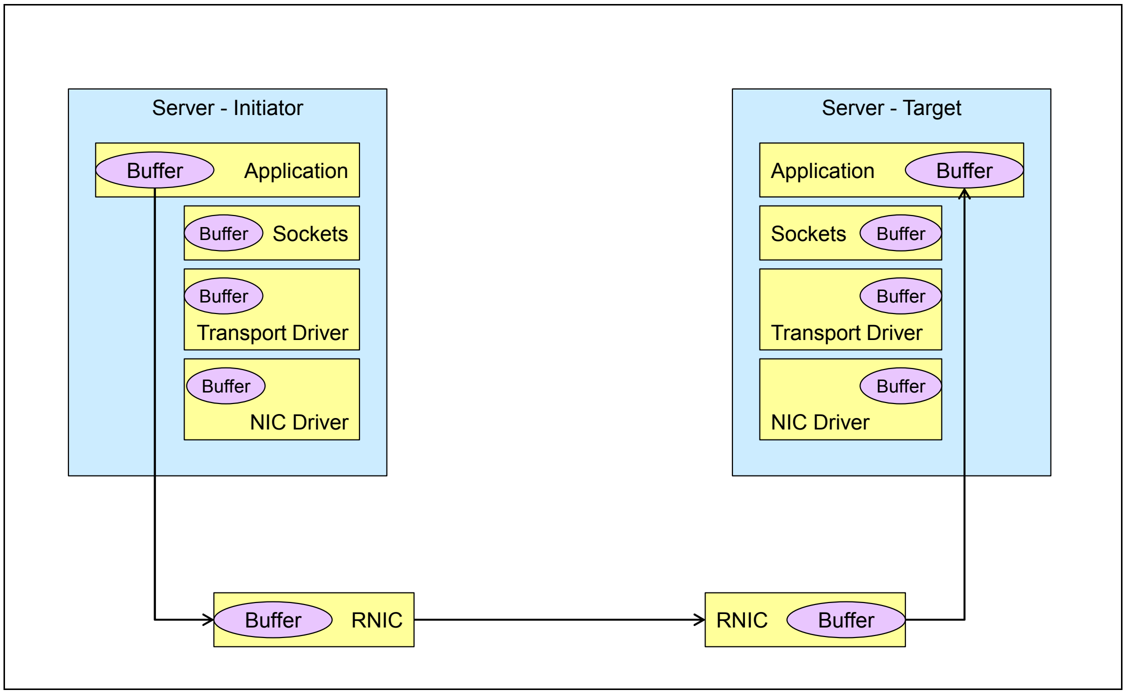


Figure 154: With OS Bypass

Zero Copy

In non-RDMA operations, the data from a read type command (for example) is transfered from the device through the HBA hardware to a buffer assigned to the HBA by the Host memory manager. See Figure 155 for a write-to-memory example. That data will be copied to the system memory in supervisor mode and then copied to the application memory. All those copy operations take CPU cycles. In RDMA mode with zero copy, the data is written to the application memory by the remote device.

For a read-from-memory operation the data flows the opposite way, but through the same steps.

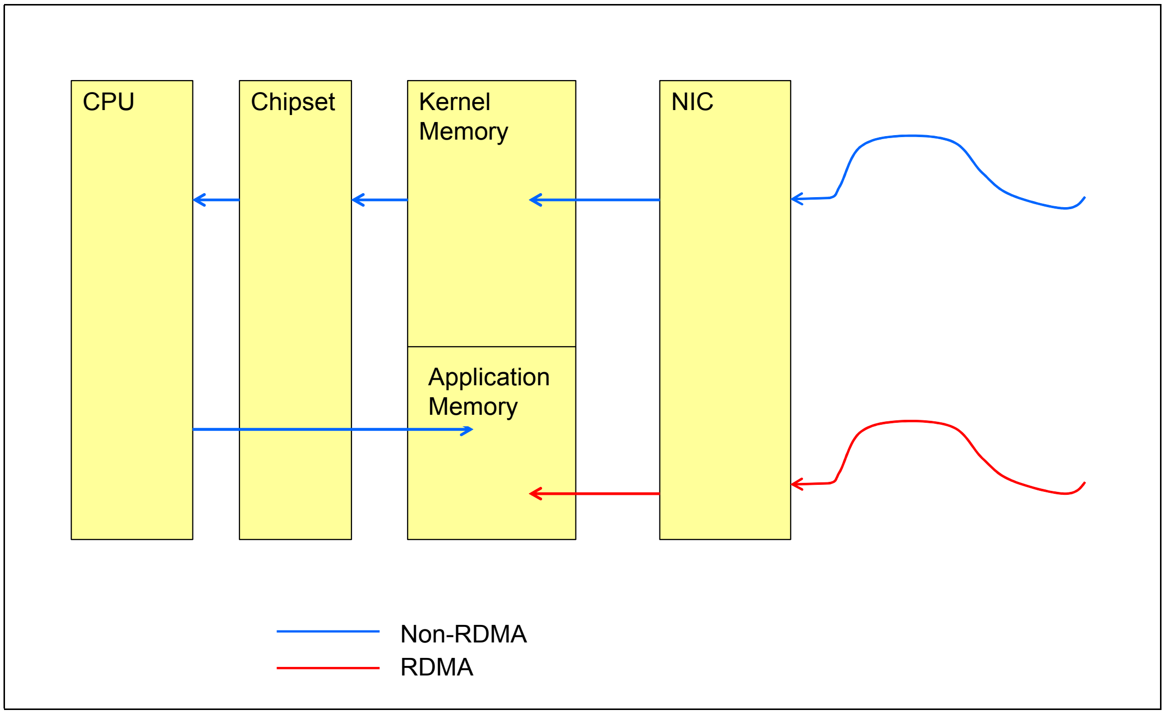


Figure 155: Zero Copy

Verbs

In English grammar, verbs define the action in our sentences. In RDMA as defined by InfiniBand, verbs define the action one device wants another device to perform. In RDMA, verbs are an abstraction which communicate an action by do not communicate how. A verb of "store" from the host to a storage device would work on a HDD, SSD, tape, CD, DVD, punch card, or any new means of storage yet to be developed. See Figure 156.

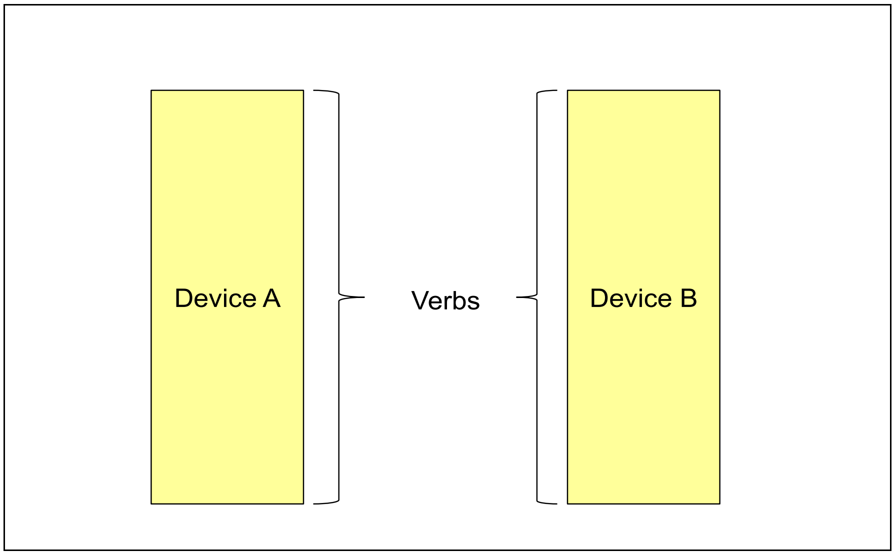


Figure 156: RDMA Verbs - an abstraction

A more formal definition from IETF is: The RDMA Verbs describe the behavior of a RNIC (RDMA Network Interface Card) hardware, firmware, and software as viewed by the host, but is not the host software and it is not a programming interface.

InfiniBand Specification Volume 1, Chapter 11, defines verbs as, "The Verbs described in this chapter provide an abstract definition of the functionality provided to a host by a host channel interface. Host CIs (Chanel Interfaces) which are compliant with this specification must exhibit the semantic behavior described by the Verbs."

Verbs	Level
Open HCA, Close HCA, Query HCA, Modify HCA	Privileged
Allocate PD, Deallocate PD	Privileged
Create AV, Query AV, Modify AV, Destroy AV	Privileged
Create QP, Modify QP, Query QP, Destroy QP, Get Special QP	Privileged
Create CQ, Query CQ, Resize CQ, Destroy CQ	Privileged
Create EE, Modify EE, Query EE, Destroy EE	Privileged
Register Memory Region, Query Memory Region, Deregister Memory Region, Reregister Memory Region	Privileged
Register Physical Memory Region, Reregister Physical Memory	Privileged
Register Shared Memory Region	Privileged
Allocate Memory Window, Query Memory Window, Deallocate Memory Window	Privileged
Bind Memory Window	User
Attach QP to Multicast Group, Detach QP from Multicast Group	Privileged
(Post) Send, (Post) Receive	User
Poll CQ	User
Request Completion Notification	User

Table 51: List of RDMA Verbs

Abbreviation	Meaning
AV	Address Vectors
CQ	Completion Queue
EE	End to End
HCA	Host Channel Adapter
PD	Protection Domain
QP	Queue Pair

Table 52: List of abbreviations from Table 51

Using these verbs, any host, regardless of Operating System, hardware, or Programming Interface, can communicate with any other host or any device regardless of type, as long as they both support RDMA as defined.

Chapter Summary

Overview of RDMA
- OS Bypass
- Zero Copy
- Verbs

InfiniBand Basics for NVMe

Chapter Contents

Overview

InfiniBand has been in use since 2000 primarily in High Performance Computing (HPC). It uses RDMA for ultra low latency and high throughput. InfiniBand uses a network to connect processors, storage, and user interface. Frequent users of HPC are:

Atmospheric modeling,
Genomics research,
Automotive crash test simulators
Oil and gas extraction models
Fluid dynamics, and
Scientific applications.

Genesis of InfiniBand

Approximately 1996, many architects realized the parallel PCI and parallel SCSI[31] would not scale well in the future. To achieve the performance and capacity of CPU and memory that was expected meant that I/O would be a major bottleneck, with no real hope of solution in the acceptable future. Further, PCI and SCSI did not work well for Inter Processor Communication (IPC), the method to transfer data directly between CPUs for High Performance Computing (HPC).

The architects of a group of companies (Dell Computing, Hitachi, Intel, NEC, Fujitsu Siemens, and Sun Microsystems) formed a task force to define a solution, called Next Generation I/O (NGIO).

The architects of a different group of companies (IBM, Compaq, Adaptec, 3Com, Cisco and Hewlett-Packard) formed a task force to develop a different solution called Future I/O.

Each of the task forces raced forward to be the first to release a workable standard. Many other companies had to have members on each task force so they could be ready regardless of which of the two competing standards won. In about 1999, they realized that neither method was the complete solution and that the fragmentation of the market was devastating and so joined to produce a single solution called InfiniBand[32].

At the same time, a software package called Virtual Interface (VI) was showing usefulness. The InfiniBand took the concepts of VI and place them into InfiniBand as hardware.

The architects of InfiniBand wanted to have multiple hosts talking to each I/O device so they bypassed PCI which has a single upstream port (one host) and implemented a switch fabric so any host could talk to any device.

They wanted to bypass the bottleneck of interrupts, I/O registers and context switching in the host so they created Remote Direct Memory Access (RDMA) where a host or I/O device could write directly into the application memory.

They wanted the infrastructure to be scalable out and scalable up so they defined each server and each I/O device as a unit; the client simply adds more units to meet the processing and storage needs.

They wanted the network to be load balancing and self-healing so they included a subnet manager which monitors the network and programs the switches for the best performance.

[31] The PCIe (serial PCI) specification was released in 2003 and the SAS (Serial Attached SCSI) specification was released in 2004.

[32] Some people claim that the name InfiniBand is a portmanteau of “Infinite Bandwidth” but the architects of both forming groups were uncomfortable with that meaning.

InfiniBand defines transports that provide:

Reliable and unreliable delivery,
Connected and connectionless delivery,
Atomic Ops,
Multicast, and
Others.

Queue Pairs

A key to this approach is that InfiniBand Architecture gives every application direct access to the messaging service. Direct access means that an application need not rely on the operating system to transfer messages.

InfiniBand calls the endpoints on each channel Queue Pairs (QPs); each QP consists of a Send Queue and a Receive Queue. If an application requires more than one connection, more QPs are created. In order to avoid involving the OS, the applications at each end of the channel must have direct access to these QPs. This is accomplished by mapping the QPs directly into each application's virtual address space. Thus, the application at each end of the connection has direct, virtual access to the channel connecting it to the application (or storage) at the other end of the channel. This is the notion of Channel I/O.

Taken altogether, InfiniBand Architecture creates private, protected channels between two disjointed virtual address spaces.
It provides:

a channel endpoint called a QP to the applications at each end of the channel, and

a means for a local application to transfer messages directly between applications residing in those disjointed virtual address spaces.

InfiniBand also provides for non-InfiniBand packets such as IPv6. These are called Raw Packets.

InfiniBand Speeds

InfiniBand speeds and encoding are shown in Table 53. This shows a x1 link speed. Links can be made faster by grouping 1, 2, 4, 8 or 12 lanes together. Grouping four lanes into a link is sometimes call a 4X link or sometimes called 4 lanes per port.

Speed Name	Clocking	4X Data Rate	Name	Encoding
SDR	2.5 Gb/s	8 Gbps	Single Data Rate	8b/10b
DDR	5.0 Gb/s	16 Gbps	Double Data Rate	8b/10b
QDR	10.0 Gb/s	32 Gbps	Quad Data Rate	8b/10b
FDR	14.0625 Gb/s	54 Gbps	14 Data Rate	64b/66b
EDR	25.78125 Gb/s	100 Gbps	Enhanced Data Rate	64b/66b
HDR	51.56 Gb/s	200 Gbps	High Data Rate	64b/66b
NDR	TBD	TBD	Next Data Rate	TBD
XDR	TBD	TBD	TBD	TBD

Table 53: InfiniBand Speeds per lane

Note in this table that the data rate is shown for a link and also takes into account the encoding.

InfiniBand Network

Figure 157 shows a single InfiniBand network.

