



Intel® Virtual RAID on CPU (Intel® VROC) IOCTLs

User Guide

IOCTLs for NVME pass-through and NVMe RAID member disks.*

Revision 1.0

September 2022

Intel Confidential



You may not use or facilitate the use of this document in connection with any infringement or other legal analysis. You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

All product plans and roadmaps are subject to change without notice.

The products described may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at [intel.com](https://www.intel.com).

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

© Intel Corporation. Intel, <Intel launch names, if mentioned in the slides below> the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries.

*Other names and brands may be claimed as the property of others.

Copyright© 2022, Intel Corporation. All rights reserved.

Contents

1	Introduction	6
1.1	Intel® Volume Management Device (VMD) Overview	6
1.2	Journaling Drives	6
2	NVMe Admin Commands.....	7
2.1	List of Supported Admin Commands.....	7
2.2	List of Supported Data Commands	7
2.3	NVMe SMART/Health Information.....	7
2.4	Identify NVMe	8
2.5	Asynchronous Events	8
2.6	NVMe Firmware Update	8
2.7	NVMe Firmware Activate	8
2.8	Get Features	9
2.9	Set Features	9
2.10	Device Self-test	9
2.11	Format NVM Command.....	9
2.12	Security Send Command.....	9
2.13	Security Receive Command.....	9
3	Public NVMe IOCTLs Support.....	11
3.1	Microsoft Firmware Update NVMe IOCTLs.....	11
3.2	Firmware Update Using SCSI Pass Through IOCTL	12
4	Private NVMe IOCTLs support.....	13
4.1	Private IOCTLs Signature and Control Code Details.	13
4.2	Private IOCTLs Data Structure.....	13
5	Private Intel® VROC IOCTLs	15
5.1	List of Private IOCTLs For RAID Volumes.....	15
5.2	NVME_GET_NUMBER_OF_RAID_VOLUMES	16
5.2.1	IOCTL Signature and Control Code:	16
5.2.2	IOCTL Data Structure:.....	16
5.3	NVME_GET_RAID_INFORMATION.....	16
5.3.1	IOCTL Signature and Control Code.....	16
5.3.2	IOCTL Data Structure.....	17
5.4	NVME_GET_RAID_CONFIGURATION	17
5.4.1	IOCTL Signature and Control Code.....	18
5.4.2	IOCTL Data Structure.....	18
5.5	NVME_GET_RAID_CONFIGURATION	19
5.5.1	IOCTL Signature and Control Code.....	19
5.5.2	IOCTL Data Structure.....	19
5.5.3	Input/Output Data Buffer Details	20
5.6	5.6 NVME_GET_NUMBER_OF_SPARE_DISKS	21
5.6.1	IOCTL signature and control code:.....	21
5.6.2	IOCTL data structure:	21
5.7	NVME_GET_SPARE_DISKS_INFORMATION.....	21
5.7.1	IOCTL signature and control code	22
5.7.2	IOCTL Data Structure.....	22

	5.7.3	Input/Output Data Buffer Structure	22
5.8		NVME_GET_NUMBER_OF_PASSTHROUGH_DISKS	23
	5.8.1	5.8.1 IOCTL Signature and Control Code	23
	5.8.2	IOCTL Data Structure	24
5.9		NVME_GET_PASSTHROUGH_DISKS_INFORMATION	24
	5.9.1	IOCTL Signature and Control Code	24
	5.9.2	IOCTL Data Structure	24
	5.9.3	Input/Output Data Buffer Structure	25
5.10		NVME_GET_NUMBER_OF_JOURNALING_DRIVES	26
	5.10.1	IOCTL Signature and Control Code:	26
	5.10.2	IOCTL Data Structure:	26
5.11		NVME_GET_JOURNALING_DRIVES_INFORMATION	27
	5.11.1	IOCTL Signature and Control Code	27
	5.11.2	IOCTL Data Structure	27
	5.11.3	Input/Output Data Buffer Structure	28
5.12		NVME_PASS_THROUGH_SRB_IO_CODE	29
	5.12.1	IOCTL Signature and Control Code	29
	5.12.2	IOCTL Data Structure	29
	5.12.3	Error Handling	31
6		Example Usage	33
6.1		References	40

Revision History

Document Number	Revision Number	Description	Revision Date
743971	1.0	Initial release	September 2022

§ §

1 Introduction

This document presents all NVMe admin commands supported by Intel® Virtual RAID on CPU (VROC) and describes how to send them to a particular NVMe device managed by the Intel Volume Management Device (VMD). These commands are passed to Intel VROC through the usage of the Intel VROC IOCTLs, outlined in this document. These IOCTLs can be used to get information about the disks in the system as well as sending NVMe private IOCTLs to one of those disks. Also included in this document are code samples showing how to send an NVMe admin command to an NVMe RAID member device.

1.1 Intel® Volume Management Device (VMD) Overview

With the Intel VROC (VMD NVMe RAID) technology, NVMe devices are grouped together under one (or more) Intel VMD domains. This means that all IOCTLs, that target one of these drives, must go through the Intel VMD controller managing the drive. Intel VROC (VMD NVMe RAID) will manage and process the IOCTLs it receives and will respond with the appropriate Return Code. For example, the Return Code can be obtained through the use of the following IOCTLs:

- NVME_GET_NUMBER_OF_PASSTHROUGH_DISKS (Described in 5.8)
- NVME_GET_PASSTHROUGH_DISKS_INFORMATION (Described in 5.9).

1.2 Journaling Drives

Intel VROC (VMD NVMe RAID), configured for premium mode, supports RAID5 write hole closure. The architecture of the feature utilizes a concept of a Journaling Drive to store all the necessary information to close the RAID5 write hole. As a result, this Journaling Drive may be subject to IOCTL commands. The Intel VROC (VMD NVMe RAID) IOCTLs supports getting information from or sending data too the Journaling Drive. For example, IOCTLs to manage a Journaling Drive are:

- NVME_GET_NUMBER_OF_JOURNALING_DRIVES (Described in 5.10)
- NVME_GET_JOURNALING_DRIVES_INFORMATION (Described in 5.11).

§§

2 NVMe Admin Commands

Intel VROC (VMD NVMe RAID) supports the following admin commands.

2.1 List of Supported Admin Commands

Command	Opcode
Get Log Page	0x02
Identify	0x06
Set Features	0x09
Get Features	0x0A
Asynchronous Event Request	0x0C
Firmware Activate	0x10
Firmware Image Download	0x11
Device Self-test	0x14
Format NVM	0x80
Security Send	0x81
Security Receive	0x82
Any vendor specific admin command	0xC0-0xFF

2.2 List of Supported Data Commands

Command	Opcode
Flush	0x00
Write Uncorrectable	0x04
Compare	0x05
Data Set Management	0x09
Any vendor specific NVM commands	0x80-0xFF

2.3 NVMe SMART/Health Information

Self-Monitoring, Analysis and Reporting Technology (S.M.A.R.T.) is an open standard used by hard-drives and hosts to monitor drive health and report potential problems. The SMART and Health Information are collected over the life of the NVMe controller and is retained across power cycles. By default, SMART monitoring is always enabled on NVMe products.

To read values of SMART and general Health Information the application sends the ADMIN_GET_LOG_PAGE IOCTL. Intel VROC (VMD NVMe RAID) uses a private IOCTL

mechanism to pass the ADMIN_GET_LOG_PAGE command to appropriate NVMe device.

2.4 Identify NVMe

The Identify command returns a data buffer that describes the NVMe controller.

To read identification parameters from NVMe device, applications will send the identify command.

Intel VROC (VMD NVMe RAID) uses a private IOCTL mechanism to pass the ADMIN_IDENTIFY command to appropriate NVMe device. This applies to NVMe device in either passthrough mode or RAID mode.

2.5 Asynchronous Events

Asynchronous events are used to notify host software of error and health information as these events occur. To enable asynchronous events to be reported by the controller, the host software needs to issue one or more Asynchronous Event Request commands to the controller.

Intel VROC (VMD NVMe RAID) uses a private IOCTL mechanism to pass the ADMIN_ASYNC_EVENT_REQUEST command to appropriate NVMe device. This applies to NVMe device in either passthrough mode or RAID mode.

2.6 NVMe Firmware Update

The ADMIN_FIRMWARE_IMAGE_DOWNLOAD private IOCTL is used to download firmware image to the controller. The new firmware image will not start to run right after the ADMIN_FIRMWARE_IMAGE_DOWNLOAD command. To select which firmware version will be executed after an NVMe device reset, the ADMIN_FIRMWARE_ACTIVATE command must be used.

