Red Hat Enterprise Linux 8.0 Beta

Configuring and managing storage hardware

Deploying and configuring single-node storage in Red Hat Enterprise Linux 8

Last Updated: 2019-04-05

Red Hat Enterprise Linux 8.0 Beta Configuring and managing storage hardware

Deploying and configuring single-node storage in Red Hat Enterprise Linux 8

Legal Notice

Abstract

This documentation collection provides instructions on how to effectively manage storage devices in Red Hat Enterprise Linux 8.

Table of Contents

THIS IS A BETA VERSION!

Thank you for your interest in Red Hat Enterprise Linux 8.0 Beta. Be aware that:

- Beta code should not be used with production data or on production systems.
- Beta does not include a guarantee of support.
- Feedback and bug reports are welcome. Discussions with your account representative, partner contact, and Technical Account Manager (TAM) are also welcome.
- Upgrades to or from a Beta are not supported or recommended.

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Please let us know how we could make it better. To do so:

- For simple comments on specific passages, make sure you are viewing the documentation in the Multi-page HTML format. Highlight the part of text that you want to comment on. Then, click the **Add Feedback** pop-up that appears below the highlighted text, and follow the displayed instructions.
- For submitting more complex feedback, create a Bugzilla ticket:
  1. Go to the Bugzilla website.
  2. As the Component, use **Documentation**.
  3. Fill in the **Description** field with your suggestion for improvement. Include a link to the relevant part(s) of documentation.
  4. Click **Submit Bug**.

CHAPTER 1. GETTING STARTED WITH PARTITIONS

As a system administrator, you can use the following procedures to create, delete, and modify various types of disk partitions.

For an overview of the advantages and disadvantages to using partitions on block devices, see the following KBase article: https://access.redhat.com/solutions/163853.

1.1. VIEWING THE PARTITION TABLE

As a system administrator, you can display the partition table of a block device to see the partition layout and details about individual partitions.

1.1.1. Viewing the partition table with parted

This procedure describes how to view the partition table on a block device using the **parted** utility.

Procedure

1. Start the interactive **parted** shell:

```
# parted block-device
```

   - Replace *block-device* with the path to the device you want to examine: for example, **/dev/sda**.

2. View the partition table:

```
(parted) print
```

3. Optionally, use the following command to switch to another device you want to examine next:

```
(parted) select block-device
```

Additional resources

- The **parted(8)** man page.

1.1.2. Example output of `parted print`

This section provides an example output of the **print** command in the **parted** shell and describes fields in the output.

Example 1.1. Output of the `print` command

```
Model: ATA SAMSUNG MZNLN256 (scsi)
Disk /dev/sda: 256GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start   End     Size    Type      File system  Flags
 1      1049kB  269MB   268MB   primary   xfs          boot
```

```
2      269MB   34.6GB  34.4GB  primary
3      34.6GB  45.4GB  10.7GB  primary
4      45.4GB  256GB   211GB   extended
5      45.4GB  256GB   211GB   logical
```

Following is a description of the fields:

`Model: ATA SAMSUNG MZNLN256 (scsi)`

The disk type, manufacturer, model number, and interface.

`Disk /dev/sda: 256GB`

The disk label type.

`Number`

The partition number. For example, the partition with minor number 1 corresponds to **`/dev/sda1`**.

`Start and End`

The location on the device where the partition starts and ends.

`Type`

Valid types are metadata, free, primary, extended, or logical.

`File system`

The file system type. If the **`File system`** field of a device shows no value, this means that its file system type is unknown. The **`parted`** utility cannot recognize the file system on encrypted devices.

`Flags`

Lists the flags set for the partition. Available flags are **`boot`**, **`root`**, **`swap`**, **`hidden`**, **`raid`**, **`lvm`**, or **`lba`**.

1.2. CREATING A PARTITION TABLE ON A DISK

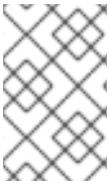
As a system administrator, you can format a block device with different types of partition tables to enable using partitions on the device.

WARNING

Formatting a block device with a partition table deletes all data stored on the device.

1.2.1. Considerations before modifying partitions on a disk

This section lists key points to consider before creating, removing, or resizing partitions.



NOTE

This section does not cover the DASD partition table, which is specific to the IBM System z architecture. For information on DASD, see the What you should know about DASD article at the IBM Knowledge Center.

