



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

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

Document Revision 1.04



Disclaimer

By using this document, in addition to any agreements you have with Intel, you accept the terms set forth below.

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.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document 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.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: <http://www.intel.com/design/literature.htm> Code names featured are used internally within Intel to identify products that are in development and not yet publicly announced for release. Customers, licensees and other third parties are not authorized by Intel to use code names in advertising, promotion or marketing of any product or services and any such use of Intel's internal code names is at the sole risk of the user.

Intel, Atom, Core, and the Intel logo are trademarks of Intel Corporation or its subsidiaries in the U.S. and other countries.

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

Copyright ©2017 Intel Corporation. All rights reserved.



1	Intel® VROC IOCTLs Overview	5
1.1	The purpose of this document	5
1.2	Intel® Volume Management Device (Intel® VMD) Overview	5
1.3	Journaling Drives	5
1.4	Backwards compatibility	5
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	8
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	10
3.1	Overview	10
3.2	Microsoft firmware update NVMe IOCTLs	10
3.3	Firmware update using SCSI pass through IOCTL	11
4	Private NVMe IOCTLs support	12
4.1	Private IOCTLs signature and control code details.	12
4.2	Private IOCTLs data structure	12
5	Private Intel® VROC IOCTLs	14
5.1	Overview.	14
5.2	List of private IOCTLs for RAID volumes	14
5.3	NVME_GET_NUMBER_OF_RAID_VOLUMES	14
5.3.1	IOCTL signature and control code:	15
5.3.2	IOCTL data structure:	15
5.4	NVME_GET_RAID_INFORMATION	15
5.4.1	IOCTL signature and control code	15
5.4.2	IOCTL data structure	16
5.5	NVME_GET_RAID_CONFIGURATION	16
5.5.1	IOCTL signature and control code	17
5.5.2	IOCTL data structure	17
5.5.3	Input/Output DataBuffer details	17
5.6	NVME_GET_NUMBER_OF_SPARE_DISKS	18
5.6.1	IOCTL signature and control code:	18
5.6.2	IOCTL data structure:	18
5.7	NVME_GET_SPARE_DISKS_INFORMATION	19
5.7.1	IOCTL signature and control code	19



5.7.2	IOCTL data structure.....	19
5.7.3	Input/Output DataBuffer structure.....	19
5.8	NVME_GET_NUMBER_OF_PASSTHROUGH_DISKS	20
5.8.1	IOCTL signature and control code:	20
5.8.2	IOCTL data structure:	21
5.9	NVME_GET_PASSTHROUGH_DISKS_INFORMATION	21
5.9.1	IOCTL signature and control code	21
5.9.2	IOCTL data structure.....	21
5.9.3	Input/Output DataBuffer structure.....	22
5.10	NVME_GET_NUMBER_OF_JOURNALING_DRIVES	23
5.10.1	IOCTL signature and control code:	23
5.10.2	IOCTL data structure:	23
5.11	NVME_GET_JOURNALING_DRIVES_INFORMATION	23
5.11.1	IOCTL signature and control code	23
5.11.2	IOCTL data structure.....	24
5.11.3	Input/Output DataBuffer structure	24
5.12	NVME_PASS_THROUGH_SRB_IO_CODE.....	25
5.12.1	IOCTL signature and control code	25
5.12.2	IOCTL data structure.....	25
5.12.3	Error handling	27
6	Example usage.....	29
7	References.....	35



1 Intel® VROC IOCTLS Overview

1.1 The purpose of this document

This document presents all NVMe admin commands supported by Intel VROC and describes how to send them to a particular NVMe device. This includes usage of Intel® VROC IOCTLS to get information about disks in the system and using those information to send NVMe private IOCTL to a given disk. Ending section consists of sample code showing how to send an NVMe admin command to an NVMe RAID member device.

1.2 Intel® Volume Management Device (Intel® VMD) Overview

In previous versions of Intel® Rapid Storage Technology enterprise (Intel® RSTe) NVMe driver, all pass-through devices had a storage controller correlated with it. If we wanted to send an NVMe pass-through IOCTL to a device, we had to send it to its controller. If we created a RAID volume, its member disks were no longer visible in device manager. Devices marked as spare behaved the same. In order to send an IOCTL to particular drives we had to target Intel® RSTe Virtual Controller, with Return Code set to value correlated with this drive. This code could be obtained by private Intel® RSTe NVMe IOCTLS.