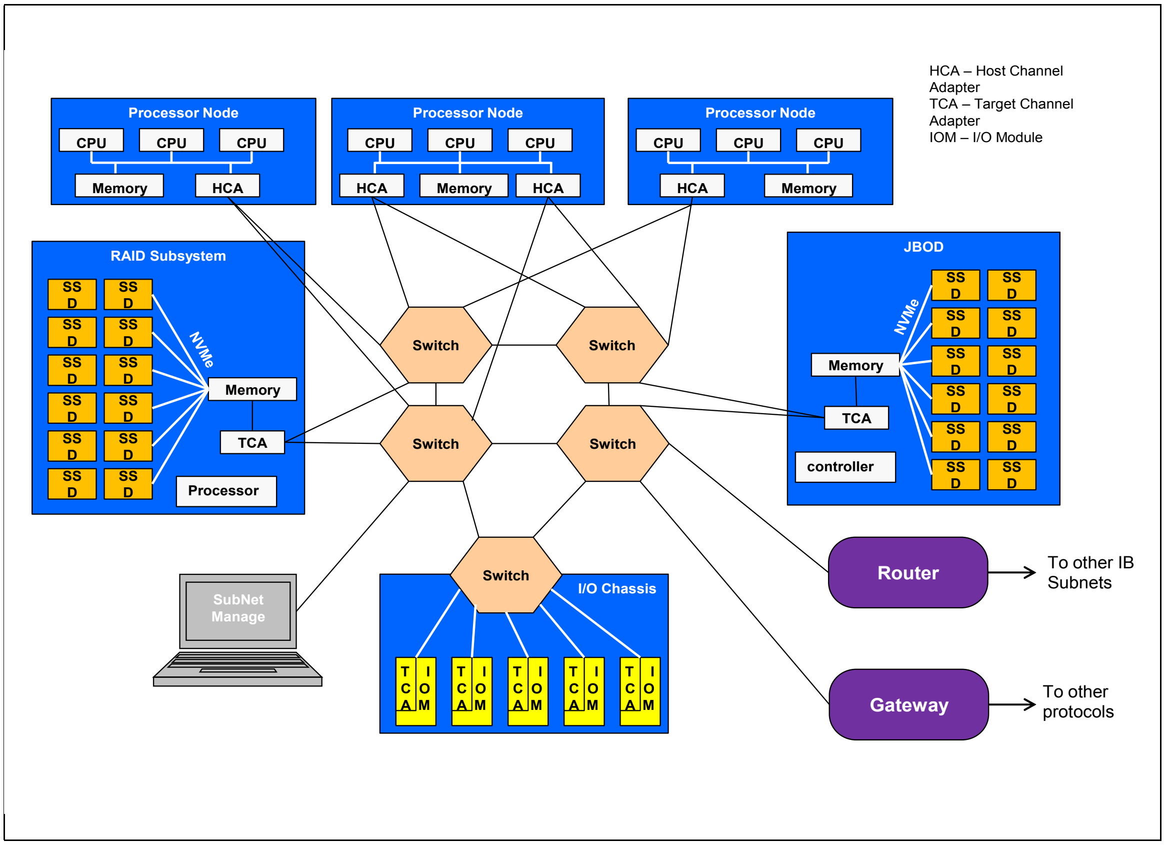


Figure 157: InfiniBand Network

A router is used to connect to other IB subnets. A gateway is used to connect to other protocols (e.g. Fibre Channel, iSCSI, SAS). The switches and interconnecting physical links between the TCAs, HCAs, routers, gateways, and subnet manager is called the IB Fabric. Connected to the Fabric are the nodes which can be processors, I/O, Routers, Gateways. The Fabric is a point-to-point switched network.

Definitions

Fabric The collection of Links, Switches, and Routers that connects a set of Channel Adapters.

Subnet A set of InfiniBandTM Architecture Ports, and associated links, that have a common Subnet ID and are managed by a common Subnet Manager. Subnets may be connected to each other through routers.

Components

Channel Adapters

Figure 157 shows the connectivity between the nodes and network are either:
HCA - Host Channel Adapters, or
TCA - Target Channel Adapters.
Each Channel Adapter (CA) has a Globally Unique Identifier (GUID) assigned by the CA vendor. Also called the Node GUID, it is persistent across power cycles and resets.

Each CA has ports. Each port is assigned a Local ID (LID) or a range of LIDs by the Subnet Manager. Each port has its own set of transmit and receive buffers.

The CA provides virtual memory address to physical memory address translation and memory access protection.

Switches

Switches route packets based on destination LID
Switches support unicast and may support multicast
Subnet manager (SM) configures switches
If multiple paths exist, subnet managers can use them for redundancy or load sharing

Intra-subnet switching

Quality of Service

Quality of Service is a flexible means of communicating the desired performance of an individual packet in its journey from the original source to the final destination. Definitions of the performance is defined by the individual customer; IB only defines how the communication works.

Virtual Lanes

Virtual Lanes (VL) are between ports. Each port has a physical link to connect to its peer. The ports also have several Virtual Lanes that share the bandwidth on the physical link. Virtual lanes have a priority such that high priority data will flow before lower priority data.

Each Virtual Lane is represented by a VL number of 0 (lowest priority) through 15 (highest priority). VL 15 is mandatory in all ports and is used by the Subnet Manager. VL 0 through 14 are used as Data Virtual Lanes. VL 0 is mandatory and the default. VL 1 through VL 14 are configured by the SM after it has determined the number supported by the devices on each end of the physical link.]

The SM will determine the number of VLs supported by both devices on a physical link and map the Service Levels from the Local Routing Header (see Figure 159 on page page 242) onto the appropriate VL. The mapping of Service Level to Virtual Lane on one Physical Link may be different from the mapping of that Service Level to Virtual Lane on the next Physical Link.

Each Virtual Lane has independent flow control so one lane does not block another.

Each Virtual Lane has a Send and a Receive Queue. The size of the Queues is vendor specific.

Packets addressed to QP 0 are Subnet management packets are use exclusively VL 15.

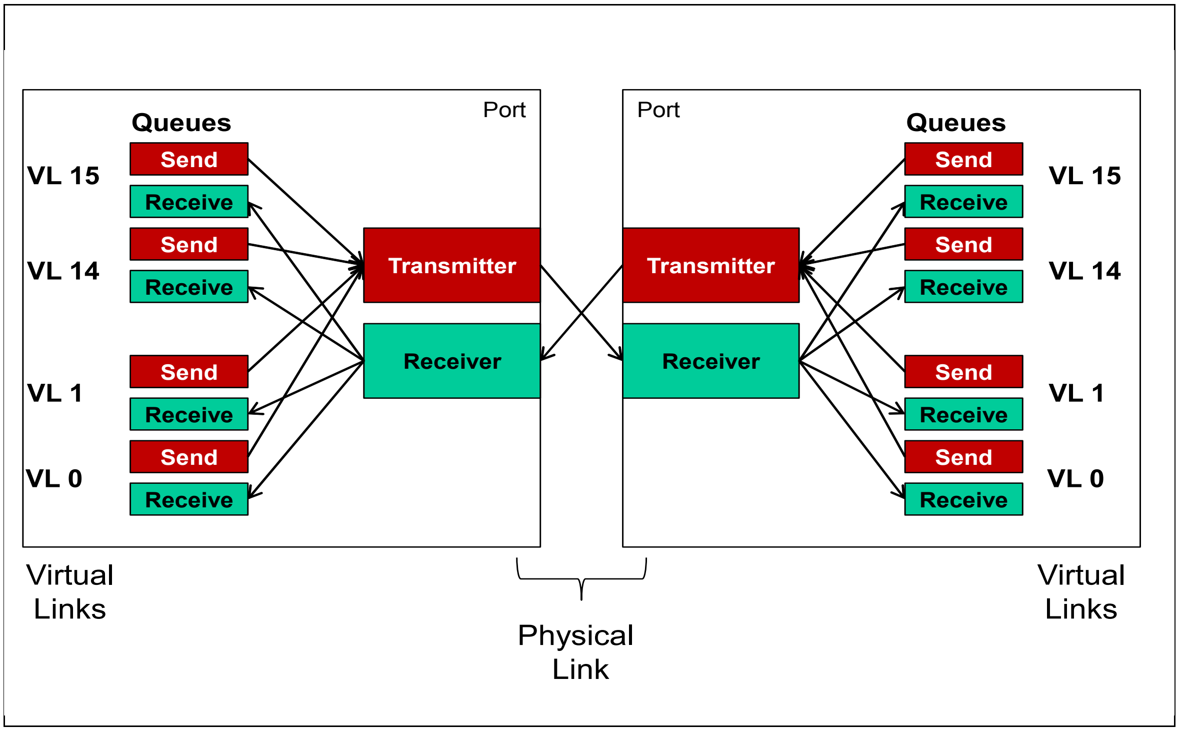


Figure 158: Virtual Links vs. Physical Link

Service Levels

Service Levels are information communicated in the Local Routing Header to indicate to the switch which virtual lanes should be used for this packet. There are 16 service levels that map to VL for each switch. The purpose and definition of each service level is client defined.

Traffic Class

Quality of service information carried in Global Routing Header. It maps to the Service Level of the Local Routing Header.

Routers

Routers are similar to switches but connect different subnets.
Routers replace packet's local router header as it passes from subnet to subnet

When using a router, source must specify the LID of the router and GID of the destination.
Each subnet is uniquely identified with a subnet ID known as the Subnet Prefix. Subnet manager programs all router ports with a subnet prefix for the subnet of that port. Ports may have multiple GIDs. Subnet prefix is the route through the router.

Queues

Queue Pair (QP) is one Send queue and one Receive queue
QP defines the end of a virtual lane
Two QP define a virtual lane
Each CA may have 2^{24} QP
QP is a private resource assigned to a single consumer

Send work queues generally hold instructions that cause data to be transferred between the consumer's memory and another consumer's memory. Receive work queues generally hold instructions about where to place data that is received from another consumer.

Completion queues hold ending status for work requests. Multiple work queues can be matched with a single completion queue.

Queue - Types of Service

Connection vs. Datagram

Connection - Each QP is associated with exactly one other QP

Datagram - Allows a single QP to send and receive messages to/from any appropriate QP on any node.

Reliable vs. Unreliable

Reliable - transport guarantees deliver of each message, in order and exactly once in absence of errors. Uses Acknowledges and negative ACKs.

Unreliable - transport does not guarantee that all date is delivered. Receiver can verify the data is not duplicated or corrupt.

Partitions

A partition describes a set of endnodes with the fabric that may communicate.

Partitioning enforces isolation among systems sharing an InfiniBand fabric. Partitioning is not related to boundaries established by subnets, switches or routers.

Each port of an endnode is a member of at least one partition and may be a member of multiple partitions.
A partition manager assigns partition keys (P Keys) to each channel adapter port.

Each P Key represents a partition. Each QP and End-to-End context is assigned to a partition and uses that P key in all packets it sends and inspects the P Key on all packets it receives. Reception of an Invalid P Key causes the packet to be discarded.

Switches and routers may optionally be used to enforce partitioning. In this case the partition manager programs the switch or router with P Key information and when the switch or router detects a packet with an invalid P key, it discards the packet.

Subnet Manager

The subnet manager (SM) is a server that queries the infrastructure to discover the available end devices and the path(s) between each pair. It configures the switches and routers for the fastest path between end devices. Then it monitors the network and will reconfigure paths between devices to balance the load or if a failure occurs.

Discovery Service

Nothing in the InfiniBand Specification, nor in the NMVe Over Fabrics Specification suggests it, but this author believes that the Subnet Manager is a logical place for the Discovery Server. The Subnet Manager discovers the devices that are on the subnet and maintains an up-to-date list of such devices are they are added, removed, or disabled due to failure. With a little software it could format that information as specified in the NVMe Over Fabrics specification and provide it to NVMe hosts when requested.

An alternative could be for NVMe to supply its own Discovery Server which interrogates the Subnet manager, or does it own discovery.

Packet Format

Figure 159 shows an InfiniBand packet.

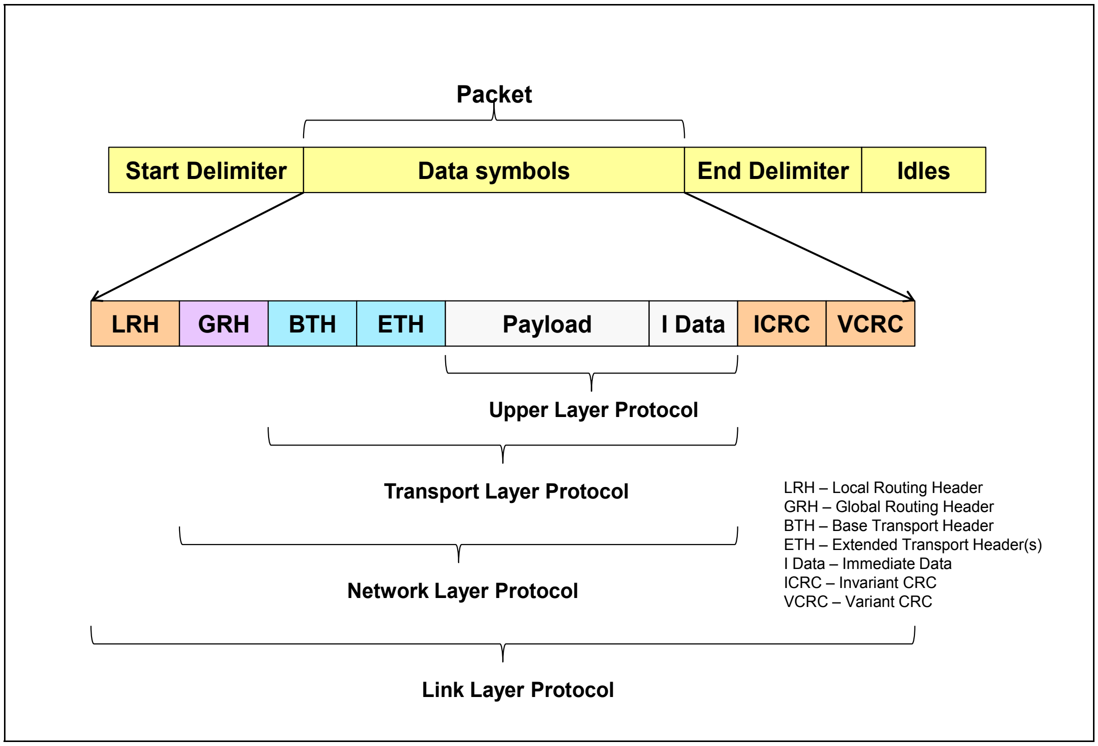


Figure 159: InfiniBand Packet Format

The start delimiter is either Start Data Packet (SDP) or Start Link Packet (SLP).

The end delimiter is either End Good Packet (EGP) or End Bad Packet (EBP).

The Start delimiters, End delimiters and Idles are either K characters in 8b/10b encoding or ordered sets in 64b/66b encoding.

The LRH (Local Routing Header) provides the address within the local subnet.

The GRH (Global Routing Header) provides the address outside the local network, if applicable. If there are no routers or gateways in the domain, the GRH will not be present.

The BTH (Base Transport Header) is present in all packets except Raw datagrams. It specifies the destination QP and indicates the operation code, packet sequence number and partition.

The various ETHs (Extended Transport Header(s)) are present depending on class of service and the operation code from the BTH.

The payload is the NVMe commands, data, or response between devices.

The I Data (Immediate Data) is optionally present in RDMA Write and SEND messages. It is the data that the application program placed in the RDMA Write or SEND message buffers.

The ICRC is a CRC check on the invariant data within the packet (data that does not change between original source and final destination).

The VCRC is a CRC check on the data within the packet that may change when going though routers or gateways.

Addressing

LID, SLID, SGID and more acronyms. But it is all easy.
LID is a 16 bit Local IDentifier in the LRH.
GID is a 128 bit Global IDentifier in the GRH.
SLID and SGID are the Source Local or Global ID.
DLID and DGID are the Destination Local or Global ID.

Figure 160 shows the format of the LRH.

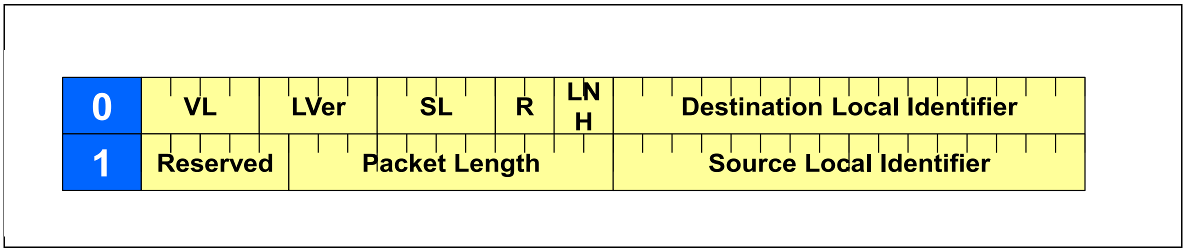


Figure 160: Local Route Header

If this packet is destined for another subnet, the DLID is the address of the router, and the LNH (Link Next Header) indicates what header is next. The two bit field is interpreted as:

MSb: 1 = IBA, 0 = non-IBA
LSb: 1 = GRH or IPv6 present, 0 = GRH or IPv6 not present

Figure 161 shows the format of the GRH (if used).

