NAME

Sys::Virt::StoragePool − Represent & manage a libvirt storage pool

DESCRIPTION

The `Sys::Virt::StoragePool` module represents a storage pool managed by libvirt. There are a variety of storage pool implementations for LVM, Local directories/filesystems, network filesystems, disk partitioning, iSCSI, and SCSI.

METHODS

my `$uuid` = `$pool`−>**get_uuid()**

Returns a 16 byte long string containing the raw globally unique identifier (UUID) for the storage pool.

my `$uuid` = `$pool`−>**get_uuid_string()**

Returns a printable string representation of the raw UUID, in the format 'XXXXXXXX−XXXX−XXXX−XXXX−XXXXXXXXXXXX'.

my `$name` = `$pool`−>**get_name()**

Returns a string with a locally unique name of the storage pool

`$pool`−>**is_active()**

Returns a true value if the storage pool is currently running

`$pool`−>**is_persistent()**

Returns a true value if the storage pool has a persistent configuration file defined

my `$xml` = `$pool`−>**get_xml_description()**

Returns an XML document containing a complete description of the storage pool's configuration

`$pool`−>**create()**

Start a storage pool whose configuration was previously defined using the `define_storage_pool` method in Sys::Virt.

`$pool`−>**undefine()**

Remove the configuration associated with a storage pool previously defined with the `define_storage pool`method in Sys::V irt. If the storage pool is running, you probably want to use the `destroy` method instead.

`$pool`−>**destroy()**

Immediately stop the storage pool.

`$flag` = `$pool`−>**get_autostart()**;

Return a true value if the storage pool is configured to automatically start upon boot. Return false, otherwise

`$pool`−>set_autostart($flag)

Set the state of the autostart flag, which determines whether the storage pool will automatically start upon boot of the host OS

`$pool`−>refresh([$flags]);

Refresh the storage pool state. Typically this will rescan the list of storage volumes. The `$flags` parameter is currently unused and if omitted defaults to zero.

`$pool`−>build([$flags]);

Construct the storage pool if it does not exist. As an example, for a disk based storage pool this would ensure a partition table exists. The`$flags` parameter allo ws control over the build operation and if omitted defaults to zero.

`$pool`−>delete([$flags]);

Delete the storage pool. The `$flags` parameter allows the data to be optionally wiped during delete and if omitted defaults to zero.

`$info` = `$pool`−>**get_info()**

Retrieve information about the current storage pool state. The returned hash reference has the following keys

state
The current status of the storage pool. See constants later.

capacity
The total logical size of the storage pool

allocation
The current physical allocation of the storage pool

available
The available space for creation of new volumes. This may be less than the difference between capacity & allocation if there are sizing / metadata constraints for volumes

my `$nnames` = `$pool`->**num_of_storage_volumes()**
Return the number of running volumes in this storage pool. The value can be used as the `maxnames` parameter to `list_storage_vol_names`.

my `@volNames` = `$pool`->list_storage_vol_names($maxnames)
Return a list of all volume names in this storage pool. The names can be used with the `get_volume_by_name` method.

my `@vols` = `$pool`->**list_volumes()**
Return a list of all volumes in the storage pool. The elements in the returned list are instances of the Sys::Virt::StorageVol class. This method requires O(n) RPC calls, so the `list_all_volumes` method is recommended as a more efficient alternative.

my `@volumes` = `$pool`->list_all_volumes($flags)
Return a list of all storage volumes associated with this pool. The elements in the returned list are instances of the Sys::Virt::StorageVol class. The `$flags` parameter can be used to filter the list of return storage volumes.

my `$vol` = `$pool`->get_volume_by_name($name)
Return the volume with a name of `$name`. The returned object is an instance of the Sys::Virt::StorageVol class.

my `$vol` = `$pool`->create_volume($xml)
Create a new volume based on the XML description passed into the `$xml` parameter. The returned object is an instance of the Sys::Virt::StorageVol class. If the optional `clonevol` is provided, data will be copied from that source volume

my `$vol` = `$pool`->clone_volume($xml, `$clonevol`);
Create a new volume based on the XML description passed into the `$xml` parameter. The returned object is an instance of the Sys::Virt::StorageVol class. The new volume will be populated with data from the specified clone source volume.

CONSTANTS

The following sets of constants may be useful in dealing with some of the methods in this package

STORAGE POOL DEFINE

The following constants can be used to control the behaviour of storage pool define operations

Sys::Virt::StoragePool::DEFINE_VALIDATE
Validate the XML document against the XML schema

POOL STATES

The following constants are useful for interpreting the `state` key value in the hash returned by `get_info`

Sys::Virt::StoragePool::STATE_INACTIVE
The storage pool is not currently active

Sys::Virt::StoragePool::STATE_BUILDING
The storage pool is still being constructed and is not ready for use yet.

Sys::Virt::StoragePool::STATE_RUNNING
The storage pool is running and can be queried for volumes

Sys::Virt::StoragePool::STATE_DEGRADED
The storage pool is running, but its operation is degraded due to a failure.

