diskinfo

Release 3.1.2

Peter Sulyok

Apr 13, 2024

CONTENTS:

CHAPTER

ONE

INTRODUCTION

This Python library can assist in collecting disk information on Linux. In more details, it can:

- collect information about a specific disk
- explore all existing disks in the system
- translate between traditional and persistent disk names
- read disk temperature
- read SMART attributes of a disk
- read partition list of a disk

1.1 Installation

Standard installation from pypi:

```
pip install diskinfo
```

The library has the following run-time requirements:

- Python version >= *3.8*
- for reading SMART data with `get_smart_data()` method, the *smartmontools* package is required
- for reading disk temperature with `get_temperature()` method, the following dependencies needs to be considered:

Disk type	Requirement
SATA HDD/SSD	*drivetemp* kernel module (Linux kernel version *5.6*+) has to be loaded
SCSI/ATA HDD	*smartmontools* has to be installed
NVME	none

- for reading disk partition information with `get_partition_list()` method, the *df* command is required.
- optionally, for the demo Rich Python library is required

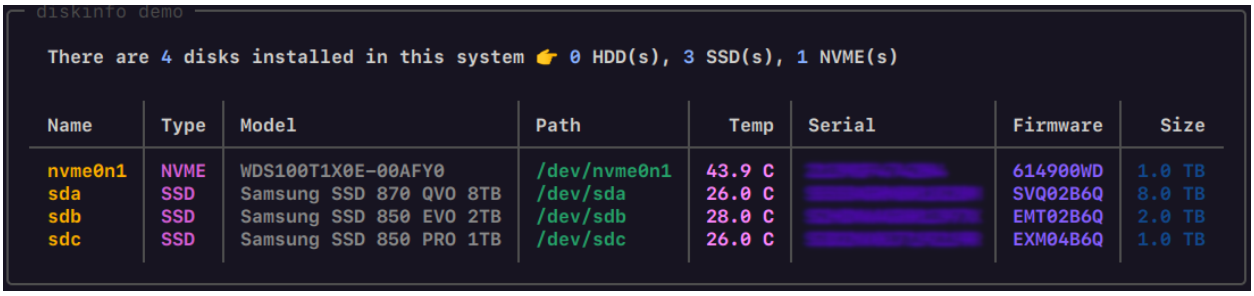
1.2 Demo

The library contains a demo application with multiple screens. In order to run demo execute the following commands:

```
pip install rich
```

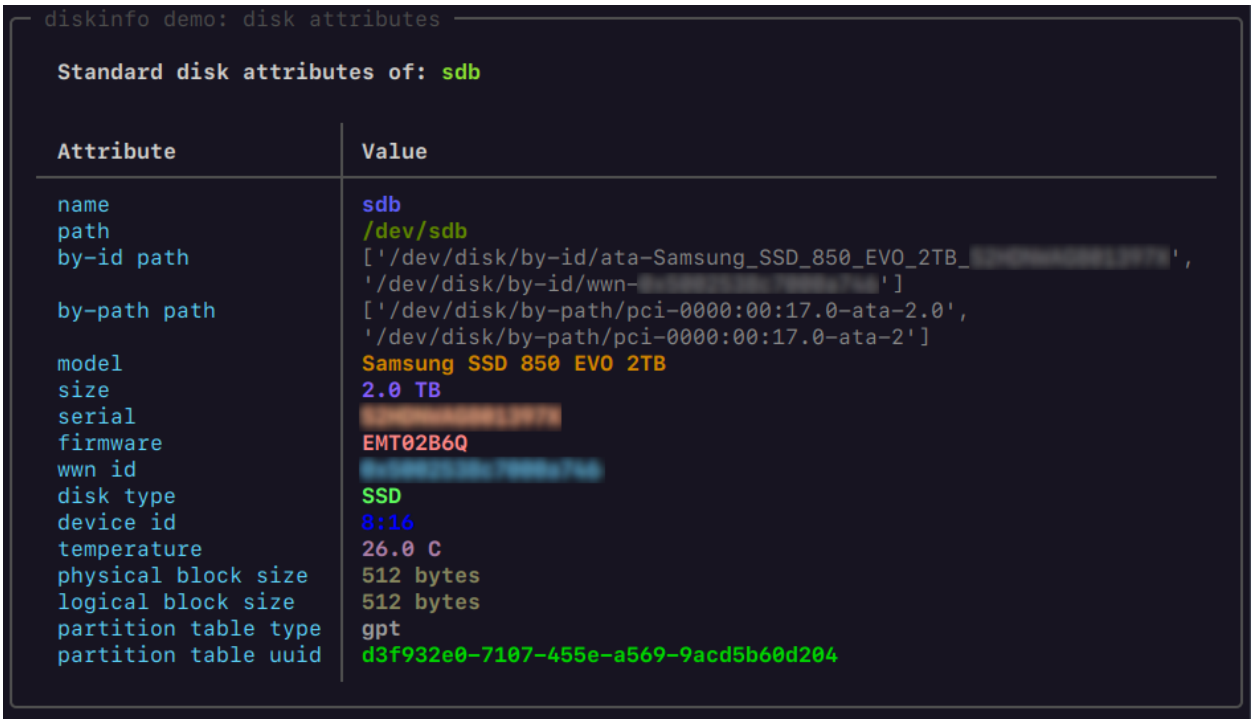
The first demo screen will list the explored disks:

```
python -m diskinfo.demo
```



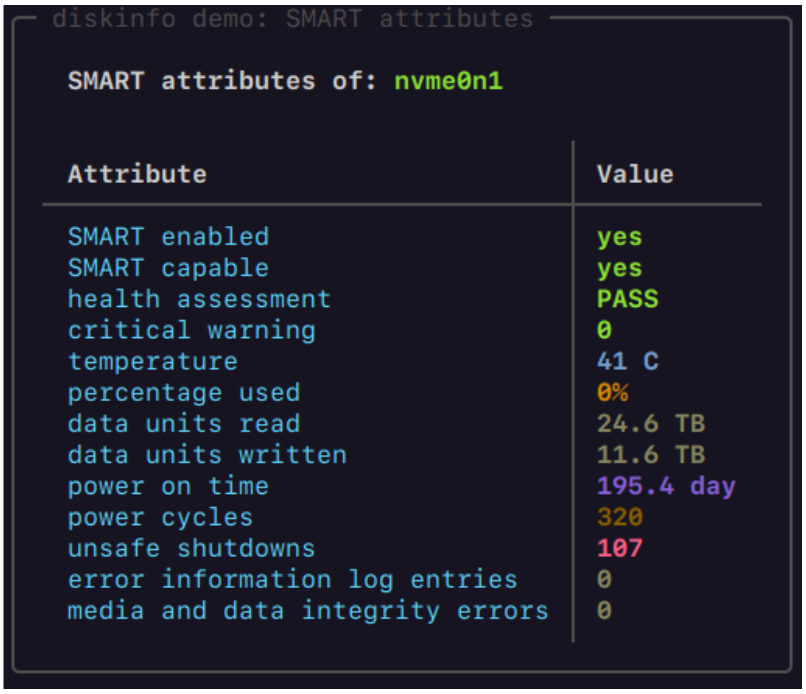
The second demo screen will display the attributes of a specified disk:

```
python -m diskinfo.demo sdb
```



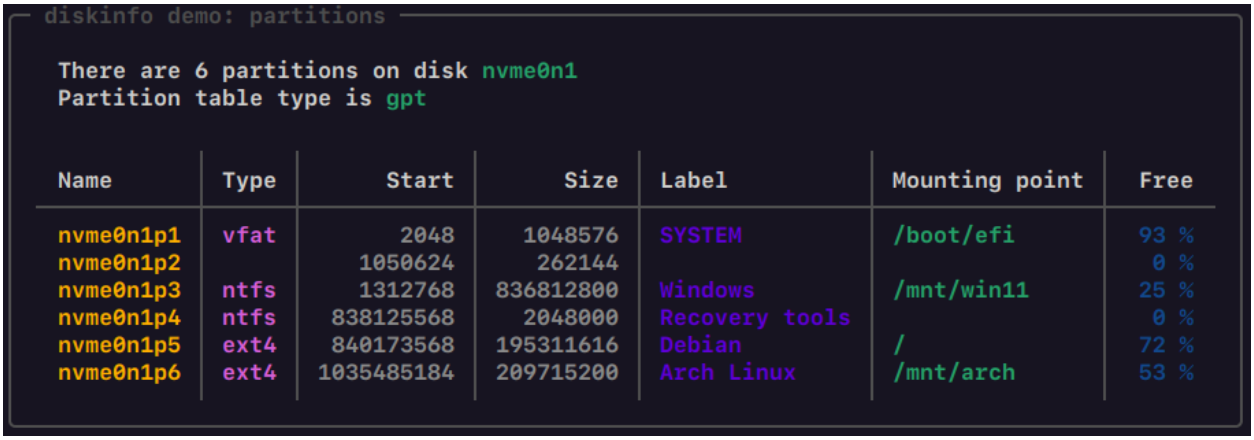
The third demo screen will display the SMART attributes of a specified disk:

```
python -m diskinfo.demo nvme0n1 -s
```



The fourth demo screen will display the list of partitions on a specified disk:

```
python -m diskinfo.demo nvme0n1 -p
```



1.3 Persistent disk names

Please note that the traditional disk names in Linux are not persistent:

```
/dev/sda
/dev/sdb
```

It means they can refer a different physical disk after a reboot. Read more about this topic at Arch Linux wiki: Persistent block device naming.