Intel VROC (VMD NVMe RAID) uses a private IOCTL mechanism to pass the ADMIN_FIRMWARE_IMAGE_DOWNLOAD command to appropriate NVMe device. This applies to NVMe device in either passthrough mode or RAID mode.

2.7 NVMe Firmware Activate

The Firmware Activate command is used to verify that a valid firmware image has been downloaded and to commit that revision to a specific firmware slot. The host may select the firmware image to activate on the next controller reset as part of this command.

Intel VROC (VMD NVMe RAID) uses a the private IOCTL mechanism to pass the ADMIN_FIRMWARE_ACTIVATE command to appropriate NVMe device. This applies to NVMe device in either passthrough mode or RAID mode.

2.8 Get Features

The Get Features command retrieves the attributes of the Feature specified.

Intel VROC (VMD NVMe RAID) uses a private IOCTL mechanism to pass the ADMIN_GET_FEATURES command to appropriate NVMe device. This applies to NVMe device in either passthrough mode or RAID mode.

2.9 Set Features

The Set Features command specifies the attributes of the Feature indicated.

Intel VROC (VMD NVMe RAID) uses a private IOCTL mechanism to pass the ADMIN_SET_FEATURES command to appropriate NVMe device. This applies to NVMe device in either passthrough mode or RAID mode.

2.10 Device Self-test

The Device Self-test command is used to start the device self-test operation or abort a device self-test operation.

Intel VROC (VMD NVMe RAID) uses a private IOCTL mechanism to pass the ADMIN_DEVICE_SELF_TEST command to appropriate NVMe device. This applies to NVMe device in either passthrough mode or RAID mode.

2.11 Format NVM Command

The Format NVM command is used to low level format the NVM media. This is used when the host wants to change the LBA data size and/or metadata size.

Intel VROC (VMD NVMe RAID) passes this command to appropriate NVMe device. This applies to NVMe device in either passthrough mode or RAID mode.

2.12 Security Send Command

The Security Send command is used to transfer security protocol data to the controller. The data structure transferred to the controller as part of this command contains security protocol specific commands to be performed by the controller.

Intel VROC (VMD NVMe RAID) passes this command to appropriate NVMe device. This applies to NVMe device in either passthrough mode or RAID mode.

2.13 Security Receive Command

The Security Receive command transfers the status and data result of one or more Security Send commands that were previously submitted to the controller. The association between a Security Receive command and previous Security Send command is dependent on the Security Protocol.



Intel VROC (VMD NVMe RAID) passes this command to appropriate NVMe device. This applies to NVMe device in either passthrough mode or RAID mode.

§§

3 *Public NVMe IOCTLS Support*

Intel VROC (VMD NVMe RAID) supports several IOCTLS described below. Support will be limited to passing described commands to the NVMe driver according to their signature.

3.1 Microsoft Firmware Update NVMe IOCTLS

The following firmware update IOCTLS are not blocked by Intel VROC (VMD NVMe RAID) driver however, they should not be used on RAID member drives:

Control code	Description
IOCTL_STORAGE_FIRMWARE_GET_INFO	Used to query storage device for firmware information. Details: https://msdn.microsoft.com/en-us/library/windows/desktop/mt718109(v=vs.85).aspx
IOCTL_STORAGE_FIRMWARE_DOWNLOAD	Used to download firmware image to target storage device. Details: https://msdn.microsoft.com/en-us/library/windows/desktop/mt718108(v=vs.85).aspx
IOCTL_STORAGE_FIRMWARE_ACTIVATE	Used to activate downloaded firmware image on target storage device. Details: https://msdn.microsoft.com/en-us/library/windows/desktop/mt718107(v=vs.85).aspx

In the case of NVMe devices under Intel VMD controller, firmware IOCTLS shall be sent to the handle obtained by the CreateFile function with file name parameter set to physical device name i.e.

```
HANDLE hDevice = CreateFile(
    _T("\\\\.\\PhysicalDrive0"),
    0,
    FILE_SHARE_READ | FILE_SHARE_WRITE,
    NULL,
    OPEN_EXISTING,
    0,
    NULL);
```



3.2 Firmware Update Using SCSI Pass Through IOCTL

With Intel VROC (VMD NVMe RAID) it is possible to update device firmware using the IOCTL SCSI_PASS_THROUGH_DIRECT command ([https://msdn.microsoft.com/en-us/library/windows/hardware/ff560521\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/hardware/ff560521(v=vs.85).aspx)).

For details, refer to the SCSI Primary Commands specification.

§§

4 *Private NVMe IOCTLs support*

For the standalone NVMe driver, the private IOCTL "NVME_PASS_THROUGH_IOCTL" is used to pass NVMe admin commands to NVMe device. The same IOCTL is supported by Intel VROC (VMD NVMe RAID). It can be sent to all NVMe devices even if it is a part of RAID volume.

If target disk is not part of any Intel VMD domain (VMD disabled), IOCTL can be sent directly to its controller as with the standalone NVMe driver (please note, this is out of scope of Intel VROC (VMD NVMe RAID) driver). Otherwise, it should be sent to one of the VMD controllers.

To address a specific disk, the special ID code should be placed in return code of IOCTL before sending. Obtaining this ID code is possible by using NVME_GET_RAID_CONFIGURATION (See: 5.5), NVME_GET_SPARE_DISKS_INFORMATION (See: 5.7), NVME_GET_PASSTHROUGH_DISKS_INFORMATION (See: 5.9) or

NVME_GET_JOURNALING_DRIVES_INFORMATION (See: 5.11), depending on target disk status.

4.1 Private IOCTLs Signature and Control Code Details.

The following signature and control code must be set in Windows IOCTL:

```
IOCTL signature: NvmeMini
IOCTL control code:
#define NVME_PASS_THROUGH_SRB_IO_CODE \
    CTL_CODE(NVME_STORPORT_DRIVER, 0x800, METHOD_BUFFERED,
    FILE_ANY_ACCESS)
IOCTL return code: vrocDiskID
```

4.2 Private IOCTLs Data Structure

```
typedef struct _NVME_PASS_THROUGH_IOCTL
{
    SRB_IO_CONTROL  SrbIoCtrl;
    ULONG           VendorSpecific[NVME_IOCTL_VENDOR_SPECIFIC_DW_SIZE];
    ULONG           NVMeCmd[NVME_IOCTL_CMD_DW_SIZE];
    ULONG           CplEntry[NVME_IOCTL_COMPLETE_DW_SIZE];
    ULONG           Direction;
    ULONG           QueueId;
    ULONG           DataBufferLen;
    ULONG           MetaDataLen;
    ULONG           ReturnBufferLen;
    UCHAR           DataBuffer[1];
} NVME_PASS_THROUGH_IOCTL, *PNVME_PASS_THROUGH_IOCTL;
```

Name		Description
SRB_IO_CONTROL SrbIoCtrl	Input	Windows specific IOCTL header Details: http://msdn.microsoft.com/en-us/library/windows/hardware/ff566339(v=vs.85).aspx
ULONG VendorSpecific[NVME_IOCTL_VENDOR_SPECIFIC_DW_SIZE];	Input	Vendor unique qualifiers for vendor unique commands
ULONG NVMeCmd[NVME_IOCTL_CMD_DW_SIZE];	Input	64-byte submission entry defined in NVMe Specification
ULONG CplEntry[NVME_IOCTL_COMPLETE_DW_SIZE];	Input	DW[0..3] of completion entry
ULONG Direction;	Input	Data transfer direction, from host to device or vice versa
ULONG QueueId;	Input	0 means using Admin queue, otherwise, IO queue is used
ULONG DataBufferLen;	Input	Transfer byte length, including Metadata, starting at DataBuffer
ULONG MetaDataLen;	Input	Set to 0 if not supported or interleaved with data
ULONG ReturnBufferLen;	Input	Returned byte length from device to host, at least the length of this structure. When data transfer required, add the length of the data.
UCHAR DataBuffer[1];	Input	Start with Metadata if present, and then regular data

§§

5 *Private Intel® VROC IOCTLs*

Intel VROC (VMD NVMe RAID) supports creating RAID volumes only with devices connected to PCIe slots with Intel VMD enabled.

Every Intel VMD domain is visible in the system as a separate storage controller. All IOCTLs to NVMe devices (RAID members, spares, journaling drives and passthrough drives) and Intel VROC RAID volumes need to be sent to the controller representing domain, which the target disk or volume is part of. The only exception is the NVME_PASS_THROUGH_IOCTL which can be sent to any Intel VMD controller.

