VIA RAID Library Manual

VIA Technologies, Inc.

2005-05-25

Table of Content

0. Introduction

This library provides some basic RAID operations based on VIA's RAID controller. User can use it to build applications to manage disk RAID. Four data structures are defined to record the information of RAID related hardware and information of all the disk arrays in system. All the APIs in this lib can be divided to the following classes:

- Get system information, including hardware information and disk array information.
- Create a new disk raid or remove an existent disk raid.
- Repair broken disk raid.
- Sync and verify of RAID1, RAID01 and RAID5.
- Migration of RAID0, RAID01, and RAID5.
- Miscellaneous functions, including initialization of raid lib, exit process, update hardware information, etc

1. Rules of Program Naming

All the definitions and function declarations are placed in a single file: <raid_lib.h>. At the start of this file, some common used data types are defined, and all the defined data type is started with character 'V'. To make things simple, the usage of all the functions and variables of raid lib is C style. And in the naming of program symbols, the following rules are applied:

- All the constants and functions' names are started with VR or vr.
- The naming style of all functions and their parameters is not "HUNGARIAN STYLE" as Windows programs used to use. To be more portable, it use a traditional one instead, e.g. vr_create_mirror.
- Symbol name prefers a concise one to a long one, e.g. it use vr_get_device_info instead of vr_get_device_information.
- In raid lib, word **array** and **raid** are different. Array includes disk raid and standalone disk, in others words, every single disk which is not in a valid raid will be regarded as a special disk array with standalone disk type.
- In raid lib, word **device** means all kinds of device attached to IDE controllers, maybe hard disk, or ATAPI device.

2. Definition of Basic Data Type

In the header file of raid lib, the following common used data types are defined. All fields of the data structure defined in this document will use these data types. The reason to define them is to be convenient when porting the raid lib to other platforms.

```
typedef int                 VINT;
typedef int                 VBOOL;
typedef char                VCHAR;
typedef unsigned char               VBYTE;
typedef unsigned short              VWORD;
typedef unsigned long               VDWORD;
typedef unsigned __int64            VDWORD64;  (In Linux, will using uint64_t data type.)
```

3. Definition of Constant

3.1 Number Ranges

The followings are the definitions of some number ranges, or the maximum count of some basic devices.

```
#define  VR_MAX_DEVICES 128
            The max number of devices per computer is 32.
#define VR_MAX_CHANNELS 16
            The max number of channels per computer is 16.
#define VR_MAX_CONTROLLERS          (VR_MAX_DEVICES / 8)
            The max number of  IDE controllers per computer is also 16.
#define VR_MAX_ARRAYSIZE   16
            The max number of devices per disk array is 32.
#define VR_MAX_ARRAYS        VR_MAX_DEVICES
            The max number of disk arrays per computer is 32.
#define VR_MAX_STRIPE_INDEX 4
            The max stripe size index number is 4.
#define VR_MAX_BLOCK_INDEX 15
            The max block size index number is 15 (used when creating RAID5).
```

3.2 Error Codes

The most APIs in raid lib will return VR_SUCCESS when running successfully. And when some errors found in the processing, a specific error code will be returned. The followings are the definitions of error codes. The details of these error codes will be described at API illustration part of this document.

```
enum {
            VR_SUCCESS                              =0,

            VR_ERR_NOT_INITED                       =100,
            VR_ERR_INIT_FAILED,
            VR_ERR_INIT_AGAIN,
            VR_ERR_INIT_DRV_ABSENT,
            VR_ERR_INIT_NO_DISK,
            VR_ERR_INIT_DRV_ERROR,
            VR_ERR_RAIDLIB_BUSY,

            VR_ERR_INVALID_INDEX                    =200,
            VR_ERR_INVALID_PARAM,
            VR_ERR_RAIDTYPE_UNSUPPORT,
            VR_ERR_RAID_BROKEN,
            VR_ERR_DISK_ABSENT,
            VR_ERR_DISK_ERROR,
            VR_ERR_DISK_NOT_ENOUGH,
            VR_ERR_CREATE_DISK_USED,
            VR_ERR_CREATE_DIFF_CTRLER,
            VR_ERR_CREATE_DISKSIZE_INVALID,
            VR_ERR_CREATE_STRIPESIZE_INVALID,
            VR_ERR_CREATE_BLOCKSIZE_INVALID,

            VR_ERR_SYNCVERIFY_FAILED                  =300,
            VR_ERR_SYNCVERIFY_DISK_INVALID,
            VR_ERR_SYNCVERIFY_ERROR_RAIDTYPE,
            VR_ERR_SYNCVERIFY_OFFSET_INVALID,
            VR_ERR_SYNCVERIFY_NUM_INVALID,
            VR_VERIFY_MISMATCH,

            VR_ERR_REPAIR_CANNOT_REPAIR               =400,
            VR_ERR_REPAIR_NONEED_REPAIR,

            VR_ERR_MIGRATION_NONEED                   =500,
            VR_ERR_MIGRATION_LOG_FAILED,
            VR_ERR_MIGRATION_FAILED,
            VR_MIGRATION_FINISH,

            VR_ERR_RAID5INIT_NONEED                   =600,
            VR_ERR_RAID5INIT_LOG_FAILED,
            VR_ERR_RAID5INIT_FAILED,
            VR_RAID5INIT_FINISH,

            VR_ERR_RAID5SYNC_NONEED                   =700,
            VR_ERR_RAID5SYNC_LOG_FAILED,
            VR_ERR_RAID5SYNC_FAILED,
            VR_RAID5SYNC_FINISH,

            VR_ERR_SYSTEM_NEED_REBOOT                 =800,
            VR_ERR_UNEXPECTED_ERROR,
            VR_HARDWARE_INFO_CHANGED,
            VR_SMART_DISK_DEGRADING,
            VR_ERR_NOT_PERMITTED,

            VR_ERR_CONFIG_UNKNOWN                     =900,
};
```

3.3 Array Type

In this version of raid lib, 4 kinds of raid will be supported: RAID0, RAID1, SPAN, and RAID01. The followings are the definitions. The type of VR_STANDALONE_DISK is a special one, in the raid lib one single disk can be regarded as a array with this type.

```
#define VR_RAID0      0
#define VR_RAID1      1
#define VR_RAID2      2
#define VR_RAID3      3
#define VR_RAID4      4
#define VR_RAID5      5
#define VR_RAID6      6
#define VR_RAID7      7
```

```
#define VR_SPAN        8
#define VR_RAID01      9 /* raid 0+1 */
#define VR_MATRIXRAID   10
#define VR_STANDALONE_DISK       0xff
```

3.4 Array Position of Disk

Every device (disk) which belongs to one disk array will have its array position. We can using it to locate one given disk, in other words, we can fetch the disk's information by using the array related locations.

```
#define VR_RAID0_STRIPE0             0  /* first stripe disk of raid0 */
#define VR_RAID1_SOURCE              0  /* source disk of raid1 */
#define VR_RAID1_MIRROR              1  /* mirror disk of raid1 */
#define VR_RAID1_SPARE               2  /* spare disk of raid1 */
#define VR_SPAN_DISK0                0  /* first disk of span raid */
#define VR_RAID01_SOURCE_STRIPE0       0
              /* first stripe disk of source sub-raid of raid0+1 */
#define VR_RAID01_MIRROR_STRIPE0       8
              /* first stripe disk of mirror sub-raid of raid0+1 */
#define VR_RAID5_DISK0             0 /* first disk of raid5 */
#define VR_MATRIXRAID_DISK0           0 /* first disk of matrix raid */
```

The array position's definition of devices in RAID01 is some special, the mirror stripe part's sequence numbers are started by 8, and not sure to be continuous with the source stripe part's.