On the other hand, there are real persistent ways to refer disk or block devices in Linux:

1. *by-id* path: it can be found in */dev/disk/by-id* directory and it is constructed with disk serial numbers:

```
/dev/disk/by-id/ata-Samsung_SSD_850_PRO_1TB_92837A469FF876
/dev/disk/by-id/wwn-0x5002539c417223be
```

2. *by-path* path: it can be found in */dev/disk/by-path* directory and it is constructed with a physical path to the disk:

```
/dev/disk/by-path/pci-0000:00:17.0-ata-3
```

There are similar persistent constructions for disk partitions, too.

1.4 How to use

The following chapters will present the six use cases listed in the *Introduction* chapter above.

1.4.1 Use case 1: collect information about a disk

Disk attributes can be collected with the creation of a `Disk` class. All disk attributes will be collected at class creation time:

```
>>> from diskinfo import Disk
>>> d = Disk("sda")
```

and later the attributes can be accessed with the help of *get* functions of the class:

```
>>> d.get_model()
'Samsung SSD 870 QVO 8TB'
>>> d.is_ssd()
True
>>> s, u = d.get_size_in_hrf()
>>> print(f"{s:.1f} {u}")
8.0 TB
>>> d.get_serial()
'S5SXNG0MB01829M'
```

The `Disk` class contains the following disk attributes:

Attribute	Description	Sample value
name	Disk name	*sda* or *nvme0n1*)
path	Disk path	*/dev/sda* or */dev/nvme0n1*
by-id path	Persistent disk path in */dev/disk/by-id* directory	
by-path path	Persistent disk path in */dev/disk/by-path* directory	
wwn	World Wide Name	*0x5002538c307370ec*
model	Disk model	*Samsung SSD 850 PRO 1TB*
serial number	Disk serial number	*S3E2NY0J723218R*
firmware	Disk firmware	*EXM04B6Q*
type	Disk type	*HDD*, *SSD* or *NVME*
size	Disk size in 512-byte blocks	
device id	Disk device id, in *'major:minor'* form	*8:0*
physical block size	Disk physical block size in bytes	*512* or *4096*
logical block size	Disk logical block size in bytes	*512*
partition table type	Type of the partition table on disk	*gpt* or *mbr*
partition table uuid	UUID of the partition table on disk	

1.4.2 Use case 2: explore disks

Disks can be explored with the creation of the *DiskInfo* class. During this process all disks will identified and their attributes will be stored:

```
>>> from diskinfo import Disk, DiskInfo
>>> di = DiskInfo()
```

After that, the number of identified disks can be read with the help of *get_disk_number()* method:

```
>>> di.get_disk_number()
4
```

and the list of the disks can be accessed (see more details in *get_disk_list()* method):

```
>>> disks = di.get_disk_list(sorting=True)
>>> for d in disks:
>>>     print(d.get_path())
/dev/nvme0n1
/dev/sda
/dev/sdb
/dev/sdc
```

The caller can also apply filters (i.e. included and excluded disk types) for both functions and can query only subset of the disks based on one or more specific *DiskType*. The list of disk can be also sorted.

1.4.3 Use case 3: translate between traditional and persistent disk names

Translation from traditional disk names to persistent ones can be done this way:

```
>>> from diskinfo import Disk
>>> d = Disk("sda")
>>> d.get_byid_path()
['/dev/disk/by-id/ata-Samsung_SSD_850_PRO_1TB_92837A469FF876', '/dev/disk/by-id/wwn-
↪0x5002539c417223be']
>>> d.get_bypath_path()
['/dev/disk/by-path/pci-0000:00:17.0-ata-3', '/dev/disk/by-path/pci-0000:00:17.0-ata-3.0
↪']
>>> d.get_serial_numner()
'92837A469FF876'
>>> d.get_wwn()
'0x5002539c417223be'
```

In the opposite direction several unique (persistent) identifier can be used to initialize *Disk* class then the traditional disk path or name can be read:

```
>>> from diskinfo import Disk
>>> d = Disk(byid_name="ata-Samsung_SSD_850_PRO_1TB_92837A469FF876")
>>> d.get_path()
'/dev/sda'
>>> d = Disk(bypath_name="pci-0000:00:17.0-ata-3")
>>> d.get_path()
'/dev/sda'
```

(continues on next page)

(continued from previous page)

```
>>> d = Disk(serial_number="92837A469FF876")
>>> d.get_path()
'/dev/sda'
>>> d = Disk(wwn="0x5002539c417223be")
>>> d.get_name()
'sda'
```

1.4.4 Use case 4: read disk temperature

After having a *Disk* class instance, the disk temperature can be read in this way:

```
>>> from diskinfo import Disk
>>> d = Disk("sda")
>>> d.get_temperature()
28
```

Please note that the drivetemp kernel module should be loaded for SSDs and HDDs (available from Linux Kernel 5.6+). NVME disks do not require anything.

1.4.5 Use case 5: read disk SMART attributes

After having a *Disk* class instance, the SMART attributes of the disk can be read with the help of *get_smart_data()* method.

```
>>> from diskinfo import Disk, DiskSmartData
>>> d = Disk("sda")
>>> sd = d.get_smart_data()
```

In case of HDDs, we can skip checking if they are in STANDBY mode:

```
>>> sd = d.get_smart_data(nocheck=True)
>>> if sd.standby_mode:
...     print("Disk is in STANDBY mode.")
... else:
...     print("Disk is ACTIVE.")
...
Disk is in STANDBY mode.
```

If we dont use the *nocheck* parameter here (when the HDD is in STANDBY mode) then the HDD will spin up and will return to ACTIVE mode. Please note if *standby_mode* is *True* then no other SMART attributes are loaded.

The most important SMART information for all disk types is the health status:

```
>>> if sd.healthy:
...     print("Disk is HEALTHY.")
... else:
...     print("Disk is FAILED!")
...
Disk is HEALTHY.
```

In case of SSDs and HDDs the traditional SMART attributes can be accessed via *smart_attributes* list:

```
>>> for item in sd.smart_attributes:
...     print(f"{item.id:>3d} {item.attribute_name}: {item.raw_value}")
...
  5 Reallocated_Sector_Ct: 0
  9 Power_On_Hours: 6356
 12 Power_Cycle_Count: 2308
177 Wear_Leveling_Count: 2
179 Used_Rsvd_Blk_Cnt_Tot: 0
181 Program_Fail_Cnt_Total: 0
182 Erase_Fail_Count_Total: 0
183 Runtime_Bad_Block: 0
187 Uncorrectable_Error_Cnt: 0
190 Airflow_Temperature_Cel: 28
195 ECC_Error_Rate: 0
199 CRC_Error_Count: 0
235 POR_Recovery_Count: 67
241 Total_LBAs_Written: 9869978356
```

See more details in *DiskSmartData* and *SmartAttribute* classes.

In case of NVME disks they have their own SMART data in *nvme_attributes* attribute:

```
>>> if d.is_nvme():
...     print(f"Power on hours: {sd.nvme_attributes.power_on_hours} h")
...
Power on hours: 1565 h
```

See the detailed list of the NVME attributes in *NvmeAttributes* class.

Please note that the *get_smart_data()* method relies on *smartctl* command. It means that the caller needs to have special access rights (i.e. *sudo* or *root*).