Sys::Virt::StoragePool::STATE_INACCESSIBLE
The storage pool is not currently accessible

DELETION MODES

Sys::Virt::StoragePool::DELETE_NORMAL
Delete the pool without any attempt to scrub data

Sys::Virt::StoragePool::DELETE_ZEROED
Fill the allocated storage with zeros when deleting

BUILD MODES

Sys::Virt::StoragePool::BUILD_NEW
Construct a new storage pool from constituent bits

Sys::Virt::StoragePool::BUILD_RESIZE
Resize an existing built storage pool preserving data where appropriate

Sys::Virt::StoragePool::BUILD_REPAIR
Repair an existing storage pool operating in degraded mode

Sys::Virt::StoragePool::BUILD_NO_OVERWRITE
Do not overwrite existing storage pool data

Sys::Virt::StoragePool::BUILD_OVERWRITE
Overwrite existing storage pool data

CREATE MODES

When creating a storage pool it can be built at the same time. The following values are therefore close to their BUILD counterparts.

Sys::Virt::StoragePool::CREATE_NORMAL
Just create the storage pool without building it.

Sys::Virt::StoragePool::CREATE_WITH_BUILD
When creating new storage pool also perform pool build without any flags.

Sys::Virt::StoragePool::CREATE_WITH_BUILD_OVERWRITE
Create the pool and perform pool build using the BUILD_OVERWRITE flag. This is mutually exclusive to CREATE_WITH_BUILD_NO_OVERWRITE.

Sys::Virt::StoragePool::CREATE_WITH_BUILD_NO_OVERWRITE
Create the pool and perform pool build using the BUILD_NO_OVERWRITE flag. This is mutually exclusive to CREATE_WITH_BUILD_OVERWRITE.

XML DOCUMENTS

The following constants are useful when requesting XML for storage pools

Sys::Virt::StoragePool::XML_INACTIVE
Return XML describing the inactive state of the storage pool.

LIST FILTERING

The following constants are used to filter object lists

Sys::Virt::StoragePool::LIST_ACTIVE
Include storage pools which are active

Sys::Virt::StoragePool::LIST_INACTIVE
Include storage pools which are inactive

Sys::Virt::StoragePool::LIST_AUTOSTART
Include storage pools which are marked for autostart

Sys::Virt::StoragePool::LIST_NO_AUTOSTART
Include storage pools which are not marked for autostart

Sys::Virt::StoragePool::LIST_PERSISTENT
Include storage pools which are persistent

Sys::Virt::StoragePool::LIST_TRANSIENT
Include storage pools which are transient

Sys::Virt::StoragePool::LIST_DIR
Include directory storage pools

Sys::Virt::StoragePool::LIST_DISK
Include disk storage pools

Sys::Virt::StoragePool::LIST_FS
Include filesystem storage pools

Sys::Virt::StoragePool::LIST_ISCSI
Include iSCSI storage pools

Sys::Virt::StoragePool::LIST_LOGICAL
Include LVM storage pools

Sys::Virt::StoragePool::LIST_MPATH
Include multipath storage pools

Sys::Virt::StoragePool::LIST_NETFS
Include network filesystem storage pools

Sys::Virt::StoragePool::LIST_RBD
Include RBD storage pools

Sys::Virt::StoragePool::LIST_SCSI
Include SCSI storage pools

Sys::Virt::StoragePool::LIST_SHEEPDOG
Include sheepdog storage pools

Sys::Virt::StoragePool::LIST_GLUSTER
Include gluster storage pools

Sys::Virt::StoragePool::LIST_ZFS
Include ZFS storage pools

Sys::Virt::StoragePool::LIST_VSTORAGE
Include VStorage storage pools

Sys::Virt::StoragePool::LIST_ISCSI_DIRECT
Include direct iSCSI pools

EVENT ID CONSTANTS

Sys::Virt::StoragePool::EVENT_ID_LIFECYCLE
Storage pool lifecycle events

Sys::Virt::StoragePool::EVENT_ID_REFRESH
Storage pool volume refresh events

LIFECYCLE CHANGE EVENTS

The following constants allow storage pool lifecycle change events to be interpreted. The events contain both a state change, and a reason though the reason is currently unused.

Sys::Virt::StoragePool::EVENT_DEFINED
Indicates that a persistent configuration has been defined for the storage pool

Sys::Virt::StoragePool::EVENT_STARTED
The storage pool has started running

Sys::Virt::StoragePool::EVENT_STOPPED
The storage pool has stopped running

Sys::Virt::StoragePool::EVENT_UNDEFINED
The persistent configuration has gone away

Sys::Virt::StoragePool::EVENT_CREATED
The underlying storage has been built

Sys::Virt::StoragePool::EVENT_DELETED
The underlying storage has been released

AUTHORS

Daniel P. Berrange <berrange@redhat.com>

COPYRIGHT

LICENSE

SEE ALSO

Sys::Virt, Sys::Virt::Error, `http://libvirt.org`