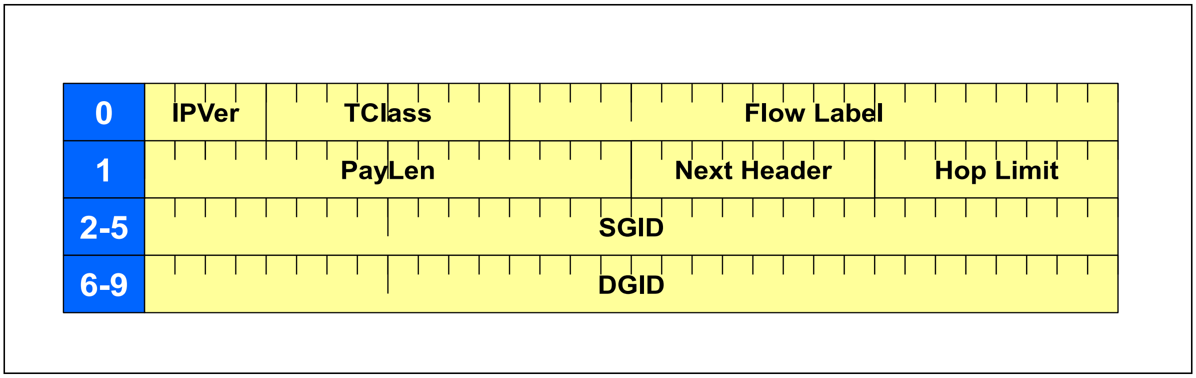


Figure 161: Global Route Header

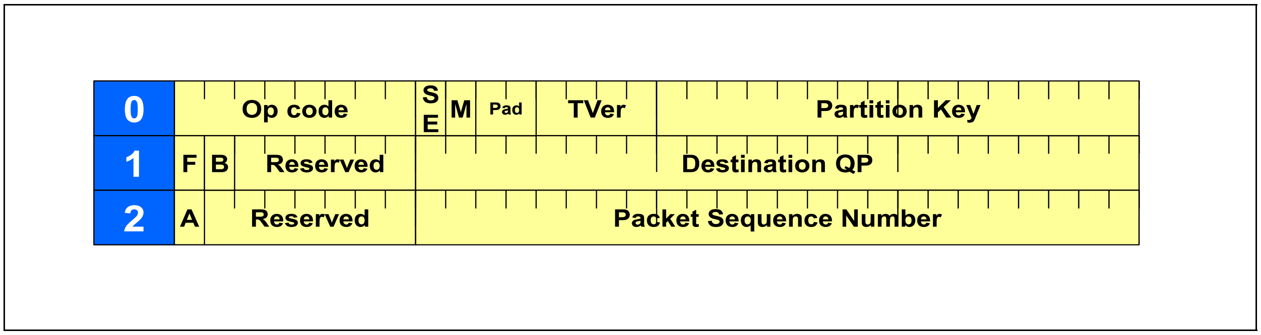


Figure 162: BTH Format

The first three bits of the Op Code indicate if this is to be handled as a reliable connection, unreliable connection, reliable datagram, unreliable datagram, congestion notification packet (CNP), or extended reliable connection.

Reliable Connection A Transport Service Type in which a Queue Pair is associated with only one other QP, such that messages transmitted by the send queue of one QP are reliably delivered to receive queue of the other QP. As such, each QP is said to be "connected" to the opposite QP.

Reliable Datagram A Transport Service Type in which a Queue Pair may communicate with multiple other QPs over a Reliable Datagram Channel. A message transmitted by an Reliable Datagram QP's send queue will be reliably delivered to the receive queue of the QP specified in the associated Work Request. Despite the name, Reliable Datagram messages are not limited to a single packet.

Unreliable Connection A Transport Service Type in which a Queue Pair is associated with only one other QP, such that messages transmitted by the send queue of one QP are, if delivered, delivered to the receive queue of the other QP. As such, each QP is said to be "connected" to the opposite QP. Messages with errors are not retried by the transport, and error handling must be provided by a higher level protocol.

Unreliable Datagram A Transport Service Type in which a Queue Pair may transmit and receive single-packet messages to/from any other QP. Ordering and delivery are not guaranteed, and delivered packets may be dropped by the receiver.

Encoding

8b/10b Encoding (Symbol Encoding)

Used with SDR, DDR, and QDR signaling rates.

Character	Encode	Name
SDP	K27.7	Start Data Packet
SLP	K28.2	Start Link Packet
Com	K28.5	Comma
EGP	K29.7	End Good Packet
EBP	K30.7	End Bad Packet
PAD	K23.7	Packet Padding
SKP	K28.0	Skip
	K28.1	Reserved
	K28.7	Reserved
	K28.3	Reserved
	K28.4	Reserved
	K28.6	Vendor Specific

Table 54: 8b/10b K Characters

Pad is used on the 8X and 12X physical links to align the physical lanes. Pad is also used for retiming repeaters to forward error conditions.

8b/10b Ordered Sets

All of the ordered sets in 8b/10b encoding begin with K28.5, the comma character.

Ordered Set	Identifie r	Size (Symbols)	Description
SKIP	K28.0	4	Skip for buffer elasticity
TS1	D10.2	16	Training Sequence 1; ensures both transmitters are active
TS2	D5.2	16	Training Sequence 2; ensures both receivers are trained
TS3	D13.2	16	Training Sequence 3; used to communicate enhanced capabilities
HRTBT	D1.2	16	Link Heartbeat
TS-T	D17/2	16	Training Sequence for Test

Table 55: 8b/10b Ordered Sets

Details of these Ordered Sets may be found in InfiniBand specification Volume 2 Rev 1.3, section 5.5.2.

64b/66b Encoding (Block Encoding)

Used with FDR, EDR, and HRD signaling rates.

“Encoding” is done by adding a two-bit Sync Header before each 64 bits of data. The Sync Header is either:
01b if the following eight bytes are data, or
10b if the following eight bytes are control.

Block	Sync Header	Byte 0	Byte 1	Byte2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Data	01b	Date 0	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7
Control	10b	Block Type	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6

Table 56: 64b/66b Encoding format

Ordered Set	Block Type (h)	Size (bytes)	Description
SDP	78	8	Start Data Packet Block
SLP	55	8	Start Link Packet Block
EGP3	B4	8	End Good Packet block with 3 data bytes
EGP7	FF	8	End Good Packet block with 7 data bytes
EBP3	AA	8	End Bad Packet block with 3 data bytes
EBP7	E1	8	End Bad Packet block with 7 data bytes
Idle	1E	8	Idle Block
SKP	4B	8	Skip Block
TS1	4B	8	Training Sequence 1
TS2	4B	8	Training Sequence 2
FTB	4B	8	Fine Tuning Block
HRTBT	4B	16	Heart Beat Ordered Set
Vendor Specific	4B	8bytes	Vendor Specific

Table 57: 64b/66b Ordered Sets

Details of these Ordered Sets may be found in InfiniBand specification Volume 2 Rev 1.3, section 5.5.3.

Mapping NVMe Commands to InfiniBand

The InfiniBand Trade Association has stated that the mapping of NVMe commands to InfiniBand is completely covered in the NVMe over Fabrics specification.

Chapter Summary

This chapter was not intended to be a detailed tutorial of InfiniBand, only enough to get you started in understanding NVMe over InfiniBand. To that end, this chapter reveals an example configuration, identification of InfiniBand components including the Subnet Manager, layout of the packet, and identification of speeds and encoding. InfiniBand specifications volume 1 and volume 2 are available at www.InfiniBandTA.org.

Chapter Contents

Ethernet Basics for NVMe

Chapter Contents

Overview

Ethernet:
is ubiquitous and high speed,
has many vendors providing a wide range of products and services, and
is accepted from the home owners to Enterprise Data Centers.
It does not support RDMA.

Ethernet is a trademarked name, and defines a Link and Physical layer protocol. It is defined by IEEE, in their 802 specifications. Information can be found at www.IEEE.org.

The Internet Protocol Suite defined by the Internet Engineering Task Force includes the higher layers such as:
Network layer (e.g. IPv4, IPv6, ICMP, IGMP, IPsec[33]), the
Transport layer (e.g. TCP, UDP, SCTP), and the
Application layer (e.g. DHCP, DNS, FTP, HTTP, SMTP).
Information can be found at www.IETF.org

In this book, I use the term "Ethernet" to include all the layers where no distinction is desired, or when all layers are included. Otherwise, I identify the intended layer or layers by the proper identifying term.

The following few pages are not intended as an Ethernet tutorial, but only to establish a common ground for adapting NVMe over Fabrics to Ethernet. All of the material in these pages is available free, or nearly free, on the Internet and in various books.

Packets, Frames and Messages

The MAC frame is defined by IEEE in 802.3 (Ethernet) as the Destination and Source addresses, the length/type field, the client data, pad (if required), and the Frame Check Sequence (FCS).

The Ethernet packet is defined by IEEE in 802.3 (Ethernet) as the MAC frame plus the preamble and Start of Frame Delimiter (SFD), and encoding.

The IETF does not define the word "frames".

The IETF defines packets as defined in 802.3.

The IETF defines the client payload of a packet as a message.

[33] RDMA - Remote Direct Memory Access
IEEE - Institute of Electrical and Electronics Engineers
IPv4 - Internet Protocol version 4 (32-bit addressing)
IPv6 - Internet Protocol version 6 (128-bit addressing)
ICMP - Internet Control Message Protocol
IGMP - Internet Group Management Protocol
IPsec - Internet Protocol Security
TCP - Transmission Control Protocol (considered reliable)
UDP - User Datagram Protocol (considered unreliable)
SCTP - Stream Control Transmission Protocol
DHCP - Dynamic Host Configuration Protocol
DNS - Domain Name System (Service)
FTP - File Transport Protocol
HTTP - HyperText Transfer Protocol
SMTP - Simple Mail Transfer Protocol
IETF - Internet Engineering Task Force

Application

To take advantage of Ethernet and also the advantages of NVMe on SSDs, two protocols were developed: RDMA over Convergent Ethernet (RoCE) and iWARP. Although sometimes presented as competing protocols, they operate very differently in different layers of the protocol stack.

RoCE was developed by the InfiniBand Trade Association. It assumes the operation of InfiniBand which includes RDMA and replaces either layers 1 (Physical) and 2 (Link layer), or layers 1, 2, and 3 (Network) with Ethernet. Thus the upper layers that map the commands to the transport and provide RDMA are left intact as InfiniBand.

iWARP, on the other hand, add two or three additional programs in the Ethernet layer 4 (Transport layer) above TCP/IP that adds RDMA to Ethernet. All layers above those two or three, and all layers below them are unmodified Ethernet.

Ethernet

As with nearly all other current interconnecting protocol, Ethernet defines a frame[34] containing different fields. These fields are added by each layer as the frame goes down each layer of the OSI stack on send, and verified, acted upon, and removed as the frame goes up through the stack on receive. The Ethernet frame and packet format is shown in Figure 163.

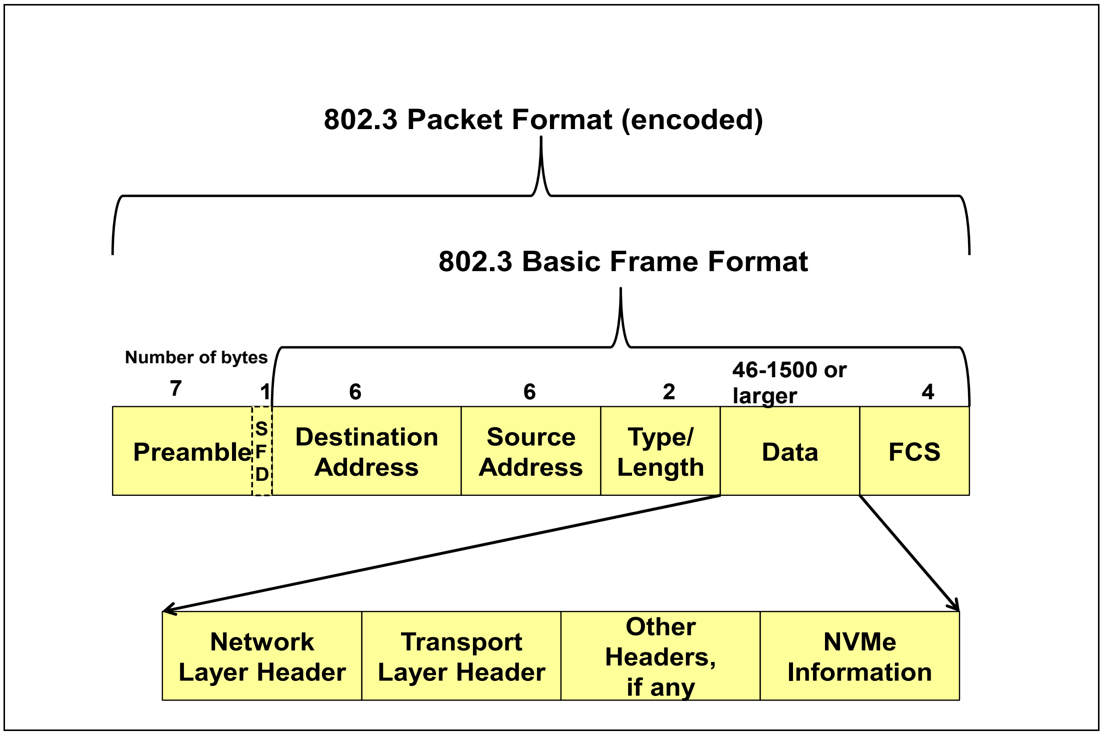


Figure 163: Example Ethernet Frame and Packet

The Preamble is seven bytes of alternating bits of 0b and 1b. The Start Frame Delimiter (SFD) is six bits of alternating 0b and 1b followed by two 1-bits. The preamble and SFD are deleted in the physical layer of the receiving device.

[34] Various other transports call these data structures frames, messages, or packets.

The Destination and Source Addresses are each 48-bit Media Access Control (MAC) addresses.

The type/length field has different uses such as identifying the type of data (e.g. IPv4, IPv6, NVMe) or the length of the data in this frame. Generally:
values from 0 to 1500 are length values,
values from 1501 to 1535 are reserved, and
values greater than 1535[35] are type.
A few type codes are:
0800h = IPv4,
86DDh = IPv6,
8870h = Jumbo Frames,
88A2h = ATA over Ethernet,
8915 = RoCEv1. For a full list of Ethertypes see http://standards-oui.ieee.org/ethertype/eth.txt.

The data field contains the payload. The minimum size is 46 bytes to allow for collision detect in the original Ethernet. The 1500 bytes is considered the standard size for Ethernet. However, special implementations can use larger frames. In the data field are the IP header, the TCP or UDP header, other headers as necessary, and the data.

The Frame Check Sequence (FCS) is a 32-bit CRC that covers everything except the Preamble and SFD.