Intel® VROC driver on the other hand has been developed with Intel® Volume Management Device (Intel® VMD) in mind. With this technology, we can group NVMe devices into VMD domains. Because of that, even if IOCTL target disk is passthrough, it should be sent to a VMD controller representing a domain it is a part of and contains a correct Return Code. This code can be obtained by new IOCTLS: NVME_GET_NUMBER_OF_PASSTHROUGH_DISKS (Described in 5.8) and NVME_GET_PASSTHROUGH_DISKS_INFORMATION (Described in 5.9).

1.3 Journaling Drives

Intel® VROC introduces a new feature: RAID5 Write Hole closure. In context of sending IOCTLS, we only need to know, that this feature introduces new possible disk usage in RAID systems: Journaling Drive. Journaling Drive as any other drive can be a target of NVMe passthrough IOCTL. To target such drive we need to send it to VMD controller representing domain it is part of and set Return Code to value correlated with that drive. Obtaining that value for Journaling Drives is possible by sending IOCTLS: NVME_GET_NUMBER_OF_JOURNALING_DRIVES (Described in 5.10) and NVME_GET_JOURNALING_DRIVES_INFORMATION (Described in 5.11).

1.4 Backwards compatibility

Most structures of IOCTLS used to send NVMe passthrough IOCTL and access RAID members and spare devices haven't changed since RSTe 4.3. The differences are that:



- they must now target Intel VMD instead of Intel® RSTe Virtual Controller,
- the `NVME_MEMBER_DISK_INFORMATION` structure has been extended and therefore the output buffer for the IOCTLS that return information about drives must be bigger (for updated `NVME_MEMBER_DISK_INFORMATION` structure layout check 5.5.3).



2 NVMe admin commands

Intel® VROC supports all admin commands currently supported by NSG standalone driver including vendor specific 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

To read values of SMART and general health information the application sends Private IOCTL ADMIN_GET_LOG_PAGE.

Intel® VROC uses private IOCTL mechanism to pass through ADMIN_GET_LOG_PAGE command to appropriate NVMe device.

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.

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 identify command.

Intel® VROC uses private IOCTL mechanism to pass through ADMIN_IDENTIFY command to appropriate NVMe device.

Standalone Intel® NVMe driver supports this IOCTL properly.

2.5 Asynchronous Events

Intel® VROC uses private IOCTL mechanism to pass through ADMIN_ASYNCHRONOUS_EVENT_REQUEST command to appropriate NVMe device. 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, host software needs to issue one or more Asynchronous Event Request commands to the controller.

2.6 NVMe Firmware update

Intel® VROC uses private IOCTL mechanism to pass through ADMIN_FIRMWARE_IMAGE_DOWNLOAD command to appropriate NVMe device. Standalone Intel® NVMe driver supports this IOCTL properly.

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 ADMIN_FIRMWARE_IMAGE_DOWNLOAD command. To select which firmware version will be executed after NVMe device reset ADMIN_FIRMWARE_ACTIVATE command must be used.

2.7 NVMe Firmware activate

Intel® VROC uses private IOCTL mechanism to pass through ADMIN_FIRMWARE_ACTIVATE command.

Standalone Intel® NVMe driver supports this IOCTL properly.

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.

2.8 Get Features

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



Intel® VROC uses private IOCTL mechanism to pass through ADMIN_GET_FEATURES command.
Intel® VROC passes this command to NVMe device in pass through mode and RAID mode.

2.9 Set Features

The Set Features command specifies the attributes of the Feature indicated.
Intel® VROC uses private IOCTL mechanism to pass through ADMIN_SET_FEATURES command.
Intel® VROC passes this command to NVMe device in pass through mode and 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 uses private IOCTL mechanism to pass through ADMIN_DEVICE_SELF_TEST command.
Intel® VROC passes this command to NVMe device in pass through mode and 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 passes this command to NVMe device in pass through mode and 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 passes this command to NVMe device in pass through mode and 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 passes this command to NVMe device in pass through mode and RAID mode.



3 Public NVMe IOCTLS support

3.1 Overview

Intel® VROC supports number of IOCTLS described below. Support will be limited to passing through described commands to NVMe driver, according to their signature.