5.1 List of Private IOCTLs For RAID Volumes

List of private Intel VROC IOCTLs and IOCTL signatures:

IOCTL name	IOCTL signature	Comment
NVME_GET_NUMBER_OF_RAID_VOLUMES	NvmeRAID	
NVME_GET_RAID_INFORMATION	NvmeRAID	
NVME_GET_RAID_CONFIGURATION	NvmeRAID	
NVME_GET_NUMBER_OF_SPARE_DISKS	NvmeRAID	Spare devices are not members of any RAID volumes.
NVME_GET_RAID_SPARE_DISKS_INFORMATION	NvmeRAID	
NVME_GET_NUMBER_OF_PASSTHROUGH_DISKS	NvmeRAID	
NVME_GET_PASSTHROUGH_DISKS_INFORMATION	NvmeRAID	
NVME_GET_NUMBER_OF_JOURNALING_DRIVES	NvmeRAID	
NVME_GET_JOURNALING_DRIVES_INFORMATION	NvmeRAID	
NVME_PASS_THROUGH_SRB_IO_CODE	NvmeRAID	Like passthrough NVME: It has the same control code but a different signature. This IOCTL will be used to send admin commands to NVMe RAID members and spare devices

5.2 NVME_GET_NUMBER_OF_RAID_VOLUMES

This IOCTL can be used to get number of Intel VROC (VMD NVMe RAID) RAID volumes present in target Intel VMD domain.

5.2.1 IOCTL Signature and Control Code:

The following signature and control code must be set in Windows IOCTL:

IOCTL signature: NvmeRAID

IOCTL control code:

```
#define NVME_GET_NUMBER_OF_RAID_VOLUMES \
    CTL_CODE(NVME_STORPORT_DRIVER, 0x805, METHOD_BUFFERED,
    FILE_ANY_ACCESS)
```

5.2.2 IOCTL Data Structure:

```
typedef struct _NVME_GET_NUMBER_OF_RAID_VOLUMES_IOCTL {
    SRB_IO_CONTROL SrbIoCtrl;
    ULONG numberOfRaidVolumes;
} NVME_GET_NUMBER_OF_RAID_VOLUMES_IOCTL,
*PNVME_GET_NUMBER_OF_RAID_VOLUMES_IOCTL;
```

Name	Direction	Description
SRB_IO_CONTROL SrbIoCtrl	Input	Windows specific IOCTL header Details: http://msdn.microsoft.com/en-us/library/windows/hardware/ff566339(v=vs.85).aspx
ULONG numberOfRaidVolumes	Output	Number of NVME Raid volumes in target VMD domain. 0 – no NVME RAID volumes.

5.3 NVME_GET_RAID_INFORMATION

This IOCTL provides general information about RAID volume:

- Model
- Firmware version
- Serial number
- Total number of RAID member devices

5.3.1 IOCTL Signature and Control Code

The following signature and control code must be set in Windows IOCTL:

IOCTL signature: NvmeRAID

```
IOCTL control code:
#define NVME_GET_RAID_INFORMATION \
    CTL_CODE(NVME_STORPORT_DRIVER, 0x806, METHOD_BUFFERED, \
    FILE_ANY_ACCESS)
```

5.3.2 IOCTL Data Structure

```
typedef struct _NVME_GET_RAID_INFORMATION_IOCTL {
    SRB_IO_CONTROL SrbIoCtrl;
    ULONG indexOfRaidVolume;
    CHAR raidType[8];
    CHAR model[40];
    CHAR firmwareVersion[8];
    CHAR serialNumber[40];
    ULONG numberOfMemberDisks;
} NVME_GET_RAID_INFORMATION_IOCTL, *PNVME_GET_RAID_INFORMATION_IOCTL;
```

Description of data structure:

Name	Direction	Description
SRB_IO_CONTROL SrbIoCtrl	Input	Windows specific IOCTL header Details: http://msdn.microsoft.com/en-us/library/windows/hardware/ff566339(v=vs.85).aspx
ULONG indexOfRaidVolume	Input	Range from 1 to numberOfRaidVolumes. To get maximum value of numberOfRaidVolumes use NVME_GET_NUMBER_OF_RAID_VOLUMES
CHAR raidType[8]	Output	8 char string null terminated describing type of RAID. Returned strings: RAID0, RAID1, RAID5, RAID10,
CHAR model[40]	Output	String presented by SSD Toolbox in "Drive Summary" tab.
CHAR firmwareVersion[8]	Output	String presented by SSD Toolbox in "Drive Summary" tab.
CHAR serialNumber[40]	Output	String presented by SSD Toolbox in "Drive Summary" tab.
ULONG numberOfMemberDisks	Output	Number of member devices in NVME RAID volume.

5.4 NVME_GET_RAID_CONFIGURATION

This IOCTL can be sent to the Intel VROC (VMD NVMe RAID) driver to get following information about NVME member devices:

- vrocDiskID
- model
- firmware version

- serial number
- detailed information about the physical location of the drive (more details in Input/Output DataBuffer details paragraph)

In response to this IOCTL, the data buffer will be filled with data about all Intel VROC (VMD NVMe RAID) RAID member devices. The size of this IOCTL depends on total number of NVMe member devices.

5.4.1 IOCTL Signature and Control Code

The following signature and control code must be set in Windows IOCTL:

IOCTL signature: "NvmeRAID"

IOCTL control code:

```
#define NVME_GET_RAID_CONFIGURATION \
        CTL_CODE(NVME_STORPORT_DRIVER, 0x807, METHOD_BUFFERED,
FILE_ANY_ACCESS)
```

5.4.2 IOCTL Data Structure

```
typedef struct _NVME_GET_RAID_CONFIGURATION_IOCTL {
    SRB_IO_CONTROL SrbIoCtrl;
    ULONG indexOfRaidVolume;
    ULONG ReturnBufferLen;
    UCHAR DataBuffer[1];
} NVME_GET_RAID_CONFIGURATION_IOCTL, *PNVME_GET_RAID_CONFIGURATION_IOCTL;
```

Description of data structure:

Name	Direction	Description
SRB_IO_CONTROL SrbIoCtrl	Input	Windows specific IOCTL header Details: http://msdn.microsoft.com/en-us/library/windows/hardware/ff566339(v=vs.85).aspx
ULONG indexOfRaidVolume	Input	Range from 1 to numberOfRaidVolumes. To get maximum value of numberOfRaidVolumes use NVME_GET_NUMBER_OF_RAID_VOLUMES
CHAR raidType[8]	Output	8 char string null terminated describing type of RAID. Returned strings: RAID0, RAID1, RAID5, RAID10,
CHAR model[40]	Output	String presented by SSD Toolbox in "Drive Summary" tab.
CHAR firmwareVersion[8]	Output	String presented by SSD Toolbox in "Drive Summary" tab.
CHAR serialNumber[40]	Output	String presented by SSD Toolbox in "Drive Summary" tab.

Name	Direction	Description
ULONG numberOfMemberDisks	Output	Number of member devices in NVME RAID volume.

5.5 NVME_GET_RAID_CONFIGURATION

This IOCTL can be sent to the Intel VROC (VMD NVMe RAID) driver to get following information about NVME member devices:

- vrocDiskID
- Model
- Firmware version
- Serial number
- Detailed information about the physical location of the drive (more details in Input/Output DataBuffer details paragraph)

In response to this IOCTL, the data buffer will be filled with data about all Intel VROC (VMD NVMe RAID) RAID member devices. The size of this IOCTL depends on total number of NVMe member devices.

5.5.1 IOCTL Signature and Control Code

The following signature and control code must be set in Windows IOCTL:

```
IOCTL signature: "NvmeRAID"
IOCTL control code:
#define NVME_GET_RAID_CONFIGURATION \
    CTL_CODE(NVME_STORPORT_DRIVER, 0x807, METHOD_BUFFERED,
    FILE_ANY_ACCESS)
```

5.5.2 IOCTL Data Structure

```
typedef struct _NVME_GET_RAID_CONFIGURATION_IOCTL {
    SRB_IO_CONTROL SrbIoCtrl;
    ULONG indexOfRaidVolume;
    ULONG ReturnBufferLen;
    UCHAR DataBuffer[1];
} NVME_GET_RAID_CONFIGURATION_IOCTL, *PNVME_GET_RAID_CONFIGURATION_IOCTL;
```

Name	Direction	Description
SRB_IO_CONTROL SrbIoCtrl	Input	Windows specific IOCTL header Details: http://msdn.microsoft.com/en-us/library/windows/hardware/ff566339(v=vs.85).aspx
ULONG indexOfRaidVolume	Input	Range from 1 to numberOfRaidVolumes. To get maximum value of numberOfRaidVolumes use NVME_GET_NUMBER_OF_RAID_VOLUMES

Name	Direction	Description
ULONG ReturnBufferLen	Input	Size of DataBuffer: ReturnBufferLen = numberOfMemberDisks * sizeof(NVME_MEMBER_DISK_INFORMATION)
UCHAR DataBuffer[1]	In/Out	Data buffer for member devices details.