1.4.6 Use case 6: read partition list

After having a *Disk* class instance, the partition list can be read with the help of *get_partition_list()* method.

```
>>> from diskinfo import Disk, DiskSmartData
>>> d = Disk("sda")
>>> plist = d.get_partition_list()
```

The return value is a list of *Partition* classes. This class provides several get functions to access the partition attributes:

```
>>> from diskinfo import Disk
>>> disk = Disk("nvme0n1")
>>> plist = disk.get_partition_list()
>>> for item in plist:
...     Disk(item.get_name())
...
nvme0n1p1
nvme0n1p2
nvme0n1p3
nvme0n1p4
```

(continues on next page)

(continued from previous page)

```
nvme0n1p5
nvme0n1p6
```

The `Partition` class contains the following partition attributes:

Attribute	Description	Sample value
name	Partition name	*sda1* or *nvme0n1p1*
Path	Partition path	*/dev/sda1* or */dev/nvme0n1p1*
by-id path	Persistent path in */dev/disk/by-id* directory	
by-path path	Persistent path in */dev/disk/by-path* directory	
by-partuuid path	Persistent path in */dev/disk/by-partuuid* directory	
by-partlabel path	Persistent path in */dev/disk/by-partlabel* directory	
by-uuid path	Persistent path in */dev/disk/by-uuid* directory	
by-label path	Persistent path in */dev/disk/by-label* directory	
Device id	Partition device id	*8:1*
Partition scheme	Partition scheme	*gtp* or *mbr*
Partition label	Partition label	*Basic data partition*
Partition UUID	Partition UUID	*acb8374d-fb60-4cb0-8ac4-273417c6f847*
Partition type	Partition type UUID	
Partition number	Partition number in the partition table	
Partition offset	Partition starting offset in 512-byte blocks	
Partition size	Partition size in 512-byte blocks	
File system label	File system label	
File system UUID	File system UUID	
File system type	File system type	*ntfs* or *ext4*)
File system version	File system version	*1.0* in case of *ext4*)
File system usage	File system usage	*filesystem* or *other*
File system free size	File system free size in 512-byte blocks	
File system mounting point	File system mounting point	*/* or */home*

CHAPTER

TWO

API REFERENCE

In this section you will find the detailed API reference about the implemented classes and functions of `diskinfo` library.

2.1 *Disk* class

class diskinfo.**Disk**(*disk_name=None*, *serial_number=None*, *wwn=None*, *byid_name=None*, *bypath_name=None*, *encoding='utf-8'*)

The Disk class contains all disk related information. The class can be initialized with specifying one of the five unique identifiers of the disk:

- a disk name (e.g. *sda* or *nvme0n1*) located in */dev/* directory.
- a disk serial number (e.g. *"92837A469FF876"*)
- a disk wwn identifier (e.g. *"0x5002638c807270be"*)
- a *by-id* name of the disk (e.g. *"ata-Samsung_SSD_850_PRO_1TB_92837A469FF876"*) located in */dev/disk/by-id/* directory
- a *by-path* name of the disk (e.g. *"pci-0000:00:17.0-ata-3"*) located in */dev/disk/by-path/* directory

Based on the specified input parameter the disk will be identified and its attributes will be collected and stored. A `ValueError` exception will be raised in case of missing or invalid disk identifier. Encoding parameter will be used for processing disk attribute strings.

Operators (<, > and ==) are also implemented for this class to compare different class instances, they use the disk name for comparison.

Note: During the class initialization the disk will not be physically accessed.

Parameters

- **disk_name** (*str*) – the disk name
- **serial_number** (*str*) – serial number of the disk
- **wwn** (*str*) – wwn identifier of the disk
- **byid_name** (*str*) – by-id name of the disk
- **bypath_name** (*str*) – by-path name of the disk
- **encoding** (*str*) – encoding (default is *utf-8*)

Raises

- **ValueError** – in case of missing or invalid parameters
- **RuntimeError** – in case of any system error

Example

This example shows how to create a *Disk* class then how to get its path and serial number:

```
>>> from diskinfo import Disk
>>> d = Disk("sda")
>>> d.get_path()
'/dev/sda'
>>> d.get_serial_number()
'S3D2NY0J819210S'
```

and here are additional ways how the *Disk* class can be initialized:

```
>>> d = Disk(serial_number="92837A469FF876")
>>> d.get_name()
'sdc'
>>> d = Disk(wwn="0x5002539c417223be")
>>> d.get_name()
'sdc'
>>> d = Disk(byid_name="ata-Samsung_SSD_850_PRO_1TB_92837A469FF876")
>>> d.get_name()
'sdc'
>>> d = Disk(bypath_name="pci-0000:00:17.0-ata-3")
>>> d.get_name()
'sdc'
```

get_byid_path()

Returns disk path in a persistent `/dev/disk/by-byid/...` form. The result could be one or more path elements in a list.

Example

An example about the use of this function:

```
>>> from diskinfo import Disk
>>> d = Disk("sdc")
>>> d.get_byid_path()
['/dev/disk/by-id/ata-Samsung_SSD_850_PRO_1TB_92837A469FF876', '/dev/disk/by-id/
↪wwn-0x5002539c417223be']
```

Return type
List[str]

get_bypath_path()

Returns disk path in a persistent `/dev/disk/by-path/...` form. The result could be one or more path elements in a list.

Example

An example about the use of this function:

```
>>> from diskinfo import Disk
>>> d = Disk("sdc")
>>> d.get_bypath_path()
['/dev/disk/by-path/pci-0000:00:17.0-ata-3', '/dev/disk/by-path/pci-0000:00:17.
↪0-ata-3.0']
```

Return type
List[str]

get_device_id()

Returns the disk device id in *'major:minor'* form.

Example

An example about the use of this function:

```
>>> from diskinfo import Disk
>>> d = Disk("sdc")
>>> d.get_device_id()
'8:32'
```

Return type
str

get_firmware()

Returns the disk firmware string.

Example

An example about the use of this function:

```
>>> from diskinfo import Disk
>>> d = Disk("sdc")
>>> d.get_firmware()
'EXM04B6Q'
```

Return type
str

get_logical_block_size()

Returns the logical block size of the disk in bytes. Typically, it is 512 bytes.

Example

An example about the use of this function:

```
>>> from diskinfo import Disk
>>> d = Disk("sdc")
>>> d.get_logical_block_size()
512
```

Return type
int

`get_model()`
Returns the disk model string.

Example

An example about the use of this function:

```
>>> from diskinfo import Disk
>>> d = Disk("sdc")
>>> d.get_model()
'Samsung SSD 850 PRO 1TB'
```

Return type
str

`get_name()`
Returns the disk name.

Example

An example about the use of this function:

```
>>> from diskinfo import Disk
>>> d = Disk(serial_number="92837A469FF876")
>>> d.get_name()
'sdc'
```

Return type
str

`get_partition_list()`
Reads partition information of the disk and returns the list of partitions. See *Partition* class for more details for a partition entry.

Returns
list of partitions

Return type
List[*Partition*]

Example

```
>>> from diskinfo import *
>>> disk=Disk("nvme0n1")
>>> plist=disk.get_partition_list()
>>> for item in plist:
...     print(item.get_name())
...
nvme0n1p1
nvme0n1p2
nvme0n1p3
nvme0n1p4
nvme0n1p5
nvme0n1p6
```

get_partition_table_type()

Returns the type of the partition table on the disk (e.g. *mbr* or *gpt*).

Example

An example about the use of this function:

```
>>> from diskinfo import Disk
>>> d = Disk("sdc")
>>> d.get_partition_table_type()
'gpt'
```

Return type

str

get_partition_table_uuid()

Returns the UUID of the partition table on the disk.

Example

An example about the use of this function:

```
>>> from diskinfo import Disk
>>> d = Disk("sdc")
>>> d.get_partition_table_uuid()
'd3f932e0-7107-455e-a569-9acd5b60d204'
```

Return type

str

get_path()

Returns the disk path.

Example

An example about the use of this function:

```
>>> from diskinfo import Disk
>>> d = Disk(serial_number="92837A469FF876")
>>> d.get_path()
'/dev/sdc'
```

Note: Please note this path is not persistent (i.e. it may refer different physical disk after a reboot).

Return type
str

`get_physical_block_size()`

Returns the physical block size of the disk in bytes. Typically, it is 512 bytes for SSDs and NVMEs, and it could be 4096 bytes for HDDs.

Example

An example about the use of this function:

```
>>> from diskinfo import Disk
>>> d = Disk("sdc")
>>> d.get_physical_block_size()
512
```

Return type
int

`get_serial_number()`

Returns the disk serial number .

Note: This is a unique and persistent identifier of the disk.

Example

An example about the use of this function:

```
>>> from diskinfo import Disk
>>> d = Disk("sdc")
>>> d.get_serial_number()
'92837A469FF876'
```

Return type
str

get_size()

Returns the size of the disk in 512-byte units.

Example

An example about the use of this function:

```
>>> from diskinfo import Disk
>>> d = Disk("sdc")
>>> s = d.get_size()
>>> print(f"Disk size: { s * 512 } bytes.")
Disk size: 1024209543168 bytes.
```

Return type
int

get_size_in_hrf(*units=0*)

Returns the size of the disk in a human-readable form.

