infi.storagemodel Documentation

Release 0.4.34

Guy Rozendorn

Nov 26, 2020

Contents

CHAPTER 1

Overview

The purpose of this module is to provide a unified, abstracted, storage model for the various operating systems.

1.1 Getting Started

To get started, first you need to get storage model:

```
>>> from infi.storagemodel import get_storage_model
>>> model = get_storage_model()
```

The model object is a global instance, so every time to call *get_storage_model()* you get the same instance. Thus function returns an instance of *StorageModel*.

The storage model is layer-based. The first layer is the *SCSI Layer*. Access to it is done by:

```
>>> scsi = model.get_scsi()
```

The function is cached, so calling it again returns the same instance.

Now lets see what objects this gives us:

```
>>> block_devices = scsi.get_all_scsi_block_devices()
>>> target_controllers = scsi.get_all_storage_controller_devices()
```

These two functions, return all the 'seen' disks and controllers by the operating systems. You can also ask for a specific device:

```
>>> device = scsi.find_scsi_block_device_by_block_access_path("/dev/sda") # on Linux
>>> device = scsi.find_scsi_block_device_by_scsi_access_path("/dev/sg0") # on Linux
```

and on Windows:

```
>>> device = scsi.find_scsi_block_device_by_scsi_access_path(r"\\?
↪\GLOBALROOT\Device\0000001a")
```

I hope you get it by now, everything is cached within the model, so if "/dev/sda" and "/dev/sg0" are actually the same device, you'll get the same instance.

Also, you can get a device by its SCSI address:

```
>>> from infi.dtypes.hctl import HCTL
>>> device.find_scsi_block_device_by_hctl(HCTL(1,0,0,1))
```

These methods return either *SCSIStorageController* or *SCSIBlockDevice*, or lists of them. Check their documentation to see what information they hold.

Now, lets get the multipath devices. In mose cases, we'd want to work with the native multipath driver, *NativeMultipathModel*:

```
>>> mpio = model.get_native_multipath()
>>> devices = mpio.get_all_multipath_devices()
```

Usually, you'd also want to differ between the multipath disks and the non-multipath disks:

```
>>> block_devices = scsi.get_all_scsi_block_devices()
>>> mp_disks = mpio.get_all_multipath_devices()
>>> non_mp_disks = mpio.filter_non_multipath_scsi_block_devices(block_devices)
```

Also, if you want disks of a specific product:

```
>>> from infi.storagemodel.vendor.infinidat.infinibox import vid_pid
>>> infinidat_mp_disks = mpio.filter_vendor_specific_devices(mp_disks, vid_pid)
>>> infinidat_non_mp_disks  = mpio.filter_vendor_specific_devices(block_devices, vid_
↪pid)
```

The `MultipathDevice` provides cross-platform abstraction of *Path* and their *LoadBalancePolicy*. The platform-specific implementation translates the configurartion into the supported *Load Balance Policies*.

That's all for now.

Oh, check the *Scanning for changes in Storage* tutorial.

1.2 Model

class StorageModel

StorageModel provides a layered view of the storage stack. The layers currently exposed by the model are:

- SCSI
- Multipath
- Disks
- Mounts
- Utils

All layers are fetched lazily and cached inside the model. When you think that the cache no longer up-to-date, you can clear it using the refresh() method

1.2.1 Getting an instance

get_storage_model()

returns a global instance of a *infi.storagemodel.base.StorageModel*.

1.3 Layers

1.3.1 SCSI Layer

class SCSIModel

1.3.2 Multipath Layer

class MultipathFrameworkModel

class NativeMultipathModel
Bases: *infi.storagemodel.base.multipath.MultipathFrameworkModel*

1.3.3 Disk Layer

class DiskModel

1.4 SCSI Objects

1.4.1 SCSI Device

class SCSIDevice

asi_context()
Returns a context for *infi.asi*

get_connectivity()
Returns a *infi.storagemodel.connectivity.FCConnectivity* instance.

get_display_name()
Returns a friendly device name. In Windows, it's PHYSICALDRIVE%d, in linux, its sdX.