3.2 Microsoft firmware update NVMe IOCTLS

Following firmware update IOCTLS are not blocked by Intel® VROC 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 case of NVMe devices under VMD controller, firmware IOCTLS shall be sent to handle, obtained by `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);
```



3.3 Firmware update using SCSI pass through IOCTL

With Intel® VROC it is possible to update device firmware using 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 SCSI Primary Commands specification.



4 Private NVMe IOCTLS support

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

If target disk isn't part of any VMD domain (VMD disabled), IOCTL can be sent directly to its controller as with standalone NVMe driver (please note, this is out of scope of Intel® VROC driver).

Otherwise it should be sent to one of the VMD controllers. To address 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: rsteDiskID

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

5.1 Overview.

Intel® VROC supports creating RAID volumes only with devices connected to PCI-E slots with VMD enabled.

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

5.2 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	Similar to pass through NVME: It has the same control code but different signature. This IOCTL will be used to send admin commands to NVMe RAID member and spare devices

IOCTLs added in Intel® VROC has been highlighted in green. They are not supported by previous versions of RSTe. Rest of the IOCTLs haven't been changed since previous versions of Intel® RSTe.

5.3 NVME_GET_NUMBER_OF_RAID_VOLUMES

This IOCTL can be used to get number of NVME RAID volumes present in target VMD domain.



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_NUMBER_OF_RAID_VOLUMES \
    CTL_CODE(NVME_STORPORT_DRIVER, 0x805, METHOD_BUFFERED, FILE_ANY_ACCESS)
```

5.3.2 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.

5.4 NVME_GET_RAID_INFORMATION

This IOCTL provides general information about RAID volume:

- Model,
- Firmware version,
- Serial number,
- Total number of RAID 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_INFORMATION \
    CTL_CODE(NVME_STORPORT_DRIVER, 0x806, METHOD_BUFFERED, FILE_ANY_ACCESS)
```



5.4.2 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.

5.5 NVME_GET_RAID_CONFIGURATION

This IOCTL can be send to Intel® VROC driver to get following information about NVME member devices:

- rsteDiskID,
- 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 RAID member devices.

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
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 DataBuffer details

Application must provide "big enough" DataBuffer to get information about all member devices.

The size of DataBuffer:

```
ReturnBufferLen = numberOfMemberDisks * sizeof(NVME_MEMBER_DISK_INFORMATION)
```

Where:

numberOfMemberDisks – is a total number of NVMe devices used to build RAID volume.

NVME_MEMBER_DISK_INFORMATION - data structure containing information about

NVMe device:

```
typedef struct _NVME_MEMBER_DISK_INFORMATION {
    ULONG rsteDiskID;
    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 rsteDiskID	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 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 <i>slotNumber</i> field has to 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 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 driver to get the following information about NVME spare devices:

- rsteDiskID,
- 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 devices marked as spare.

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 DataBuffer: ReturnBufferLen = numberOfSpareDisks * sizeof(NVME_DISK_INFORMATION)
UCHAR DataBuffer[1]	In/Out	Data buffer for member devices details.

5.7.3 Input/Output DataBuffer structure

Application must provide "big enough" DataBuffer to get information about all spare devices.



The size of DataBuffer:

```
ReturnBufferLen = numberOfSpareDisks * sizeof(NVME_DISK_INFORMATION)
```

Where:

numberOfSpareDisks – is a total number of NVMe devices marked as spare.

NVME_DISK_INFORMATION - data structure containing information about NVMe disk:

```
typedef struct _NVME_DISK_INFORMATION {
    ULONG rsteDiskID;
    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 rsteDiskID	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 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 <i>slotNumber</i> field has to 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 target VMD domain.

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 driver to get the following information about NVME passthrough devices:

- rsteDiskID,
- 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.

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 DataBuffer structure

Application must provide "big enough" DataBuffer to get information about all passthrough devices.

The size of DataBuffer:

```
ReturnBufferLen = numberOfPassthroughDisks * sizeof(NVME_DISK_INFORMATION)
```

Where:

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

NVME_DISK_INFORMATION - data structure containing information about NVMe disk:

```
typedef struct _NVME_DISK_INFORMATION {
    ULONG rsteDiskID;
    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 rsteDiskID	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
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 <i>slotNumber</i> field has to 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 total number of NVMe devices that are used as journaling drives in 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 driver to get the following information about NVME journaling drives:

- rsteDiskID,
- 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 journaling drives.