Parameters
units (*int*) – unit system will be used in result:

- 0 metric units (default)
- 1 IEC units
- 2 legacy units

Read more about units here.

Returns
size of the disk, proper unit

Return type
Tuple[float, str]

Example

An example about the use of this function:

```
>>> from diskinfo import Disk
>>> d = Disk("sdc")
>>> s,u = d.get_size_in_hrf()
>>> print(f"{s:.1f} {u}")
1.0 TB
```

get_smart_data(*nocheck=False*, *sudo=False*, *smartctl_path='/usr/sbin/smartctl'*)

Returns SMART data of the disk. This function will execute `smartctl` command from smartmontools package, it needs to be installed.

Note: `smartctl` command needs root priviledge for reading device SMART attributes. This function has to be used as *root* user or called with `sudo=True` parameter.

In case of HDDs, the `smartctl` command will access the disk directly and the HDD could be woken up. If the `nocheck=True` parameter is used then the current power state of the disk will be preserved.

Parameters

- **nocheck** (*bool*) – No check should be applied for a HDDs ("`-n standby`" argument will be used)
- **sudo** (*bool*) – sudo command should be used, default value is `False`
- **smartctl_path** (*str*) – Path for `smartctl` command, default value is `/usr/sbin/smartctl`

Returns

SMART information of the disk (see more details at *DiskSmartData* class) or None if *smartctl* cannot be executed

Return type

DiskSmartData

Example

The example show the use of the function:

```
>>> from diskinfo import Disk, DiskSmartData
>>> d = Disk("sda")
>>> d = d.get_smart_data()
```

In case of SSDs and HDDs the traditional SMART attributes can be accessed via *smart_attributes* list:

```
>>> for item in d.smart_attributes:
...     print(f"{item.id:>3d} {item.attribute_name}: {item.raw_value}")
...
  5 Reallocated_Sector_Ct: 0
  9 Power_On_Hours: 6356
 12 Power_Cycle_Count: 2308
177 Wear_Leveling_Count: 2
179 Used_Rsvd_Blk_Cnt_Tot: 0
181 Program_Fail_Cnt_Total: 0
182 Erase_Fail_Count_Total: 0
183 Runtime_Bad_Block: 0
187 Uncorrectable_Error_Cnt: 0
190 Airflow_Temperature_Cel: 28
195 ECC_Error_Rate: 0
199 CRC_Error_Count: 0
235 POR_Recovery_Count: 67
241 Total_LBAs_Written: 9869978356
```

In case of NVME disks they have their own SMART data in *nvme_attributes* field:

```
>>> if d.is_nvme():
...     print(f"Power on hours: {d.nvme_attributes.power_on_hours} h")
...
Power on hours: 1565 h
```

get_temperature(*sudo=False*, *smartctl_path='/usr/sbin/smartctl'*)

Returns the current disk temperature. The method will try to read disk temperature from the Linux kernel HWMON interface first then will try to execure the *smartctl* command. The method has the following requirements:

Disk type	Requirement
SATA HDD/SSD	*drivetemp* kernel module (Linux kernel version *5.6*+) has to be loaded
SCSI/ATA HDD	*smartmontools* has to be installed
NVME	none

Note: Please note that reading disk temperature from HWMON kernel interface will not access the disk and will not change its power state (e.g. HDD can be in STANDBY state) but reading disk temperature with *smartctl* will do.

Parameters

- **sudo** (*bool*) – `sudo` command should be used for `smartctl`, default value is `False`
- **smartctl_path** (*str*) – Path for `smartctl` command, default value is `/usr/sbin/smartctl`

Returns
disk temperature in C degree or None if the temperature cannot be determined

Return type
float

Example

An example about the use of this function:

```
>>> from diskinfo import Disk
>>> d = Disk("sdc")
>>> d.get_temperature(sudo=True)
28.5
```

`get_type()`

Returns the type of the disk. One of the constants in *DiskType* class:

- `DiskType.HDD` for hard disks (with spinning platters)
- `DiskType.SSD` for SDDs on SATA or USB interface
- `DiskType.NVME` for NVME disks
- `DiskType.LOOP` for LOOP disks

Example

An example about the use of this function:

```
>>> from diskinfo import Disk
>>> d = Disk("sdc")
>>> d.get_type()
2
```

Return type
int

get_type_str()

Returns the name of the disk type. See the return values in *DiskType* class:

- `DiskType.HDD_STR` for hard disks (with spinning platters)
- `DiskType.SSD_STR` for SDDs on SATA or USB interface
- `DiskType.NVME_STR` for NVME disks
- `DiskType.LOOP_STR` for LOOP disks

Raises
RuntimeError – in case of unknown disk type.

Return type
str

Example

An example about the use of this function:

```
>>> from diskinfo import Disk
>>> d = Disk("sdc")
>>> d.get_type_str()
'SSD'
```

get_wwn()

Returns the world-wide name (WWN) of the disk. Read more about WWN here.

Note: This is a unique and persistent identifier of the disk.

Example

An example about the use of this function:

```
>>> from diskinfo import Disk
>>> d = Disk("sdc")
>>> d.get_wwn()
'0x5002539c417223be'
```

Return type
str

is_hdd()

Returns *True* if the disk type is HDD, otherwise *False*.

Example

An example about the use of this function:

```
>>> from diskinfo import Disk
>>> d = Disk("sdc")
>>> d.is_hdd()
False
```

Return type
bool

`is_loop()`

Returns *True* if the disk type is LOOP, otherwise *False*.

Example

An example about the use of this function:

```
>>> from diskinfo import Disk
>>> d = Disk("loop0")
>>> d.is_loop()
True
```

Return type
bool

`is_nvme()`

Returns *True* if the disk type is NVME, otherwise *False*.

Example

An example about the use of this function:

```
>>> from diskinfo import Disk
>>> d = Disk("sdc")
>>> d.is_nvme()
False
```

Return type
bool

`is_ssd()`

Returns *True* if the disk type is SSD, otherwise *False*.

Example

An example about the use of this function:

```
>>> from diskinfo import Disk
>>> d = Disk("sdc")
>>> d.is_ssd()
True
```

Return type
bool

2.2 *DiskType* class

class diskinfo.**DiskType**

Constant values for disk types and for their names.

HDD = 1

Hard disk type.

HDD_STR = 'HDD'

Name of the hard disk type.

LOOP = 8

LOOP disk type.

LOOP_STR = 'LOOP'

Name of the LOOP disk type.

NVME = 4

NVME disk type.

NVME_STR = 'NVME'

Name of the NVME disk type.

SSD = 2

SSD disk type.

SSD_STR = 'SSD'

Name of the SSD disk type.

2.3 *DiskSmartData* class

class diskinfo.**DiskSmartData**

This class presents all collected SMART data for a disk. This class is created by *get_smart_data()* method. There are several disk type specific data attributes in this class, they are available only for a specific disk type(s).

find_smart_attribute_by_id(*id_val*)

Finds a SMART attribute by the *id* and returns its index in *smart_attributes* list, or -1 if not found.

Parameters
id_val (*int*) – SMART attribute *id* value

Returns
an index of the attribute in *smart_attributes* list, or -1 if not found

Return type
int

Example

An example about the use of this function:

```
>>> from diskinfo import Disk, DiskSmartData
>>> d = Disk("sda")
>>> sd = d.get_smart_data(sudo=True)
>>> sd.find_smart_attribute_by_id(5)
0
```

find_smart_attribute_by_name(*name_val*)

Finds an attribute by *name* and return with its index in *smart_attributes* list.

Parameters
name_val (*int*) – SMART attribute name value

Returns
an index of the attribute in *smart_attributes* list, or -1 if not found

Return type
int

Example

An example about the use of this function:

```
>>> from diskinfo import Disk, DiskSmartData
>>> d = Disk("sda")
>>> sd = d.get_smart_data(sudo=True)
>>> sd.find_smart_attribute_by_name("Power_On_Hours")
1
```

healthy: bool

The health status of the disk. Valid for all disk types. Meaning:

- *True*: the disk is reported healthy (i.e. overall-health self-assessment test is PASSED).
- *False*: the disk is reported failed (i.e. overall-health self-assessment test is FAILED).

nvme_attributes: *NvmeAttributes*

NVME attributes for a disk. Valid only for NVME disks. See more details in *NvmeAttributes* class.

smart_attributes: List[*SmartAttribute*]

List of the SMART attributes of a disk. Valid only for HDDs and SSDs, in case of NVME disk it will be empty. Methods *find_smart_attribute_by_id()* and *find_smart_attribute_by_name()* will find an item on this list based on an *id* or a *name*.

smart_capable: bool

The disk is SMART capable.

smart_enabled: bool

SMART is enabled for the disk.

standby_mode: bool

Standby flag for disk. Valid only for HDDs:

- *True* means the disk is in STANDBY state
- *False* means the disk is ACTIVE or IDLE

> **Warning:** When an HDD is in STANDBY state (i.e. this flag is *True*) then other SMART attributes will not be updated!