5.5.3 Input/Output Data Buffer Details

Application must provide a large enough data buffer to get information about all member devices.

The size of data buffer:

ReturnBufferLen = numberOfMemberDisks *
sizeof(NVME_MEMBER_DISK_INFORMATION)

Where: numberOfMemberDisks is the total number of NVMe devices used to build RAID volume.

NVME_MEMBER_DISK_INFORMATION is the data structure containing the information

NVMe device:

```
typedef struct _NVME_MEMBER_DISK_INFORMATION {
    ULONG vrocDiskID;
    CHAR diskModel[40];
    CHAR firmwareVersion[8];
    CHAR serialNumber[40];
    ULONG socketNumber;
    ULONG vmdControllerNumber;
    ULONG rootPortOffset;
    ULONG slotNumber;
} NVME_MEMBER_DISK_INFORMATION, *PNVME_MEMBER_DISK_INFORMATION;
```

Name	Direction	Description
ULONG vrocDiskID	Output	32-bit ID of disk. This ID must be used to send NVMe admin requests to individual NVMe devices marked as spare.
CHAR diskModel[40]	Output	String presented by SSD Toolbox in "Drive Summary" tab
CHAR firmwareVersion[8]	Output	String presented by SSD Toolbox in "Drive Summary" tab
CHAR serialNumber[40]	Output	String presented by SSD Toolbox in "Drive Summary" tab
ULONG socketNumber	Output	CPU socket number.
ULONG vmdControllerNumber	Output	VMD domain number.

Name	Direction	Description
ULONG rootPortOffset	Output	In case of direct attached NVMe drives, offset is a 0-based number of the slot, where the drive is attached, within a given VMD domain. In case of switch attached drives, this field will be equal to switch slot number and a slotNumber field must be used to identify the drives instead.
ULONG slotNumber	Output	Slot identifier.

5.6 NVME_GET_NUMBER_OF_SPARE_DISKS

This IOCTL can be used to get total number of NVMe devices marked as spare drives in target Intel VMD domain.

5.6.1 IOCTL signature and control code:

The following signature and control code must be set in Windows IOCTL:

IOCTL signature: NvmeRAID

IOCTL control code:

```
#define NVME_GET_NUMBER_OF_SPARE_DISKS \
    CTL_CODE(NVME_STORPORT_DRIVER, 0x808, METHOD_BUFFERED,
    FILE_ANY_ACCESS)
```

5.6.2 IOCTL data structure:

```
typedef struct _NVME_GET_NUMBER_OF_SPARE_DISKS_IOCTL {
    SRB_IO_CONTROL SrbIoCtrl;
    ULONG numberOfSpareDisks;
} NVME_GET_NUMBER_OF_SPARE_DISKS_IOCTL,
*PNVME_GET_NUMBER_OF_SPARE_DISKS_IOCTL;
```

Name	Direction	Description
SRB_IO_CONTROL SrbIoCtrl	Input	Windows specific IOCTL header Details: http://msdn.microsoft.com/en-us/library/windows/hardware/ff566339(v=vs.85).aspx
ULONG numberOfSpareDisks	Output	Total number of NVMe devices marked as spare in target VMD domain.

5.7 NVME_GET_SPARE_DISKS_INFORMATION

This IOCTL can be send to Intel VROC (VMD NVMe RAID) driver to get the following information about NVMe spare devices:

- vrocDiskID
- Model

- Firmware version
- Serial number
- Detailed information about the physical location of the drive (more details in Input/Output Data Buffer details paragraph)

In response to this IOCTL data buffer will be filled with data about all NVMe devices marked as spare. The size of this IOCTL depends on total number of NVMe spare devices in system.

5.7.1 IOCTL signature and control code

The following signature and control code must be set in Windows IOCTL:

IOCTL signature: NvmeRAID

IOCTL control code:

```
#define NVME_GET_SPARE_DISKS_INFORMATION \
    CTL_CODE(NVME_STORPORT_DRIVER, 0x809, METHOD_BUFFERED, \
    FILE_ANY_ACCESS)
```

5.7.2 IOCTL Data Structure

```
typedef struct _NVME_GET_SPARE_DISKS_INFORMATION_IOCTL {
    SRB_IO_CONTROL SrbIoCtrl;
    ULONG ReturnBufferLen;
    UCHAR DataBuffer[1];
} NVME_GET_SPARE_DISKS_INFORMATION_IOCTL,
*PNVME_GET_SPARE_DISKS_INFORMATION_IOCTL;
```

Name	Direction	Description
SRB_IO_CONTROL SrbIoCtrl	Input	Windows specific IOCTL header Details: http://msdn.microsoft.com/en-us/library/windows/hardware/ff566339(v=vs.85).aspx
ULONG ReturnBufferLen	Input	Size of data buffer: ReturnBufferLen = numberOfSpareDisks * sizeof(NVME_DISK_INFORMATION)
UCHAR DataBuffer[1]	In/Out	Data buffer for member devices details.

5.7.3 Input/Output Data Buffer Structure

Application must provide a large enough data buffer to get information about all spare devices.

The size of data buffer:

ReturnBufferLen = numberOfSpareDisks * sizeof(NVME_DISK_INFORMATION)

Where:

numberOfSpareDisks is the total number of NVMe devices marked as spare.

NVME_DISK_INFORMATION is the data structure containing information about the NVMe disk:

```
typedef struct _NVME_DISK_INFORMATION {
    ULONG vrocDiskID;
    CHAR diskModel[40];
    CHAR firmwareVersion[8];
    CHAR serialNumber[40];
    ULONG socketNumber;
    ULONG vmdControllerNumber;
    ULONG rootPortOffset;
    ULONG slotNumber;
} NVME_DISK_INFORMATION, *PNVME_DISK_INFORMATION;
```

Name	Direction	Description
ULONG vrocDiskID	Output	32-bit ID of disk. This ID must be used to send NVMe admin requests to individual NVMe devices marked as spare.
CHAR diskModel[40]	Output	String presented by SSD Toolbox in "Drive Summary" tab
CHAR firmwareVersion[8]	Output	String presented by SSD Toolbox in "Drive Summary" tab
CHAR serialNumber[40]	Output	String presented by SSD Toolbox in "Drive Summary" tab
ULONG socketNumber	Output	CPU socket number.
ULONG vmdControllerNumber	Output	VMD domain number.
ULONG rootPortOffset	Output	In the case of direct attached NVMe drives, the offset is a 0-based number of the slot, where the drive is attached, within a given VMD domain. In case of switch attached drives, this field will be equal to switch slot number and a slotNumber field must be used to identify the drives instead.
ULONG slotNumber	Output	Slot identifier.

5.8 NVME_GET_NUMBER_OF_PASSTHROUGH_DISKS

This IOCTL can be used to get total number of NVMe devices that are not member devices of any volume nor marked as spare in the target Intel VMD domain.

5.8.1 5.8.1 IOCTL Signature and Control Code

The following signature and control code must be set in Windows IOCTL:

IOCTL signature: NvmeRAID

IOCTL control code:

```
#define NVME_GET_NUMBER_OF_PASSTHROUGH_DISKS \
    CTL_CODE(NVME_STORPORT_DRIVER, 0x80A, METHOD_BUFFERED, \
    FILE_ANY_ACCESS)
```

5.8.2 IOCTL Data Structure

```
typedef struct _NVME_GET_NUMBER_OF_PASSTHROUGH_DISKS_IOCTL {
    SRB_IO_CONTROL SrbIoCtrl;
    ULONG numberOfPassthroughDisks;
} NVME_GET_NUMBER_OF_PASSTHROUGH_DISKS_IOCTL,
*PNVME_GET_NUMBER_OF_PASSTHROUGH_DISKS_IOCTL;
```

Name	Direction	Description
SRB_IO_CONTROL SrbIoCtrl	Input	Windows specific IOCTL header Details: http://msdn.microsoft.com/en-us/library/windows/hardware/ff566339(v=vs.85).aspx
ULONG numberOfPassthrough Disks	Output	Total number of NVME passthrough devices in target VMD domain.