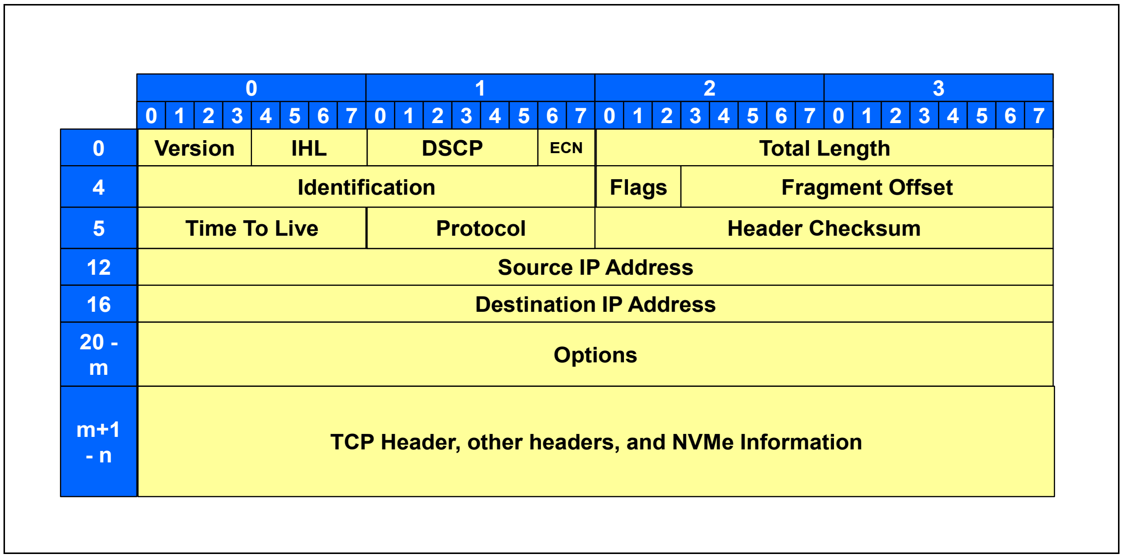
Network Layer

The network layer is where the source and destination IP addresses are communicated. This allows frames to go to any device on any subnet if it is not otherwise prohibited (e.g. security). With this, NVMe devices communicating over Ethernet can reach any device on a Ethernet network.

IPv4

IPv4 (Figure 164) provides for a 32 bit address.

[35] 1536 decimal is equal to 600 hexadecimal.



<table>
<tr><th></th><th colspan="8">0</th><th colspan="8">1</th><th colspan="8">2</th><th colspan="8">3</th></tr>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th></tr>
<tr><td>0</td><td colspan="4">Version</td><td colspan="4">IHL</td><td colspan="6">DSCP</td><td colspan="2">ECN</td><td colspan="16">Total Length</td></tr>
<tr><td>4</td><td colspan="16">Identification</td><td colspan="3">Flags</td><td colspan="13">Fragment Offset</td></tr>
<tr><td>5</td><td colspan="8">Time To Live</td><td colspan="8">Protocol</td><td colspan="16">Header Checksum</td></tr>
<tr><td>12</td><td colspan="32">Source IP Address</td></tr>
<tr><td>16</td><td colspan="32">Destination IP Address</td></tr>
<tr><td>20 - m</td><td colspan="32">Options</td></tr>
<tr><td>m+1 - n</td><td colspan="32">TCP Header, other headers, and NVMe Information</td></tr>
</table>

Figure 164: IPv4 Header

Version is 4h for IPv4

IHL - Internet Header Length in 32 bit words (points to the beginning of the data)

DSCP - (with ECN was called "Type of Service" in RFC 791. Updated in RFC2474. Equivalent to "Traffic Class" in IPv6) Differentiated Services Code Point.

ECN - Explicit Congestion Notification. Defined in RFC 3168.

Total Length - length of the datagram in octets including Internet header and data.

Identification - assigned by the sender to aid in assembling of fragments of a datagram.

Flags - control flags

Fragment Offset - where in the datagram this fragment belongs, measured in units of 8 octets.

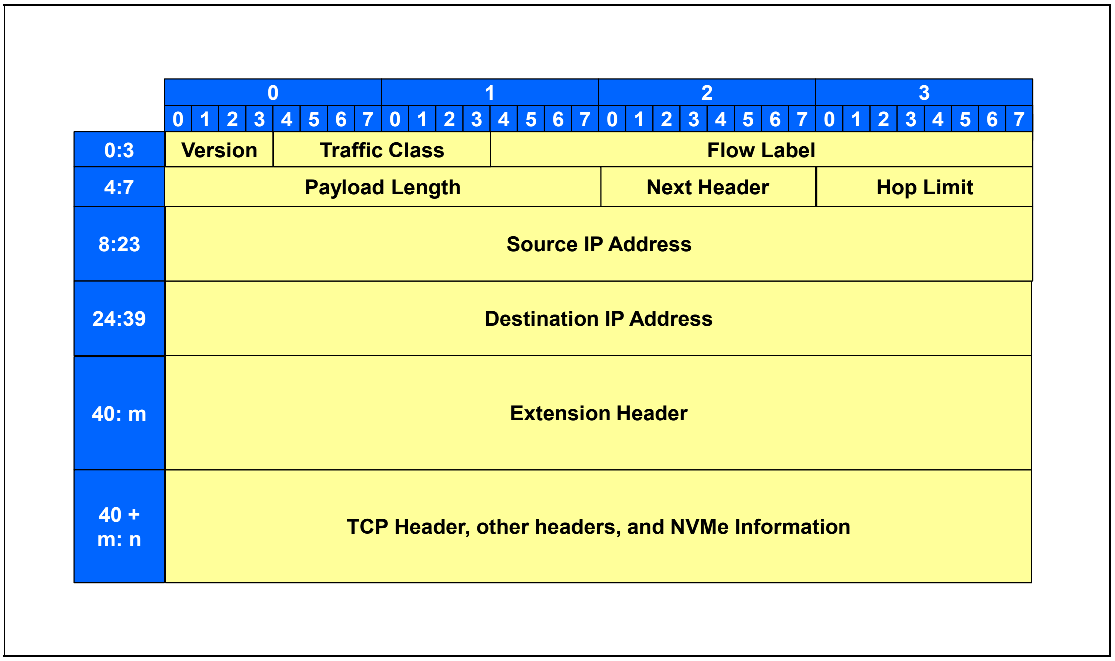
Time to Live is to prevent a frame from bouncing around in the Ethernet. The original idea was that this field contained a value that decremented and when it reached zero, the frame was discarded. The current implementation is usually to set it to a hop count and it decrements on each hop. When the count reaches zero, the frame is discarded. Called "Hop Limit" in IPv6 header

Protocol - what protocol is being used, assigned by Internet Assigned Numbers Authority (IANA). Check www.iana.org for list of assigned numbers. Called "Next Header" in IPv6 header.

Header Checksum - a 16-bit CRC of the header only. Must be verified and recomputed at each point the header is processed, because some fields change at each note (e.g. Time to Live).

IPv6

IPv6 (Figure 165) creates addresses of 128 bits allowing a great growth in the number of devices, and in the connectivity of those devices.



<table>
<tr><th></th><th colspan="2">0</th><th colspan="2">1</th><th colspan="2">2</th><th colspan="2">3</th></tr>
<tr><th></th><th>0 1 2 3</th><th>4 5 6 7</th><th>0 1 2 3</th><th>4 5 6 7</th><th>0 1 2 3</th><th>4 5 6 7</th><th>0 1 2 3</th><th>4 5 6 7</th></tr>
<tr><td>0:3</td><td>Version</td><td colspan="2">Traffic Class</td><td colspan="5">Flow Label</td></tr>
<tr><td>4:7</td><td colspan="4">Payload Length</td><td colspan="2">Next Header</td><td colspan="2">Hop Limit</td></tr>
<tr><td>8:23</td><td colspan="8">Source IP Address</td></tr>
<tr><td>24:39</td><td colspan="8">Destination IP Address</td></tr>
<tr><td>40: m</td><td colspan="8">Extension Header</td></tr>
<tr><td>40 + m: n</td><td colspan="8">TCP Header, other headers, and NVMe Information</td></tr>
</table>

Figure 165: IPv6 Header

Version - 6h = IPv6

Traffic Class - (was "Type of Service" and then "DSCP" in IPv4) Provide information of priority of frames for congestion management or discarding of frames.

Flow Label - used to identify non-default quality of service for this packet

Payload Length - length of packet following this IPv6 header, in octets

Next Header - Type of header immediately following this one. Uses same value as IPv4 **Protocol** Field.

Hop Limit is similar to the **Time to Live** in the IPv4 network header. The field is decremented each time this frame goes through a switch or router. When the field reaches zero, the frame is discarded.

Transport Layer

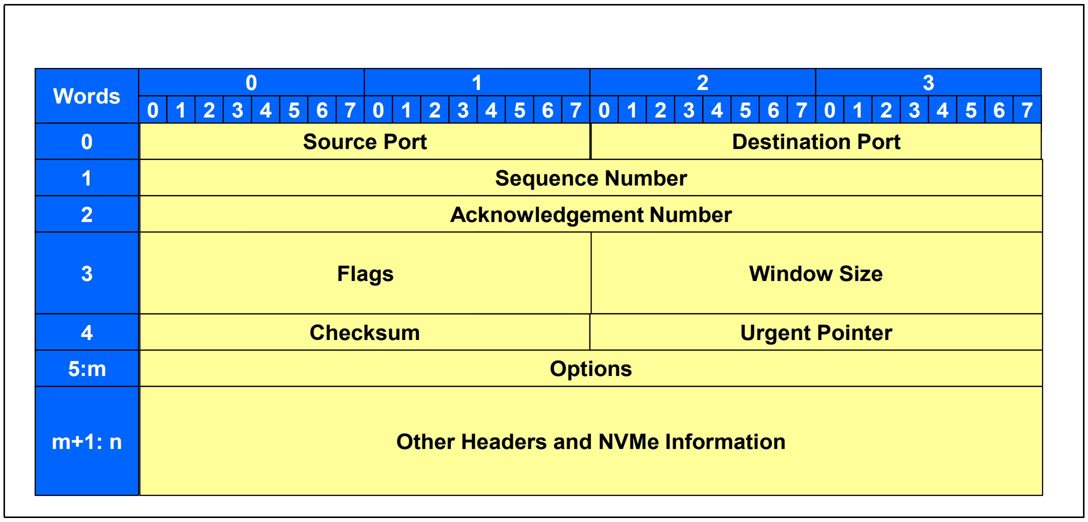
Very simply put, the Network layer directs the packet to the proper node or device; the Transport layer directs it to the proper port[36] on the device. The two main protocols in the Transport Layer are Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). TCP provides a reliable transport meaning that each received packet it checked and acknowledged; if not properly acknowledged, the sender can go into error recovery. UDP provides an efficient, unreliable transport meaning that there is no acknowledgment sent from the packet receiver to the packet sender. For each TCP and UDP, there may also be higher layer protocols to verify the information sent was received, but there are outside the scope of Ethernet 802.3 and Internet Protocol Suite.

[36] This is referring to the 64K ports in the TCP or UDP stack, not the 1, 2, or 4 physical connection points which are also called ports. These port referenced here are assigned to certain protocols, for example Ports 20 and 21 are for FTP and Port 80 is for TCP.

The network layer contained the node addresses (source and destination), the transport layer contains the port addresses (source and destination).

TCP

Figure 166 shows the TCP header. Because TCP is a reliable[37] connection-based[38] transport, it has the Sequence Number field and the Acknowledge Number to ensure that every frame is received correctly and every ACK arrives to its destination.



Words	0								1								2								3							
	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
0	Source Port																Destination Port															
1	Sequence Number																															
2	Acknowledgement Number																															
3	Flags																Window Size															
4	Checksum																Urgent Pointer															
5:m	Options																															
m+1: n	Other Headers and NVMe Information																															

Figure 166: TCP Header

Sequence Number is the sequence number of the first data octet in this segment.

Acknowledgement Number is the value of the next sequence number the sender of the segment is expecting to receive.

Window size is the number of data octets beginning the sender of this segment is able to accept.

UDP

See Figure 167. On the other hand, UDP is an unreliable, connectionless transport so it does not have the Numbers. Frames and ACKs could be lost and it would not be detected except perhaps at a higher layer.

[37] Reliable means that the frame sender is notified that the frame is notified that each frame arrived without errors. This is usually accomplished with Acknowledges sent to the frame sender from the frame receiver.

[38] Connection-based means that a separate protocol associates the originator with the responder before the information frames are sent. The resources used for the connection are reserved to that connection.

	0		1		2		3	
	0 1 2 3	4 5 6 7	0 1 2 3	4 5 6 7	0 1 2 3	4 5 6 7	0 1 2 3	4 5 6 7
0	Source Port				Destination Port			
1	Length				Checksum			
2-n	Other Headers and NVMe information							

Figure 167: UDP Header

Discovery Service

Whereas InfiniBand and Fibre Channel are expected to have stable configurations and are designed with that in mind (they have a device which has a list of all devices in the domain), Ethernet is expected to see devices added and removed frequently (and therefore does not try to keep a current list of active devices in one device which may be queried).

Ethernet used in NVMe configurations will probably be treated, in this respect, more like InfiniBand and Fibre Channel and have stable configurations. It may be reasonable to create a NVMe Discovery Server that keeps a current list of devices on the subnet which may be queried by the NVMe Discovery Service. Using means outside the specifications it may be possible to keep the discovery time to a minimum (assigning IP addresses to slots rather than devices, having each tray or rack have a very limited range of IP addresses).

RoCE

For this discussion of RoCE, please reference Figure 140 on page 210.

RoCE v1 vs. RoCE v2

RoCE v1 replaces the Physical and Link Layer of InfiniBand with the equivalent layers of Ethernet. This means that the Local Routing Header (LRH) is replaced by the MAC address of Ethernet. It also means that the packet cannot cross from one subnet to another. Many InfiniBand implementations are on one subnet, so this is not a major limitation.

RoCE v2 replaces the Physical, Link, and Network layers with the equivalent layers of Ethernet. The Global Routing Header (GRH) of IB is replaced by a IP address. This gives the capability for the packet to cross from one subnet to another.

RoCE v2 also uses, or rides on, UDP which is a Ethernet layer 4 (Transport Layer) protocol.

RoCE was created by the InfiniBand Trade Association (www.InfiniBandTA.org) and is defined in Annexes 16 and 17 of the IB specification. Both of those Annexes are standalone documents. The specification defines RoCE and InfiniBand over a non-IB defined link and physical layer. Convention today uses only Ethernet.

IB virtual lanes are not supported. Instead, IB Service Levels 0-7 are mapped to Ethernet priorities 0-7. IB Service Levels 8-15 are reserved.

Every RoCE packet contains a GRH. IB normally does not have the GRH unless the destination is on a different subnet that the source. Every RoCE packet contains a MAC header in place of the IB LRH. Every RoCE packet contains the Ethernet Frame Check Sum (FSC) as defined for Ethernet.

Figure 168 shows the development of RoCE from the InfiniBand viewpoint.