get_hctl()
Returns a *infi.dtypes.hctl.HCTL* object

get_scsi_access_path()
Returns a string path for the device

- In Windows, it's something under globalroot like block_device_path
- In linux, it's /dev/sgX

get_scsi_ata_information()
Returns the SCSI ata information of the device as a dict of dicts for SATL and identify device

get_scsi_inquiry_pages()
Returns an immutable dict-like object of available inquiry pages from this device. For example:

```
>>> device.get_scsi_inquiry_pages()[0x80].product_serial_number
```

get_scsi_product_id()
Returns the stripped T10 product identifier string, as give in SCSI Standard Inquiry

get_scsi_revision()
Returns the stripped T10 revision string, as give in SCSI Standard Inquiry

get_scsi_serial_number()
Returns the SCSI serial string of the device or an empty string (“”) if not available

get_scsi_standard_inquiry()
Returns the standard inquiry data

get_scsi_test_unit_ready()
Returns True if the device is ready, False if got NOT_READY check condition

get_scsi_vendor_id()
Returns the stripped T10 vendor identifier string, as give in SCSI Standard Inquiry

get_scsi_vendor_id_or_unknown_on_error()
Returns (‘<unknown>’, ‘<unknown>’) on unexpected error instead of raising exception

get_scsi_vid_pid()
Returns a tuple of the vendor_id and product_id

get_scsi_vid_pid_rev()
Returns a tuple of the vendor_id, product_id and revision

1.4.2 SCSI Block Device

class SCSIBlockDevice
Bases: *infi.storagemodel.base.scsi.SCSIDevice*

1.4.3 SCSI Storage Controller Device

class SCSIStorageController
Bases: *infi.storagemodel.base.scsi.SCSIDevice*

1.5 Multipath Objects

1.5.1 Multipath Device

class MultipathBlockDevice

asi_context()
Returns an infi.asi context

get_block_access_path()
Returns a path for the device

get_disk_drive()
Returns a *infi.storagemodel.base.disk.DiskDrive* instance.

Raises *infi.storagemodel.base.disk.NoSuchDisk* if not found.

get_display_name()
Returns a string represtation for the device

get_paths()
Returns a list of *infi.storagemodel.base.multipath.Path* instances

get_policy()
Returns an instance of *infi.storagemodel.base.multipath.LoadBalancePolicy*

get_scsi_ata_information()
Returns the SCSI ata information of the device as a dict of dicts for SATL and identify device

get_scsi_inquiry_pages()
Returns an immutable dict-like object of available inquiry pages from this device. For example:

```
>>> device.get_scsi_inquiry_pages()[0x80].product_serial_number
```

get_scsi_product_id()
Returns the stripped T10 product identifier string, as give in SCSI Standard Inquiry

get_scsi_revision()
Returns the stripped T10 revision string, as give in SCSI Standard Inquiry

get_scsi_serial_number()
Returns the SCSI serial string of the device or an empty string ("") if not available

get_scsi_standard_inquiry()
Returns the standard inquiry data

get_scsi_test_unit_ready()
Returns True if the device is ready, False if got NOT_READY check condition

get_scsi_vendor_id()
Returns the stripped T10 vendor identifier string, as give in SCSI Standard Inquiry

get_scsi_vendor_id_or_unknown_on_error()
Returns ('<unknown>', '<unknown>') on unexpected error instead of raising exception

get_scsi_vid_pid()
Returns a tuple of the vendor_id and product_id

get_scsi_vid_pid_rev()
Returns a tuple of the vendor_id, product_id and revision

get_vendor()
Returns a get_vendor-specific implementation from the factory based on the device's SCSI vid and pid

1.5.2 Multipath Path

class Path

get_alua_state()
Returns the ALUA (Asymmetric Logical Unit Access) value

get_connectivity()
Returns an *infi.storagemodel.connectivity.FCConnectivity* instance.

get_display_name()
Returns the path name (currently the same as *get_path_id*).

get_hctl()
Returns a *infi.dtypes.hctl.HCTL* instance

get_io_statistics()
Returns a *infi.storagemodel.base.multipath.PathStatistics* instance

get_path_id()
Returns depending on the operating system:

- sdX on linux
- PathId on Windows

get_state()
Returns either "up" or "down".

1.5.3 Load Balance Policies

class LoadBalancePolicy
Base class of all available load balancing policies

class FailoverOnly(*active_path_id*)
Bases: *infi.storagemodel.base.multipath.LoadBalancePolicy*

Load balancing policy where the alternative paths are used only in case the active path fails.

class RoundRobin
Bases: *infi.storagemodel.base.multipath.LoadBalancePolicy*

Load balancing policy where all paths are used in a balanced way.

class RoundRobinWithSubset(*active_path_ids*)
Bases: *infi.storagemodel.base.multipath.LoadBalancePolicy*

Load balancing policy where a subset of the paths are used in a balanced way.

class RoundRobinWithTPGSSubset(*active_path_ids*)
Bases: *infi.storagemodel.base.multipath.RoundRobinWithSubset*

Load balancing policy where only paths that are active/optimized according to TPGS are used

class RoundRobinWithExplicitSubset(*active_path_ids*)
Bases: *infi.storagemodel.base.multipath.RoundRobinWithSubset*

Load balancing policy where an explicitly-given subset of the paths are used

class WeightedPaths(*weights*)
Bases: *infi.storagemodel.base.multipath.LoadBalancePolicy*

Load balancing policy that assigns a weight to each path. The weight indicates the relative priority of a given path. The larger the number, the lower ranked the priority.

class LeastBlocks
Bases: *infi.storagemodel.base.multipath.LoadBalancePolicy*

Load balancing policy that sends I/O down the path with the least number of data blocks currently being processed

class LeastQueueDepth
Bases: *infi.storagemodel.base.multipath.LoadBalancePolicy*

Load balancing policy that sends I/O down the path with the fewest currently outstanding I/O requests.