5.9 NVME_GET_PASSTHROUGH_DISKS_INFORMATION

This IOCTL can be send to Intel VROC (VMD NVMe RAID) driver to get the following information about NVME passthrough devices:

- vrocDiskID
- Model
- Firmware version
- Serial number

Detailed information about the physical location of the drive (more details in Input/Output DataBuffer details paragraph)

In response to this IOCTL data buffer will be filled with data about all NVME passthrough devices. The size of this IOCTL depends on total number of NVMe passthrough devices in target VMD domain.

5.9.1 IOCTL Signature and Control Code

The following signature and control code must be set in Windows IOCTL:

IOCTL signature: NvmeRAID

IOCTL control code:

```
#define NVME_GET_PASSTHROUGH_DISKS_INFORMATION \
    CTL_CODE(NVME_STORPORT_DRIVER, 0x80B, METHOD_BUFFERED, \
    FILE_ANY_ACCESS)
```

5.9.2 IOCTL Data Structure

```
typedef struct _NVME_GET_PASSTHROUGH_DISKS_INFORMATION_IOCTL {
    SRB_IO_CONTROL SrbIoCtrl;
```

```

        ULONG ReturnBufferLen;
        UCHAR DataBuffer[1];
    } NVME_GET_PASSTHROUGH_DISKS_INFORMATION_IOCTL,
    *PNVME_GET_PASSTHROUGH_DISKS_INFORMATION_IOCTL;

```

Name	Direction	Description
SRB_IO_CONTROL SrbIoCtrl	Input	Windows specific IOCTL header Details: http://msdn.microsoft.com/en-us/library/windows/hardware/ff566339(v=vs.85).aspx
ULONG ReturnBufferLen	Input	Size of DataBuffer: ReturnBufferLen = numberOfPassthroughDisks * sizeof(NVME_DISK_INFORMATION)
UCHAR DataBuffer[1]	In/Out	Data buffer for member devices details.

5.9.3 Input/Output Data Buffer Structure

Application must provide large enough data buffer to get information about all passthrough devices.

The size of data buffer: ReturnBufferLen = numberOfPassthroughDisks *
sizeof(NVME_DISK_INFORMATION)

Where:

numberOfPassthroughDisks is the total number of NVMe passthrough devices in target VMD domain.

NVME_DISK_INFORMATION is the data structure containing information about NVMe disk:

```

typedef struct _NVME_DISK_INFORMATION {
    ULONG vrocDiskID;
    CHAR diskModel[40];
    CHAR firmwareVersion[8];
    CHAR serialNumber[40];
    ULONG socketNumber;
    ULONG vmdControllerNumber;
    ULONG rootPortOffset;
    ULONG slotNumber;
} NVME_DISK_INFORMATION, *PNVME_DISK_INFORMATION;

```

Name	Direction	Description
ULONG vrocDiskID	Output	32-bit ID of disk. This ID must be used to send NVMe admin requests to individual NVMe passthrough devices.
CHAR diskModel[40]	Output	String presented by SSD Toolbox in "Drive Summary" tab
CHAR firmwareVersion[8]	Output	String presented by SSD Toolbox in "Drive Summary" tab

Name	Direction	Description
CHAR serialNumber[40]	Output	String presented by SSD Toolbox in "Drive Summary" tab
ULONG socketNumber	Output	CPU socket number.
ULONG vmdControllerNumber	Output	VMD domain number.
ULONG rootPortOffset	Output	In case of direct attached NVMe drives, offset is a 0-based number of the slot, where the drive is attached, within a given VMD domain. In case of switch attached drives, this field will be equal to switch slot number and a slotNumber field must be used to identify the drives instead.
ULONG slotNumber	Output	Slot identifier.

5.10 NVME_GET_NUMBER_OF_JOURNALING_DRIVES

This IOCTL can be used to get the total number of NVMe devices that are used as journaling drives in the target VMD domain.

5.10.1 IOCTL Signature and Control Code:

The following signature and control code must be set in Windows IOCTL:

IOCTL signature: NvmeRAID

IOCTL control code:

```
#define NVME_GET_NUMBER_OF_JOURNALING_DRIVES \
    CTL_CODE(NVME_STORPORT_DRIVER, 0x80C, METHOD_BUFFERED, \
    FILE_ANY_ACCESS)
```

5.10.2 IOCTL Data Structure:

```
typedef struct _NVME_GET_NUMBER_OF_JOURNALING_DRIVES_IOCTL {
    SRB_IO_CONTROL SrbIoCtrl;
    ULONG numberOfJournalingDrives;
} NVME_GET_NUMBER_OF_JOURNALING_DRIVES_IOCTL,
*PNVME_GET_NUMBER_OF_JOURNALING_DRIVES_IOCTL;
```

Name	Direction	Description
SRB_IO_CONTROL SrbIoCtrl	Input	Windows specific IOCTL header Details: http://msdn.microsoft.com/en-us/library/windows/hardware/ff566339(v=vs.85).aspx
ULONG numberOfJournalingDrives	Output	Total number of NVMe journaling drives in target VMD domain.

5.11 NVME_GET_JOURNALING_DRIVES_INFORMATION

This IOCTL can be send to Intel VROC (VMD NVMe RAID) driver to get the following information about NVME journaling drives:

- vrocDiskID
- Model
- Firmware version
- Serial number
- Detailed information about the physical location of the drive (more details in Input/Output DataBuffer details paragraph)

In response to this, the IOCTL data buffer will be filled with data about all NVME journaling drives.

The size of this IOCTL depends on the total number of NVMe journaling drives in the target VMD domain.

5.11.1 IOCTL Signature and Control Code

The following signature and control code must be set in Windows IOCTL:

IOCTL signature: NvmeRAID

IOCTL control code:

```
#define NVME_GET_JOURNALING_DRIVES_INFORMATION \
        CTL_CODE(NVME_STORPORT_DRIVER, 0x80D, METHOD_BUFFERED, \
        FILE_ANY_ACCESS)
```

5.11.2 IOCTL Data Structure

```
typedef struct _NVME_GET_JOURNALING_DRIVES_INFORMATION_IOCTL {
    SRB_IO_CONTROL SrbIoCtrl;
    ULONG ReturnBufferLen;
    UCHAR DataBuffer[1];
} NVME_GET_JOURNALING_DRIVES_INFORMATION_IOCTL, *
PNVME_GET_JOURNALING_DRIVES_INFORMATION_IOCTL;
```

Name	Direction	Description
SRB_IO_CONTROL SrbIoCtrl	Input	Windows specific IOCTL header Details: http://msdn.microsoft.com/en-us/library/windows/hardware/ff566339(v=vs.85).aspx
ULONG ReturnBufferLen	Input	Size of data buffer: ReturnBufferLen = numberOfJournalingDrives * sizeof(NVME_DISK_INFORMATION)
UCHAR DataBuffer[1]	In/Out	Data buffer for journaling drives details.

5.11.3 Input/Output Data Buffer Structure

Application must provide a large enough data buffer to get information about all passthrough devices.

The size of DataBuffer: $\text{ReturnBufferLen} = \text{numberOfJournalingDrives} * \text{sizeof}(\text{NVME_DISK_INFORMATION})$

Where:

numberOfJournalingDrives is the number of NVMe journaling drives in target domain.

NVME_DISK_INFORMATION is the data structure containing information about the NVMe disk:

```
typedef struct _NVME_DISK_INFORMATION {
    ULONG vrocDiskID;
    CHAR diskModel[40];
    CHAR firmwareVersion[8];
    CHAR serialNumber[40];
    ULONG socketNumber;
    ULONG vmdControllerNumber;
    ULONG rootPortOffset;
    ULONG slotNumber;
} NVME_DISK_INFORMATION, *PNVME_DISK_INFORMATION;
```

Name	Direction	Description
ULONG vrocDiskID	Output	32 bit ID of disk. This ID must be used to send NVMe admin requests to individual NVMe journaling drive.
CHAR diskModel[40]	Output	String presented by SSD Toolbox in "Drive Summary" tab
CHAR firmwareVersion[8]	Output	String presented by SSD Toolbox in "Drive Summary" tab
CHAR serialNumber[40]	Output	String presented by SSD Toolbox in "Drive Summary" tab
ULONG socketNumber	Output	CPU socket number.
ULONG vmdControllerNumber	Output	VMD domain number.
ULONG rootPortOffset	Output	In case of direct attached NVMe drives, offset is a 0-based number of the slot, where the drive is attached, within a given VMD domain. In case of switch attached drives, this field will be equal to switch slot number and a slotNumber field must be used to identify the drives instead.
ULONG slotNumber	Output	Slot identifier.