The maximum number of partitions
The number of partitions on a device is limited by the type of the partition table:

- On a device formatted with the **Master Boot Record (MBR)** partition table, you can have either:
  - Up to four primary partitions, or
  - Up to three primary partitions, one extended partition, and multiple logical partitions within the extended.
- On a device formatted with the **GUID Partition Table (GPT)**, the maximum number of partitions is 128. While the GPT specification allows for more partitions by growing the area reserved for the partition table, common practice used by the **`parted`** utility is to limit it to enough area for 128 partitions.

The maximum size of a partition
The size of a partition on a device is limited by the type of the partition table:

- On a device formatted with the **Master Boot Record (MBR)** partition table, the maximum size is 2TiB.
- On a device formatted with the **GUID Partition Table (GPT)**, the maximum size is 8ZiB.

If you want to create a partition larger than 2TiB, the disk must be formatted with GPT.

Size alignment
The **`parted`** utility enables you to specify partition size using multiple different suffixes:

MiB, GiB, or TiB

Size expressed in powers of 2.

- The starting point of the partition is aligned to the exact sector specified by size.
- The ending point is aligned to the specified size minus 1 sector.

MB, GB, or TB

Size expressed in powers of 10.
The starting and ending point is aligned within one half of the specified unit: for example, ±500KB when using the MB suffix.

1.2.2. Comparison of partition table types

This section compares the properties of different types of partition tables that you can create on a block device.

Table 1.1. Partition table types

Partition table	Maximum number of partitions	Maximum partition size
Master Boot Record (MBR)	4 primary, or 3 primary and 12 logical inside an extended partition	2TiB

Partition table	Maximum number of partitions	Maximum partition size
GUID Partition Table (GPT)	128	8ZiB

1.2.3. Creating a partition table on a disk with parted

This procedure describes how to format a block device with a partition table using the **parted** utility.

Procedure

1. Start the interactive **parted** shell:

```
# parted block-device
```

   - Replace *block-device* with the path to the device where you want to create a partition table: for example, **/dev/sda**.

2. Determine if there already is a partition table on the device:

```
(parted) print
```

   If the device already contains partitions, they will be deleted in the next steps.

3. Create the new partition table:

```
(parted) mklabel table-type
```

   - Replace *table-type* with with the intended partition table type:
     - **msdos** for MBR
     - **gpt** for GPT

   Example 1.2. Creating a GPT table

   For example, to create a GPT table on the disk, use:

```
(parted) mklabel gpt
```

   The changes start taking place as soon as you enter this command, so review it before executing it.

4. View the partition table to confirm that the partition table exists:

```
(parted) print
```

5. Exit the **parted** shell:

```
(parted) quit
```

Additional resources

- The **parted(8)** man page.

Next steps

- Create partitions on the device. See Section 1.3, “Creating a partition” for details.

1.3. CREATING A PARTITION

As a system administrator, you can create new partitions on a disk.

1.3.1. Considerations before modifying partitions on a disk

This section lists key points to consider before creating, removing, or resizing partitions.

NOTE

This section does not cover the DASD partition table, which is specific to the IBM System z architecture. For information on DASD, see the What you should know about DASD article at the IBM Knowledge Center.

The maximum number of partitions
The number of partitions on a device is limited by the type of the partition table:

- On a device formatted with the **Master Boot Record (MBR)** partition table, you can have either:
  - Up to four primary partitions, or
  - Up to three primary partitions, one extended partition, and multiple logical partitions within the extended.
- On a device formatted with the **GUID Partition Table (GPT)**, the maximum number of partitions is 128. While the GPT specification allows for more partitions by growing the area reserved for the partition table, common practice used by the **parted** utility is to limit it to enough area for 128 partitions.

The maximum size of a partition
The size of a partition on a device is limited by the type of the partition table:

- On a device formatted with the **Master Boot Record (MBR)** partition table, the maximum size is 2TiB.
- On a device formatted with the **GUID Partition Table (GPT)**, the maximum size is 8ZiB.

If you want to create a partition larger than 2TiB, the disk must be formatted with GPT.

Size alignment
The **parted** utility enables you to specify partition size using multiple different suffixes:

MiB, GiB, or TiB

Size expressed in powers of 2.