2.4 *SmartAttribute* class

class diskinfo.**SmartAttribute**(*_id*, *attribute_name*, *flag*, *value*, *worst*, *thresh*, *_type*, *updated*, *when_failed*, *raw_value*)

This class implements a SMART attribute. It stores ten values from *smartctl* command.

Example

This class presents one line in the following sample output (from a`smartctrl` command):

```
ID# ATTRIBUTE_NAME          FLAG     VALUE WORST THRESH TYPE      UPDATED  WHEN_
↪FAILED RAW_VALUE
  5 Reallocated_Sector_Ct   0x0033   100   100   010    Pre-fail  Always       -
↪    0
  9 Power_On_Hours          0x0032   095   095   000    Old_age   Always       -
↪    23268
 12 Power_Cycle_Count       0x0032   092   092   000    Old_age   Always       -
↪    7103
177 Wear_Leveling_Count     0x0013   099   099   000    Pre-fail  Always       -
↪    20
179 Used_Rsvd_Blk_Cnt_Tot   0x0013   100   100   010    Pre-fail  Always       -
↪    0
181 Program_Fail_Cnt_Total  0x0032   100   100   010    Old_age   Always       -
↪    0
182 Erase_Fail_Count_Total  0x0032   100   100   010    Old_age   Always       -
↪    0
183 Runtime_Bad_Block       0x0013   100   099   010    Pre-fail  Always       -
↪    0
187 Uncorrectable_Error_Cnt 0x0032   100   100   000    Old_age   Always       -
↪    0
190 Airflow_Temperature_Cel 0x0032   072   045   000    Old_age   Always       -
↪    28
195 ECC_Error_Rate          0x001a   200   200   000    Old_age   Always       -
↪    0
199 CRC_Error_Count         0x003e   100   100   000    Old_age   Always       -
↪    0
235 POR_Recovery_Count      0x0012   099   099   000    Old_age   Always       -
```

(continues on next page)

(continued from previous page)

```
↪   218
241 Total_LBAs_Written      0x0032   099   099   000    Old_age   Always       -   ␣
↪   51650461687
```

Parameters

- **_id** (*int*) –
- **attribute_name** (*str*) –
- **flag** (*int*) –
- **value** (*int*) –
- **worst** (*int*) –
- **thresh** (*int*) –
- **_type** (*str*) –
- **updated** (*str*) –
- **when_failed** (*str*) –
- **raw_value** (*int*) –

attribute_name: `str`

Name of the SMART attribute.

flag: `int`

SMART attribute handling flag.

id: `int`

ID of the SMART attribute (*1-255*).

raw_value: `int`

The raw value for the SMART attribute, defined by the manufacturer.

thresh: `int`

Lowest value (defined by manufacturer) that `worst` is allowed to reach for a SMART attribute.

type: `str`

Type of the SMART attribute:

- *Pre-fail*: the attribute is considered a critical one
- *Old_age*: the attribute does not fail the disk

updated: `str`

Indicator when the SMART attribute is updated (values are *Always* or *Offline*).

value: `int`

Normalized value of SMART attribute (*0-255*). Notes:

- values *0*, *254*, *255* are reserved for internal use
- value *253* means not used yet
- many cases it is starting at *value=100* case then dropping to *value=1*

when_failed: *str*

Usually it is blank, or it contains the last operational hour when this SMART attribute failed.

worst: *int*

Lowest value of the SMART attribute.

2.5 *NvmeAttributes* class

class diskinfo.**NvmeAttributes**(*critical_warning*, *temperature*, *available_spare*, *available_spare_threshold*, *percentage_used*, *data_units_read*, *data_units_written*, *host_read_commands*, *host_write_commands*, *controller_busy_time*, *power_cycles*, *power_on_hours*, *unsafe_shutdowns*, *media_and_data_integrity_errors*, *error_information_log_entries*, *warning_composite_temperature_time*, *critical_composite_temperature_time*)

This class implements NVME attributes. Read more about NVME attributes:

- smartmontools: NVME support
- NVME standard v1.4 (page 121-125)

Parameters

- **critical_warning** (*int*) –
- **temperature** (*int*) –
- **available_spare** (*int*) –
- **available_spare_threshold** (*int*) –
- **percentage_used** (*int*) –
- **data_units_read** (*int*) –
- **data_units_written** (*int*) –
- **host_read_commands** (*int*) –
- **host_write_commands** (*int*) –
- **controller_busy_time** (*int*) –
- **power_cycles** (*int*) –
- **power_on_hours** (*int*) –
- **unsafe_shutdowns** (*int*) –
- **media_and_data_integrity_errors** (*int*) –
- **error_information_log_entries** (*int*) –
- **warning_composite_temperature_time** (*int*) –
- **critical_composite_temperature_time** (*int*) –

available_spare: *int*

Contains a normalized percentage (0% to 100%) of the remaining spare capacity available.

available_spare_threshold: int

When the Available Spare falls below the threshold indicated in this field, an asynchronous event completion may occur. The value is indicated as a normalized percentage (0% to 100%).

controller_busy_time: int

Contains the amount of time (in minutes) the controller is busy with I/O commands.

critical_composite_temperature_time: int

Contains the amount of time in minutes that the controller is operational and the Composite Temperature is greater than or equal to the Critical Composite Temperature Threshold.

critical_warning: int

This attributes indicates critical warnings for the state of the controller.

data_units_read: int

Contains the number of 512-byte blocks the host has read from the NVME controller. The value reported in thousands (i.e. 1 means 1000 units of 512-byte blocks) and rounded up. Value 0 means that this attribute is not reported.

data_units_written: int

Contains the number of 512-byte blocks the host has written to the NVME controller. The value reported in thousands (i.e. 1 means 1000 units of 512-byte blocks) and rounded up. Value 0 means that this attribute is not reported.

error_information_log_entries: int

Contains the number of Error Information log entries over the life of the controller.

host_read_commands: int

Contains the number of read commands completed by the controller.

host_write_commands: int

Contains the number of write commands completed by the controller.

media_and_data_integrity_errors: int

Contains the number of occurrences where the controller detected an unrecovered data integrity error.

percentage_used: int

Contains a vendor specific estimate of the percentage of NVM subsystem life used based on the actual usage and the manufacturer's prediction of NVM life. A value of 100 indicates that the estimated endurance of the NVM in the NVM subsystem has been consumed, but may not indicate an NVM subsystem failure.

power_cycles: int

Contains the number of the power cycles.

power_on_hours: int

Contains the number of power-on hours.

temperature: int

Contains the current composite temperature value of the NVME disk.

unsafe_shutdowns: int

Contains the number of unsafe shutdowns.

warning_composite_temperature_time: int

Contains the amount of time in minutes that the controller is operational and the Composite Temperature is greater than or equal to the Warning Composite Temperature Threshold.

2.6 *Partition* class

class diskinfo.**Partition**(*name*, *dev_id*)

This class is the implementation of a partition entry. It is created by the *get_partition_list()* method. All partition attributes are collected at class creation time and these attributes can be accessed through get functions of the class later.

Note:

1. The class creation and the get functions will not generate disk operations and will not change the power state of the disk.
2. The class uses the *df* command to find the mounting point and available space of a file system.

Parameters

- **name** (*str*) – name of the partition (e.g. *sda1*)
- **dev_id** (*str*) – device id of the partition (e.g. *8:1*)

Raises

- **ValueError** – in case of invalid input parameters
- **FileNotFoundError** – if the *df* command cannot be executed

Example

This example shows the basic use of the class:

```
>>> from diskinfo import Disk
>>> disk = Disk("nvme0n1")
>>> plist = disk.get_partition_list()
>>> for item in plist:
...     print(item.get_name())
...
nvme0n1p1
nvme0n1p2
nvme0n1p3
nvme0n1p4
nvme0n1p5
nvme0n1p6
```

get_byid_path()

Returns the *by-id* persistent path of the partition. The result could be on or more path elements.