3.5 Device Status

Within the data structure of device information, there's a WORD field of device status, and the probable values of device status are defined as the following:

```
#define VR_DEVICE_STATUS_NEED_REBOOT             0x0001
              // need system reboot to enable the disk
#define VR_DEVICE_STATUS_NEED_SYNC             0x0002
              // is a mirror disk, need sync
#define VR_DEVICE_STATUS_DEGRADING             0x0004
              // reliability degrading, risk of data loss
#define VR_DEVICE_STATUS_ERROR                 0x0008
              // disk failed
#define VR_DEVICE_STATUS_NEED_CLEAR_PARTITIONS      0x0010
              // need to clear partition info on this disk
#define VR_DEVICE_STATUS_UNPLUGGED             0x0020
              // device is currently Unplugged
#define VR_DEVICE_STATUS_NEW_PLUG_IN             0x0040
              // device is newly plugged in
```

3.6 Array Status

Within the data structure of disk array information, there's a WORD field of device status, and the probable values of array status are defined as the following:

```
#define VR_ARRAY_STATUS_BROKEN      0x0001 // array is broken, function failed
#define VR_ARRAY_STATUS_NEED_REBOOT 0x0002 // need reboot to enable the array
#define VR_ARRAY_STATUS_NEED_SYNC   0x0004 // some mirror disks in the array
                                           // need sync
#define VR_ARRAY_STATUS_WARNING     0x0008 // general warning condition
#define VR_ARRAY_STATUS_NEED_INIT   0x0010 // only for RAID5, need init
```

3.7 Device Type

The devices attached to the RAID controllers can have 3 types: ATA device, SATA device or ATAPI device. The followings is the definitions:

```
#define VR_ATA_DEVICE    0
#define VR_ATAPI_DEVICE 1
#define VR_SATA_DEVICE  2
```

3.8 System Device Status

The device attached to the RAID controllers may contain system files of current running OS. The following 3 status values are defined to indicate the instance:

```
#define VR_DEVICE_NOT_SYS_DISK        0 /* has no system files */
#define VR_DEVICE_MAYBE_SYS_DISK      1 /* maybe has system files */
#define VR_DEVICE_SYS_DISK            2 /* has system files */
```

3.9 Configuration Parameters

User can use vr_config_param function to set/get parameters of raid lib. The followings are the entry parameters:

```
#define VR_CONFIG_GET_RAID5OPT 1
#define VR_CONFIG_SET_RAID5OPT 2
#define VR_CONFIG_GET_USERINFO 3
```

4. Data Structures

4.1 vr_controller_info_t

```
typedef struct  _vr_controller_info {
    VCHAR chipName[32];          // chipset name
    VCHAR manufactory[32];       // manufactory name
    VBYTE support_PIO;        // max PIO mode
    VBYTE support_MutiDMA;    // max Muti-word DMA mode
    VBYTE support_UltraDMA;   // max UltraDMA mode
    VBYTE channel_Num;        // how many channels
    VBOOL bSupportRaid0Migration; // whether this controller support migration
                                  // process of RAID0
    VBOOL bSupportRAID5;        // whether this controller support RAID5 functions
    VINT  index;                // internal index
} vr_controller_info_t;
```

This data type is used to record the information of Raid controller. And raid lib use it to deliver controllers' details to application. The field chipName and manufactory are hardware , and the next 4 fields are about hardware capabilities.

4.2 vr_channel_info_t

```
typedef struct _vr_channel_info {
    VINT   index;                          // internal index
```

```
        VINT   ctrlerIndex;                  // of which controller

        VDWORD io_Base;           // data/command base address
        VDWORD control_Base;       // control/status base address for this channel
        VDWORD BusMaster_Base;     // bus master control registers' base address
        VBYTE cableType;          // 0: 40 pin; 1: 80 pin; 2: SATA cable
        VBYTE bMaster_Exist;      // whether master drive exist on this channel
        VBYTE bSlave_Exist;       // whether slave drive exist on this channel

        VBOOL bSlaveValid;        // whether the slave device of this channel is valid
    } vr_channel_info_t;
```

This data type is used to record the of every IDE channel. And raid lib use it to deliver channels’ details to application. All the fields are used to describe hardware capabilities.

4.3 vr_device_info_t

```
    typedef struct _vr_device_info {
        VCHAR serialNumber[32];
        VCHAR firmwareRevison[16];
        VCHAR modelName[64];
        VCHAR minorRevisonNumber[64];

         // CHS parameters
        VDWORD cylinderNumber;
        VDWORD headNumber;
        VDWORD sectorNumberPerTrack;

        // following 8 bytes define the capcity (in sector) of the disk
        VDWORD capacityLow;        // (Unit: sector)
        VDWORD capacityHigh;       // (Unit: sector)

        // if any following supportxxxMode member has a value 0xFF, it will
        // mean the mode is totally unsupported, a value other than 0xFF means
        // the max mode number.
        VBYTE supportPIOTransferMode;
        VBYTE supportMultiDMAMode;
        VBYTE supportUltraDMAMode;

        // current transfer mode
        // bit 5,6,7: 0,PIO; 1,Multi-DMA; 2,Ultra-DMA
        // bit 0-4: Mode number
        VBYTE currentTransferMode;

        VBYTE deviceType; // the possible values are:
                          // VR_ATA_DEVICE, VR_ATAPI_DEVICE,
                          // VR_SATA_DEVICE

        // device status
        VWORD status; /* the invalid value are defined in this file, and all
                start with VR_DEVICE_STATUS */

        // physical position info
        VBYTE ctrler_index;      /* which controller */
        VBYTE chan_index;        /* which channel in the controller */
        VBOOL master;           /* master or slave */
        VINT     index;         /* index of all devices in system */
        VBYTE systemDisk;        /* does the physical device contain system
                                    files of current running OS ?
                                    the probably value are:
                                    VR_DEVICE_NOT_SYS_DISK
                                    VR_DEVICE_MAYBE_SYS_DISK
                                    VR_DEVICE_SYS_DISK
                                    they are defined in this file */

        // device's disk-array-relevant info
        VBOOL  bDevInRaid;          // whether the device in a valid raid
        VWORD  array_position;      // positon within the raid
        VINT      array_index;      // index of the raid in which the device be
        VDWORD realCapacityLow;        // (Unit: sector)
        VDWORD realCapacityHigh;       // (Unit: sector)
    } vr_device_info_t;
```

This data type is used to record the information of every ATA or ATAPI device. And raid lib use it to deliver devices' details to application. All the field of this data structure can be separated to 7 classes:

a. Hardware identification information
b. CHS information
c. Capability in sectors
d. Hardware capabilities
e. Current status
f. Physical position information
g. Array related information

The possible values of field: status is listed at section 3.5 of this document.

4.4 vr_array_info_t

```
typedef struct _vr_array_info {
    VWORD status;

    VBYTE raidType; //  value 0xFF means
              // a stand-alone disk. When it's a stand-alone disk,
              // only diskNum has meaning, and diskNum should
              // always be 1 .
    VBYTE diskNum;// count of valid disk members.

    VDWORD capacityLow;// (Unit: sector)
    VDWORD capacityHigh;// (Unit: sector)

    VDWORD stripeSize; // valid when raid is raid0 or raid01, in  Kbytes
    VDWORD blockSize; // valid when raid is RAID5, in Kbytes
    VBOOL  bNeedMigration; // the raid need migration
                           // only valid when raid0/raid5/matrixRaid
    VBOOL  bNeedInit;     // the raid need initialization, only valid for RAID5
    VBOOL  bOptimized;     // only for RAID5, this RAID5 access was optimized
    VBYTE systemDisk;    /* does the devices within this disk array contain system
                            files of current running OS ? The probably value are:
                                VR_DEVICE_NOT_SYS_DISK
                                VR_DEVICE_MAYBE_SYS_DISK
                                VR_DEVICE_SYS_DISK
                              they are defined in this file */
```

```
        VBYTE raid_index;// only raid index, no meaning with stand-alone disk
        VINT    index; // all device index, including all raid and stand-alone disk
} vr_array_info_t;
```

This data type is used to record the information of every disk array in current system. And raid lib use it to deliver disk arrays' details to application.

The possible values of field: status is listed at section 3.6 of this document.

5. Application Interfaces of Raid Features

In the raid lib, all the APIs will return a integer value to indicate the result, expect for vr_exit. The return values' definition can be referred at section 3.2 of this document. Int the parameters of some functions, all the device and disk array will be specified by its internal index. And all the output values will be transferred by pointers.

5.1 Initialize and Exit

5.1.1 Initialize function:

Declaration:

```
VINT  vr_init (void);
```

Description:

Application should use this API to initialize the raid lib before using other APIs.