5.12 NVME_PASS_THROUGH_SRB_IO_CODE

Intel VROC (VMD NVMe RAID) will provide an IOCTL to pass through admin commands to all NVMe devices.

Warning: All IOCTLs for NVMe devices need to be sent to Intel VMD instead of directly to the NVMe devices.

List of admin commands which can be sent to NVMe devices using private IOCTL:

Command	Hex
ADMIN_GET_LOG_PAGE	0x02
ADMIN_IDENTIFY	0x06
ADMIN_SET_FEATURES	0x09
ADMIN_GET_FEATURES	0x0A
ADMIN_ASYNCHRONOUS_EVENT_REQUEST	0x0C
ADMIN_FIRMWARE_ACTIVATE	0x10
ADMIN_FIRMWARE_IMAGE_DOWNLOAD	0x11
ADMIN_FORMAT_NVM	0x80
ADMIN_SECURITY_SEND	0x81
ADMIN_SECURITY_RECEIVE	0x82
Any vendor specific command	0xC0-0xFF

5.12.1 IOCTL Signature and Control Code

The following signature and control code must be set in Windows IOCTL:

IOCTL signature: NvmeRAID

IOCTL control code:

```
#define NVME_PASS_THROUGH_SRB_IO_CODE \
    CTL_CODE(NVME_STORPORT_DRIVER, 0x800, METHOD_BUFFERED, \
    FILE_ANY_ACCESS)
```

5.12.2 IOCTL Data Structure

To simplify the passthrough IOCTL interface, Intel® VROC will reuse the NVME_PASS_THROUGH_IOCTL data structure introduced in the standalone NVMe driver. To properly identify the RAID member disk, the property "ReturnCode" in SRB_IO_CONTROL data structure will be used.

Application will have to write value of vrocDiskID returned by NVME_GET_RAID_CONFIGURATION to the property "ReturnCode."

```
typedef struct _SRB_IO_CONTROL {
    ULONG HeaderLength;
    UCHAR Signature[8];
    ULONG Timeout;
    ULONG ControlCode;
```

```

        ULONG ReturnCode;    //vrocDiskID must be written here before
IOCTL send
        ULONG Length;
    } SRB_IO_CONTROL, *PSRB_IO_CONTROL;

typedef struct _NVME_PASS_THROUGH_IOCTL {
SRB_IO_CONTROL SrbIoCtrl;
ULONG        VendorSpecific[NVME_IOCTL_VENDOR_SPECIFIC_DW_SIZE];
ULONG        NVMeCmd[NVME_IOCTL_CMD_DW_SIZE];
ULONG        CplEntry[NVME_IOCTL_COMPLETE_DW_SIZE];
ULONG        Direction;
ULONG        QueueId;
ULONG        DataBufferLen;
ULONG        MetaDataLen;
ULONG        ReturnBufferLen;
UCHAR        DataBuffer[1];
} NVME_PASS_THROUGH_IOCTL, *NVME_PASS_THROUGH_IOCTL;

```

Name		Description
SRB_IO_CONTROL SrbIoCtrl	Input	Windows specific IOCTL header Details: http://msdn.microsoft.com/en-us/library/windows/hardware/ff566339(v=vs.85).aspx IMPORTANT: "ReturnCode" property must be filled with vrocDiskID. vrocDiskID is a 32-bit ID of RAID member disk. This ID must be used to send NVMe requests to individual NVMe member disks. NVME_GET_RAID_CONFIGURATION, NVME_GET_SPARE_DISK_INFORMATION, NVME_GET_PASSTHROUGH_DISKS_INFORMATION or NVME_GET_JOURNALING_DRIVES_INFORMATION must be used to get this ID.
ULONG VendorSpecific[NVME_IOCTL_VENDOR_SPECIFIC_DW_SIZE];	Input	Vendor unique qualifiers for vendor unique commands
ULONG NVMeCmd[NVME_IOCTL_CMD_DW_SIZE];	Input	64-byte submission entry defined in NVMe Specification
ULONG CplEntry[NVME_IOCTL_COMPLETE_DW_SIZE];	Input	DW[0..3] of completion entry
ULONG Direction;	Input	Data transfer direction, from host to device or vice versa
ULONG QueueId;	Input	0 means using Admin queue, otherwise, IO queue is used
ULONG DataBufferLen;	Input	Transfer byte length, including Metadata, starting at DataBuffer

Name		Description
ULONG MetaDataLen;	Input	Set to 0 if not supported or interleaved with data
ULONG ReturnBufferLen;	Input	Returned byte length from device to host, at least the length of this structure. When data transfer is required, add the length of the data.
UCHAR DataBuffer[1];	Input	Start with Metadata if present, and then regular data

Note: If NVMe passthrough IOCTL is used to update firmware on a device which is a RAID member disk, a full system reboot is required. If a reboot is not performed, an attempt to send device reset command like STORAGE_BUS_RESET_REQUEST can end with RAID failure.

5.12.3 Error Handling

There are three levels of status that user applications receive after calling the DeviceIoControl API. The first level is the return code of the API. The second level is the ReturnCode field of SRB_IO_CONTROL structure, which is marked down by the miniport driver. The third level is the completion status included in the completion entry after the request has been issued to the controller. It is recommended that user applications investigate all three levels of status to ensure the request is completed successfully.

The following status is noted in the ReturnCode of SRB_IO_CONTROL structure by the miniport driver when the request is processed by the driver. User applications need to examine the ReturnCode to find out if the driver has discovered any errors in the request.

When the ReturnCode is NVME_IOCTL_SUCCESS, which indicates the request had been issued to the Controller and user applications need to examine the completion status of CplEntry. Otherwise, the request had not been issued to controller due to error.

```
enum _IOCTL_STATUS
{
    NVME_IOCTL_SUCCESS,
    NVME_IOCTL_INVALID_IOCTL_CODE,
    NVME_IOCTL_INVALID_SIGNATURE,
    NVME_IOCTL_INSUFFICIENT_IN_BUFFER,
    NVME_IOCTL_INSUFFICIENT_OUT_BUFFER,
    NVME_IOCTL_UNSUPPORTED_ADMIN_CMD,
    NVME_IOCTL_UNSUPPORTED_NVM_CMD,
    NVME_IOCTL_INVALID_ADMIN_VENDOR_SPECIFIC_OPCODE,
    NVME_IOCTL_INVALID_NVM_VENDOR_SPECIFIC_OPCODE,
    NVME_IOCTL_ADMIN_VENDOR_SPECIFIC_NOT_SUPPORTED, //AVSCC=0
    NVME_IOCTL_NVM_VENDOR_SPECIFIC_NOT_SUPPORTED, // NVSCC=0
    NVME_IOCTL_INVALID_DIRECTION_SPECIFIED, // when Direction is greater
    than 3
    NVME_IOCTL_INVALID_META_BUFFER_LENGTH,
    NVME_IOCTL_PRP_TRANSLATION_ERROR,
    NVME_IOCTL_INVALID_PATH_TARGET_ID,
    NVME_IOCTL_FORMAT_NVM_PENDING, // Only one Format NVM at a time
    NVME_IOCTL_FORMAT_NVM_FAILED,
    NVME_IOCTL_INVALID_NAMESPACE_ID
};
```



With the `ReturnCode`, there are three levels of status codes user applications can examine after calling the `DeviceIoControl` API:

- Level 1: Returned status of `DeviceIoControl` API
- Level 2: `ReturnCode` of `SRB_IO_CONTROL` structure
- Level 3: Status Field of Completion Entry

When the driver receives the request, it will always mark `SrbStatus` as `SRB_STATUS_SUCCESS`. In case of any errors, driver will mark down proper status code in `ReturnCode`. Therefore, the basic scenario that user applications need to follow to identify any errors after calling `DeviceIoControl` is:

1. When `DeviceIoControl` returns with error, `GetLastError` is used to find more details.
2. When `DeviceIoControl` returns successfully, `ReturnCode` needs to be examined to see if the driver reports any errors.
3. When `ReturnCode` is `NVME_IOCTL_SUCCESS`, the Status Field of Completion Entry serves as the final status of the completed NVMe command.



6 Example Usage

Example application shown in this chapter cover usage of 4 out of 8 IOCTLs presented in the previous chapters. It shows the recommended procedure when sending NVMe IOCTLs to member devices.