- The starting point of the partition is aligned to the exact sector specified by size.
- The ending point is aligned to the specified size minus 1 sector.

MB, GB, or TB

Size expressed in powers of 10.
The starting and ending point is aligned within one half of the specified unit: for example, ±500KB when using the MB suffix.

1.3.2. Partition types

This section describes different attributes that specify the type of a partition.

Partition types or flags
The partition type, or flag, is used by a running system only rarely. However, the partition type matters to on-the-fly generators, such as **`systemd-gpt-auto-generator`**, which use the partition type to, for example, automatically identify and mount devices.

- The **`parted`** utility provides some control of partition types by mapping the partition type to *flags*. The parted utility can handle only certain partition types: for example LVM, swap, or RAID.
- The **`fdisk`** utility supports the full range of partition types by specifying hexadecimal codes.

Partition file system type
The **`parted`** utility optionally accepts a file system type argument when creating a partition. The value is used to:

- Set the partition flags on MBR, or
- Set the partition UUID type on GPT. For example, the **`swap`**, **`fat`**, or **`hfs`** file system types set different GUIDs. The default value is the Linux Data GUID.

The argument does not modify the file system on the partition in any way. It only differentiates between the supported flags or GUIDs.

The following file system types are supported:

- **`xfs`**
- **`ext2`**
- **`ext3`**
- **`ext4`**
- **`fat16`**
- **`fat32`**
- **`hfs`**
- **`hfs+`**
- **`linux-swap`**
- **`ntfs`**
- **`reiserfs`**

1.3.3. Creating a partition with parted

This procedure describes how to create a new partition on a block device using the **parted** utility.

Prerequisites

- There is a partition table on the disk. For details on how to format the disk, see Section 1.2, “Creating a partition table on a disk”.
- If the partition you want to create is larger than 2TiB, the disk must be formatted with the GUID Partition Table (GPT).

Procedure

1. Start the interactive **parted** shell:

```
# parted block-device
```

   - Replace *block-device* with the path to the device where you want to create a partition: for example, **/dev/sda**.

2. View the current partition table to determine if there is enough free space:

```
(parted) print
```

   - If there is not enough free space, you can resize an existing partition. For more information, see Section 1.5, “Resizing a partition”.
   - From the partition table, determine:
     - The start and end points of the new partition
     - On MBR, what partition type it should be.

3. Create the new partition:

```
(parted) mkpart part-type name fs-type start end
```

   - Replace *part-type* with with **primary**, **logical**, or **extended** based on what you decided from the partition table. This applies only to the MBR partition table.
   - Replace *name* with an arbitrary partition name. This is required for GPT partition tables.
   - Replace *fs-type* with any one of **xfs**, **ext2**, **ext3**, **ext4**, **fat16**, **fat32**, **hfs**, **hfs+**, **linux-swap**, **ntfs**, or **reiserfs**. The *fs-type* parameter is optional. Note that **parted** does not create the file system on the partition.
   - Replace *start* and *end* with the sizes that determine the starting and ending points of the partition, counting from the beginning of the disk. You can use size suffixes, such as **512MiB**, **20GiB**, or **1.5TiB**. The default size megabytes.

Example 1.3. Creating a small primary partition

For example, to create a primary partition from 1024MiB until 2048MiB on an MBR table, use:

```
(parted) mkpart primary 1024MiB 2048MiB
```

```
```

The changes start taking place as soon as you enter this command, so review it before executing it.

4. View the partition table to confirm that the created partition is in the partition table with the correct partition type, file system type, and size:

```
(parted) print
```

5. Exit the **parted** shell:

```
(parted) quit
```

6. Use the following command to wait for the system to register the new device node:

```
# udevadm settle
```

7. Verify that the kernel recognizes the new partition:

```
# cat /proc/partitions
```

Additional resources

- The **parted(8)** man page.

1.3.4. Setting a partition type with fdisk

This procedure describes how to set a partition type, or flag, using the **fdisk** utility.

Prerequisites

- There is a partition on the disk.

Procedure

1. Start the interactive **fdisk** shell:

```
# fdisk block-device
```

   - Replace *block-device* with the path to the device where you want to set a partition type: for example, **/dev/sda**.

2. View the current partition table to determine the minor partition number:

```
Command (m for help): print
```

   You can see the current partition type in the **Type** column and its corresponding type ID in the **Id** column.