Input parameters:

None

Return value:

VR_SUCCESS :

Initialize the raid lib successfully

VR_ERR_INIT_FAILED :

Some errors found when initializing the raid lib

5.1.2 Exit function

Declaration:

```
void vr_exit (void);
```

Description:

Application should use this API to free some allocated resource before exit.

Input parameters:

None

Return value:

None

5.2 Get System Information

5.2.1 Get the version number of RAID library

Declaration:

```
VINT vr_get_version (VINT *pmajor, VINT *pminor,
                                        VINT *pmicro);
```

Description:

Query the version number of current RAID library.

Input parameters:

VINT *pmajor :

Pointer of variable to get the major version number

VINT *pminor :

Pointer of variable to get the minor version number

VINT *pmicro :

Pointer of variable to get the micro version number

Return value:

VR_SUCCESS :

Get the number successfully.

VR_ERR_INVALID_PARAM :

Input parameter is invalid: the pointer is NULL.

5.2.2 Get the number of raid controllers in system

Declaration:

`VINT  vr_get_controller_num (VINT *pnumber);`

Description:

Application can query the count of raid controllers in current system.

Input parameters:

VINT *pnumber :

Pointer of variable to get the number

Return value:

VR_SUCCESS :

Get the number successfully.

VR_ERR_NOT_INITED :

Raid lib hasn’t been initialized.

VR_ERR_INVALID_PARAM :

Input parameter is invalid: the pointer is NULL.

5.2.3 Get information of raid controllers

Declaration:

`VINT vr_get_controller_info (VINT index, vr_controller_info_t *pinfo);`

Description:

Application can fetch the information of one specific raid controller.

Input parameters:

VINT index :

Index to raid controllers, specify which controller.

vr_controller_info_t * pinfo :

Pointer to a vr_controller_info_t data structure to get the information

Return value:

VR_SUCCESS :

Get the information successfully.

VR_ERR_NOT_INITED :

Raid lib hasn’t been initialized.

VR_ERR_INVALID_INDEX :

The input index is invalid.

VR_ERR_INVALID_PARAM :

Input parameter is invalid: the pointer is NULL.

5.2.4 Get information of IDE channels

Declaration:

```
VINT  vr_get_channel_info (VINT ctrler_index, VINT chan_index,
                                        vr_channel_info_t* pinfo);
```

Description:

Application can fetch the information of one specific raid controller.

Input parameters:

VINT ctrler_index :

Index to raid controllers, specify which controller.

VINT chan_index :

Index to IDE channel, specify which channel of this controller.

vr_channel_info_t *pinfo :

Pointer to a vr_channel_info_t data structure to get the information

Return value:

VR_SUCCESS :

Get the information successfully.

VR_ERR_NOT_INITED :

Raid lib hasn't been initialized.

VR_ERR_INVALID_INDEX :

The input index is invalid.

VR_ERR_INVALID_PARAM :

Input parameter is invalid: the pointer is NULL.

5.2.5 Get one device's information by its physical address

Declaration:

```
VINT vr_get_device_info_by_phy_address (VINT ctrler_index,
                        VINT chan_index, VINT master, vr_device_info_t* pinfo);
```

Description:

Application can fetch the information of one specific device, which is located by physical address.

Input parameters:

VINT ctrler_index :

Index to raid controllers, specify which controller.

VINT chan_index :

Index to IDE channel, specify which channel of this controller.

VINT master :

Specify the master device or slave master of this IDE channel.

vr_device_info_t *pinfo :

Pointer to a vr_device_info_t data structure to get the information

Return value:

VR_SUCCESS :

Get the information successfully.

VR_ERR_NOT_INITED :

Raid lib hasn't been initialized.

VR_ERR_INVALID_INDEX :

The input index is invalid.

VR_ERR_DISK_ABSENT :

The specific device is absent.

VR_ERR_INVALID_PARAM :

Input parameter is invalid: the pointer is NULL.

5.2.6 Get the number of all devices in system

Declaration:

```
VINT vr_get_device_num (VINT *pnumber);
```

Description:

Application can query the count of all devices in current system.

Input parameters:

VINT *pnumber :

Pointer of variable to get the number.

Return value:

VR_SUCCESS :

Get the number successfully.

VR_ERR_NOT_INITED :

Raid lib hasn't been initialized.

VR_ERR_INVALID_PARAM :

Input parameter is invalid: the pointer is NULL.

5.2.7 Get one device's information by index

Declaration:

VINT vr_get_device_info (VINT index, vr_device_info_t* pinfo);

Description:

Application can fetch the information of one specific device, which is located by index of all devices in current system.

Input parameters:

VINT index :

Index to all devices in system, specify which device.

vr_device_info_t *pinfo :

Pointer to a vr_device_info_t data structure to get the information.

Return value:

VR_SUCCESS :

Get the information successfully.

VR_ERR_NOT_INITED :

Raid lib hasn't been initialized.

VR_ERR_INVALID_INDEX :

The input index is invalid.

VR_ERR_INVALID_PARAM :

Input parameter is invalid: the pointer is NULL.

5.2.8 Get the number of all disk arrays in system

Declaration:

VINT vr_get_array_num (VINT only_raid, VINT *pnumber);

Description:

Application can query the count of all disk arrays in current system. This API can also tell application the count of all disk raid (not including standalone disks).

Input parameters:

VINT only_raid :

Specify the count number is for only disk raid or not.

VINT *pnumber :

Pointer of variable to get the number.

Return value:

VR_SUCCESS :

Get the number successfully.

VR_ERR_NOT_INITED :

Raid lib hasn't been initialized.

VR_ERR_INVALID_PARAM :

Input parameter is invalid: the pointer is NULL.

5.2.9 Get information of disk arrays

Declaration:

VINT vr_get_array_info (VINT array_index, vr_array_info_t* pinfo);

Description:

Application can fetch the information of one specific disk array, which is located by index of all disk arrays in current system.

Input parameters:

VINT array_index :

Index to all disk arrays in system, specify which disk array.

vr_array_info_t *pinfo :

Pointer to a vr_array_info_t data structure to get the information

Return value:

VR_SUCCESS :

Get the information successfully.

VR_ERR_NOT_INITED :

Raid lib hasn't been initialized.

VR_ERR_INVALID_INDEX :

The input index is invalid.

VR_ERR_INVALID_PARAM :

Input parameter is invalid: the pointer is NULL.

5.2.10 Get one device's information by its array position

Declaration:

VINT vr_get_device_info_by_array_pos (VINT array_index, VINT pos,
vr_device_info_t* pinfo)

Description:

Application can fetch the information of one specific device, which is located by this device's position in the specific disk array.

Input parameters:

VINT array_index :

Index to all disk arrays in system, specify which disk array.

VINT pos :

Device's position within the specific disk array, the definitions of array position are at section 3.4 of this document.

vr_device_info_t *pinfo :

Pointer to a vr_device_info_t data structure to get the information.

Return value:

VR_SUCCESS :

Get the information successfully.

VR_ERR_NOT_INITED :

Raid lib hasn't been initialized.

VR_ERR_INVALID_INDEX :

The input array index or array position is invalid.

VR_ERR_INVALID_PARAM :

Input parameter is invalid: the pointer is NULL.

5.2.11 Query whether the system support dynamic RAID

Declaration:

VBOOL vr_is_system_support_dynamic_raid (void);

Description:

Query whether the OS and RAID lib support dynamic RAID process.

Input parameters:

None

Return value:

TRUE :

Support it.

FALSE :

Can not support it.

5.2.12 Query whether the system need reboot

Declaration:

VBOOL vr_is_system_need_reboot (void);

Description:

Query whether the system need reboot to update the RAID information and active the changed RAID. See details in section 6.7 of this document.

Input parameters:

None

Return value:

TRUE :

Need reboot.

FALSE :

Need not reboot.

5.2.13 Raid lib configuration information

Declaration:

`VINT vr_config_param (VINT function, void *param);`

Description:

Get/Set Raid lib configuration.