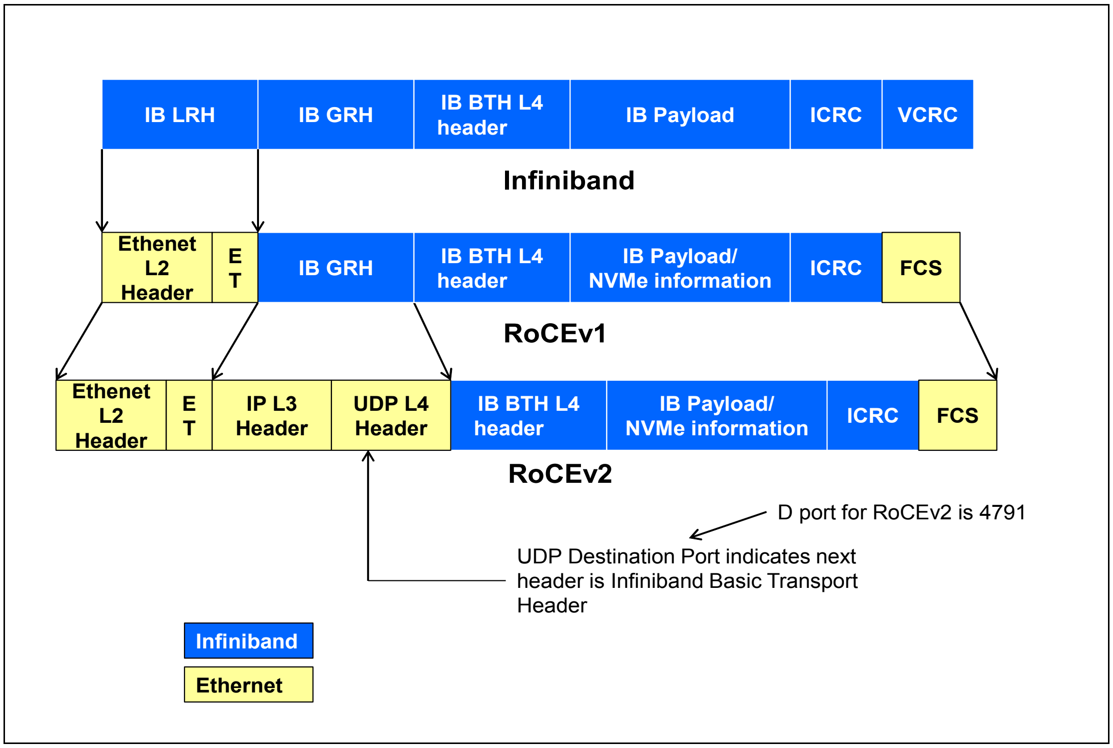


Figure 168: RoCE and InfiniBand

The top row shows the pure InfiniBand packet. To create RoCEv1, Ethernet layers 1 and 2 replace the IB LRH, and the Ethernet FCS replaces the VCRC of IB.

To create RoCEv2, the Ethernet L3 (network) and Ethernet L4 (transport) replaces the IB GRH.

iWARP

Caution: While each specification is consistent is its use of words, phrases and acronyms, they are not consistent from one standard to another. For instance, ULP, is defined as the protocol layer above the protocol layer defined in the specification. For RFC 5040, the ULP is an OS or Application, for RFC 5044, the ULP is a DDP segment and what RFC 5040 calls the ULP is called the consumer message. In this discussion, I have used the terms as used in the specifications and added clarification notes.

Note: Some people say that iWARP stands for Internet Wide Area RDMA Protocol. The IEFT standards committee who creates the Internet Standards, including iWARP claim it has no translation and then lists uses for iWARP that do not fit the suggested acronym.

Note: The term iWarp also referred to an experimental parallel supercomputer architecture developed by Intel and Carnegie Mellon University in 1988. This iWarp had NOTHING to do with the current iWARP developed by the IETF. At least the capitalization is different, which give a clues to which standard is being referenced - if people are consistent in their capitalization.

You can purchase Ethernet HBAs for $6.80 or for $800.00 or anywhere between. A major difference is where the processing is done; A - on the low cost HBAs, most of the processing is done in the host processor causing more work for it and limiting the HBA speed, and B - on the high cost HBAs, most of the processing is done in the HBA freeing up the host processor to do more of its work and handing the processing in hardware making it very fast. This moving of the TCP/IP stack from software to the HBA hardware is called TCP Offload Engine (TOE). TOE is widely used on Fibre Channel and iSCSI where high performance is desired. It is assumed that for SSDs connected via Ethernet and iWARP, TOE will be used.

Documents

All RFC documents are available from ietf.org.

RFC 5040 - RDMA Protocol Specification
RFC 5041 - DDP over Reliable Transports
RFC 5042 - DDP/RDMAP Security
RFC 5043 - Stream Control Transmission Protocol (SCTP)
RFC 5044 - Marker PDU Aligned Framing for TCP Specification
RFC 6580 - IANA egistries for the Remote Direct Dataa Placement
RFC 6581 - Enhanced RDMA
RFC 76306 - RDMA Protocol Extensions

Draft-hilland-iwarp-verbs-v1.0 - RDMA Protocol Verbs.

iWARP Operation

For this discussion of iWARP, please reference Figure 140 on page 210.

Also please reference Figure 163 on page 251. The “other headers” referenced are RDMAP, DDP, and MPA in iWARP.

Transmission of Commands and Status use NVMe capsules. Transmission of data uses RDMA.

Reference Figure 169, the layers of iWARP. These layers ride on TCP/IP which is Ethernet’s was of providing reliable transport over multiple subnets.

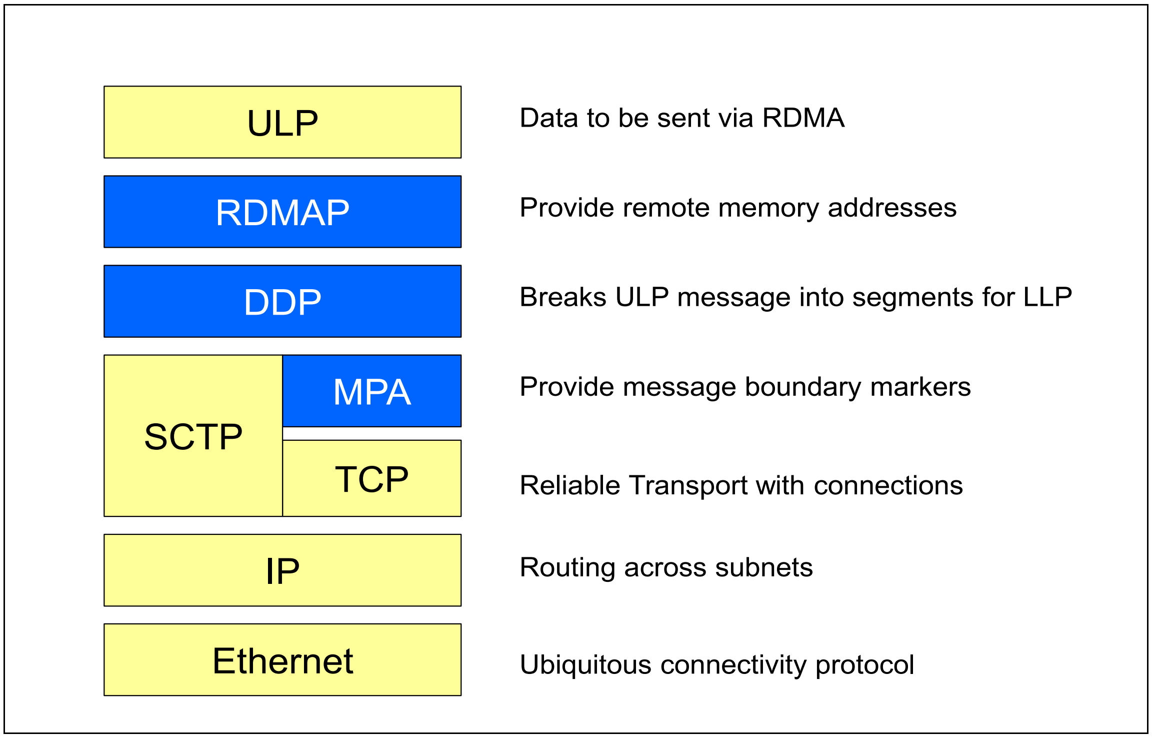

ULP
Data to be sent via RDMA
RDMAP
Provide remote memory addresses
DDP
Breaks ULP message into segments for LLP
SCTP
MPA
Provide message boundary markers
TCP
Reliable Transport with connections
IP
Routing across subnets
Ethernet
Ubiquitous connectivity protocol

Figure 169: iWARP layers

The ULP references all layers above iWARP. It represents the application and operating system. The ULP builds the data that is to be sent across the fabric.

Ethernet, and most other current transports keep the host memory addresses in the host and allow the HBA to access the host memory to store (read commands) or fetch (write commands) the data.

RDMA transfers that responsibility to the remote device, therefore the remote device must have the actual memory address. The RDMAP block sends a header to the remote device with the memory addresses of the host memory space. DDP breaks the ULP messages into sizes for the LLP. MPA is used over TCP, but not over SCTP. It provides markers showing where one message ends and the next one begins.

Buffers

Tagged Buffers

When using a Tagged Buffer, the data sink (data receiver) will advertise the location of the buffer in its memory, and assign a Steering Tag (STag) for the data stream. This memory address may be in system memory, but more likely in application memory. How the data sink sends the information to the data source is outside the DDP Specification. Because the data is transferred directly to the proper location in the application's memory, there is no need for the operation system to verify the data or copy it to the application memory space. This is called zero copy.

Each lower level frame will include in its header the STag and an offset in that buffer. The data will be sent by the data source in order, but if it gets out of order in the transmission, it can be reassembled in the application memory. See Figure 172 for the DDP header for a Tagged Buffer.

Each data stream will have a different STag.

Untagged Buffers

If the data sink does not advertise a buffer with STag to the data source, the data source will use the Untagged Buffer header. Figure 173 shows the DDP header for an Untagged Buffer. The Queue Number will identify the buffer. The Message Sequence Number, starting at 1h, identifies which message this is in this queue. The Message Offset is where in the buffer this DDP segment is to be placed.

For a single ULP message, each DDP segment must have the same Queue Number and the same Message Sequence Number. Within each ULP message, the Message Offset will be different.

Operation Details

RDMAP

The Remote Direct Memory Access Protocol (RDMAP) adds the header shown in Figure 170. It has the 64-bit memory address for the source and destination, an Steering Tag (STag) to identify the source buffer and another one to identify the destination buffer, and a length in bytes. This allows the end device to control the transfer to or from the host memory address space.

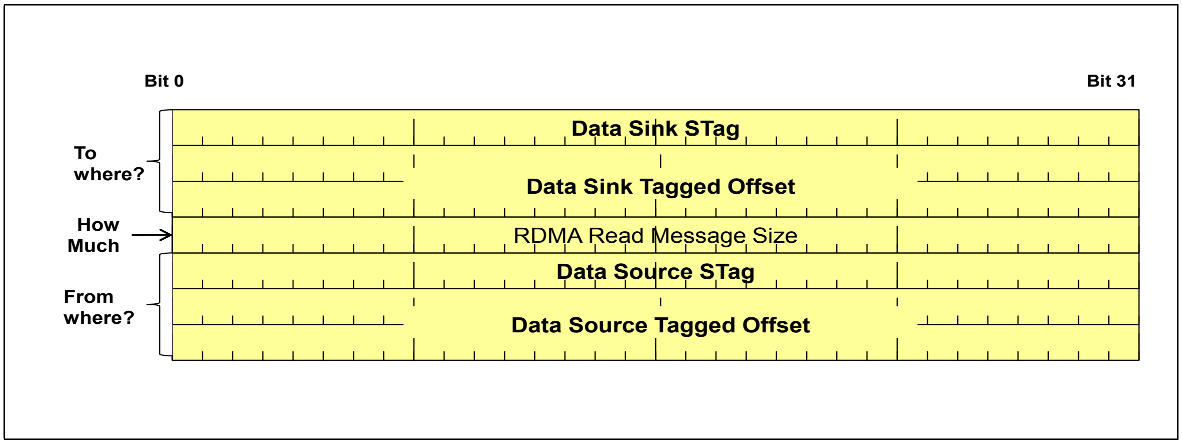


Figure 170: RDMAP Header

DDP

The Direct Data Placement (DDP) software breaks the message from the ULP into sizes that the LLP can handle. To identify which buffer this DDP segment is a part of, and where it goes in the buffer, DDP adds a header. See Figure 171 for a diagram of the segmentation, Figure 172 for the Tagged Buffer header, and Figure 173 for the untagged buffer header.

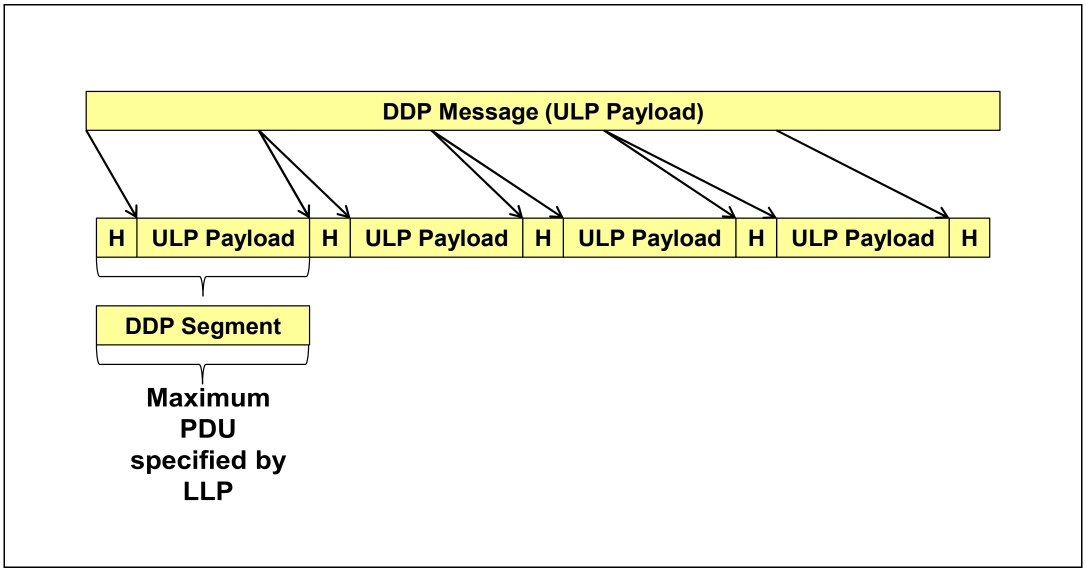


Figure 171: DDP Segmentation and Reassembly

Figure 172 shows the DDP header when using Tagged Buffers.

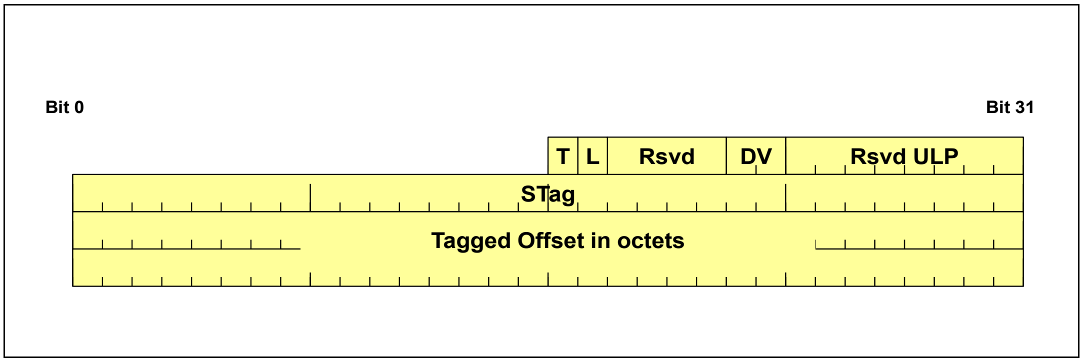


Figure 172: DDP Header - Tagged Buffer

The STag (Steering Tag) identifies the buffer and is the same value at in the RDMAP header. The Tagged Offset in octets is where in the buffer does this DDP Segment go.

Figure 173 shows the DDP Header for untagged buffer. Because there could be DDP segments destined to different buffers mixed in the incoming stream, there has to be some method to identify which buffer and where in that buffer this data is to be placed.