This application looks for any Miniport with at least 1 volume (using NVME_GET_NUMBER_OF_RAID_VOLUMES). If it finds one, it gets number of devices of the first volume of that Miniport (by using NVME_GET_RAID_INFORMATION). This information is needed to allocate a large enough buffer for the next IOCTL: NVME_GET_RAID_CONFIGURATION, which returns information about all member devices of that volume. From the data returned by this IOCTL, the application gets vrocDiskID of the first member disk. Finally, IOCTL NVME_PASS_THROUGH_SRB_IO_CODE is sent to that disk. In this example, the payload of this IOCTL is the command ADMIN_IDENTIFY. After receiving it, the applications print information about the VID, SSVID, serial number, model number and firmware version of the target disk. During execution it also prints the SCSI port on which it finds the miniport, number of volumes on that port, member device count of the first found volume, and serial number and vrocDiskID of the first disk returned by NVME_GET_RAID_CONFIGURATION.

This code uses nvme.h header, which available on the internet

```
#include <windows.h>
#include <winioctl.h>
#include <ntddscsi.h>
#include <stdio.h>
#include "nvme.h"

typedef struct _NVME_GET_NUMBER_OF_RAID_VOLUMES_IOCTL {
    SRB_IO_CONTROL SrbIoCtrl;
    ULONG numberOfRaidVolumes;
} NVME_GET_NUMBER_OF_RAID_VOLUMES_IOCTL,
*PNVME_GET_NUMBER_OF_RAID_VOLUMES_IOCTL;

typedef struct _NVME_PASS_THROUGH_IOCTL {
    SRB_IO_CONTROL SrbIoCtrl;
    ULONG VendorSpecific[6];
    ULONG NVMeCmd[16];
    ULONG CplEntry[4];
    ULONG Direction;
    ULONG QueueId;
    ULONG DataBufferLen;
    ULONG MetaDataLen;
    ULONG ReturnBufferLen;
    UCHAR DataBuffer[1];
} NVME_RAID_PASS_THROUGH_IOCTL, *PNVME_RAID_PASS_THROUGH_IOCTL;

typedef struct _NVME_GET_RAID_CONFIGURATION_IOCTL {
    SRB_IO_CONTROL SrbIoCtrl;
    ULONG indexOfRaidVolume;
    ULONG ReturnBufferLen;
    UCHAR DataBuffer[1];
}
```

```

} NVME_GET_RAID_CONFIGURATION_IOCTL,
*PNVME_GET_RAID_CONFIGURATION_IOCTL;

typedef struct _NVME_GET_RAID_INFORMATION_IOCTL {
    SRB_IO_CONTROL SrbIoCtrl;
    ULONG indexOfRaidVolume;
    CHAR raidType[8];
    CHAR model[40];
    CHAR firmwareVersion[8];
    CHAR serialNumber[40];
    ULONG numberOfMemberDisks;
} NVME_GET_RAID_INFORMATION_IOCTL,
*PNVME_GET_RAID_INFORMATION_IOCTL;

typedef struct _NVME_MEMBER_DISK_INFORMATION {
    ULONG vrocDiskID;
    CHAR diskModel[40];
    CHAR firmwareVersion[8];
    CHAR serialNumber[40];
    ULONG socketNumber;
    ULONG vmdControllerNumber;
    ULONG rootPortOffset;
    ULONG slotNumber;
} NVME_MEMBER_DISK_INFORMATION, *PNVME_MEMBER_DISK_INFORMATION;

#define NVME_GET_NUMBER_OF_RAID_VOLUMES \
    CTL_CODE(NVME_STORPORT_DRIVER, 0x805, METHOD_BUFFERED, \
    FILE_ANY_ACCESS)

#define NVME_GET_RAID_INFORMATION \
    CTL_CODE(NVME_STORPORT_DRIVER, 0x806, METHOD_BUFFERED, \
    FILE_ANY_ACCESS)

#define NVME_GET_RAID_CONFIGURATION \
    CTL_CODE(NVME_STORPORT_DRIVER, 0x807, METHOD_BUFFERED, \
    FILE_ANY_ACCESS)

#define NVME_PASS_THROUGH_SRB_IO_CODE \
    CTL_CODE(NVME_STORPORT_DRIVER, 0x800, METHOD_BUFFERED, \
    FILE_ANY_ACCESS)

#define NVME_FROM_DEV_TO_HOST 2 /* Transfer data from device to \
host */
#define NVME_RAID_SIG_STR "NvmeRAID"
#define NVME_RAID_SIG_STR_LEN 8
#define NVME_STORPORT_DRIVER 0xE000
#define NVME_PT_TIMEOUT 40

int main()
{
    HANDLE          hDevice = INVALID_HANDLE_VALUE;
    DWORD           scsiPort;
    WCHAR           buffer[MAX_PATH];

```

```

LPTSTR          pszTxt = _T("\\\\.\\Scsi");
LPCTSTR         pszFormat = _T("%s%d");

BOOL bRet = 0;
NVME_GET_NUMBER_OF_RAID_VOLUMES_IOCTL
numberOfRaidVolumesIoctl;
NVME_GET_RAID_INFORMATION_IOCTL raidInformationIoctl;
PNVME_GET_RAID_CONFIGURATION_IOCTL raidConfigurationIoctl;
PNVME_RAID_PASS_THROUGH_IOCTL nvmePassThroughIoctl;
PNVMe_COMMAND pCmd;
PADMIN_IDENTIFY_COMMAND_DW10 dwl0;
DWORD dwReturned;
DWORD dwError = ERROR_SUCCESS;
ULONG numberOfDisks;
PNVME_MEMBER_DISK_INFORMATION diskInfo;
PADMIN_IDENTIFY_CONTROLLER pidCtrlr;
UCHAR *tmp;
int counter;

for (scsiPort = 0; scsiPort < 16; scsiPort++)
{
    //Format the device name string to something like
    "\\.\Scsi0:"
    wsprintf(buffer, pszFormat, pszTxt, scsiPort);
    hDevice = CreateFile(buffer,
        GENERIC_READ | GENERIC_WRITE,
        FILE_SHARE_READ | FILE_SHARE_WRITE,
        NULL,
        OPEN_EXISTING,
        FILE_ATTRIBUTE_NORMAL,
        NULL);

    if (hDevice != INVALID_HANDLE_VALUE)
    {
        /* Sending NVME_GET_RAID_VOLUMES to find any
        controller with RAID volumes */

        //Set up the structure.
        ZeroMemory(&numberOfRaidVolumesIoctl,
            sizeof(NVME_GET_NUMBER_OF_RAID_VOLUMES_IOCTL));
        numberOfRaidVolumesIoctl.SrbIoCtrl.ControlCode =
            NVME_GET_NUMBER_OF_RAID_VOLUMES;
        numberOfRaidVolumesIoctl.SrbIoCtrl.HeaderLength =
            sizeof(SRB_IO_CONTROL);

        memcpy((UCHAR*)(&numberOfRaidVolumesIoctl.SrbIoCtrl.Signature[0])
            , NVME_RAID_SIG_STR, NVME_RAID_SIG_STR_LEN);
        numberOfRaidVolumesIoctl.SrbIoCtrl.Timeout =
            NVME_PT_TIMEOUT;
        numberOfRaidVolumesIoctl.SrbIoCtrl.Length =
            sizeof(numberOfRaidVolumesIoctl) - sizeof(SRB_IO_CONTROL);
        numberOfRaidVolumesIoctl.SrbIoCtrl.ReturnCode = 0;
    }
}

```

```

        //Call the IOCTL
        bRet = DeviceIoControl(hDevice, IOCTL_SCSI_MINIPORT,
&numberOfRaidVolumesIoctl,
sizeof(NVME_GET_NUMBER_OF_RAID_VOLUMES_IOCTL),
&numberOfRaidVolumesIoctl,
sizeof(NVME_GET_NUMBER_OF_RAID_VOLUMES_IOCTL), &dwReturned,
NULL);

        dwError = GetLastError();

        //If any RAID volumes are found, get it is
configuration
        if (numberOfRaidVolumesIoctl.numberOfRaidVolumes > 0)
        {

                printf("Found %lu volumes on SCSI port %lu\n",
numberOfRaidVolumesIoctl.numberOfRaidVolumes, scsiPort);
                /* Sending NVME_GET_RAID_INFORMATION to get
information of member disk count of volume which is needed to */

                //Set up the structure.
                ZeroMemory(&raidInformationIoctl,
sizeof(NVME_GET_RAID_INFORMATION_IOCTL));
                raidInformationIoctl.SrbIoCtrl.ControlCode =
NVME_GET_RAID_INFORMATION;
                raidInformationIoctl.SrbIoCtrl.HeaderLength =
sizeof(SRB_IO_CONTROL);

                memcpy((UCHAR*)&raidInformationIoctl.SrbIoCtrl.Signature[0]),
NVME_RAID_SIG_STR, NVME_RAID_SIG_STR_LEN);
                raidInformationIoctl.SrbIoCtrl.Timeout =
NVME_PT_TIMEOUT;
                raidInformationIoctl.SrbIoCtrl.Length =
sizeof(raidInformationIoctl) - sizeof(SRB_IO_CONTROL);
                raidInformationIoctl.SrbIoCtrl.ReturnCode = 0;
                raidInformationIoctl.indexOfRaidVolume = 1;

                //Call the IOCTL
                bRet = DeviceIoControl(hDevice,
IOCTL_SCSI_MINIPORT, &raidInformationIoctl,
sizeof(NVME_GET_RAID_INFORMATION_IOCTL), &raidInformationIoctl,
sizeof(NVME_GET_RAID_INFORMATION_IOCTL), &dwReturned, NULL);
                if(dwError = GetLastError())
                        printf("dwError: %lu\n", dwError);

                numberOfDisks =
raidInformationIoctl.numberOfMemberDisks;
                printf("Number of NVMe RAID volume member disks:
%lu\n", numberOfDisks);

                /* Sending NVME_GET_RAID_CONFIGURATION */