Input parameters:

VINT function :

Function number to configure the raid lib. The valid values are:

VR_CONFIG_GET_RAID5OPT 1
VR_CONFIG_SET_RAID5OPT 2
VR_CONFIG_GET_USERINFO 3

void *param:

Pointer to variable which has the value to set or fetch the value

Return value:

VR_SUCCESS :

Get the information successfully.

VR_ERR_NOT_INITED :

Raid lib hasn't been initialized.

VR_ERR_CONFIG_UNKNOWN

Invalid function entry number

5.3 Create and Remove Disk Raid

By now, only 4 kinds of raid can be created: RAID0, RAID1, SPAN and RAID0+1. In VIA's raid lib, there are 5 functions to create them, one general interface and 4 specific interfaces. Using raid lib, user can also remove a existent disk raid of any type. If one disk raid in system is a mirror raid, user can add or remove a spare disk to/from it.

5.3.1 General interface to create a disk raid

Declaration:

```
VINT  vr_create_raid (VINT type, VINT disk_num, VINT disks[ ],
                      unsigned long param , VDWORD paramBlockSize);
```

Description:

This function can create a disk raid of all supported type, including RAID0, RAID1, SPAN RAID, RAID0+1.

Input parameters:

VINT type :

Specify which kind of disk raid to be created.

VINT disk_num :

Specify using how many disks to create the raid.

VINT disks[ ] :

The array of disk's internal index, and the valid range of this array is the second parameter: disk_num.

unsigned long param :

This is the extra parameter of specific raid type if needed. The following is the list of all the meanings according each raid type:

RAID0 (Stripe):	the upper 30 bits indicate the stripe size, in Kbytes. And the lowest bit of it indicate whether this new RAID need migration.
RAID1 (Mirror):	whether to keep the data of source disk
SPAN:	whether to keep the data of first disk
RAID0+1:	stripe size, in Kbytes.
RAID5:	the upper 30 bits indicate the stripe size, in Kbytes. And the bit0 of it indicate whether this new RAID need migration. Bit1 indicate whether this RAID5 will use the optimized algorithm.

VDWORD paramBlockSize :

This parameter will be used when creating RAID5, and it stands for the block size index. Its values will be from 0 to 15, and the block size will be 256k to 1536kbytes.

Return value:

VR_SUCCESS :

Create the disk raid successfully.

VR_ERR_NOT_INITED :

Raid lib hasn't been initialized.

VR_ERR_RAIDLIB_BUSY :

Some other functions are using the Raid lib and this call can not get the lock.

VR_ERR_RAIDTYPE_UNSUPPORT :

The parameter 'type' has a invalid value.

VR_ERR_DISK_NOT_ENOUGH :

The specified disks is not enough.

VR_ERR_INVALID_INDEX :

The input disk index is invalid.

VR_ERR_CREATE_DISK_USED :

One of the specified disks has already been in one existent disk raid.

VR_ERR_DISK_ERROR :

One of the specified disks has error status and can not be used.

VR_ERR_CREATE_DIFF_CTRLER :

All the specified disks are not in a single raid controller.

VR_ERR_CREATE_STRIPESIZE_INVALID :

When the raid type is RAID0, the input parameter stripesize is invalid.

VR_ERR_CREATE_DISKSIZE_INVALID :

When the raid type is RAID1 and the user want to keep the source disk's data, the mirror or spare disk's capacity is smaller than source disk's.

5.3.2 Create a stripe raid (RAID0)

Declaration:

VINT vr_create_stripe(VINT disk_num, VINT disks[ ], VDWORD stripe_size,
VBOOL need_migration);

Description:

This function can create a disk raid with type VR_RAID0.

Input parameters:

VINT disk_num :

Specify using how many disks to create the raid.

VINT disks[ ] :

The array of disk's internal index, and the valid range of this array is the first parameter: disk_num.

VDWORD stripe_size :

This is the stripe size of this raid, in Kbytes.

VBOOL need_migration :

Whether the new created RAID need migration.

Return value:

VR_SUCCESS :

Create the disk raid successfully.

VR_ERR_NOT_INITED :

Raid lib hasn't been initialized.

VR_ERR_RAIDLIB_BUSY :

Some other functions are using the Raid lib and this call can not get the lock.

VR_ERR_DISK_NOT_ENOUGH :

The specified disks is not enough.

VR_ERR_INVALID_INDEX :

The input disk index is invalid.

VR_ERR_CREATE_DISK_USED :

One of the specified disks has already been in one existent disk raid.

VR_ERR_DISK_ERROR :

One of the specified disks has error status and can not be used.

VR_ERR_CREATE_DIFF_CTRLER :

All the specified disks are not in a single raid controller.

VR_ERR_CREATE_STRIPESIZE_INVALID :

The input parameter stripe_size is invalid.

5.3.3 Create a mirror raid (RAID1)

Declaration:

```
VINT vr_create_mirror(VINT source, VINT mirror, VINT spare,
                      VBOOL keep_source);
```

Description:

This function can create a disk raid with type VR_RAID1.

Input parameters:

VINT source :

Specify the source disk's index.

VINT mirror :

Specify the source disk's index.

VINT spare :

Specify the source disk's index, if user will not add a spare disk, this parameter should be set to –1.

VBOOL keep_souce :

Whether to keep the data of source disk

Return value:

VR_SUCCESS :

Create the disk raid successfully.

VR_ERR_NOT_INITED :

Raid lib hasn't been initialized.

VR_ERR_RAIDLIB_BUSY :

Some other functions are using the Raid lib and this call can not get the lock.

VR_ERR_INVALID_INDEX :

The input disk index is invalid.

VR_ERR_CREATE_DISK_USED :

One of the specified disks has already been in one existent disk raid.

VR_ERR_DISK_ERROR :

One of the specified disks has error status and can not be used.

VR_ERR_CREATE_DIFF_CTRLER :

All the specified disks are not in a single raid controller.

VR_ERR_CREATE_DISKSIZE_INVALID :

When the raid type is RAID1 and the user want to keep the source disk's data, the mirror or spare disk's capacity is smaller than source disk's.

5.3.4 Create a span raid

Declaration:

VINT vr_create_span(VINT disk_num, VINT disks[ ], VBOOL keep_first);

Description:

This function can create a disk raid with type VR_SPAN.

Input parameters:

VINT disk_num :

Specify using how many disks to create the raid.

VINT disks[ ] :

The array of disk's internal index, and the valid range of this array is the first parameter: disk_num.

VBOOL keep_first :

Whether to keep the data of the first disk.

Return value:

VR_SUCCESS :

Create the disk raid successfully.

VR_ERR_NOT_INITED :

Raid lib hasn't been initialized.

VR_ERR_RAIDLIB_BUSY :

Some other functions are using the Raid lib and this call can not get the lock.

VR_ERR_DISK_NOT_ENOUGH :

The specified disks is not enough.

VR_ERR_INVALID_INDEX :

The input disk index is invalid.

VR_ERR_CREATE_DISK_USED :

One of the specified disks has already been in one existent disk raid.

VR_ERR_DISK_ERROR :

One of the specified disks has error status and can not be used.

VR_ERR_CREATE_DIFF_CTRLER :

All the specified disks are not in a single raid controller.

5.3.5 Create a RAID0+1 raid

Declaration:

VINT vr_create_raid01 (VINT disk_num, VINT disks[ ],
VDWORD stripe_size);

Description:

This function can create a disk raid with type VR_RAID0.

Input parameters:

VINT disk_num :

Specify using how many disks to create the raid. The number of disks should be bigger or equal to 4 and it should be a even number.

VINT disks[ ] :

The array of disk's internal index, and the valid range of this array is the first parameter: disk_num.

VDWORD stripe_size :

This is the stripe size of this raid, in Kbytes.

Return value:

VR_SUCCESS :

Create the disk raid successfully.

VR_ERR_NOT_INITED :

Raid lib hasn’t been initialized.

VR_ERR_RAIDLIB_BUSY :

Some other functions are using the Raid lib and this call can not get the lock.

VR_ERR_DISK_NOT_ENOUGH :

The specified disks is not enough, or the number is not a even number.

VR_ERR_INVALID_INDEX :

The input disk index is invalid.

VR_ERR_CREATE_DISK_USED :

One of the specified disks has already been in one existent disk raid.