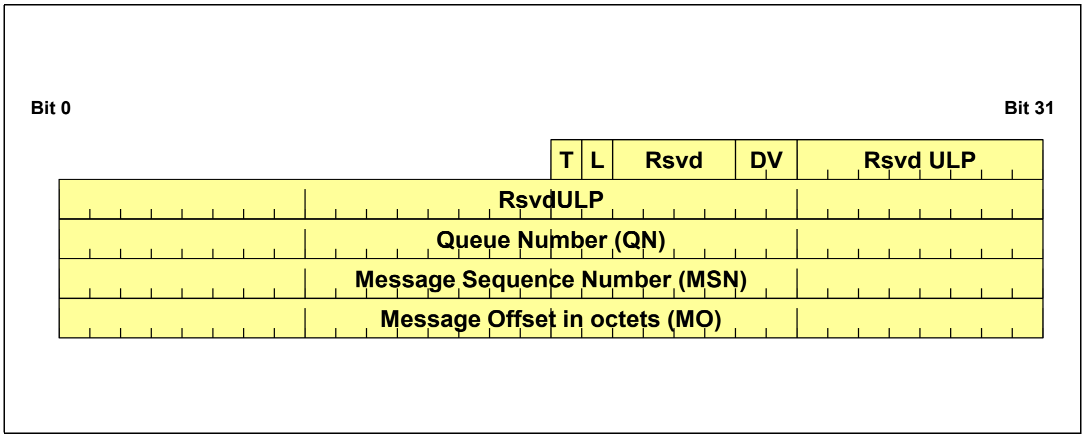


Figure 173: DDP Header - Untagged Buffer

In the Tagged and Untagged Buffer header:

T = Tagged Buffer
L = Last segment of this message
DV is version
RSVD ULP is reserved for the ULP and will be sent unmodified by the DDP.

MPA

The Marker Protocol data unit Aligned (MPA) framing software fits between the DDP and TCP layers. It is not used if Stream Control Transmission Protocol (SCTP) is being used instead of TCP.

In MPA specification RFC 5044, an Upper Layer Protocol Data Unit (ULPDU) is a DDP Segment. A DDP Segment consists of a DDP header and up to the maximum size the transport can handle in a single frame.

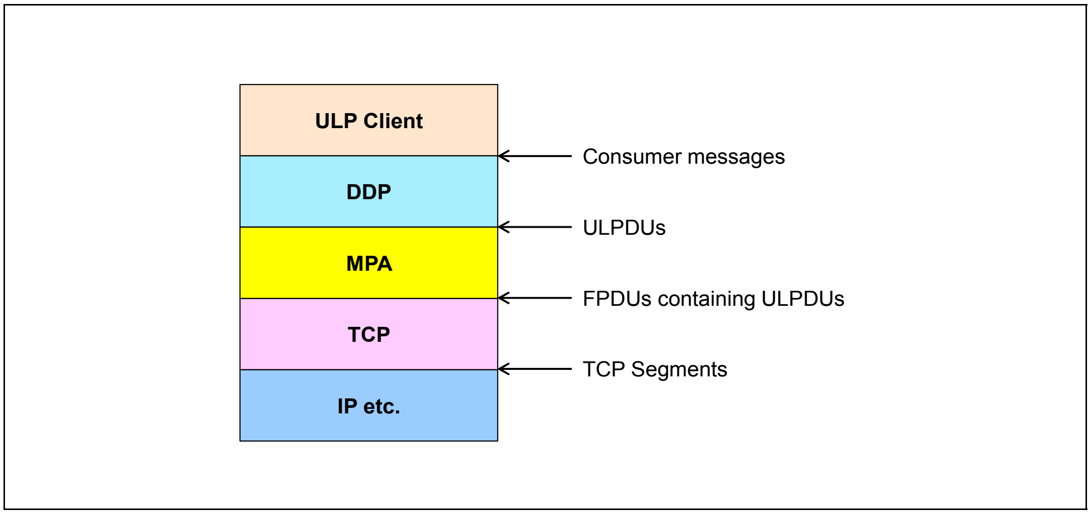


Figure 174: Layers as known by MPA

Chapter Summary

This chapter is not intended to be a comprehensive tutorial of Ethernet or the Internet Protocol Suite, but only to introduce Ethernet, the Protocol Suite, RoCE, and iWARP to you. It covers the messages and packets, the Network and Transport layers and then adding RoCE and iWARP to Ethernet.

Initially this chapter also covered addressing (URL to IP to MAC) and RAPR, but it was suggested that it should be removed. If you would like to see that in this chapter, please notify me at hcurley@indra.com.

Chapter Contents

Fibre Channel Basics for NVMe

Chapter Contents

Overview

Documents

NVMe documents are available at NVMExpress.Org
NVM Express 1.2.1 Gold
NVMe over Fabrics 1.0 Specification

Fibre Channel documents are available at www.t11.org
Fibre Channel specifications available for purchase
FC-NVMe Rev 1.13

Introduction

Note: This chapter is not an in-depth study of Fibre Channel. It is intended to give the reader an overview of how Fibre Channel works for the purpose of understanding NVMe over Fabrics over Fibre Channel.

Fibre Channel was originally started in 1988 as a high-end interface between data processing end devices (Initiators and Target/Logical Units in SCSI) that would have the fastest transfer rate, longest distance reach, very reliable, and greatest flexibility. Over time it had the greatest acceptance for high-end storage devices in general purpose, enterprise computing environments.

Today, there are three well-established, high-end interfaces (Fibre Channel, InfinBand, and Ethernet including iSCSI)[39]. There is much collaboration between the standards committees who define these interfaces, and also between the engineers who develop the high speed components.

From the beginning of Fibre Channel, the interconnect has been called the fabric. The fabric consists of:

- Switches,
- Directors (high end, highly reliable switches), and
- Media:
  - Copper
  - Optical

Figure 175 shows an example multi-switch fabric. Switches currently have up to 384 ports each.
N-Ports are Node Ports (Initiator or Target)
F-Ports are Fabric Ports (Switch connection to N-Port)
E-Ports are Expansion Ports (Switch to Switch connection).

[39] Perhaps PCIe with Thunderbolt and Display Port should be named here also. However the three listed above support storage devices which Display Port is not designed to do. Thunderbolt has not received the widespread acceptance yet of the three listed above.

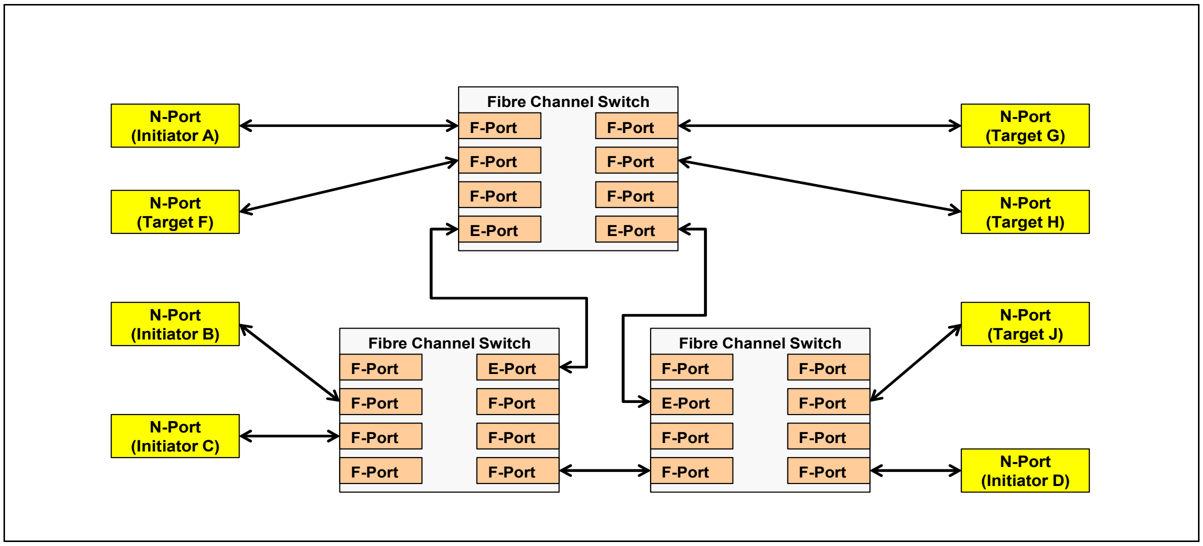


Figure 175: Multi-Switch Fabric

Initialization of Fibre Channel

Following the power on, each device will initialize internally, checking its internal operation and loading operating state registers. This is all device specific and no signals will appear on the interface.

Devices will perform speed negotiation on each physical link. Transmitter training, if performed, is requested by Administrative action or the speed negotiation state machine.

If the fabric has more than one switch, perform selection of Principle Switch. Principle Switch will then set the Domain ID for each switch.

Please reference Figure 176. Each N-Port will do a FLOGI (Fabric Login) to its connected F-Port using the Extended Link Service (ELS)[40]. The N-Port will communicate its parameters; the F-Port will respond with the operational parameters to be used (the best between the N-Port and the F-Port), and the N-Port ID to be used as the source ID.

Both devices will initialize their buffer-to-buffer credits and buffers.

Initiator N-Ports will then send queries to the Fabric requesting Topology information. The Fabric will reply with a list of N-Ports attached to each switch. The Fabric also communicates certain characteristics of each device.

N-Ports will perform PLOGI (Port Login) with each N-Port it wishes to communicate.

Each N-Port will perform a Process Login if required for the FC-4 protocol. NVMe requires either an explicit or implicit Process Login between connected N-Ports. Explicit process login is done with a Fibre Channel Link Service PRLI; Implicit Process Login is done by some means outside the scope of Fibre Channel Standards.

[40] One N-Port may be connected to another N-Port making a point to point link. However, it would have no switches and therefore not be a fabric.

The Fabric is now ready for normal operations.

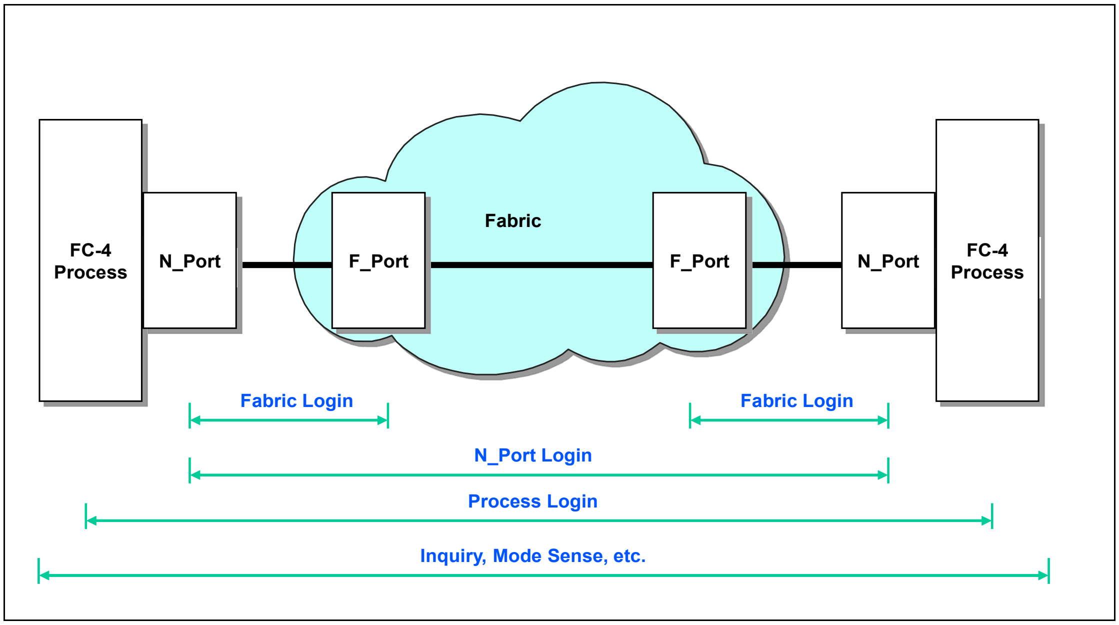


Figure 176: Levels of Login

Initialization of Upper Layer Protocol (ULP)

After the nodes have completed the initialization protocol for Fibre Channel the nodes will initialize the ULP. If it is SCSI, the initialization would include:

Test Unit Ready,
Inquiry,
Mode Sense,
Mode Select,
Report LUNs, etc commands.

If the ULP is NVMe, the initialization would include:

Fabric Command Connect to create the queues,
Admin Commands:
Identify Controller and
Identify Namespace,
Set Features, etc.

Operation in Fibre Channel

The host will create a work activity for the storage device. This would be called a command in most protocols. The FC HBA[41] driver will build an information unit (IU) and put it in the payload of the FC frame. The FC HBA driver will then build the header, optional headers and CRC for the FC frame. The FC HBA driver will then send the frame to the FC hardware which puts the Frame delimiters around the frame.

[41] Fibre Channel Host Bus Adapter

The FC hardware port will verify that enough buffers exist in the peer port at the other end of the electrical or optical cable. If enough buffers exist, the FC port will send the frame.

The switch that received the frame will look at the frame header, decide the necessary routing, and forward the frame toward its destination.

Figure 177, below shows the Fibre Channel Frame. The frame delimiters are green. The header specifies the type of frame and the source and destination address. The Payload contains the Fibre Channel Information Unit (if operating in Fibre Channel mode) or the NVMe capsule (if operating in NVMe over Fabrics mode). The CRC is the frame check sequence.

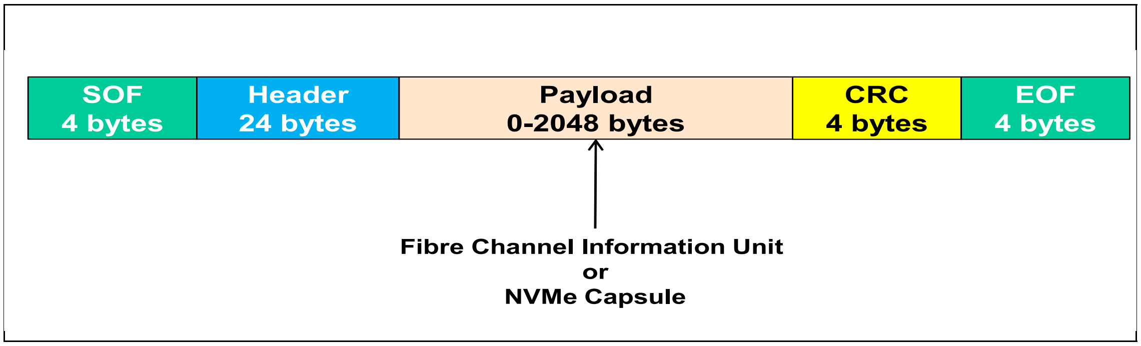


Figure 177: Fibre Channel Frame

Selected Details

Classes of Service

One of the ways that Fibre Channel tried to be everything for everybody was the definition of classes of service. As originally defined, there were dedicated connections with and without confirmed delivery, virtual connections, datagrams, multicast and internal Fabric only classes. Class 3 and Class F (see below) were by far the most widely used classes and eventually the lesser used classes became obsolete.

The remaining classes are all connectionless, meaning:
they contain the source and destination address in each frame,
they do not:
create a connection before sending frames, and
do not reserved necessary resources for one connection only,
but they may:
encounter routing delays, and
route the frames out of order.

Class 2

Class 2 is connectionless, but does require acknowledgment for each frame sent and received.

Class 3

Class 3 (also called Datagram) is connectionless, but data is not confirmed. If a frame is lost or corrupt in transfer, it is up to an upper layer to resolve the issue.