Size of this IOCTL depends on total number of NVMe journaling drives in 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 DataBuffer: ReturnBufferLen = numberOfJournalingDrives * sizeof(NVME_DISK_INFORMATION)
UCHAR DataBuffer[1]	In/Out	Data buffer for journaling drives details.

5.11.3 Input/Output DataBuffer structure

Application must provide “big enough” DataBuffer to get information about all passthrough devices.

The size of DataBuffer:

```
ReturnBufferLen = numberOfJournalingDrives * sizeof(NVME_DISK_INFORMATION)
```

Where:

numberOfJournalingDrives – is a number of NVMe journaling drives in target domain.
NVME_DISK_INFORMATION - data structure containing information about NVMe disk:

```
typedef struct _NVME_DISK_INFORMATION {
    ULONG rsteDiskID;
    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 rsteDiskID	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 <i>slotNumber</i> field has to be used to identify the drives instead.
ULONG slotNumber	Output	Slot identifier.

5.12 NVME_PASS_THROUGH_SRB_IO_CODE

Intel® VROC will provide an IOCTL to pass through admin commands to all NVMe devices.

Warning: all IOCTLs for NVMe devices need to be send to Intel® VMD, instead of sending directly to NVMe devices.

List of admin commands which can be send 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 passthrough IOCTL interface Intel® VROC will reuse NVME_PASS_THROUGH_IOCTL data structure introduced in standalone NVMe driver. To properly identify RAID member disk, a property "ReturnCode" in SRB_IO_CONTROL data structure will be used.

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



```
typedef struct _SRB_IO_CONTROL {
    ULONG HeaderLength;
    UCHAR Signature[8];
    ULONG Timeout;
    ULONG ControlCode;
    ULONG ReturnCode;    //rsteDiskID 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 rsteDiskID. rsteDiskID 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
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
-------------------------	-------	---

Note:

If NVMe pass through IOCTL was used to update firmware on a device which is a RAID member disk, a full system reboot is required. Any try 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 DeviceIoControl API. First, it's the return code of the API. Second, it's the ReturnCode field of SRB_IO_CONTROL structure, which is marked down by miniport driver. The third level is the completion status included in the completion entry after the request had been issued to the controller. It's recommended that user applications look into all three levels of status to ensure the request is completed successfully.

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

When 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 certain 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 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 always marks SrbStatus as SRB_STATUS_SUCCESS no matter what. In case of any errors, driver just marks down proper status code in ReturnCode. Therefore, the basic scenario user applications need to follow to identify any errors after calling DeviceIoControl is:

1. When DeviceIoControl returns with error, GetLastError is used to find out 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 previous chapter. It shows 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 big enough buffer for the next IOCTL: NVME_GET_RAID_CONFIGURATION, which returns information about all member devices of that volume. From data returned by this IOCTL application gets `rsteDiskID` of the first member disk. Finally IOCTL NVME_PASS_THROUGH_SRB_IO_CODE is being sent to that disk. In this example, payload of this IOCTL is command ADMIN_IDENTIFY. After receiving it, applications prints information about VID, SSVID, serial number, model number and firmware version of target disk. During execution it also prints SCSI port on which it found miniport, number of volumes on that port, member devices count of the first found volume and serial number and `rsteDiskID` of the first disk returned by NVME_GET_RAID_CONFIGURATION.

This code uses `nvme.h` header, which available in the internet¹

```
#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;
```

¹ <https://svn.openfabrics.org/svnrepo/nvmewin/trunk/source/nvme.h>



```

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 rsteDiskID;
    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 dw10;
    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:"
    wprintf(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's 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\nrsteDiskId:%lx\n", diskInfo->serialNumber, diskInfo-
>rsteDiskID);

/* 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->rsteDiskID;
        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;
    }

```

Example usage



```
        CloseHandle(hDevice);
    }
}
return 0;
}
```



7 References

NVM_Express_1_1b.pdf

<https://svn.openfabrics.org/svnrepo/nvmewin/trunk/source/nvme.h>