VR_ERR_DISK_ERROR :

One of the specified disks has error status and can not be used.

VR_ERR_CREATE_DIFF_CTRLER :

All the specified disks are not in a single raid controller.

VR_ERR_CREATE_STRIPESIZE_INVALID :

When the raid type is RAID0, the input parameter stripesize is invalid.

5.3.6 Create a RAID5

Declaration:

```
VINT vr_create_raid5(VINT disk_num, VINT disks[], VDWORD stripe_size,
                     VDWORD block_size, VBOOL need_migration,
                     VBOOL bopt);
```

Description:

This function can create a disk raid with type VR_RAID0.

Input parameters:

VINT disk_num :

Specify using how many disks to create the raid.

VINT disks[ ] :

The array of disk’s internal index, and the valid range of this array is the first parameter: disk_num.

VDWORD stripe_size :

This is the stripe size of this raid, in Kbytes.

VDWORD block_size :

This is the block size index of this raid, from 0 to VR_MAX_BLOCK_INDEX.

VBOOL need_migration :

Whether the new created RAID need migration.

VBOOL bopt :

It indicate whether this RAID5 will use the optimized algorithm.

Return value:

VR_SUCCESS :

Create the disk raid successfully.

VR_ERR_NOT_INITED :

Raid lib hasn’t been initialized.

VR_ERR_RAIDLIB_BUSY :

Some other functions are using the Raid lib and this call can not get the lock.

VR_ERR_DISK_NOT_ENOUGH :

The specified disks is not enough.

VR_ERR_INVALID_INDEX :

The input disk index is invalid.

VR_ERR_CREATE_DISK_USED :

One of the specified disks has already been in one existent disk raid.

VR_ERR_DISK_ERROR :

One of the specified disks has error status and can not be used.

VR_ERR_CREATE_DIFF_CTRLER :

All the specified disks are not in a single raid controller.

VR_ERR_CREATE_STRIPESIZE_INVALID :

The input parameter stripe_size is invalid.

VR_ERR_CREATE_BLOCKSIZE_INVALID :

The input parameter block_size is invalid.

5.3.7 Add a spare disk to a mirror raid

Declaration:

VINT vr_mirror_add_spare (VINT array_index, VINT disk_index);

Description:

This function can add a spare disk to the specified mirror raid, if this mirror raid has no spare disk.

Input parameters:

VINT array_index :

Index to all disk arrays in system, specify a existent mirror raid. And this mirror raid should has no spare disk.

VINT disk_index :

Specify which disk to be added.

Return value:

VR_SUCCESS :

Add spare disk to the mirror raid successfully.

VR_ERR_NOT_INITED :

Raid lib hasn't been initialized.

VR_ERR_RAIDLIB_BUSY :

Some other functions are using the Raid lib and this call can not get the lock.

VR_ERR_INVALID_INDEX :

The input disk index is invalid or the array index is invalid.

VR_ERR_INVALID_PARAM :

The specified disk raid is not a mirror raid, or the mirror raid has the spare disk.

VR_ERR_RAID_BROKEN :

The specified mirror raid is a broken one.

VR_ERR_CREATE_DISK_USED :

The specified disk has already been in one existent disk raid.

VR_ERR_DISK_ERROR :

The specified disk has error status and can not be used.

VR_ERR_CREATE_DIFF_CTRLER :

The specified disk are not in the same raid controller as the mirror raid's.

VR_ERR_CREATE_DISKSIZE_INVALID :

The specified disk's capacity is not enough.

5.3.8 Remove the spare disk from a mirror raid

Declaration:

VINT vr_mirror_remove_spare (VINT array_index);

Description:

This function can remove the spare disk of the specified mirror raid if it has.

Input parameters:

VINT array_index :

Index to all disk arrays in system, specify a existent mirror raid. And this mirror raid should has a spare disk.

Return value:

VR_SUCCESS :

Remove spare disk from the mirror raid successfully.

VR_ERR_NOT_INITED :

Raid lib hasn't been initialized.

VR_ERR_RAIDLIB_BUSY :

Some other functions are using the Raid lib and this call can not get the lock.

VR_ERR_INVALID_INDEX :

The input array index is invalid.

VR_ERR_INVALID_PARAM :

The specified disk raid is not a mirror raid, or the mirror raid has no spare disk.

VR_ERR_RAID_BROKEN :

The specified mirror raid is a broken one.

5.3.9 Remove a existent raid

Declaration:

VINT vr_remove_raid (VINT array_index);

Description:

Remove a existent disk raid from system.

Input parameters:

VINT array_index :

Index to all disk arrays in system, specify which disk array.

Return value:

VR_SUCCESS :

Remove the specified disk raid successfully.

VR_ERR_NOT_INITED :

Raid lib hasn't been initialized.

VR_ERR_RAIDLIB_BUSY :

Some other functions are using the Raid lib and this call can not get the lock.

VR_ERR_INVALID_INDEX :

The input array index is invalid.

VR_ERR_INVALID_PARAM :

The specified array is a standalone disk.

5.4 Repair Broken Disk Raid

Not all the broken disk raids can be repaired. The following instances can not be repaired:

a. A broken RAID1 (mirror raid) that its source disk **AND** its mirror disk are all invalid.
b. A broken RAID0+1 that the two disks in one of its stripe disk pair (means a source stripe disk and its corresponding mirror stripe disk) are all invalid.
c. All broken RAID0 and SPAN raids can not be repaired.
d. Above 2 disks in RAID5 are invalid, this RAID5 cannot be repaired.

To repair a broken RAID1 can use its spare disk if it exist and be valid. There are 4 interfaces: one general interface and 3 specific interface to deal with the corresponding instance.

5.4.1 General interface to repair a broken disk raid

Declaration:

VINT vr_repair_broken_raid (VINT array_index, VINT disk_num,
VINT disks[ ]);

Description:

Repair a broken disk raid in the system. To repair a broken mirror raid with a valid spare disk, if the parameter disk_num is 0 the lib will use the spare disk to repair it, and if the parameter disk_num is 1 the lib will use the specified free disk.

Input parameters:

VINT array_index :

Index to all disk arrays in system, specify which disk array.

VINT disk_num :

Specify using how many disks to be added to the broken disk raid.

VINT disks[ ] :

The array of disk's internal index, and the valid range of this array is the second parameter: disk_num.

Return value:

VR_SUCCESS :

Repair the broken disk raid successfully.

VR_ERR_NOT_INITED :

Raid lib hasn't been initialized.

VR_ERR_RAIDLIB_BUSY :

Some other functions are using the Raid lib and this call can not get the lock.

VR_ERR_INVALID_INDEX :

The input array index is invalid, or input disk index is invalid.

VR_ERR_INVALID_PARAM :

The specified array is a standalone disk.

VR_ERR_REPAIR_NONEED_REPAIR :

The specified disk raid is not a broken one, or a broken mirror raid with its spare disk being invalid.

VR_ERR_REPAIR_CANNOT_REPAIR :

The specified broken raid can not be repaired.

VR_ERR_DISK_NOT_ENOUGH :

The specified disks is not enough to repair the broken raid.

VR_ERR_RAIDTYPE_UNSUPPORT :

The specified broken raid's type is not supported by now.

VR_ERR_CREATE_DISK_USED :

One of the specified disks has already been in one existent disk raid.

VR_ERR_DISK_ERROR :

One of the specified disks has error status and can not be used.

VR_ERR_CREATE_DIFF_CTRLER :

The specified disk are not in the same raid controller as the broken raid's.

VR_ERR_CREATE_DISKSIZE_INVALID :

The specified disk's capacity is not enough.

VR_ERR_UNEXPECTED_ERROR :

Some system error found when updating the internal disk array information.

5.4.2 Repair a broken mirror raid with spare disk

Declaration:

`VINT  vr_repair_mirror_with_spare (VINT array_index);`

Description:

Repair a broken RAID1(mirror raid) in the system using its spare disk. This broken mirror raid should has a valid spare disk.

Input parameters:

VINT array_index :

Index to all disk arrays in system, specify which disk array.

Return value:

VR_SUCCESS :

Repair the broken disk raid successfully.

VR_ERR_NOT_INITED :

Raid lib hasn't been initialized.