Class F

Class F is internal to the fabric only and is similar to Class 2 (multiplexed and confirmed delivery).

Fibre Channel Vocabulary

Node is an end device (e.g. server, disk, disk array, tape, HBA) with one or more ports. Nearly all Fibre Channel Nodes contain at least two ports for fail over or performance enhancement.

Port is a connection point of a node, a link control facility.

Physical Link is the transmitter and/or receiver and the fibre that connects them.

Frame is the smallest data structure that can be sent in Fibre Channel, may contain up to 2K bytes of information.

Sequence is one or more frames that makes up an information unit, always uni-directional. Non-data sequences are single frame only, data sequences may have one or more frames

Exchange - one or more sequences that makes up an server request. Exchanges may be uni-directional or bi-directional.

Information Unit - a command, data, response or control sent as part of a frame.

Server Request - the command, data, and ending status to process a command. Called an I/O Process in SCSI-2. Server Request is defined in SAM-6.

Word - a string of four contiguous bytes within an encoded frame occurring on boundaries that are zero modulo 4 from a specified reference (low order two address bits are 00b).

Flow Control

Flow control is credit based as is flow control in PCIe. However, the implementation is VERY different.

Buffer-to-Buffer Flow Control

Buffer-to-Buffer flow control is from one device, across one physical link, to the peer device. Each pair of devices attached by a single link will keep track of available buffers using Buffer-to-Buffer flow control. The mechanism is called Credit Based Flow Control. In Fibre Channel, each credit indicates one frame buffer of 2K bytes - the maximum size payload for Fibre Channel frames.

The sender of frames has a counter that counts up for each RRDY primitive[42] received and counts down for each frame sent. Using this, the frame sender knows how many frame buffers the receiver has available.

[42] Primitive are 40-bit control characters that primarily work from the link layer of one device, across the physical link, to the link layer of the next device. Primitives are not frames.

Figure 178 shows the counter in yellow. The contents of that counter in the sender of frames is the number of frame buffers available at the receiver. The numbers on the right are the sequential steps of RRDYs and Frames sent.

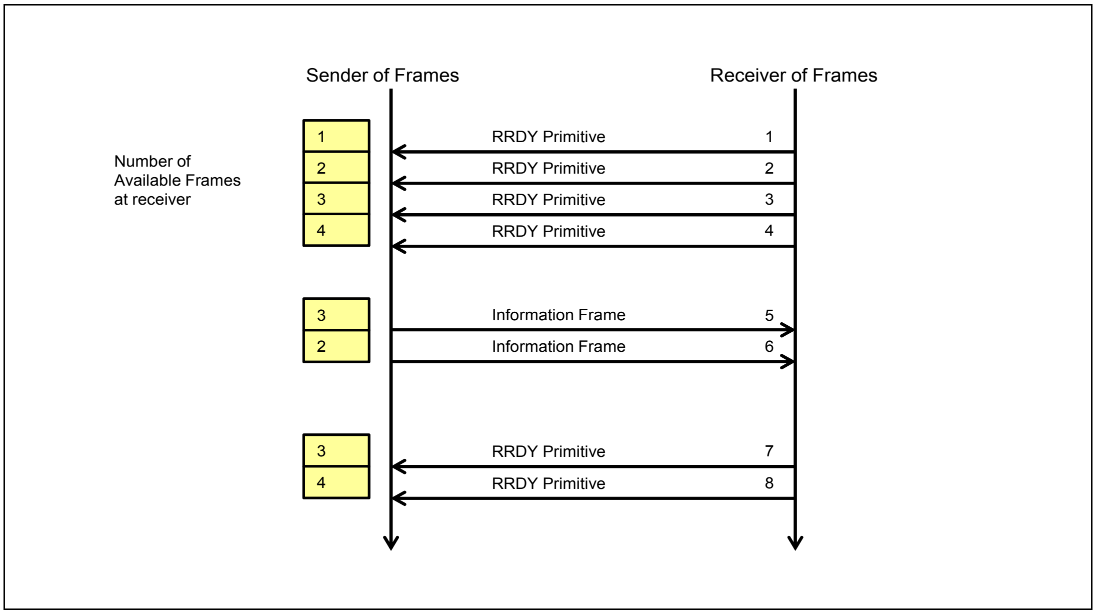


Figure 178: Flow Control

Buffer-to-Buffer Flow Control is used for all classes of frames.

End to End

When using Class 2 or Class F (the acknowledged classes) End-to-End flow control is also used. The end devices agree on a number of buffers. For each frame sent, the sender decrements its counter. For each ACK received, the sender of frames increases its counter.

Flow Control Initialization

As part of initialization for Fibre Channel, the Node Ports (N-Ports) will do a Fabric Login (FLOGI) to the connected switch. This included an Extended Link Service with the Node Port operational parameters; the Switch will respond with the best parameters that both devices can operate.

After all the N-Ports have performed the FLOGI, they will log in with other N-Ports with a Port Login (PLOGI) and exchange parameters between end devices.

One of the parameters for both FLOGI and PLOGI is the Buffer-to-Buffer Credit.

A parameter for PLOGI in Class 2 only is the End-to-End Credit.

Exchanges, Sequences, and Frames

Figure 179 shows the relationship between Exchanges, Sequences and Frames. Each frame and each sequence is uni-directional only. Exchanges may be bi-directional.

Sequence Initiative

The Sequence Initiator bit controls which N-Port is allowed to initiate the next Sequence for the Exchange. The Sequence Initiator may **hold** the initiative to transmit the next Sequence of the Exchange, or the Sequence Initiator may **transfer** the initiative to transmit the next Sequence of the Exchange. The decision to hold or transfer initiative shall be indicated by Sequence Initiative bit in Frame Control field in the FC Header.

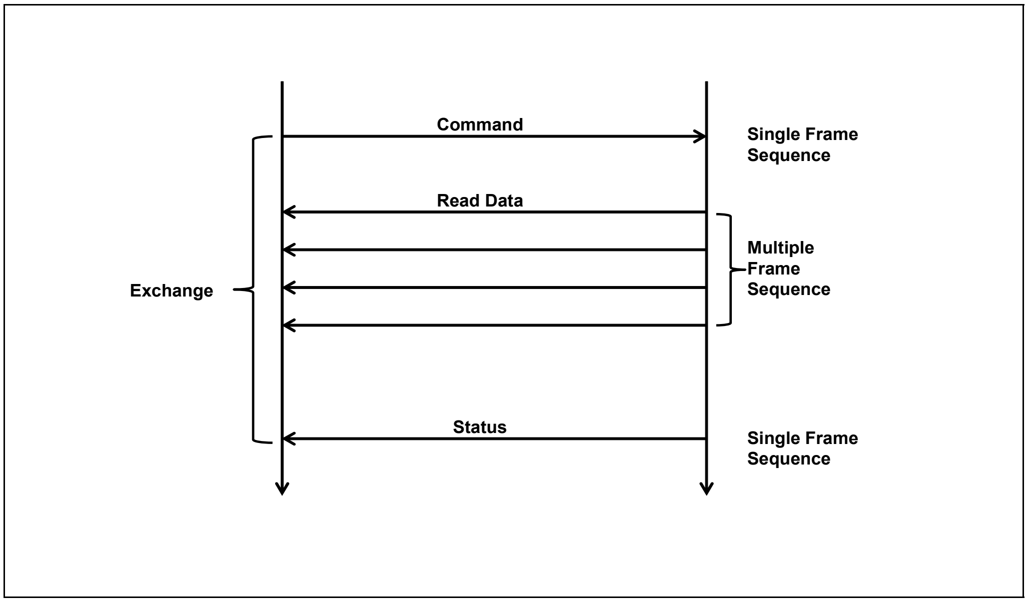


Figure 179: Exchanges, Sequences, and Frames

In Figure 179, the originator/initiator that sent the Command Frame would transfer the Sequence Initiative. If the command was a read type of command, the responder/target would then transfer the Read Data Frames and hold the Sequence Initiative. When transferring the Status Frame, the responder would transfer the Sequence Initiative.

F/M/L

Also in the Frame Control field of the FC Header are the First and Last bits.

The First bit indicates that this frame is part of the First Sequence of the Exchange or is not a part of the first frame. It is set for all frames that are part of the first Sequence of the Exchange.

The Last bit shall be set to 1b on the last frame of the last Sequence of the Exchange. It may be set to 1b on other frames of the last Sequence of the Exchange. However, once set to 1b it must remain set on all subsequent frames of the Sequence.

In Figure 179 above, the Command frame would have the F bit set, the read data frames would have neither the F or L bit set, and the Status frame would have the L bit set.

Fibre Channel Levels

FC-4

FC-4 defines the mapping of the Upper Level Protocol (ULP) to Fibre Channel. This is the layer that defines how NVMe Over Fabrics maps to FC constructs.

FC-3

FC-3 is "common services." Services such as Basic Link Services and Extended Link Services (ELS) are in the FC-3 level.

FC-2

FC-2 is the data transport level. It defines the exchanges, sequences and frames. FC-1 and FC-2 are defined in the FC-FS standards

FC-1

FC-1 defines the encoding and decoding of characters, special characters and primitives. FC-1 performs Forward Error Correction (FEC) for 32GFC.
FC-FS-4 specifics encoding as 8b/10b (each 8 bit byte is encoded into a 10 bit character), 64b/66b (each 8 bytes are pre-pended with a two bit synchronization header), and 256b/257b (16 bytes encoded and scrambled with a pre-pended single bit header).

FC-0

FC-0 defines the physical media, connectors, cables and speeds. FC-0 is defined in the FC-PI standard.

Frame contents

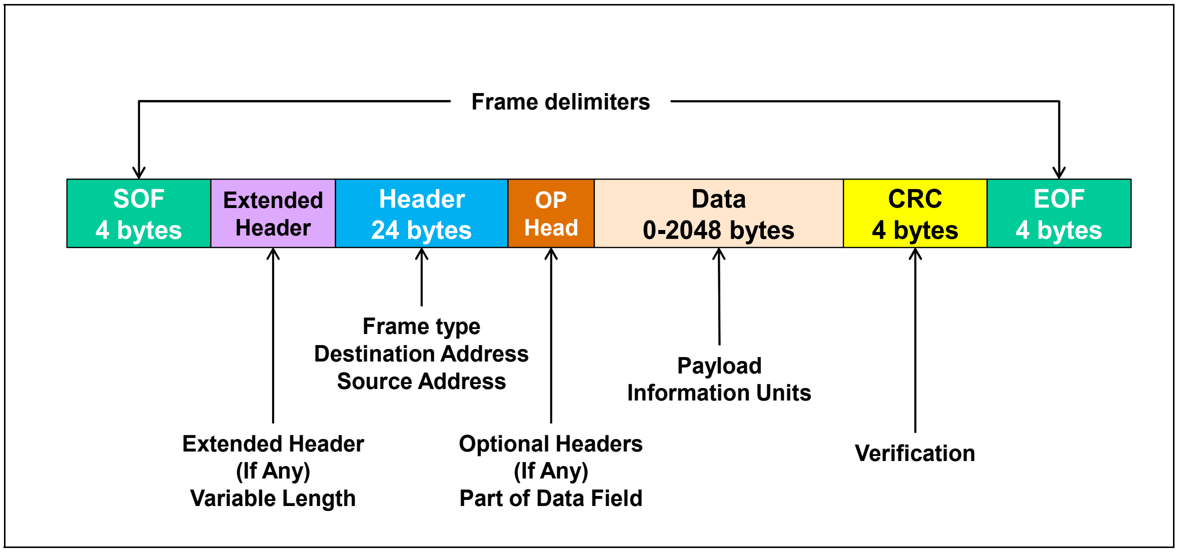


Figure 180: FC Frame Contents

SOF - Start of Frame delimiter identifies the class of service and is used for Sequence control. Examples are:
Start of Frame Initiate Class 2,
Start of Frame Normal Class 3, and
Start of Frame Fabric.

Extended Header supports special functionality of FC-2. Their use will be further defined here if they are used in NVMe Over Fabrics.

The header contains the type of information, the source and destination addresses, and information to reorder data that gets our of order in the fabric.

Optional Headers are special use information in the data field. Their use will be further defined here if they are used in NVMe Over Fabrics.

The Data field is the Payload and contains the command, data, status or control Information Unit from the upper layer protocol.

CRC - verifies the preceding data for integrity. Check includes the Extended headers, header, Optional headers, and data fields. CRC does not include the frame delimiters. CRC is calculated before encoding and scrambling.

EOF - End of Frame delimiter indicates the end of the frame and additional processing necessary. Examples are:
EOF Normal,
EOF Terminate (sequence associated with this sequence ID is complete,
EOF Abort (partial frame transmitted due to a link malfunction, no reply, and
EOF Invalid (replaces EOF Normal or EOF Terminate if the frame contents is invalid).

Frame Header

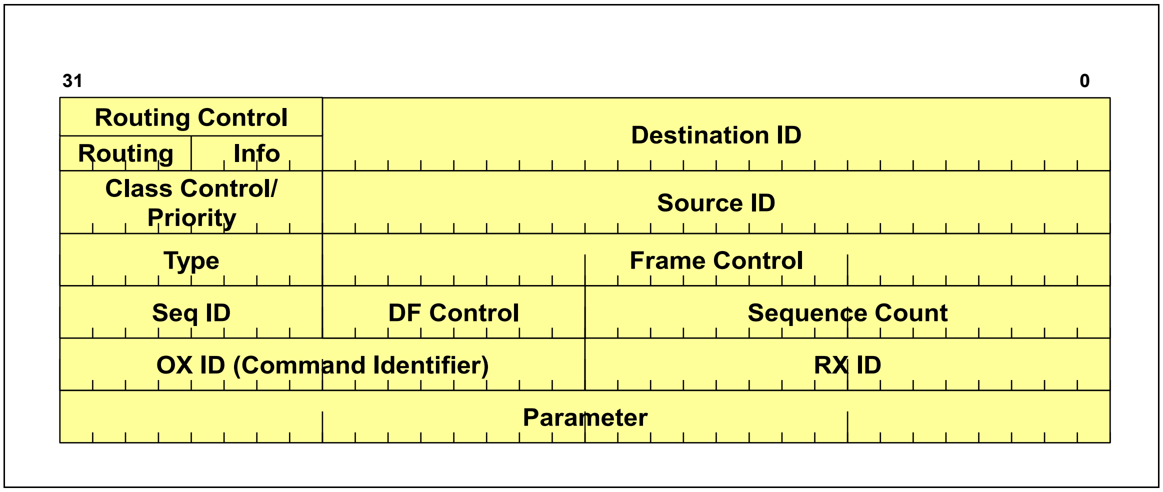


Figure 181: FC Frame Header

Routing Control gives the fabric and receiver a general idea of the type of information being sent in this packet, such as Link Control, ULP Unsolicited Control (command), Solicited Data, or Video.

Destination ID and Source ID are the 24-bit FC addresses of the end devices.

The Type field for NVMe over FC is set to:
08h for Frames of NVMe over FC Exchanges using IU types specified in Table 59 or Table 60, or

28h for all other NVMe over FC frames.[43]

The Frame Control is a bucket of bits that assist in processing this frame. Examples of bits are: this frame is from the originator of the Exchange or the responder of the Exchange. This frame is the first, middle, or last frame of this sequence. The ACK form is ACK 1 (ACK every frame), or ACK 0 (one ACK for the Sequence). There are 0, 1, 2, or 3 fill bytes.

The Seq ID and Sequence count identify the sequence for this frame.

DF Control identifies if any optional headers are present in the data field.

OX ID is Exchange Originator assigned ID. This is similar to SCSI or SATA Queue Tag or Command Identifier.