3. Enter the partition type command and select a partition using its minor number:

```
Command (m for help): type
Partition number (1,2,3 default 3): 2
```

4. Optionally, list the available hexadecimal codes:

```
Hex code (type L to list all codes): L
```

5. Set the partition type:

```
Hex code (type L to list all codes): 8e
```

6. Write your changes and exit the **fdisk** shell:

```
Command (m for help): write
The partition table has been altered.
Syncing disks.
```

7. Verify your changes:

```
# fdisk --list block-device
```

1.4. REMOVING A PARTITION

As a system administrator, you can remove a disk partition that is no longer used to free up disk space.

WARNING

Removing a partition deletes all data stored on the partition.

1.4.1. Considerations before modifying partitions on a disk

This section lists key points to consider before creating, removing, or resizing partitions.

NOTE

This section does not cover the DASD partition table, which is specific to the IBM System z architecture. For information on DASD, see the What you should know about DASD article at the IBM Knowledge Center.

The maximum number of partitions
The number of partitions on a device is limited by the type of the partition table:

- On a device formatted with the **Master Boot Record (MBR)** partition table, you can have either:
  - Up to four primary partitions, or
  - Up to three primary partitions, one extended partition, and multiple logical partitions within the extended.
- On a device formatted with the **GUID Partition Table (GPT)**, the maximum number of partitions

is 128. While the GPT specification allows for more partitions by growing the area reserved for the partition table, common practice used by the **parted** utility is to limit it to enough area for 128 partitions.

The maximum size of a partition
The size of a partition on a device is limited by the type of the partition table:

- On a device formatted with the **Master Boot Record (MBR)** partition table, the maximum size is 2TiB.
- On a device formatted with the **GUID Partition Table (GPT)**, the maximum size is 8ZiB.

If you want to create a partition larger than 2TiB, the disk must be formatted with GPT.

Size alignment
The **parted** utility enables you to specify partition size using multiple different suffixes:

MiB, GiB, or TiB

Size expressed in powers of 2.

- The starting point of the partition is aligned to the exact sector specified by size.
- The ending point is aligned to the specified size minus 1 sector.

MB, GB, or TB

Size expressed in powers of 10.
The starting and ending point is aligned within one half of the specified unit: for example, ±500KB when using the MB suffix.

1.4.2. Removing a partition with parted

This procedure describes how to remove a disk partition using the **parted** utility.

Procedure

1. Start the interactive **parted** shell:

```
# parted block-device
```

   - Replace *block-device* with the path to the device where you want to remove a partition: for example, **/dev/sda**.

2. View the current partition table to determine the minor number of the partition to remove:

```
(parted) print
```

3. Remove the partition:

```
(parted) rm minor-number
```

   - Replace *minor-number* with the minor number of the partition you want to remove: for example, **3**.

The changes start taking place as soon as you enter this command, so review it before executing it.

4. Confirm that the partition is removed from the partition table:

```
(parted) print
```

5. Exit the **parted** shell:

```
(parted) quit
```

6. Verify that the kernel knows the partition is removed:

```
# cat /proc/partitions
```

7. Remove the partition from the **/etc/fstab** file if it is present. Find the line that declares the removed partition, and remove it from the file.

8. Regenerate mount units so that your system registers the new **/etc/fstab** configuration:

```
# systemctl daemon-reload
```

9. If you have deleted a swap partition or removed pieces of LVM, remove all references to the partition from the kernel command line in the **/etc/default/grub** file and regenerate GRUB configuration:

   - On a BIOS-based system:

```
# grub2-mkconfig --output=/etc/grub2.cfg
```

   - On a UEFI-based system:

```
# grub2-mkconfig --output=/etc/grub2-efi.cfg
```

10. To register the changes in the early boot system, rebuild the **initramfs** file system:

```
# dracut --force --verbose
```

Additional resources

- The **parted(8)** man page

1.5. RESIZING A PARTITION

As a system administrator, you can extend a partition to utilize unused disk space, or shrink a partition to use its capacity for different purposes.

1.5.1. Considerations before modifying partitions on a disk

This section lists key points to consider before creating, removing, or resizing partitions.

NOTE

This section does not cover the DASD partition table, which is specific to the IBM System z architecture. For information on DASD, see the What you should know about DASD article at the IBM Knowledge Center.