VR_ERR_RAIDLIB_BUSY :

Some other functions are using the Raid lib and this call can not get the lock.

VR_ERR_INVALID_INDEX :

The input array index is invalid.

VR_ERR_INVALID_PARAM :

The specified array is not a RAID1(mirror raid).

VR_ERR_REPAIR_NONEED_REPAIR :

The specified disk raid is not a broken one, or a broken mirror raid with its spare disk being invalid.

VR_ERR_REPAIR_CANNOT_REPAIR :

The specified broken raid can not be repaired.

VR_ERR_DISK_NOT_ENOUGH :

The spare disk of this broken mirror raid is invalid.

VR_ERR_UNEXPECTED_ERROR :

Some system error found when updating the internal disk array information.

5.4.3 Repair a broken mirror raid with a free disk

Declaration:

VINT vr_repair_mirror_with_new_disk (VINT array_index, VINT disk_index);

Description:

Repair a broken RAID1(mirror raid) in the system using a new free disk.

Input parameters:

VINT array_index :

Index to all disk arrays in system, specify which disk array.

VINT disk_index :

Specify which disk to be added to the broken mirror raid.

Return value:

VR_SUCCESS :

Repair the broken disk raid successfully.

VR_ERR_NOT_INITED :

Raid lib hasn't been initialized.

VR_ERR_RAIDLIB_BUSY :

Some other functions are using the Raid lib and this call can not get the lock.

VR_ERR_INVALID_INDEX :

The input array index is invalid, or input disk index is invalid.

VR_ERR_INVALID_PARAM :

The specified array is not a RAID1(mirror raid).

VR_ERR_REPAIR_NONEED_REPAIR :

The specified disk raid is not a broken one, or a broken mirror raid with its spare disk being invalid.

VR_ERR_REPAIR_CANNOT_REPAIR :

The specified broken raid can not be repaired.

VR_ERR_CREATE_DISK_USED :

The specified disk has already been in one existent disk raid.

VR_ERR_DISK_ERROR :

The specified disk has error status and can not be used.

VR_ERR_CREATE_DIFF_CTRLER :

The specified disk are not in the same raid controller as the broken raid's.

VR_ERR_CREATE_DISKSIZE_INVALID :

The specified disk's capacity is not enough.

VR_ERR_UNEXPECTED_ERROR :

Some system error found when updating the internal disk array information.

5.4.4 Repair a broken RAID0+1

Declaration:

VINT vr_repair_broken_raid01 (VINT array_index, VINT disk_num,
VINT disks[ ]);

Description:

Repair a broken RAID0+1.

Input parameters:

VINT array_index :
Index to all disk arrays in system, specify which disk array.

VINT disk_num :
Specify using how many disks to be added to the broken RAID0+1.

VINT disks[ ] :
The array of disk's internal index, and the valid range of this array is the second parameter: disk_num.

Return value:

VR_SUCCESS :
Repair the broken disk raid successfully.

VR_ERR_NOT_INITED :
Raid lib hasn't been initialized.

VR_ERR_RAIDLIB_BUSY :
Some other functions are using the Raid lib and this call can not get the lock.

VR_ERR_INVALID_INDEX :
The input array index is invalid, or input disk index is invalid.

VR_ERR_INVALID_PARAM :
The specified array is not a RAID0+1.

VR_ERR_REPAIR_NONEED_REPAIR :
The specified disk raid is not a broken one.

VR_ERR_REPAIR_CANNOT_REPAIR :
The specified broken raid can not be repaired.

VR_ERR_DISK_NOT_ENOUGH :
The specified disks is not enough to repair the broken raid.

VR_ERR_CREATE_DISK_USED :
One of the specified disks has already been in one existent disk raid.

VR_ERR_DISK_ERROR :
One of the specified disks has error status and can not be used.

VR_ERR_CREATE_DIFF_CTRLER :
The specified disk are not in the same raid controller as the broken raid's.

VR_ERR_CREATE_DISKSIZE_INVALID :
The specified disk's capacity is not enough.

VR_ERR_UNEXPECTED_ERROR :
Some system error found when updating the internal disk array information.

5.4.5 Repair a broken RAID5 with a free disk

Declaration:

VINT vr_repair_raid5_with_new_disk (VINT array_index, VINT disk_index);

Description:

Repair a broken RAID5 in the system using a new free disk.

Input parameters:

VINT array_index :
Index to all disk arrays in system, specify which disk array.

VINT disk_index :
Specify which disk to be added to the broken mirror raid.

Return value:

VR_SUCCESS :

Repair the broken disk raid successfully.

VR_ERR_NOT_INITED :

Raid lib hasn't been initialized.

VR_ERR_RAIDLIB_BUSY :

Some other functions are using the Raid lib and this call can not get the lock.

VR_ERR_INVALID_INDEX :

The input array index is invalid, or input disk index is invalid.

VR_ERR_INVALID_PARAM :

The specified array is not a RAID1(mirror raid).

VR_ERR_REPAIR_NONEED_REPAIR :

The specified disk raid is not a broken one, or a broken mirror raid with its spare disk being invalid.

VR_ERR_REPAIR_CANNOT_REPAIR :

The specified broken raid can not be repaired.

VR_ERR_CREATE_DISK_USED :

The specified disk has already been in one existent disk raid.

VR_ERR_DISK_ERROR :

The specified disk has error status and can not be used.

VR_ERR_CREATE_DIFF_CTRLER :

The specified disk are not in the same raid controller as the broken raid's.

VR_ERR_CREATE_DISKSIZE_INVALID :

The specified disk's capacity is not enough.

VR_ERR_UNEXPECTED_ERROR :

Some system error found when updating the internal disk array information.

5.5 Sync and Verify Mirror Raid

A newly created RAID1(mirror raid) or RAID01 probably need to be synchronized. And the verify action can check whether one RAID1 or RAID01 need to do sync. In here, some functions work at a disk mirror pair, which means the source disk and the mirror disk of a RAID1, or a source stripe disk and its corresponding mirror stripe disk.

5.5.1 Synchronize a disk mirror pair

Declaration:

```
VINT  vr_sync_disk_pair   (VINT source_disk, VINT mirror_disk,
                           VDWORD64 start_sector,VWORD sector_num);
```

Description:

Synchronize the specified part of one disk mirror pair which includes the specified source disk and mirror disk.

Input parameters:

VINT source_disk :

Index of the source disk of the specified disk mirror pair.

VINT source_disk :

Index of the mirror disk of the specified disk mirror pair.

VDWORD64 start_sector :

Specify from which sector of the disks to start the sync action, this is a 64-bit width integer.

VWORD sector_num :

Specify how many sectors of the disks to do the sync action, this is a WORD value and should not be larger than 256.

Return value:

VR_SUCCESS :

Sync the disk mirror pair successfully.

VR_ERR_NOT_INITED :

Raid lib hasn't been initialized.

VR_ERR_INVALID_INDEX :

The input disk index is invalid.

VR_ERR_SYNCVERIFY_DISK_INVALID :

The input source disk and mirror disk is not a disk mirror pair, or the source disk and the mirror disk are reversed.

VR_ERR_DISK_ERROR :

One of the specified disks has error status and can not be used.

VR_ERR_DRIVER_NEED_REBOOT :

The system should reboot or the raid driver should reload to update the system information, before do this action.

VR_ERR_SYNCVERIFY_NUM_INVALID :

The parameter sector_num is larger than 256.

VR_ERR_SYNCVERIFY_OFFSET_INVALID :

The sector range specified by the parameters are beyond the capacity of disks.

VR_ERR_SYNCVERIFY_FAILED :

The raid driver fails when doing this action.

5.5.2 Verify a disk mirror pair

Declaration:

VINT vr_verify_disk_pair (VINT source_disk, VINT mirror_disk,
DWORD64 start_sector, VWORD sector_num);

Description:

Verify the specified part of one disk mirror pair which includes the specified source disk and mirror disk, if found the data different, it will mark the disk raid as needing to sync.

Input parameters:

VINT source_disk :

Index of the source disk of the specified disk mirror pair.

VINT source_disk :

Index of the mirror disk of the specified disk mirror pair.