Example

An example about use of the function:

```
>>> from diskinfo import Disk
>>> disk = Disk("nvme0n1")
>>> plist = disk.get_partition_list()
>>> for item in plist:
...     print(item.get_name(), "-", item.get_byid_path())
...
nvme0n1p1 - ['/dev/disk/by-id/nvme-WDS100T1X0E-00AFY0_2140GF374501-part1',
 '/dev/disk/by-id/nvme-eui.e8238fa6bf540001001b555a49bfc681-part1']
nvme0n1p2 - ['/dev/disk/by-id/nvme-WDS100T1X0E-00AFY0_2140GF374501-part2',
 '/dev/disk/by-id/nvme-eui.e8238fa6bf540001001b555a49bfc681-part2']
nvme0n1p3 - ['/dev/disk/by-id/nvme-eui.e8238fa6bf540001001b555a49bfc681-part3',
'/dev/disk/by-id/nvme-WDS100T1X0E-00AFY0_2140GF374501-part3']
nvme0n1p4 - ['/dev/disk/by-id/nvme-WDS100T1X0E-00AFY0_2140GF374501-part4',
 '/dev/disk/by-id/nvme-eui.e8238fa6bf540001001b555a49bfc681-part4']
nvme0n1p5 - ['/dev/disk/by-id/nvme-WDS100T1X0E-00AFY0_2140GF374501-part5',
 '/dev/disk/by-id/nvme-eui.e8238fa6bf540001001b555a49bfc681-part5']
nvme0n1p6 - ['/dev/disk/by-id/nvme-WDS100T1X0E-00AFY0_2140GF374501-part6',
 '/dev/disk/by-id/nvme-eui.e8238fa6bf540001001b555a49bfc681-part6']
```

Return type
List[str]

`get_bylabel_path()`

Returns the *by-label* persistent path of the partition (see sample value in the example). The result could be empty if the filesystem does not have a label.

Example

An example about use of the function:

```
>>> from diskinfo import Disk
>>> disk = Disk("nvme0n1")
>>> plist = disk.get_partition_list()
>>> for item in plist:
...     print(item.get_name(), "-", item.get_bylabel_path())
...
nvme0n1p1 - /dev/disk/by-label/SYSTEM
nvme0n1p2 -
nvme0n1p3 - /dev/disk/by-label/Windows
nvme0n1p4 - /dev/disk/by-label/Recovery tools
nvme0n1p5 - /dev/disk/by-label/Debian
nvme0n1p6 - /dev/disk/by-label/Arch Linux
```

Return type
str

get_bypartlabel_path()

Returns the *by-partlabel* persistent path of the partition (see sample value in the example). The result could be empty if the partition does not have a label.

Note: This is a GTP partition specific value.

Example

An example about use of the function:

```
>>> from diskinfo import Disk
>>> disk = Disk("nvme0n1")
>>> plist = disk.get_partition_list()
>>> for item in plist:
...     print(item.get_name(), "-", item.get_bypartlabel_path())
...
nvme0n1p1 - /dev/disk/by-partlabel/EFI system partition
nvme0n1p2 - /dev/disk/by-partlabel/Microsoft reserved partition
nvme0n1p3 - /dev/disk/by-partlabel/Basic data partition
nvme0n1p4 - /dev/disk/by-partlabel/Basic data partition
nvme0n1p5 -
nvme0n1p6 -
```

Return type
str

get_bypartuuid_path()

Returns the *by-partuuid* persistent path of the partition (see sample values in the example).

Note: This is a GTP partition specific value.

Example

An example about use of the function:

```
>>> from diskinfo import Disk
>>> disk = Disk("nvme0n1")
>>> plist = disk.get_partition_list()
>>> for item in plist:
...     print(item.get_name(), "-", item.get_bypartuuid_path())
...
nvme0n1p1 - /dev/disk/by-partuuid/acb8374d-fb60-4cb0-8ac4-273417c6f847
nvme0n1p2 - /dev/disk/by-partuuid/59417232-6e42-4c03-b258-2d20ddb0486a
nvme0n1p3 - /dev/disk/by-partuuid/ec51c644-3ad7-44a5-9750-fc577a3d1ccf
nvme0n1p4 - /dev/disk/by-partuuid/9192cde2-b90d-4fd1-99ed-a3584f66c87c
nvme0n1p5 - /dev/disk/by-partuuid/6fd87857-265c-489c-8401-a47944c940f2
nvme0n1p6 - /dev/disk/by-partuuid/d5e53353-1943-4827-9b46-63459432f51c
```

Return type
str

`get_bypath_path()`

Returns the *by-path* persistent path of the partition (see sample value in the example).

Example

An example about use of the function:

```
>>> from diskinfo import Disk
>>> disk = Disk("nvme0n1")
>>> plist = disk.get_partition_list()
>>> for item in plist:
...     print(item.get_name(), "-", item.get_bypath_path())
...
nvme0n1p1 - /dev/disk/by-path/pci-0000:02:00.0-nvme-1-part1
nvme0n1p2 - /dev/disk/by-path/pci-0000:02:00.0-nvme-1-part2
nvme0n1p3 - /dev/disk/by-path/pci-0000:02:00.0-nvme-1-part3
nvme0n1p4 - /dev/disk/by-path/pci-0000:02:00.0-nvme-1-part4
nvme0n1p5 - /dev/disk/by-path/pci-0000:02:00.0-nvme-1-part5
nvme0n1p6 - /dev/disk/by-path/pci-0000:02:00.0-nvme-1-part6
```

Return type
str

`get_byuuid_path()`

Returns the *by-uuid* persistent path of the partition (see sample value in the example). The result could be empty if the partition does not have a file system.

Example

An example about use of the function:

```
>>> from diskinfo import Disk
>>> disk = Disk("nvme0n1")
>>> plist = disk.get_partition_list()
>>> for item in plist:
...     print(item.get_name(), "-", item.get_byuuid_path())
...
nvme0n1p1 - /dev/disk/by-uuid/6432-935A
nvme0n1p2 -
nvme0n1p3 - /dev/disk/by-uuid/0CA833E3A833CA4A
nvme0n1p4 - /dev/disk/by-uuid/784034274033EB10
nvme0n1p5 - /dev/disk/by-uuid/d54d33ea-d892-44d9-ae24-e3c6216d7a32
nvme0n1p6 - /dev/disk/by-uuid/a0b1c6e7-2541-4e89-93eb-898f6d544a1e
```

Return type
str

`get_fs_free_size()`

Returns the free size of the file system in 512-byte blocks. The result could be 0 if the partition does not contain a file system.

Example

An example about use of the function:

```
>>> from diskinfo import Disk
>>> disk = Disk("nvme0n1")
>>> plist = disk.get_partition_list()
>>> for item in plist:
...     print(item.get_name(), "-", item.get_fs_free_size())
...
nvme0n1p1 - 971968
nvme0n1p2 - 0
nvme0n1p3 - 214591944
nvme0n1p4 - 0
nvme0n1p5 - 141095712
nvme0n1p6 - 114470872
```

Return type
int

`get_fs_free_size_in_hrf(units=0)`

Returns the free size of the file system in human-readable form. The result could be 0 if the partition does not contain a file system.

Parameters
units (*int*) – unit system will be used for the calculation and in the result:

- 0 metric units (default)
- 1 IEC units
- 2 legacy units

Read more about units here.

Returns
size in human-readable form, proper unit

Return type
Tuple[float, str]

Example

An example about use of the function:

```
>>> from diskinfo import Disk
>>> disk = Disk("nvme0n1")
>>> plist = disk.get_partition_list()
>>> for item in plist:
...     s, u = size_in_hrf(item.get_fs_free_size()*512)
...     print(item.get_name(), "-", f"{s:.1f} {u}")
```

(continues on next page)

(continued from previous page)

```
...
nvme0n1p1 - 497.6 MB
nvme0n1p2 - 0.0 B
nvme0n1p3 - 109.9 GB
nvme0n1p4 - 0.0 B
nvme0n1p5 - 72.2 GB
nvme0n1p6 - 58.6 GB
```

get_fs_label()

Returns the label of the file system. The result could be empty if the file system does not have a label.

Example

An example about use of the function:

```
>>> from diskinfo import Disk
>>> disk = Disk("nvme0n1")
>>> plist = disk.get_partition_list()
>>> for item in plist:
...     print(item.get_name(), "-", item.get_fs_label())
...
nvme0n1p1 - SYSTEM
nvme0n1p2 -
nvme0n1p3 - Windows
nvme0n1p4 - Recovery tools
nvme0n1p5 - Debian
nvme0n1p6 - Arch Linux
```

Return type

str

get_fs_mounting_point()

Returns the mounting point of the file system. The result could be empty if the partition does not contain any file system, or it is not mounted.