1.6 Disk Objects

1.6.1 Disk Drive

class DiskDrive

create_guid_partition_table(*alignment_in_bytes=None*)
Creates a GUID partition table and returns it (*infi.storagemodel.base.partition.GUIDPartitionTable*)

create_mbr_partition_table(*alignment_in_bytes=None*)
Creates an MBR partition table and returns it (*infi.storagemodel.base.partition.MBRPartitionTable*)

delete_partition_table()
Deletes the partition table from the disk

get_block_access_path()
Returns the block access path for the disk

get_partition_table()
Returns the disk's partition table (*infi.storagemodel.base.partition.PartitionTable*). Raises *ValueError* if there is no partition table on disk.

get_size_in_bytes()
Returns the disk size in bytes

get_storage_device()
Returns the storage device that is represented by this disk drive - either a *infi.storagemodel.base.multipath.MultipathDevice* or *infi.storagemodel.base.scsi.SCSIBlockDevice*

is_empty()
Returns True if the disk has no partition table.

1.7 Partition Objects

1.7.1 Partition Table

class PartitionTable
Base class for representing patition tables

create_partition_for_whole_table(*file_system_object*, *alignment_in_bytes=None*)
Creates a partition that fills the entire drive. The partition is set to use the given filesystem, but does not get formatted by this method.

Changes are written immediately on disk. The partition table is re-read and the cache for the current object is cleared.

Returns a *infi.storagemodel.base.partition.Partition* object

classmethod create_partition_table(*disk_drive*, *alignment_in_bytes=None*)
Creates a partition table of the requested class on the given *infi.storagemodel.base.disk.DiskDrive*. No partitions are created inside the partition table.

Changes are written immediately on disk. The partition table is re-read and the cache for the current object is cleared.

Returns The newly created *infi.storagemodel.base.partition.Partition* object

get_disk_drive()
Returns the *infi.storagemodel.base.disk.DiskDrive* that holds the partition

get_partitions()
Returns a list of *infi.storagemodel.base.partition.Partition* objects inside the partition table

is_empty()
Returns True if there are no partitions in the partition table

Master Boot Record

class MBRPartitionTable
Represents a Master Boot Record partition table

create_partition_for_whole_table(*file_system_object*, *alignment_in_bytes=None*)
Creates a partition that fills the entire drive. The partition is set to use the given filesystem, but does not get formatted by this method.

Changes are written immediately on disk. The partition table is re-read and the cache for the current object is cleared.

Returns a *infi.storagemodel.base.partition.Partition* object

classmethod create_partition_table(*disk_drive*, *alignment_in_bytes=None*)
Creates a partition table of the requested class on the given *infi.storagemodel.base.disk.DiskDrive*. No partitions are created inside the partition table.

Changes are written immediately on disk. The partition table is re-read and the cache for the current object is cleared.

Returns The newly created *infi.storagemodel.base.partition.Partition* object

get_disk_drive()
Returns the *infi.storagemodel.base.disk.DiskDrive* that holds the partition

get_partitions()
Returns a list of *infi.storagemodel.base.partition.Partition* objects inside the partition table

is_empty()
Returns True if there are no partitions in the partition table

1.8 Mount Objects

1.8.1 Mount

class Mount
Represents a non-persistent mount in the operating system

get_block_access_path()
Returns the block access path of the device to be mounted

get_filesystem()
Returns the *infi.storagemodel.base.filesystem.FileSystem* object that requested to be mounted

get_mount_options()
Returns filesystem-specific mount options

get_mount_point()
Returns the mount point

Persistent Mount

class PersistentMount
Represents a persistent mount in the operating system

get_block_access_path()
Returns the block access path of the device to be mounted

get_filesystem()
Returns the *infi.storagemodel.base.filesystem.FileSystem* object that requested to be mounted

get_mount_options()
Returns filesystem-specific mount options

get_mount_point()
Returns the mount point

1.9 File System Objects

1.9.1 Filesystem

class FileSystem
Represents a Filesystem that can be formatted and mounted

format(*block_device*, **args*, ***kwargs*)
Formats the device with this filesystem.

block_device: either a *infi.storagemodel.base.scsi.SCSIBlockDevice*, *infi.storagemodel.base.multipath.MultipathDevice* or *infi.storagemodel.base.partition.Partition*

Raises *infi.storagemodel.errors.StorageModelError* if the format has failed

get_label(*block_access_path*)
Returns the block device label, or an empty string if there's no label.