```

```

        //Set up the structure.
        raidConfigurationIoctl =
(PNVME_GET_RAID_CONFIGURATION_IOCTL) malloc(
sizeof(NVME_GET_RAID_CONFIGURATION_IOCTL) + numberOfDisks *
sizeof(NVME_MEMBER_DISK_INFORMATION) );
        if (raidConfigurationIoctl == NULL){
            CloseHandle(hDevice);
            return -1;
        }
        ZeroMemory(raidConfigurationIoctl,
sizeof(NVME_GET_RAID_CONFIGURATION_IOCTL) + numberOfDisks *
sizeof(NVME_MEMBER_DISK_INFORMATION) );
        raidConfigurationIoctl->SrbIoCtrl.ControlCode =
NVME_GET_RAID_CONFIGURATION;
        raidConfigurationIoctl->SrbIoCtrl.HeaderLength =
sizeof(SRB_IO_CONTROL);
        memcpy((UCHAR*)(&raidConfigurationIoctl->SrbIoCtrl.Signature[0]), NVME_RAID_SIG_STR,
NVME_RAID_SIG_STR_LEN);
        raidConfigurationIoctl->SrbIoCtrl.Timeout =
NVME_PT_TIMEOUT;
        raidConfigurationIoctl->SrbIoCtrl.Length =
sizeof(NVME_GET_RAID_CONFIGURATION_IOCTL) + (numberOfDisks *
sizeof(NVME_MEMBER_DISK_INFORMATION)) - sizeof(SRB_IO_CONTROL);
        raidConfigurationIoctl->SrbIoCtrl.ReturnCode = 0;
        raidConfigurationIoctl->indexOfRaidVolume = 1;
        raidConfigurationIoctl->ReturnBufferLen =
numberOfDisks * sizeof(NVME_MEMBER_DISK_INFORMATION);

        //Call the IOCTL
        bRet = DeviceIoControl(hDevice,
IOCTL_SCSI_MINIPORT, raidConfigurationIoctl,
sizeof(NVME_GET_RAID_CONFIGURATION_IOCTL) + (numberOfDisks *
sizeof(NVME_MEMBER_DISK_INFORMATION)),
        raidConfigurationIoctl,
sizeof(NVME_GET_RAID_CONFIGURATION_IOCTL) + (numberOfDisks *
sizeof(NVME_MEMBER_DISK_INFORMATION)), &dwReturned, NULL);
        if (dwError = GetLastError())
            printf("dwError: %lu\n", dwError);

        diskInfo = (PNVME_MEMBER_DISK_INFORMATION)
raidConfigurationIoctl->DataBuffer;
        printf("Serial: %s\nvrocDiskId:%lx\n",diskInfo->serialNumber,diskInfo->vrocDiskID);

        /* Sending NVME_PASSTHROUGH_IOCTL with
ADMIN_IDENTIFY command payload */
        nvmePassThroughIoctl =
(PNVME_RAID_PASS_THROUGH_IOCTL)
malloc(sizeof(NVME_RAID_PASS_THROUGH_IOCTL) +
sizeof(ADMIN_IDENTIFY_CONTROLLER));
        if (nvmePassThroughIoctl == NULL) {
            free(raidConfigurationIoctl);

```

```

        CloseHandle(hDevice);
        return -1;
    }
    ZeroMemory(nvmePassThroughIoctl,
sizeof(NVME_RAID_PASS_THROUGH_IOCTL) +
sizeof(ADMIN_IDENTIFY_CONTROLLER));
    nvmePassThroughIoctl->SrbIoCtrl.ControlCode =
NVME_PASS_THROUGH_SRB_IO_CODE;
    nvmePassThroughIoctl->SrbIoCtrl.HeaderLength =
sizeof(SRB_IO_CONTROL);
    memcpy((UCHAR*)&nvmePassThroughIoctl-
>SrbIoCtrl.Signature[0]), NVME_RAID_SIG_STR,
NVME_RAID_SIG_STR_LEN);
    nvmePassThroughIoctl->SrbIoCtrl.Timeout =
NVME_PT_TIMEOUT;
    nvmePassThroughIoctl->SrbIoCtrl.Length =
sizeof(NVME_RAID_PASS_THROUGH_IOCTL) +
sizeof(ADMIN_IDENTIFY_CONTROLLER) - sizeof(SRB_IO_CONTROL);
    nvmePassThroughIoctl->SrbIoCtrl.ReturnCode =
diskInfo->vrocDiskID;
    pCmd = (PNVMe_COMMAND)nvmePassThroughIoctl-
>NVMeCmd;
    pCmd->CDW0.OPC = ADMIN_IDENTIFY;
    dw10 = (PADMIN_IDENTIFY_COMMAND_DW10)&(pCmd-
>CDW10);
    dw10->CNS = 1;
    nvmePassThroughIoctl->QueueId = 0; // Admin queue
    nvmePassThroughIoctl->DataBufferLen = 0;
    nvmePassThroughIoctl->Direction =
NVME_FROM_DEV_TO_HOST;
    nvmePassThroughIoctl->ReturnBufferLen =
sizeof(ADMIN_IDENTIFY_CONTROLLER) +
sizeof(NVME_RAID_PASS_THROUGH_IOCTL);
    nvmePassThroughIoctl->VendorSpecific[0] =
(DWORD)0;
    nvmePassThroughIoctl->VendorSpecific[1] =
(DWORD)0;
    memset(nvmePassThroughIoctl->DataBuffer, 0x55,
sizeof(ADMIN_IDENTIFY_CONTROLLER));

    //Call the IOCTL
    bRet = DeviceIoControl(hDevice,
IOCTL_SCSI_MINIPORT, nvmePassThroughIoctl,
sizeof(ADMIN_IDENTIFY_CONTROLLER) +
sizeof(NVME_RAID_PASS_THROUGH_IOCTL),
    nvmePassThroughIoctl,
sizeof(ADMIN_IDENTIFY_CONTROLLER) +
sizeof(NVME_RAID_PASS_THROUGH_IOCTL), &dwReturned, NULL);
    if (dwError = GetLastError())
        printf("dwError: %lu\n", dwError);

    //Print returned payload

```

```

        pIdCtrlr = (PADMIN_IDENTIFY_CONTROLLER)
nvmePassThroughIoctl->DataBuffer;

        printf("IdentifyController: NL_IOCTL_IDENTIFY:
SUCCESS!!!\n");
        printf("IdentifyController: VID = 0x%x, SSVID =
0x%x \n",
                pIdCtrlr->VID, pIdCtrlr->SSVID);
        tmp = pIdCtrlr->SN;
        printf("IdentifyController: serialNum: ");
        counter = 20;
        while (counter--)
        {
            printf("%c", *tmp++);
        }
        printf("\n");
        tmp = pIdCtrlr->MN;
        printf("IdentifyController: modelNum: ");
        counter = 40;
        while (counter--)
        {
            printf("%c", *tmp++);
        }
        printf("\n");
        tmp = pIdCtrlr->FR;
        printf("IdentifyController: firmwarRev: ");
        counter = 8;
        while (counter--)
        {
            printf("%c", *tmp++);
        }
        printf("\n");

        free(raidConfigurationIoctl);
        free(nvmePassThroughIoctl);
        CloseHandle(hDevice);
        break;
    }

    CloseHandle(hDevice);
}

return 0;
}

```

6.1 References

Title	Link
NVM_Express_1_1b.pdf	https://svn.openfabrics.org/svnrepo/nvmewin/trunk/source/nvme.h

§§