Example

An example about use of the function:

```
>>> from diskinfo import Disk
>>> disk = Disk("nvme0n1")
>>> plist = disk.get_partition_list()
>>> for item in plist:
...     print(item.get_name(), "-", item.get_fs_mounting_point())
...
nvme0n1p1 - /boot/efi
nvme0n1p2 -
nvme0n1p3 - /mnt/win11
nvme0n1p4 -
```

(continues on next page)

(continued from previous page)

```
nvme0n1p5 - /
nvme0n1p6 - /mnt/arch
```

Return type
str

`get_fs_type()`

Returns the type of the file system. The result could be empty if the partition does not contain a file system.

Example

An example about use of the function:

```
>>> from diskinfo import Disk
>>> disk = Disk("nvme0n1")
>>> plist = disk.get_partition_list()
>>> for item in plist:
...     print(item.get_name(), "-", item.get_fs_type())
...
nvme0n1p1 - vfat
nvme0n1p2 -
nvme0n1p3 - ntfs
nvme0n1p4 - ntfs
nvme0n1p5 - ext4
nvme0n1p6 - ext4
```

Return type
str

`get_fs_usage()`

Returns the usage of the file system. The result could be empty if the partition does not contain a file system. Vlaid values are`filesystem` or *other* for special partitions (e.g. for a swap partition).

Example

An example about use of the function:

```
>>> from diskinfo import Disk
>>> disk = Disk("nvme0n1")
>>> plist = disk.get_partition_list()
>>> for item in plist:
...     print(item.get_name(), "-", item.get_fs_usage())
...
nvme0n1p1 - filesystem
nvme0n1p2 -
nvme0n1p3 - filesystem
nvme0n1p4 - filesystem
nvme0n1p5 - filesystem
nvme0n1p6 - filesystem
```

Return type
str

`get_fs_uuid()`

Returns the UUID of the file system. The result could be empty if the partition does not contain a file system.

Example

An example about use of the function:

```
>>> from diskinfo import Disk
>>> disk = Disk("nvme0n1")
>>> plist = disk.get_partition_list()
>>> for item in plist:
...     print(item.get_name(), "-", item.get_fs_uuid())
...
nvme0n1p1 - 6432-935A
nvme0n1p2 -
nvme0n1p3 - 0CA833E3A833CA4A
nvme0n1p4 - 784034274033EB10
nvme0n1p5 - d54d33ea-d892-44d9-ae24-e3c6216d7a32
nvme0n1p6 - a0b1c6e7-2541-4e89-93eb-898f6d544a1e
```

Return type
str

`get_fs_version()`

Returns the version of the file system. The result could be empty if the partition does not contain a file system or does not have a version.

Example

An example about use of the function:

```
>>> from diskinfo import Disk
>>> disk = Disk("nvme0n1")
>>> plist = disk.get_partition_list()
>>> for item in plist:
...     print(item.get_name(), "-", item.get_fs_version())
...
nvme0n1p1 - FAT32
nvme0n1p2 -
nvme0n1p3 -
nvme0n1p4 -
nvme0n1p5 - 1.0
nvme0n1p6 - 1.0
```

Return type
str

get_name()

Returns the name of the partition (e.g. *sda1* or *nvme0n1p1*).

Example

An example about use of the function:

```
>>> from diskinfo import Disk
>>> disk = Disk("nvme0n1")
>>> plist = disk.get_partition_list()
>>> for item in plist:
...     print(item.get_name())
...
nvme0n1p1
nvme0n1p2
nvme0n1p3
nvme0n1p4
nvme0n1p5
nvme0n1p6
```

Return type

str

get_part_device_id()

Returns the device id of the partition in *major:minor* form.

Example

An example about use of the function:

```
>>> from diskinfo import Disk
>>> disk = Disk("nvme0n1")
>>> plist = disk.get_partition_list()
>>> for item in plist:
...     print(item.get_name(), "-", item.get_part_device_id())
...
nvme0n1p1 - 259:1
nvme0n1p2 - 259:2
nvme0n1p3 - 259:3
nvme0n1p4 - 259:4
nvme0n1p5 - 259:5
nvme0n1p6 - 259:6
```

Return type

str

get_part_label()

Returns the label of the partition. The result could be empty if the partition does not have a label.

Note: This is a GTP partition specific value.

Example

An example about use of the function:

```
>>> from diskinfo import Disk
>>> disk = Disk("nvme0n1")
>>> plist = disk.get_partition_list()
>>> for item in plist:
...     print(item.get_name(), "-", item.get_part_label())
...
nvme0n1p1 - EFI system partition
nvme0n1p2 - Microsoft reserved partition
nvme0n1p3 - Basic data partition
nvme0n1p4 - Basic data partition
nvme0n1p5 -
nvme0n1p6 -
```

Return type
str

get_part_number()

Returns the number of the partition.

Example

An example about use of the function:

```
>>> from diskinfo import Disk
>>> disk = Disk("nvme0n1")
>>> plist = disk.get_partition_list()
>>> for item in plist:
...     print(item.get_name(), "-", item.get_part_number())
...
nvme0n1p1 - 1
nvme0n1p2 - 2
nvme0n1p3 - 3
nvme0n1p4 - 4
nvme0n1p5 - 5
nvme0n1p6 - 6
```

Return type
int

get_part_offset()

Returns the starting offset of the partition in 512-byte blocks.

Example

An example about use of the function:

```
>>> from diskinfo import Disk
>>> disk = Disk("nvme0n1")
>>> plist = disk.get_partition_list()
>>> for item in plist:
...     print(item.get_name(), "-", item.get_part_offset())
...
nvme0n1p1 - 2048
nvme0n1p2 - 1050624
nvme0n1p3 - 1312768
nvme0n1p4 - 838125568
nvme0n1p5 - 840173568
nvme0n1p6 - 1035485184
```

Return type
int

get_part_scheme()

Returns the scheme of the partition. The result could be *gtp* or *mbr*.

Example

An example about use of the function:

```
>>> from diskinfo import Disk
>>> disk = Disk("nvme0n1")
>>> plist = disk.get_partition_list()
>>> for item in plist:
...     print(item.get_name(), "-", item.get_part_scheme())
...
nvme0n1p1 - gpt
nvme0n1p2 - gpt
nvme0n1p3 - gpt
nvme0n1p4 - gpt
nvme0n1p5 - gpt
nvme0n1p6 - gpt
```

Return type
str

get_part_size()

Returns the size of the partition in 512-byte blocks.

Example

An example about use of the function:

```
>>> from diskinfo import *
>>> disk = Disk("nvme0n1")
>>> plist = disk.get_partition_list()
>>> for item in plist:
...     print(item.get_name(), "-", item.get_part_size())
...
nvme0n1p1 - 1048576
nvme0n1p2 - 262144
nvme0n1p3 - 836812800
nvme0n1p4 - 2048000
nvme0n1p5 - 195311616
nvme0n1p6 - 209715200
```

Return type
int

get_part_size_in_hrf(*units=0*)

Returns the size of the partition in human-readable form.

Parameters
units (*int*) – unit system will be used for the calculation and in the result:

- 0 metric units (default)
- 1 IEC units
- 2 legacy units

Read more about units here.

Returns
size in human-readable form, proper unit

Return type
Tuple[float, str]

Example

An example about use of the function:

```
>>> from diskinfo import *
>>> disk = Disk("nvme0n1")
>>> plist = disk.get_partition_list()
>>> for item in plist:
...     s, u = item.get_part_size_in_hrf()
...     print(item.get_name(), "-", "{s:.1f} {u}")
...
nvme0n1p1 - 536.9 MB
nvme0n1p2 - 134.2 MB
nvme0n1p3 - 428.4 GB
nvme0n1p4 - 1.0 GB
```

(continues on next page)

(continued from previous page)

```
nvme0n1p5 - 100.0 GB
nvme0n1p6 - 107.4 GB
```

get_part_type()

Returns the UUID of the partition type. See available GTP partition types listed on wikipedia.

Note: This is a GTP partition specific value.

Example

An example about use of the function:

```
>>> from diskinfo import Disk
>>> disk = Disk("nvme0n1")
>>> plist = disk.get_partition_list()
>>> for item in plist:
...     print(item.get_name(), "-", item.get_part_type())
...
nvme0n1p1 - c12a7328-f81f-11d2-ba4b-00a0c93ec93b
nvme0n1p2 - e3c9e316-0b5c-4db8-817d-f92df00215ae
nvme0n1p3 - ebd0a0a2-b9e5-4433-87c0-68b6b72699c7
nvme0n1p4 - de94bba4-06d1-4d40-a16a-bfd50179d6ac
nvme0n1p5 - 0fc63daf-8483-4772-8e79-3d69d8477de4
nvme0n1p6 - 0fc63daf-8483-4772-8e79-3d69d8477de4
```

Return type
str

get_part_uuid()

Returns the UUID of the partition.