The maximum number of partitions
The number of partitions on a device is limited by the type of the partition table:

- On a device formatted with the **Master Boot Record (MBR)** partition table, you can have either:
  - Up to four primary partitions, or
  - Up to three primary partitions, one extended partition, and multiple logical partitions within the extended.
- On a device formatted with the **GUID Partition Table (GPT)**, the maximum number of partitions is 128. While the GPT specification allows for more partitions by growing the area reserved for the partition table, common practice used by the **`parted`** utility is to limit it to enough area for 128 partitions.

The maximum size of a partition
The size of a partition on a device is limited by the type of the partition table:

- On a device formatted with the **Master Boot Record (MBR)** partition table, the maximum size is 2TiB.
- On a device formatted with the **GUID Partition Table (GPT)**, the maximum size is 8ZiB.

If you want to create a partition larger than 2TiB, the disk must be formatted with GPT.

Size alignment
The **`parted`** utility enables you to specify partition size using multiple different suffixes:

MiB, GiB, or TiB

Size expressed in powers of 2.

- The starting point of the partition is aligned to the exact sector specified by size.
- The ending point is aligned to the specified size minus 1 sector.

MB, GB, or TB

Size expressed in powers of 10.
The starting and ending point is aligned within one half of the specified unit: for example, ±500KB when using the MB suffix.

1.5.2. Resizing a partition with parted

This procedure resizes a disk partition using the **`parted`** utility.

Prerequisites

- If you want to shrink a partition, back up the data that are stored on it.

WARNING

Shrinking a partition might result in data loss on the partition.

- If you want to resize a partition to be larger than 2TiB, the disk must be formatted with the GUID Partition Table (GPT). For details on how to format the disk, see Section 1.2, “Creating a partition table on a disk”.

Procedure

1. If you want to shrink the partition, shrink the file system on it first so that it is not larger than the resized partition. Note that XFS does not support shrinking.
2. Start the interactive **parted** shell:

```
# parted block-device
```

   - Replace *block-device* with the path to the device where you want to resize a partition: for example, **/dev/sda**.

3. View the current partition table:

```
(parted) print
```

   From the partition table, determine:

   - The minor number of the partition
   - The location of the existing partition and its new ending point after resizing

4. Resize the partition:

```
(parted) resizepart minor-number new-end
```

   - Replace *minor-number* with the minor number of the partition that you are resizing: for example, **3**.
   - Replace *new-end* with the size that determines the new ending point of the resized partition, counting from the beginning of the disk. You can use size suffixes, such as **512MiB**, **20GiB**, or **1.5TiB**. The default size megabytes.

   Example 1.4. Extending a partition

   For example, to extend a partition located at the beginning of the disk to be 2GiB in size, use:

```
(parted) resizepart 1 2GiB
```

   The changes start taking place as soon as you enter this command, so review it before executing it.

5. View the partition table to confirm that the resized partition is in the partition table with the correct size:

```
(parted) print
```

6. Exit the **parted** shell:

```
(parted) quit
```

7. Verify that the kernel recognizes the new partition:

```
# cat /proc/partitions
```

8. If you extended the partition, extend the file system on it as well. See (reference) for details.

Additional resources

- The **parted(8)** man page.

CHAPTER 2. OVERVIEW OF PERSISTENT NAMING ATTRIBUTES

As a system administrator, you need to refer to storage volumes using persistent naming attributes to build storage setups that are reliable over multiple system boots.

2.1. DISADVANTAGES OF NON-PERSISTENT NAMING ATTRIBUTES

Red Hat Enterprise Linux provides a number of ways to identify storage devices. It is important to use the correct option to identify each device when used in order to avoid inadvertently accessing the wrong device, particularly when installing to or reformatting drives.

Traditionally, non-persistent names in the form of **/dev/sd*(major number)(minor number)*** are used on Linux to refer to storage devices. The major and minor number range and associated **sd** names are allocated for each device when it is detected. This means that the association between the major and minor number range and associated **sd** names can change if the order of device detection changes.