VDWORD64 start_sector :

Specify from which sector of the disks to start the verify action, this is a 64-bit width integer.

VWORD sector_num :

Specify how many sectors of the disks to do the verify action, this is a WORD value and should not be larger than 256.

Return value:

VR_SUCCESS :

Verify the disk mirror pair successfully.

VR_ERR_NOT_INITED :

Raid lib hasn't been initialized.

VR_ERR_INVALID_INDEX :

The input disk index is invalid.

VR_ERR_SYNCVERIFY_DISK_INVALID :

The input source disk and mirror disk is not a disk mirror pair, or the source disk and the mirror disk are reversed.

VR_ERR_DISK_ERROR :

One of the specified disks has error status and can not be used.

VR_ERR_DRIVER_NEED_REBOOT :

The system should reboot or the raid driver should reload to update the system information, before do this action.

VR_ERR_SYNCVERIFY_NUM_INVALID :

The parameter sector_num is larger than 256.

VR_ERR_SYNCVERIFY_OFFSET_INVALID :

The sector range specified by the parameters are beyond the capacity of disks.

VR_ERR_SYNCVERIFY_FAILED :

The raid driver fails when doing this action.

VR_VERIFY_MISMATCH :

The data of the source disk and the mirror disk are different, and the lib will mark the corresponding disk raid as needing to sync.

5.5.3 Mark a disk mirror pair to be synchronized

Declaration:

VINT vr_mark_disk_pair_as_synced (VINT source_disk, VINT mirror_disk);

Description:

If the data of the source disk and the mirror disk are same though they havn't been synchronized, this function can just mark them as having been synchronized.

Input parameters:

VINT source_disk :

Index of the source disk of the specified disk mirror pair.

VINT source_disk :

Index of the mirror disk of the specified disk mirror pair.

Return value:

VR_SUCCESS :

Mark the disk mirror pair successfully.

VR_ERR_NOT_INITED :

Raid lib hasn't been initialized.

VR_ERR_INVALID_INDEX :

The input disk index is invalid.

VR_ERR_SYNCVERIFY_DISK_INVALID :

The input source disk and mirror disk is not a disk mirror pair, or the source disk and the mirror disk are reversed.

5.5.4 Mark a disk raid to be synchronized

Declaration:

VINT vr_mark_raid_syned (VINT array_index);

Description:

If the data of all disk mirror pairs of one disk raid are same though they haven't been synchronized, this function can just mark the disk raid as having been synchronized.

Input parameters:

VINT array_index :

Index to all disk arrays in system, specify which disk array.

Return value:

VR_SUCCESS :

Mark the disk raid successfully.

VR_ERR_NOT_INITED :

Raid lib hasn't been initialized.

VR_ERR_INVALID_INDEX :

The input disk array index is invalid.

VR_ERR_SYNCVERIFY_ERROR_RAIDTYPE :

The specified disk array is not RAID1 and RAID01.

VR_ERR_SYNCVERIFY_DISK_INVALID

One of the disks in the disk raid is invalid.

5.6 Migration of RAID

If a new created RAID0/RAID5/RAID01 want to keep the first disk's data, the migration process can move the data of first data to the entire RAID and during the process the RAID can still be accessed and the origin data of first data can still read and write. The supported function do one step process of migration every time when being called, and user should call this function continuously until the function return the finishing status.

5.6.1 Do one step of migration process of RAID

Declaration:

```
VINT  vr_migration_raid (VINT array_index, VDWORD64 *pcur_position,
                                  VWORD *ppercent);
```

Description:

Do one step process of the migration of the specified RAID. After every calling of this function, user can get the current position (in sectors) and current percent of all the migration process.

Input parameters:

VINT array_index :

Index to all disk arrays in system, specify which disk array.

VDWORD64 *pcur_position :

Pointer to one VDWORD64 variable to save current position (in sectors) of the migration process after this calling.

VWORD *ppercent :

Pointer to one VWORD variable to save current percent of the migration process after this calling.

Return value:

VR_SUCCESS :

Current step of migration process finished successfully.

VR_ERR_NOT_INITED :

Raid lib hasn't been initialized.

VR_ERR_INVALID_INDEX :

The input disk array index is not a valid RAID0.

VR_ERR_RAIDTYPE_UNSUPPORT :

The input disk array index is not a valid RAID0, RAID5, RAID01.

VR_ERR_MIGRATION_NONEED :

The specified RAID need not migration.

VR_ERR_RAID_BROKEN :

The specified RAID is a broken one.

VR_ERR_DRIVER_NEED_REBOOT :

The system should reboot or the raid driver should reload to update the system information, before do this action.

VR_ERR_MIGRATION_FAILED :

Some error found when the driver do the migration action.

VR_ERR_MIGRATION_LOG_FAILED :

Some error found when the lib write the migration log information to devices.

VR_MIGRATION_FINISH :

The entire migration process of current RAID has finished.

5.7 Other Operations of RAID5

If a new RAID5 disk array was created, the disks in it can be initialized to a parity valid status. When a broken RAID5 was repaired by adding a new free disk, this new added disk should be synchronized with old data in other disks. And RAID5 can be verified to check whether the parity values are valid.

5.7.1 Do one step of initialization process of RAID5

Declaration:

VINT vr_init_raid5 (VINT array_index, VDWORD64 *pcur_position,
VWORD *ppercent);

Description:

Do one step process of the initialization of the specified RAID5. After every calling of this function, user can get the current position (in sectors) and current percent of all the migration process.

Input parameters:

VINT array_index :

Index to all disk arrays in system, specify which disk array.

VDWORD64 *pcur_position :

Pointer to one VDWORD64 variable to save current position (in sectors) of the migration process after this calling

VWORD *ppercent :

Pointer to one VWORD variable to save current percent of the migration process after this calling

Return value:

VR_SUCCESS :

Current step of initialization process finished successfully.

VR_ERR_NOT_INITED :

Raid lib hasn't been initialized.

VR_ERR_INVALID_INDEX :

The input disk array index is not a valid RAID5.

VR_ERR_RAIDTYPE_UNSUPPORT :

The input disk array index is not a valid RAID5.

VR_ERR_RAID5INIT_NONEED :

The specified RAID need not init.

VR_ERR_RAID_BROKEN :

The specified RAID5 is a broken one.

VR_ERR_DRIVER_NEED_REBOOT :

The system should reboot or the raid driver should reload to update the system information, before do this action.

VR_ERR_ RAID5INIT _FAILED :

Some error found when the driver doing the operation.

VR_ERR_ RAID5INIT _LOG_FAILED :

Some error found when the lib writing the log information to devices.

VR_ RAID5INIT _FINISH :

The entire init process of current RAID5 has finished.

5.7.2 Do one step of synchronization process of RAID5

Declaration:

VINT vr_sync_raid5 (VINT array _index, VDWORD64 *pcur_position,
VWORD *ppercent);

Description:

Do one step process of the synchronization of the specified RAID5. After every calling of this function, user can get the current position (in sectors) and current percent of all the migration process.

Input parameters:

VINT array_index :

Index to all disk arrays in system, specify which disk array.

VDWORD64 *pcur_position :

Pointer to one VDWORD64 variable to fetch current position (in sectors) of the sync process after this calling

VWORD *ppercent :

Pointer to one VWORD variable to fetch current percent of the migration process after this calling

Return value:

VR_SUCCESS :

Current step of sync process finished successfully.

VR_ERR_NOT_INITED :

Raid lib hasn't been initialized.

VR_ERR_INVALID_INDEX :

The input disk array index is not a valid RAID5.

VR_ERR_RAIDTYPE_UNSUPPORT :

The input disk array index is not a valid RAID5.

VR_ERR_RAID5SYNC_NONEED :

The specified RAID need not sync.

VR_ERR_RAID_BROKEN :

The specified RAID5 is a broken one.

VR_ERR_DRIVER_NEED_REBOOT :

The system should reboot or the raid driver should reload to update the system information, before do this action.

VR_ERR_ RAID5SYNC _FAILED :

Some error found when the driver doing the operation.

VR_ERR_ RAID5SYNC _LOG_FAILED :

Some error found when the lib writing the log information to devices.

VR_ RAID5SYNC _FINISH :

The entire sync process of current RAID5 has finished.