RX ID is Exchange Responder assigned ID.

The Parameter field carries additional information required based on the Type field.

Mapping NVMe onto Fibre Channel

General Information

Commands and responses may get out of order in the fabric. To ensure fused commands are processed in the proper order, a Command Sequence Number (CSN) is added. For ERSP, a Response Sequence Number (RSN) is added. Good responses (SCT = 0h and Status Code = 00h) are processed as soon as they are received by the initiator.

The CSN is 32-bit counter that is set to zero for each NVMe connection and incremented for each command. The next command after it reaches its maximum value causes the counter to wrap back to zero. Receipt of each command is indicated by:

an NVMe XFER RDY,
The first Data Frame for the command,
A response or Error response,
An ACK, or
An ABTS.

The RSN is a 32-bit counter that is set to zero for the first NVMe ERSP for each connection and is incremented for each ERSP. ERSP is sent:

when the target detects an error,
when the target port SQ is 90% or more full (to update the SQ Head pointer),
when the preset ERSP ratio is met (to update the SQ Head pointer),
in response to a fused command, or
when the target port determines it wants to send one.

Link Services

FC standard FC-LS defines about 70 Extended Link Services provided to assist in connectivity and operation.[44] FC-NVMe adds three new Link Services and carries forward two others.

[43] See FC-FS-5 on www.t11.org for the full list.

ELS Command Code	Description	Abbreviation
01h	NVMe LS Reject	NVMe RJT
02h	NVMe LS Accept	NVMe ACC
03h	Create Association	CASS
05h	Create I/O Connection	CIOC
05	Disconnect	DISC

Table 58: NVMe Link Services

These can be identified as FC-NVMe by the FC header fields (See Figure 181 on page 274):
Routing Control.Routing = 3h,
Routing Control.Info = 2h for request and 3h for response, and
Type = 28h (FC-NVMe FC-4 Link Service Frame).

FC Header for NVMe IUs

Please refer to Figure 181, Table 59, and Table 60.

The left most column in the tables are identifiers for that line in the table. They are not used in the protocol.

Routing Control is the value for the first byte of the header.

The Content field is what goes into the FC Frame payload as the NVMe Capsule.

The F/M/L field is defined in “F/M/L” on page 272. The SI field is defined in “Sequence Initiative” on page 272. The M/O field is Mandatory or Optional.

IU	Description	Routing Control	Content	F/M/L	SI	M/O
T1	Command	06h	NVMe CMD	F	T	M
T2	Command	06h	NVMe CMD	F	H	O
T3	Data Out	01h	NVMe Data	M	T	M
T4	Confirm	03h	NVMe Confirm	L	T	O

Table 59: NVMe IUs sent to Targets

T1 is the normal command sending sequence. On read commands, the SI is transferred because the next sequence is for the target to send the read data. On write commands, the SI is transferred because the next sequence is for the target to send a XFER RDY.

T2 is used if the NVMe XFER RDY is disabled, such as on the first transfer for a write command.

[44] Basic Link Services are defined in the FC-FS documents. FC-4 Link Services are defined in the FC-4 documents.

IU	Description	Routing Control	Content	F/M/L	SI	M/O
I1	Data Out request	05h	NVMe XFER RDY	M	T	M
I2	Data In	01h	NVMe Data	M	H	M
I3	Response	07h	NVMe Response	L	T	M
I4	Extended Response	08h	NVMe Error Response	L	T	M
I5	Extended Response with Confirmation Request	08h	NVMe Error Response	M	T	O

Table 60: NVMe IUs sent to Initiators

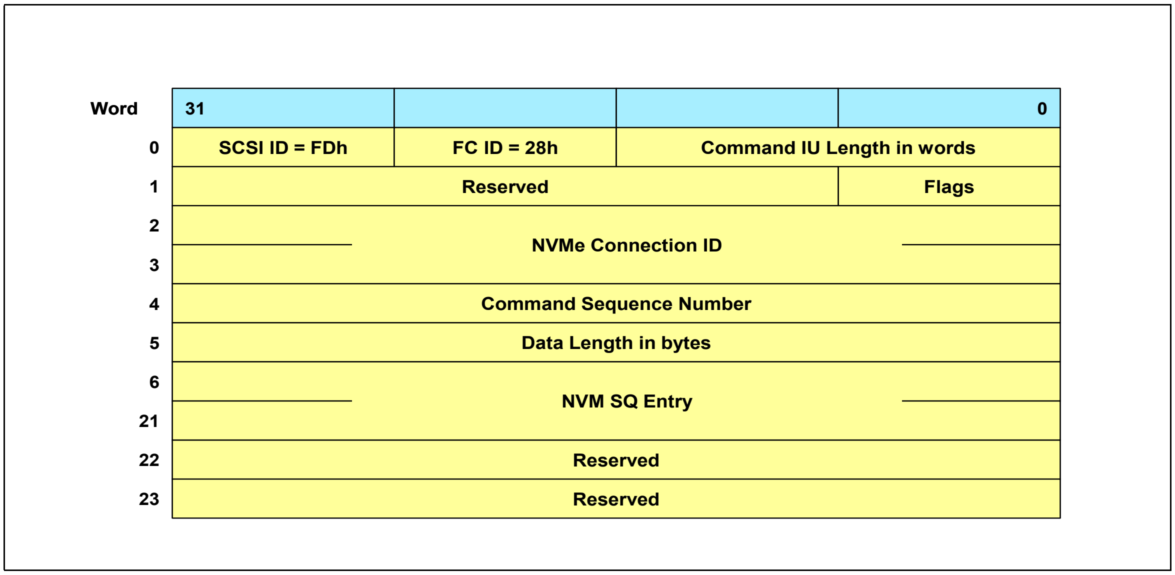
Fibre Channel defines a process where the target can request a confirmation from the host when it receives the status. That request is indicated by I5 and the Confirmation is indicated with T4.

FC Payload

The following five data structures are not intended to duplicate the FC-NVMe standard, they are shown here for ease of reading this material. Each of these IUs is preceded by SOF and the 24-byte header, and followed by the CRC and EOF.

The following five Information Units are patterned after Fiber Channel Protocol (FCP-4) for SCSI commands. Changes are made to adapt them to NVMe.

NVMe Command IU



Word	31			0
0	SCSI ID = FDh	FC ID = 28h	Command IU Length in words	
1	Reserved			Flags
2	NVMe Connection ID			
3				
4	Command Sequence Number			
5	Data Length in bytes			
6	NVM SQ Entry			
21				
22	Reserved			
23	Reserved			

Figure 182: Command IU

The “F” in SCSI ID = FDh means that it uses an extended logical unit address method with a length field of 11b; the Dh means the extended addressing method is defined in Fibre Channel FC-FS-5.

The FC ID = 28h is defined in Fibre Channel to mean NVMe over Fabrics over FC.

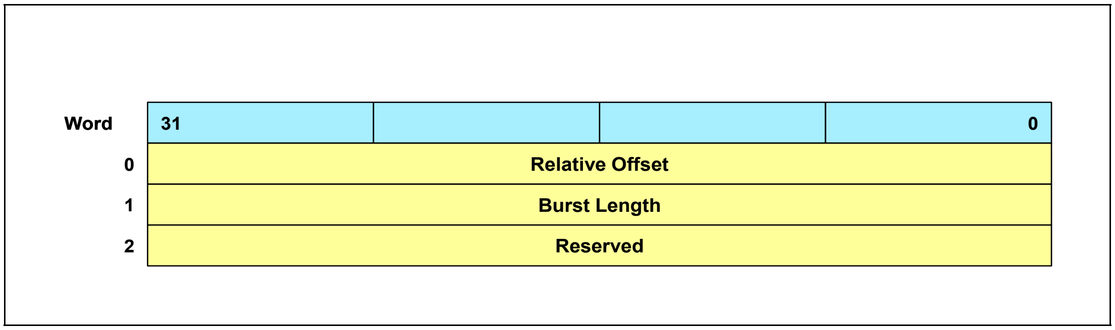
The Flags bits
    0 = Write operation
    1 = Read operation

The Command Sequence Number is explained in “General Information” on page 275.

The NVM SQ Entry is detailed in “Common Command Fields” on page 51, “Admin Commands” on page 69, and “NVMe Commands” on page 125.

NVMe XFER RDY IU

The XFER RDY Information Unit from the target tells the initiator what write data the target can accept (Relative Offset) and how much (Burst Length). The target may send multiple XFER RDY IUs, but the total length for them must equal the length specified in the write command.



Word	31			0
0	Relative Offset			
1	Burst Length			
2	Reserved			

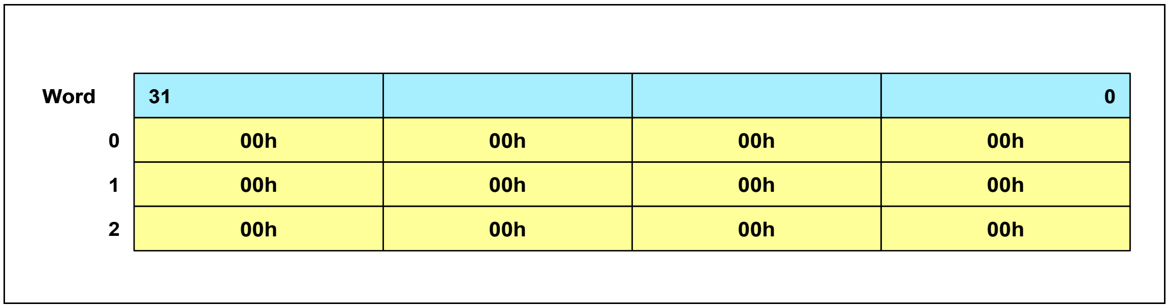
Figure 183: XFER RDY IU

NVMe Data IU

There is no format for the NVMe Data IU. It is just an integer number of 4-byte words.

NVMe RSP IU

The Response (or RSP) IU is used to report good status is shown in Figure 184. Good status is defined as SCT = 000b and Status = 00h. The identification of which command this status references is in the FC header as the Source and Destination ID and OX_ID which is equivalent to the tag.



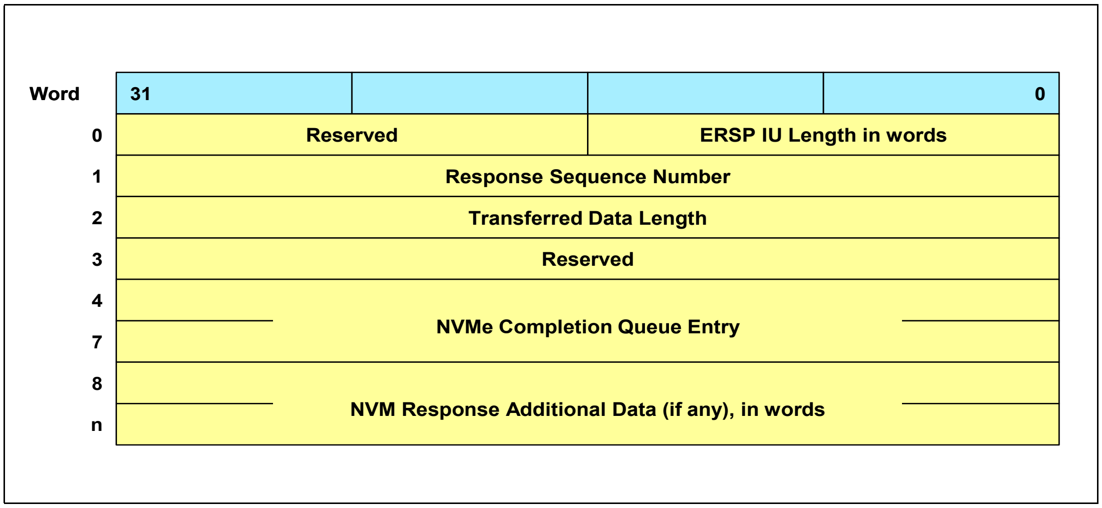
Word	31			0
0	00h	00h	00h	00h
1	00h	00h	00h	00h
2	00h	00h	00h	00h

Figure 184: RSP IU

Good status is efficient, but it does not give the SQ head pointer update. For that, use the ERSP IU.

NVMe ERSP IU

The NVMe ERSP is the status presented when an error occurs or when it is necessary to update the SQ Head Pointer. The NVMe Completion Queue Entry, Figure 37 page 65, shows the Command Identifier, SQ Head Pointer, Status Code Type, and Status Code. The SQ ID is not necessary in NVMe over Fabrics because the SQ and CQ have a one-to-one relationship and both have the same ID. Whatever the ID of the CQ this status arrived in is the same ID as SQ that sent the command.



Word	31			0
0	Reserved		ERSP IU Length in words	
1	Response Sequence Number			
2	Transferred Data Length			
3	Reserved			
4 7	NVMe Completion Queue Entry			
8 n	NVM Response Additional Data (if any), in words			

Figure 185: ERSP IU

The ERSP IU must be sent:

every "n" completions where "n" is the ratio established when the queue was created,
on any completion error,
when the SQ is 90% full, and
when the target thinks it appropriate.

NVMe Confirmation IU

There is no payload for Confirmation. It flows from the initiator port in answer to a NVMe RSP IU or a NVMe ERSP IU to let the target port know that the response was received by the initiator.

Discovery Service

Fibre Channel Standards, NVMe Specifications, NVMe Over Fabrics Specifications, and FC-NVMe Standard provide no information about how the Discovery Service is to be implemented (nor should they). It may make sense to begin by looking at the Fibre Channel Director Server at FC address FF FF FCh. This optional server within the fabric will probably be present in all medium to large switched FC networks. It has within it a Name Server and an IP Address Server, both of which may be queried.

Some information is placed in the Directory Server during a Fabric Login (FLOGI). Other information must be explicitly placed in the Directory Server. Other nodes can read the information to discover what nodes are available. For instance an NVMe host may login to the Directory Server and issue a Get Log Page 70h command to discover what NVMe devices are on the network.

Discovery Service Steps (Informative)

1. Perform FLOGI to log into the Fabric.
2. Perform PLOGI to the Name Server at well-known address of FF FF FCh.
3. Register your information with the Name Server.
4. Register for RSCN (Request State Change Notification) when the topology changes.
5. Request a list of devices with Type code set to 28h, see FC-GS-8.
6. For each device returned
    1) Perform PLOGI with that device,
    2) Perform PRLI (process login), and
    3) Issue NVMe Get Log page.
7. For each log page with Subsystem type = 01h, perform step 6 to find additional devices.[45]

FC-NVMe rev 0.7 defines the Discovery Log Page for NVMe over FC. See Figure 152 on page 224.
Transport type = 02h
Address Family = 04h
Subsystem Type = 01h for referral or 02h for NVM subsystem
Transport Service ID = none
Transport address = “nn-WWNN:pn-WWPN” where
WWNN is World Wide Node Name of the target port, and
WWPN is World Wide Port Name of the target port.
Transport Specific Address Subtype shall be set to all zeros.

Summary

Fibre Channel is a high speed, high reliable, long reach, high end transport primarily used in Enterprise data centers for connecting Storage Area Networks (SANs). It does not support RDMA. FC has been in successful operation for two decades and has proven its reliably and speed.

[45] There should be no additional devices because the FC Name Server should have all the devices in the domain. Checking the link to additional NVMe Discovery Services simply verifies that there are no more devices.

FC-NVMe is a standard to map NVMe queues and commands into Fibre Channel frames, taking advantage of the highly efficient NVMe queues and NVMe commands. When this text was written, the FC-NVMe standard was in the process of being created and this information is incomplete and may change. Check the latest FC-NVMe standard from www.t11.org.

Chapter Contents