Raises *infi.storagemodel.errors.LabelNotSupported* if operation not supported by the filesystem

get_name()
Returns the string name of the filesystem

mount(*block_access_path*, *mount_point*, *mount_options_dict={}*)
Mounts a device to the mount point, with the given options dictionary.

block_device_path: the block access path of the storage device

mount_point: the path to the mount point

mount_options_dict: filesystem-specific mount options

Raises *infi.storagemodel.errors.MountPointDoesNotExist* if the mount point does not exist

Raises *infi.storagemodel.errors.MountPointInUse* if the mount point is in use by another mount

Raises *infi.storagemodel.errors.AlreadyMounted* if the device is already mounted

Returns a *infi.storagemodel.mount.Mount* object

resize(*size_in_bytes*)
Resize a filesystem. On platforms where resizing isn't neccessary, this method does nothing (e.g. Windows)

set_label(*block_access_path*, *label*)
Sets a filesystem label on the specific block device.

Raises *infi.storagemodel.errors.InvalidLabel* if the label is too long

Raises *infi.storagemodel.errors.LabelNotSupported* if not supported by the filesystem

unmount(*block_access_path*, *mount_point*)
Unmount the filesystem from the mount point.

mount_point: path to the mount point

Raises *infi.storagemodel.errors.NotMounted* if the mount point argument is not a mounted path

1.10 Connectivity

1.10.1 Fiber Channel

class FCConnectivity(*device*, *local_port*, *remote_port*)
Fibre Channel Connectivity Information

1.11 Vendor Information

1.11.1 Infinidat

class InfiniBoxInquiryMixin

class InfiniBoxVolumeMixin

class SophisticatedMixin

class InfinidatNAA(*data*)

class InfinidatFiberChannelPort(*relative_target_port_identifer*, *target_port_group*)

1.12 Scanning for changes in Storage

Doing a rescan on the SCSI bus is an expensive operation.

Usually, issuing a rescan on an operating system does not report on comlpetion, and it takes some time for the upper layers to finish their work.

Since rescan is usually done when there's a change in connectivity or mapping, this moddule provides an interfaces that blocks until the condition you're expending is True.

The function `rescan_and_wait_for()` accepts a predicate and blocks until the predicate returns True, or timeout is raised.

Let's see some examples.

1.12.1 Examples

Waiting for a new device

The most common use case is that a volume has been mapped, and you want to wait for it to appear.

For that we have the *DiskExists* predicate:

class DiskExists(*scsi_serial_number*)
Returns True if a disk was discovered with the given scsi_serial_number

Waiting for a new disk is straight-forward:

```
>>> from infi.storagemodel import get_storage_model()
>>> from infi.storagemodel.predicates import DiskExists
>>> get_storage_model().rescan_and_wait_for(DiskExists("123456"))
```

This predicate works for both multipath disks and non-multipath disks

Waiting for a device to be gone

If you want to rescan after unmapping a volume, use the *DiskNotExists* predicate:

class DiskNotExists(*scsi_serial_number*)
Returns True if a disk with the given scsi_serial_number has gone away

Fiber Channel mappings

If you performed a connectivity change, you can wait for it to happen.

```
>>> from infi.storagemodel.predicates import FiberChannelMappingExists
>>> predicate = FiberChannelMappingExists("01020304060708", "0a:0b:0c:0d:0e:0f:0g:0h",
↪ lun_number=0)
>>> get_storage_model().rescan_and_wait_for(predicate)
```

The complete description of the predicates for this use case:

class FiberChannelMappingExists(*initiator_wwn*, *target_wwn*, *lun_number*)
Returns True if a lun mapping was discovered

class FiberChannelMappingNotExists(*initiator_wwn*, *target_wwn*, *lun_number*)
Returns True if a lun un-mapping was discovered

The wwn argument can take any WWN format you can think of (lower-case, upper-case, with "-"/":" separators or not).

Waiting for several conditions

You can also wait for several conditions together:

```
>>> from infi.storagemodel.predicates import DiskExists, DiskNotExists, PredicateList
>>> get_storage_model().rescan_and_wait_for([DiskExists("123"), DiskNotExists("456")])
```

1.13 Errors and Exceptions

Python Module Index

i

Index

I

L

M

N

P

R

S

U

W