5.7.3 Do one step of verify process of RAID5

Declaration:

```
VINT vr_verify_raid5 (VINT array_index, VDWORD64 *pcur_position,
                                    VWORD *ppercent);
```

Description:

Do one step process of the verify of the specified RAID5. After every calling of this function, user can get the current position (in sectors) and current percent of all the migration process.

Input parameters:

VINT array_index :

Index to all disk arrays in system, specify which disk array.

VDWORD64 *pcur_position :

Pointer to one VDWORD64 variable to save current position (in sectors) of the migration process after this calling

VWORD *ppercent :

Pointer to one VWORD variable to save current percent of the migration process after this calling

Return value:

VR_SUCCESS :

Current step of migration process finished successfully.

VR_ERR_NOT_INITED :

Raid lib hasn't been initialized.

VR_ERR_INVALID_INDEX :

The input disk array index is not a valid RAID5.

VR_ERR_RAIDTYPE_UNSUPPORT :

The input disk array index is not a valid RAID5.

VR_ERR_RAID_BROKEN :

The specified RAID5 is a broken one.

VR_ERR_DRIVER_NEED_REBOOT :

The system should reboot or the raid driver should reload to update the system information, before do this action.

VR_ERR_SYNCVERIFY_FAILED :

Some error found when the driver doing the operation.

VR_RAID5SYNC_FINISH :

The entire verify process of current RAID5 has finished.

5.8 Update Internal Hardware and Array Information

Because the devices attached to the raid controllers can be hot-plugged, the hardware information and disk array information will probably change. Applications using this raid lib should keep a timer callback which to call vr_update_hardware_info to check the changing of hardware, if found the hardware's status had changed it should re-fetch all the information from raid lib.

The function vr_exec_SMART will be used to check devices' status by sending SMART command to them, and it should not be used in application's timer functions.

5.8.1 Function to check the hardware's change

Declaration:

VINT vr_update_hardware_info (void);

Description:

Check the hardware status of all the devices attached to the raid controllers. If some devices' status has changed, .the application should re-fetch all the information and update their display. When the return value is VR_ERR_RAIDLIB_BUSY, application should not stop, and the next timer callback function can call it again.

Input parameters:

None

Return value:

VR_SUCCESS :

No device has changed.

VR_ERR_NOT_INITED :

Raid lib hasn't been initialized.

VR_ERR_RAIDLIB_BUSY :

Some other functions are using the Raid lib and this call can not get the lock.

VR_UPDATE_INFO_CHANGED :

Some devices has changed.

5.8.2 Send SMART command to every hard disk

Declaration:

VINT vr_exec_SMART (void);

Description:

Send SMART command to every hard-disk in system. Check whether the hard-disks have degrading status.

Input parameters:

None

Return value:

VR_SUCCESS :

No device has degrading status.

VR_ERR_NOT_INITED :

Raid lib hasn’t been initialized.

VR_ERR_RAIDLIB_BUSY :

Some other functions are using the Raid lib and this call can not get the lock.

VR_SMART_DISK_DEGRADING :

Some devices have degrading status.

6. Important Notes

6.1 When the internal data will change

Because some functions will change the internal data of disk array information, after calling these functions the RAID application that using the RAID lib should re-fetch the information of hardware and disk arrays, and update the program’s display. These functions are:

- VINT vr_create_raid (VINT type, VINT disk_num, VINT disks[], unsigned long param);
- VINT vr_create_mirror (VINT source, VINT mirror, VINT spare, VINT keep_source);
- VINT vr_create_stripe (VINT disk_num, VINT disks[], unsigned long stripe_size, VBOOL need_migration);
- VINT vr_create_span (VINT disk_num, VINT disks[], VINT keep_first);
- VINT vr_create_raid01 (VINT disk_num, VINT disks[], unsigned long stripe_size);
- VINT vr_create_raid5(VINT disk_num, VINT disks[], VDWORD stripe_size, VDWORD block_size, VBOOL need_migration, VBOOL bopt);
- VINT vr_mirror_add_spare (VINT array_index, VINT disk_index);
- VINT vr_mirror_remove_spare (VINT array_index);
- VINT vr_remove_raid (VINT array_index);
- VINT vr_mark_disk_pair_as_synced (VINT source_disk, VINT mirror_disk);
- VINT vr_mark_raid_syned (VINT array_index);
- VINT vr_repair_broken_raid (VINT array_index, VINT disk_num, VINT disks[]);
- VINT vr_repair_mirror_with_spare (VINT array_index);
- VINT vr_repair_mirror_with_new_disk (VINT array_index, VINT disk_index);
- VINT vr_repair_broken_raid01 (VINT array_index, VINT disk_num, VINT disks[]);
- VINT vr_sync_disk_pair (VINT source_disk, VINT mirror_disk, VDWORD64 start_sector,VWORD sector_num);

  Note: Only after the last part of sync process, the application need update the information.
- VINT vr_verify_disk_pair (VINT source_disk, VINT mirror_disk, VDWORD64 start_sector,VWORD sector_num);

  Note: Only when the return value is VR_VERIFY_MISMATCH, the application need update the information.
- VINT vr_migration_raid (VINT array_index, VDWORD64 *pcur_position, VWORD *ppercent);

  Note: Only after the entire of migration process has finished, the application need update the information.
- VINT vr_update_hardware_info (void);

  Note: Only when the return value is VR_UPDATE_INFO_CHANGED, the application need update the information.
- VINT vr_exec_SMART (void);

  Note: Only when the return value is VR_SMART_DISK_DEGRADING, the application need update the information.

6.2 Device's capacity and real capacity

In the data structure vr_device_info_t, the field **capacityLow** and **capacityHigh** compose a 64bit integer to record the capacity in sectors of this device. And the field **realCapacityLow** and **realCapacityHigh** compose another 64bit capacity value which means the actual used capacity in the disk raid. Of course, only when the device is in a valid disk raid, the later value is valid.

6.3 Stripe size and stripe size index

The stripe size of RAID0 and RAID0+1 has 5 values: 4Kbytes, 8Kbytes, 16Kbytes, 32Kbytes, 64Kbytes. In the raid lib, stripe size index's values are defined as: 0, 1, 2, 3, 4. User can convert the stripe size index to stripe size (in Kbytes) using macro GET_STRIPE_SIZE. This is the definition of the macro:

```
#define GET_STRIPE_SIZE( I ) ((1 << ( I )) * 4)
```

6.4 The spare disk of RAID1 (mirror raid)

In the raid lib, the RAID1 or mirror raid can be appended one and only one spare disk. APIs support the adding and removing of it.

6.5 Controller limit of RAID

In the raid lib, it is ruled that all the disks in a disk raid should be in one raid controller. So when to create disk raid or append a new disk to disk raid, user should assure that all the disks in the parameters are in one single raid controller.

6.6 The index values in the data structures

The field **index** of data structure vr_device_info_t means the index of all the devices in current system.

The field **index** of data structure vr_array_info_t means the index of all disk arrays in current system, including all the disk arrays with standalone disk type.

The field **raid_index** of vr_array_info_t means the index of all the disk raids in system, not including the standalone disks.

6.7 System reboot problem

When the system and RAID lib can not support dynamic RAID, or when the system disk has been changed, the application need reboot the system (or reload the driver in Linux) to update system device configuration. When the RAID status was changed (e.g. create, remove or change RAID, repair a broken RAID), application should check whether the system need reboot, and ask user whether to reboot the system at once. If the user don't want reboot the system at once, the application can also continue to work, but user can not use the changed device in system.

6.8 Usage of the raid lib

Windows Release:

The released package of this RAID library will contain a header file: raid_lib.h, an import library file: raid_lib.lib, and multi DLL files for every different OS. User should link the raid_lib.lib file when compiling the application, and place the correct DLL file according the current OS in the same directory of the executable file or system folders. By now, the RAID library can support Windows 98, Windows 2000, Windows XP and Windows Server2003.

Linux Release:

The released package will contain the following files:

raid_lib.h	Header file
libVIAraid.a	Static library

Notes:

To use RAID library under Linux, the application must support the following function entry:

```
int MessageBox (void * parent, const char * msg, const char * title, int flag);
```