

Intel® Virtual RAID on CPU Private UEFI Volume Information Protocol

User Guide

Revision 2.0

October 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.

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, 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 Summary 4
1.1 Protocol GUID 4
1.2 Protocol Interface Structure 4
1.2.1 Parameters 5
1.2.2 Related Definitions 5
1.3 Description 5
1.4 Example 6
1.4.1 Example Sample Code Output 10

1 Summary

Intel® Virtual RAID on CPU (Intel® VROC) version 8.0 provides a new protocol for retrieving RAID volume information.

Intel® VROC volume information protocol can be used to retrieve information for all RAID volumes created through Intel® Rapid Storage Technology enterprise (Intel® RSTe), UEFI mode only, and Intel® VROC. Changes can also be seen using the Human Interface Infrastructure (HII), Intel® RSTe, and Intel® VROC UEFI tools.

1.1 Protocol GUID

```
// 6139796A-3DF0-4838-827B-427E7DC8281C}

#define EFI_VROC_VOLUME_INFO_PROTOCOL_GUID \

      { 0x6139796a, 0x3df0, 0x4838,{
0x82,0x7b,0x42,0x7e,0x7d,0xc8,0x28,0x1c} }
```

1.2 Protocol Interface Structure

```
#pragma pack( 1 )

typedef struct _VROC_VOLUME_INFO_PROTOCOL

{

    EFI_STRING  VendorID;

    EFI_STRING  ProductID;

    EFI_STRING  VolumeName;

    UINT64      TotalBlocks;

    UINTN       BlockSize;

    UINT8       RaidLevel;

    UINT8       VolumeType;

} VROC_VOLUME_INFO_PROTOCOL;

#pragma pack()
```

1.2.1 Parameters

1. VendorID – Volume vendor identification string.
2. ProductID – Product identification string.
3. VolumeName – Name of a RAID volume.
4. TotalBlocks – Total size of volume, in blocks.
5. BlockSize – Logical block size in bytes.
6. RaidLevel – RAID Volume level (see related definitions below).
7. VolumeType – RAID Volume type, determines if volume was created as VROC, SATA or SSATA (see related definitions below).

1.2.2 Related Definitions

- Available RAID levels:

```
#define VROC_VOLUME_RAID0     0   // RAID volume level 0 (Stripe)

#define VROC_VOLUME_RAID1     1   // RAID volume level 1 (Mirror)

#define VROC_VOLUME_RAID5     5   // RAID volume level 5 (Parity)

#define VROC_VOLUME_RAID10   10   // RAID volume level 10 (RAID level 0 +
1)
```

- Available volume types:

```
#define VROC_VOLUME_SATA      0   // RAID volume created on SATA
controller

#define VROC_VOLUME_sSATA     1   // RAID volume created on sSATA
controller

#define VROC_VOLUME_VROC      2   // RAID volume created on VROC
controller

#define VROC_VOLUME_sSATA2    3   // RAID volume created on sSATA2
controller
```

1.3 Description

This protocol is installed on each handle of the Intel® RSTe/Intel® VROC RAID volume.

The caller should not try to deallocate or change values of the memory under returned pointer, as this can lead to undefined behavior. It is the Intel driver's responsibility to allocate and free this memory area whenever necessary.

If caller service intends to keep device information, i.e. for a lifetime of a global structure instance, a copy of the interface memory should be made, and protocol

should be closed. This is dictated by the fact that protocol data can change in time if device enumeration is done (i.e. create or delete volume).

To calculate device capacity, TotalBlocks and BlockSize parameters can be used. The following example shows how to get disk size in bytes:

```
UINT64 sizeInBytes = result->TotalBlocks * result->BlockSize;
```

1.4 Example

The below code is just a sample code. It shows the usage of Intel® VROC volume info protocol from a UEFI Shell application.

```
#include <Uefi.h>
#include <Library/UefiApplicationEntryPoint.h>
#include <Library/UefiBootServicesTableLib.h>
#include <Library/UefiLib.h>

/**
  VROC Private Volume Info Protocol GUID.
  {6139796A-3DF0-4838-827B-427E7DC8281C}
**/
#define EFI_VROC_VOLUME_INFO_PROTOCOL_GUID \
      { 0x6139796a, 0x3df0, 0x4838,{
0x82,0x7b,0x42,0x7e,0x7d,0xc8,0x28,0x1c} }

#define VROC_VOLUME_RAID0           0
#define VROC_VOLUME_RAID1           1
#define VROC_VOLUME_RAID5           5
#define VROC_VOLUME_RAID10         10

#define VROC_VOLUME_SATA            0
#define VROC_VOLUME_sSATA           1
#define VROC_VOLUME_VROC            2
#define VROC_VOLUME_sSATA2          3

/**
  VROC Private Volume Info Protocol GUID global variable.
**/
EFI_GUID gEfiVROCVolumeInfoProtocolGuid =
EFI_VROC_VOLUME_INFO_PROTOCOL_GUID;

/**
  RSTe Volume Info protocol interface.
**/
#pragma pack( 1 )
typedef struct _VROC_VOLUME_INFO_PROTOCOL
{
    EFI_STRING  VendorID;
    EFI_STRING  ProductID;
    EFI_STRING  VolumeName;
    UINT64      TotalBlocks;
    UINTN       BlockSize;
    UINT8       RaidLevel;
    UINT8       VolumeType;
} VROC_VOLUME_INFO_PROTOCOL;
#pragma pack()
```