Note: This is a GTP partition specific value.

Example

An example about use of the function:

```
>>> from diskinfo import Disk
>>> disk = Disk("nvme0n1")
>>> plist = disk.get_partition_list()
>>> for item in plist:
...     print(item.get_name(), "-", item.get_part_uuid())
...
nvme0n1p1 - acb8374d-fb60-4cb0-8ac4-273417c6f847
nvme0n1p2 - 59417232-6e42-4c03-b258-2d20ddb0486a
nvme0n1p3 - ec51c644-3ad7-44a5-9750-fc577a3d1ccf
nvme0n1p4 - 9192cde2-b90d-4fd1-99ed-a3584f66c87c
```

(continues on next page)

(continued from previous page)

```
nvme0n1p5 - 6fd87857-265c-489c-8401-a47944c940f2
nvme0n1p6 - d5e53353-1943-4827-9b46-63459432f51c
```

Return type
str

`get_path()`

Returns the path of the partition (e.g. */dev/sda1* or */dev/nvme0n1p1*).

Note: This is not a persistent path!

Example

An example about use of the function:

```
>>> from diskinfo import Disk
>>> disk = Disk("nvme0n1")
>>> plist = disk.get_partition_list()
>>> for item in plist:
...     print(item.get_path())
...
/dev/nvme0n1p1
/dev/nvme0n1p2
/dev/nvme0n1p3
/dev/nvme0n1p4
/dev/nvme0n1p5
/dev/nvme0n1p6
```

Return type
str

2.7 *DiskInfo* class

`class diskinfo.DiskInfo`

This class implements disk exploration functionality. At class initialization time all existing disks will be explored automatically (empty loop devices will be skipped). In a next step, *get_disk_number()* method can return the number of identified disk and *get_disk_list()* method can return the list of the identified disks. In both cases disk type filters can be applied to get only a subset of the discovered disks. The filters are set of *DiskType* values.

Operator `in` is also implemented for this class. Caller can check if a *Disk* class instance can be found on the list of the identified disks.

Example

A code example about the basic use of the class and the use of the `in` operator.

```
>>> from diskinfo import Disk, DiskType, DiskInfo
>>> di = DiskInfo()
>>> n = di.get_disk_number(included={DiskType.SSD}, excluded={DiskType.HDD})
>>> print(f"Number of SSDs: {n}")
Number of SSDs: 3
>>> d = Disk("sda")
>>> print(d in di)
True
```

get_disk_list(*included=None*, *excluded=None*, *sorting=False*, *rev_order=False*)

Returns the list of identified disks. The caller can specify inclusive and exclusive filters for disk types. If no filters are specified the default behavior is to include all disk types and to exclude nothing. The list can be sorted based on the disk *name* in alphabetical order. Caller can also request sorting in reverse order.

Parameters

- **included** (*set*) – filter set for included disk types
- **excluded** (*set*) – filter set for excluded disk types
- **sorting** (*bool*) – disk list will be sorted based on *name* string
- **rev_order** (*bool*) – sorting in reverse order

Returns
list of the (filtered) disks

Return type
List[*Disk*]

Raises
ValueError – if the same disk type is on both included and excluded filter sets

Example

A code example about using filters and sorting: it will list the device path of the sorted list of the HDDs:

```
>>> from diskinfo import DiskType, DiskInfo
>>> di = DiskInfo()
>>> disks = di.get_disk_list(included={DiskType.HDD}, sorting=True)
>>> for d in disks:
...     print(d.get_path())
...
/dev/sda
/dev/sdb
/dev/sdc
```

get_disk_number(*included=None*, *excluded=None*)

Returns the number of the disks. The caller can specify inclusive and exclusive filters for disk types. If no filters are specified then the default behavior is to include all disk types and to exclude nothing.

Parameters

- **included** (*set*) – filter set for included disk types

- **excluded** (*set*) – filter set for excluded disk types

Returns
number of the (filtered) disks

Return type
int

Raises
ValueError – if the same disk type is on both included and excluded filter sets

Example

A code example about using filters: it counts the number of SSDs excluding NVME disks.

```
>>> from diskinfo import DiskType, DiskInfo
>>> di = DiskInfo()
>>> n = di.get_disk_number(included={DiskType.SSD}, excluded={DiskType.HDD})
>>> print(f"Number of SSDs: {n}")
Number of SSDs: 3
```

2.8 Utility functions

diskinfo.**_read_file**(*path*, *encoding='utf-8'*)

Reads the text content of the specified file. The function will hide IOError and FileNotFound exceptions during the file operations. The result bytes will be read with the specified encoding and stripped.

Parameters

- **path** (*str*) – file path
- **encoding** (*str*) – encoding (default is *utf-8*)

Returns
file content text

Return type
str

Example

An example aboout the use of the function:

```
>>> from diskinfo import *
>>> _read_file("/sys/block/sda/dev")
'8:0'
```

diskinfo.**_read_udev_property**(*path*, *udev_property*, *encoding='utf-8'*)

Reads a property from an *udev* data file. The function will hide IOError and py:obj:*FileNotFound* exceptions during the file operations. The result string will be decoded and stripped.

Parameters

- **path** (*str*) – path of the udev data file (e.g. */run/udev/data/b8:0*)
- **udev_property** (*str*) – udev property string

- **encoding** (*str*) – encoding (default is *utf-8*)

Returns
udev property value

Return type
str

Raises
ValueError – in case of empty input parameters

Example

An example about the use of the function:

```
>>> from diskinfo import *
>>> _read_udev_property("/run/udev/data/b259:0", "ID_MODEL=")
'WDS100T1X0E-00AFY0'
```

diskinfo.**_read_udev_path**(*path*, *path_type*, *encoding='utf-8'*)

Reads one or more path elements from an udev data file. It will hide IOError and FileNotFound exceptions during the file operations. The result path elements will be decoded and stripped.

Parameters

- **path** (*str*) – path of the udev data file (e.g. */run/udev/data/b8:0*)
- **path_type** (*int*) – type of the path to find/load from udev data file. Valid values are:
  - 0 *by-id* path
  - 1 *by-path* path
  - 2 *by-partuuid* path
  - 3 *by-partlabel* path
  - 4 *by-label* path
  - 5 *by-uuid* path
- **encoding** (*str*) – encoding (default is *utf-8*)

Returns
path elements

Return type
List[str]

Raises
ValueError – in case of empty or invalid input parameters

Example

An example about the use of the function:

```
>>> from diskinfo import *
>>> _read_udev_path("/run/udev/data/b259:0", 1)
['/dev/disk/by-path/pci-0000:02:00.0-nvme-1']
```

diskinfo.**size_in_hrf**(*size_value*, *units=0*)

Returns the size in a human-readable form.

Parameters

- **size_value** (*int*) – number of bytes
- **units** (*int*) – unit system will be used for the calculation and in the result:
  - 0 metric units (default)
  - 1 IEC units
  - 2 legacy units

  Read more about units here.

Returns
size in human-readable form, proper unit

Return type
Tuple[float, str]

Raises
ValueError – in case of invalid input parameters (negative size, invalid units)

Example

An example about the use of the function:

```
>>> from diskinfo import *
>>> size = 12839709879873
>>> s, u = size_in_hrf()
>>> print(f"{s:.1f} {u}")
12.8 TB
>>> s, u = size_in_hrf(size, units=1)
>>> print(f"{s:.1f} {u}")
11.7 TiB
```

diskinfo.**time_in_hrf**(*time*, *unit=0*, *short_format=False*)

Returns the amount of time in a human-readable form.

Parameters

- **time** (*int*) – time value
- **unit** (*int*) – unit of the input time value
  - 0 seconds
  - 1 minutes
  - 2 hours

  - 3 days
  - 4 years
- **short_format** (*bool*) – result unit in short format (e.g. *min* instead of *minute*)

Returns
time in human-readable form, proper unit

Return type
Tuple[float, str]

Raises
ValueError – in case of invalid input parameters (negative time, invalid unit)

Example

An example about the use of the function:

```
>>> from diskinfo import *
>>> hours = 6517
>>> t, u = time_in_hrf(hours, unit=2)
>>> print(f"{t:.1f} {u}")
271.5 day
>>> days = 2401
>>> t, u = time_in_hrf(hours, unit=3, short_format=True)
>>> print(f"{t:.1f} {u}")
6.6 yr
```

CHAPTER

THREE

INDICES AND TABLES

- genindex
- search

INDEX

Symbols

A

C

D

E

F

G