Such a change in the ordering might occur in the following situations:

- The parallelization of the system boot process detects storage devices in a different order with each system boot.
- A disk fails to power up or respond to the SCSI controller. This results in it not being detected by the normal device probe. The disk is not accessible to the system and subsequent devices will have their major and minor number range, including the associated **sd** names shifted down. For example, if a disk normally referred to as **sdb** is not detected, a disk that is normally referred to as **sdc** would instead appear as **sdb**.
- A SCSI controller (host bus adapter, or HBA) fails to initialize, causing all disks connected to that HBA to not be detected. Any disks connected to subsequently probed HBAs are assigned different major and minor number ranges, and different associated **sd** names.
- The order of driver initialization changes if different types of HBAs are present in the system. This causes the disks connected to those HBAs to be detected in a different order. This might also occur if HBAs are moved to different PCI slots on the system.
- Disks connected to the system with Fibre Channel, iSCSI, or FCoE adapters might be inaccessible at the time the storage devices are probed, due to a storage array or intervening switch being powered off, for example. This might occur when a system reboots after a power failure, if the storage array takes longer to come online than the system take to boot. Although some Fibre Channel drivers support a mechanism to specify a persistent SCSI target ID to WWPN mapping, this does not cause the major and minor number ranges, and the associated **sd** names to be reserved; it only provides consistent SCSI target ID numbers.

These reasons make it undesirable to use the major and minor number range or the associated **sd** names when referring to devices, such as in the **/etc/fstab** file. There is the possibility that the wrong device will be mounted and data corruption might result.

Occasionally, however, it is still necessary to refer to the **sd** names even when another mechanism is used, such as when errors are reported by a device. This is because the Linux kernel uses **sd** names (and also SCSI host/channel/target/LUN tuples) in kernel messages regarding the device.

2.2. FILE SYSTEM AND DEVICE IDENTIFIERS

This sections explains the difference between persistent attributes identifying file systems and block devices.

File system identifiers
File system identifiers are tied to a particular file system created on a block device. The identifier is also stored as part of the file system. If you copy the file system to a different device, it still carries the same file system identifier. On the other hand, if you rewrite the device, such as by formatting it with the **mkfs** utility, the device loses the attribute.

File system identifiers include:

- Unique identifier (UUID)
- Label

Device identifiers
Device identifiers are tied to a block device: for example, a disk or a partition. If you rewrite the device, such as by formatting it with the **mkfs** utility, the device keeps the attribute, because it is not stored in the file system.

Device identifiers include:

- World Wide Identifier (WWID)
- Partition UUID
- Serial number

Recommendations

- Some file systems, such as logical volumes, span multiple devices. Red Hat recommends accessing these file systems using file system identifiers rather than device identifiers.

2.3. DEVICE NAMES MANAGED BY THE UDEV MECHANISM IN /DEV/DISK/

This section lists different kinds of persistent naming attributes that the **udev** service provides in the **/dev/disk/** directory.

The **udev** mechanism is used for all types of devices in Linux, not just for storage devices. In the case of storage devices, Red Hat Enterprise Linux contains **udev** rules that create symbolic links in the **/dev/disk/** directory. This enables you to refer to storage devices by:

- Their content
- A unique identifier
- Their serial number.

Although **udev** naming attributes are persistent, in that they do not change on their own across system reboots, some are also configurable.

2.3.1. File system identifiers

The UUID attribute in /dev/disk/by-uuid/

Entries in this directory provide a symbolic name that refers to the storage device by a **unique identifier** (UUID) in the content (that is, the data) stored on the device. For example:

```
/dev/disk/by-uuid/3e6be9de-8139-11d1-9106-a43f08d823a6
```

You can use the UUID to refer to the device in the **/etc/fstab** file using the following syntax:

```
UUID=3e6be9de-8139-11d1-9106-a43f08d823a6
```

You can configure the UUID attribute when creating a file system, and you can also change it later on.

The Label attribute in /dev/disk/by-label/

Entries in this directory provide a symbolic name that refers to the storage device by a **label** in the content (that is, the data) stored on the device.

For example:

```
/dev/disk/by-label/Boot
```

You can use the label to refer to the device in the **/etc/fstab** file using the following syntax:

```
LABEL=Boot
```

You can configure the Label attribute when creating a file system, and you can also change it later on.

2.3.2. Device identifiers

The WWID attribute in /dev/disk/by-id/

The World Wide Identifier (WWID) is a persistent, **system-independent identifier** that the SCSI Standard requires from all SCSI devices. The WWID identifier is guaranteed to be unique for every storage device, and independent of the path that is used to access the device. The identifier is a property of the device but is not stored in the content (that is, the data) on the devices.