Summary

```
/**
  The user Entry Point for Application.

  @param[in] ImageHandle    The firmware allocated handle for the EFI
image.
  @param[in] SystemTable    A pointer to the EFI System Table.

  @retval    EFI_SUCCESS    The entry point is executed successfully.
  @retval    other          Some error occurs when executing this entry
point.

**/
EFI_STATUS
EFIAPI
UefiMain(
    IN EFI_HANDLE         ImageHandle,
    IN EFI_SYSTEM_TABLE   *SystemTable
)
{
    EFI_HANDLE                * pHandleBuf = NULL;
    UINTN                       nHandles = 0;
    UINTN                       nIdx = 0;
    EFI_STATUS                  nStatus = EFI_SUCCESS;
    VROC_VOLUME_INFO_PROTOCOL * result = NULL;

    Print(L"\nIntel(R) VROC Volume Info Protocol Example.\n\n");

    Print(L"Searching for RSTE_VOLUME_INFO_PROTOCOL instances...\n");

    nStatus = gBS->LocateHandleBuffer(
        ByProtocol,
        &gEfiVROCVolumeInfoProtocolGuid,
        NULL,
        &nHandles,
        &pHandleBuf
    );

    if (nStatus == EFI_OUT_OF_RESOURCES)
    {
        Print(L"ERROR: There is not enough pool memory to store the
matching results.\n");
        return nStatus;
    }

    if (nStatus == EFI_INVALID_PARAMETER)
    {
        Print(L"ERROR: One of the parameters has an invalid value.\n");
        return nStatus;
    }

    if (nStatus == EFI_NOT_FOUND)
    {
        Print(L"No instance of VROC volume info protocol was found.\n");
        return nStatus;
    }

    Print(L"Found %d instances.\n\n", nHandles);
    for (nIdx = 0; nIdx < nHandles; nIdx++)
```

```
    {
        if (pHandleBuf[nIdx])
        {
            Print(L"[%d] Opening RSTE_VOLUME_INFO_PROTOCOL.\n", nIdx);

            nStatus = gBS->OpenProtocol(
                pHandleBuf[nIdx],
                &gEfiVROCVolumeInfoProtocolGuid,
                (void**)&result,
                ImageHandle,
                NULL,
                EFI_OPEN_PROTOCOL_BY_HANDLE_PROTOCOL
            );

            if (nStatus == EFI_INVALID_PARAMETER)
            {
                Print(L"[%d] ERROR: One of the parameters has an invalid
value.\n", nIdx);
                continue;
            }

            if (nStatus == EFI_ACCESS_DENIED || nStatus ==
EFI_ALREADY_STARTED)
            {
                Print(L"[%d] ERROR: Already opened with BY_DRIVER or
EXCLUSIVE attribute.\n", nIdx);
                continue;
            }

            if (result)
            {
                UINT64 sizeInBytes = result->TotalBlocks * result-
>BlockSize;

                Print(L"[%d] %s %s (%s)\n",
                    nIdx,
                    result->VendorID,
                    result->ProductID,
                    result->VolumeName
                );

                Print(L"      Total blocks: %ld\n", result->TotalBlocks);
                Print(L"        Block size: %d\n",  result->BlockSize);
                Print(L"     Size in bytes: %ld\n", sizeInBytes);
                Print(L"        Raid Level: %d\n",  result->RaidLevel);

                switch (result->VolumeType)
                {
                case VROC_VOLUME_SATA:
                    Print(L"       Volume Type: SATA\n");
                    break;
                case VROC_VOLUME_sSATA:
                    Print(L"       Volume Type: SSATA\n");
                    break;
                default:
                    Print(L"       Volume Type: VROC\n");
                    break;
                }
            }
```

```
                    else
                    {
                        Print(L"[%d] ERROR: No interface returned.\n", nIdx);
                    }

                    Print(L"[%d] Closing VROC_VOLUME_INFO_PROTOCOL.\n\n", nIdx);

                    gBS->CloseProtocol(
                        pHandleBuf[nIdx],
                        &gEfiVROCVolumeInfoProtocolGuid,
                        ImageHandle,
                        NULL
                    );
                }
            }

            gBS->FreePool(pHandleBuf);
            return EFI_SUCCESS;
        }
```

1.4.1 Example Sample Code Output

```
FS0:\> rstevRSTeVolumeInfo.efi

Intel(R) VROC Volume Info Potocol Example.

Searching for RSTE_VOLUME_INFO_PROTOCOL instances...
Found 3 instances.

[0] Opening RSTE_VOLUME_INFO_PROTOCOL.
[0] Intel VROC RAID 1 Volume (Volume001)
      Total blocks: 742340608
        Block size: 512
     Size in bytes: 380078391296
        Raid Level: 1
       Volume Type: VROC
[0] Closing VROC_VOLUME_INFO_PROTOCOL.

[1] Opening RSTE_VOLUME_INFO_PROTOCOL.
[1] Intel RAID 1 Volume (Volume002)
      Total blocks: 927924224
        Block size: 512
     Size in bytes: 475097202688
        Raid Level: 1
       Volume Type: SATA
[1] Closing VROC_VOLUME_INFO_PROTOCOL.

[2] Opening RSTE_VOLUME_INFO_PROTOCOL.
[2] Intel RAID 0 Volume (Volume003)
      Total blocks: 92776448
        Block size: 512
     Size in bytes: 47501541376
        Raid Level: 0
       Volume Type: SATA
[2] Closing VROC_VOLUME_INFO_PROTOCOL.

FS0:\>
```

§§