This identifier can be obtained by issuing a SCSI Inquiry to retrieve the Device Identification Vital Product Data (page **0x83**) or Unit Serial Number (page **0x80**).

Red Hat Enterprise Linux automatically maintains the proper mapping from the WWID-based device name to a current **/dev/sd** name on that system. Applications can use the **/dev/disk/by-id/** name to reference the data on the disk, even if the path to the device changes, and even when accessing the device from different systems.

Example 2.1. WWID mappings

WWID symlink	Non-persistent device	Note
/dev/disk/by-id/scsi-3600508b400105e210000900000490000	**/dev/sda**	A device with a page **0x83** identifier
/dev/disk/by-id/scsi-SSEAGATE_ST373453LW_3HW1RHM6	**/dev/sdb**	A device with a page **0x80** identifier

WWID symlink	Non-persistent device	Note
`/dev/disk/by-id/ata-SAMSUNG_MZNLN256HMHQ-000L7_S2WDNX0J336519-part3`	`/dev/sdc3`	A disk partition

In addition to these persistent names provided by the system, you can also use **udev** rules to implement persistent names of your own, mapped to the WWID of the storage.

The Partition UUID attribute in /dev/disk/by-partuuid

The Partition UUID (PARTUUID) attribute identifies partitions as defined by GPT partition table.

Example 2.2. Partition UUID mappings

PARTUUID symlink	Non-persistent device
`/dev/disk/by-partuuid/4cd1448a-01`	`/dev/sda1`
`/dev/disk/by-partuuid/4cd1448a-02`	`/dev/sda2`
`/dev/disk/by-partuuid/4cd1448a-03`	`/dev/sda3`

The Path attribute in /dev/disk/by-path/

This attribute provides a symbolic name that refers to the storage device by the **hardware path** used to access the device.

WARNING

The Path attribute is unreliable, and Red Hat does not recommend using it.

2.4. THE WORLD WIDE IDENTIFIER WITH DM MULTIPATH

This section describes the mapping between the World Wide Identifier (WWID) and non-persistent device names in a Device Mapper Multipath configuration.

If there are multiple paths from a system to a device, DM Multipath uses the WWID to detect this. DM Multipath then presents a single "pseudo-device" in the **`/dev/mapper/wwid`** directory, such as **`/dev/mapper/3600508b400105df70000e00000ac0000`**.

The command **`multipath -l`** shows the mapping to the non-persistent identifiers:

- ***Host*:*Channel*:*Target*:*LUN***
- **/dev/sd** name
- ***major*:*minor*** number

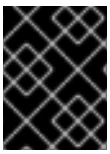
Example 2.3. WWID mappings in a multipath configuration

An example output of the **multipath -l** command:

```
3600508b400105df70000e00000ac0000 dm-2 vendor,product
[size=20G][features=1 queue_if_no_path][hwhandler=0][rw]
\_ round-robin 0 [prio=0][active]
 \_ 5:0:1:1 sdc 8:32  [active][undef]
 \_ 6:0:1:1 sdg 8:96  [active][undef]
\_ round-robin 0 [prio=0][enabled]
 \_ 5:0:0:1 sdb 8:16  [active][undef]
 \_ 6:0:0:1 sdf 8:80  [active][undef]
```

DM Multipath automatically maintains the proper mapping of each WWID-based device name to its corresponding **/dev/sd** name on the system. These names are persistent across path changes, and they are consistent when accessing the device from different systems.

When the **user_friendly_names** feature of DM Multipath is used, the WWID is mapped to a name of the form **/dev/mapper/mpath*N***. By default, this mapping is maintained in the file **/etc/multipath/bindings**. These **mpath*N*** names are persistent as long as that file is maintained.



IMPORTANT

If you use **user_friendly_names**, then additional steps are required to obtain consistent names in a cluster.

2.5. LIMITATIONS OF THE UDEV DEVICE NAMING CONVENTION

The following are some limitations of the **udev** naming convention:

- It is possible that the device might not be accessible at the time the query is performed because the **udev** mechanism might rely on the ability to query the storage device when the **udev** rules are processed for a **udev** event. This is more likely to occur with Fibre Channel, iSCSI or FCoE storage devices when the device is not located in the server chassis.
- The kernel might send **udev** events at any time, causing the rules to be processed and possibly causing the **/dev/disk/by-*/** links to be removed if the device is not accessible.
- There might be a delay between when the **udev** event is generated and when it is processed, such as when a large number of devices are detected and the user-space **udevd** service takes some amount of time to process the rules for each one. This might cause a delay between when the kernel detects the device and when the **/dev/disk/by-*/** names are available.
- External programs such as **blkid** invoked by the rules might open the device for a brief period of time, making the device inaccessible for other uses.

2.6. LISTING PERSISTENT NAMING ATTRIBUTES

This procedure describes how to find out the persistent naming attributes of non-persistent storage devices.

Procedure

- To list the UUID and Label attributes, use the **lsblk** utility:

```
$ lsblk --fs storage-device
```

For example:

Example 2.4. Viewing the UUID and Label of a file system

```
$ lsblk --fs /dev/sda1

NAME FSTYPE LABEL UUID
MOUNTPOINT
sda1 xfs    Boot  afa5d5e3-9050-48c3-acc1-bb30095f3dc4 /boot
```

- To list the PARTUUID attribute, use the **lsblk** utility with the **--output +PARTUUID** option:

```
$ lsblk --output +PARTUUID
```

For example:

Example 2.5. Viewing the PARTUUID attribute of a partition

```
$ lsblk --output +PARTUUID /dev/sda1

NAME MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT PARTUUID
sda1   8:1    0  512M  0 part /boot      4cd1448a-01
```

- To list the WWID attribute, examine the targets of symbolic links in the **/dev/disk/by-id/** directory. For example:

Example 2.6. Viewing the WWID of all storage devices on the system

```
$ file /dev/disk/by-id/*

/dev/disk/by-id/ata-QEMU_HARDDISK_QM00001:
symbolic link to ../../sda
/dev/disk/by-id/ata-QEMU_HARDDISK_QM00001-part1:
symbolic link to ../../sda1
/dev/disk/by-id/ata-QEMU_HARDDISK_QM00001-part2:
symbolic link to ../../sda2
/dev/disk/by-id/dm-name-rhel_rhel8-root:
symbolic link to ../../dm-0
/dev/disk/by-id/dm-name-rhel_rhel8-swap:
symbolic link to ../../dm-1
/dev/disk/by-id/dm-uuid-LVM-
```

```
QIWtEHtXGobe5bewlIUDivKOz5ofkgFhP0RMFsNyySVihqEl2cWWbR7MjXJolD6g:
symbolic link to ../../dm-1
/dev/disk/by-id/dm-uuid-LVM-
QIWtEHtXGobe5bewlIUDivKOz5ofkgFhXqH2M45hD2H9nAf2qfWSrlRLhzfMyOKd:
symbolic link to ../../dm-0
/dev/disk/by-id/lvm-pv-uuid-atlr2Y-vuMo-ueoH-CpMG-4JuH-AhEF-
wu4QQm:                          symbolic link to ../../sda2
```

2.7. MODIFYING PERSISTENT NAMING ATTRIBUTES

This procedure describes how to change the UUID or Label persistent naming attribute of a file system.

NOTE

Changing **udev** attributes happens in the background and might take a long time. The **udevadm settle** command waits until the change is fully registered, which ensures that your next command will be able to utilize the new attribute correctly.

In the following commands:

- Replace *new-uuid* with the UUID you want to set; for example, **1cdfbc07-1c90-4984-b5ec-f61943f5ea50**. You can generate a UUID using the **uuidgen** command.
- Replace *new-label* with a label; for example, **backup_data**.

Prerequisites

- If you are modifying the attributes of an XFS file system, unmount it first.

Procedure

- To change the UUID or Label attributes of an **XFS** file system, use the **xfs_admin** utility:

```
# xfs_admin -U new-uuid -L new-label storage-device
# udevadm settle
```

- To change the UUID or Label attributes of an **ext4**, **ext3**, or **ext2** file system, use the **tune2fs** utility:

```
# tune2fs -U new-uuid -L new-label storage-device
# udevadm settle
```

- To change the UUID or Label attributes of a swap volume, use the **swaplabel** utility:

```
# swaplabel --uuid new-uuid --label new-label swap-device
# udevadm settle
```