LIBRBD (PYTHON)

The *rbd* python module provides file-like access to RBD images.

EXAMPLE: CREATING AND WRITING TO AN IMAGE

To use *rbd*, you must first connect to RADOS and open an IO context:

```
cluster = rados.Rados(conffile='my_ceph.conf')
cluster.connect()
ioctx = cluster.open_ioctx('mypool')
```

Then you instantiate an :class:rbd.RBD object, which you use to create the image:

```
rbd_inst = rbd.RBD()
size = 4 * 1024**3  # 4 GiB
rbd_inst.create(ioctx, 'myimage', size)
```

To perform I/O on the image, you instantiate an :class:rbd.Image object:

```
image = rbd.Image(ioctx, 'myimage')
data = 'foo' * 200
image.write(data, 0)
```

This writes ‘foo’ to the first 600 bytes of the image. Note that data cannot be :type:unicode - *Librbd* does not know how to deal with characters wider than a :c:type:char.

In the end, you’ll want to close the image, the IO context and the connection to RADOS:

```
image.close()
ioctx.close()
cluster.shutdown()
```

To be safe, each of these calls would need to be in a separate :finally block:

```
cluster = rados.Rados(conffile='my_ceph_conf')
try:
    ioctx = cluster.open_ioctx('my_pool')
    try:
        rbd_inst = rbd.RBD()
        size = 4 * 1024**3  # 4 GiB
        rbd_inst.create(ioctx, 'myimage', size)
        image = rbd.Image(ioctx, 'myimage')
        try:
            data = 'foo' * 200
            image.write(data, 0)
        finally:
            image.close()
    finally:
        ioctx.close()
finally:
    cluster.shutdown()
```

This can be cumbersome, so the **Rados**, **Ioctx**, and **Image** classes can be used as context managers that close/shutdown automatically (see **PEP 343**). Using them as context managers, the above example becomes:

```
with rados.Rados(conffile='my_ceph.conf') as cluster:
    with cluster.open_ioctx('mypool') as ioctx:
        rbd_inst = rbd.RBD()
        size = 4 * 1024**3  # 4 GiB
```

```
        rbd_inst.create(ioctx, 'myimage', size)
        with rbd.Image(ioctx, 'myimage') as image:
            data = 'foo' * 200
            image.write(data, 0)
```

API REFERENCE

This module is a thin wrapper around librbd.

It currently provides all the synchronous methods of librbd that do not use callbacks.

Error codes from librbd are turned into exceptions that subclass **Error**. Almost all methods may raise **Error** (the base class of all rbd exceptions), **PermissionError** and **IOError**, in addition to those documented for the method.

A number of methods have string arguments, which must not be unicode to interact correctly with librbd. If unicode is passed to these methods, a **TypeError** will be raised.

class rbd.**RBD**

This class wraps librbd CRUD functions.

clone(*p_ioctx*, *p_name*, *p_snapname*, *c_ioctx*, *c_name*, *features=0*, *order=None*)

Clone a parent rbd snapshot into a COW sparse child.

Parameters:
- **p_ioctx** – the parent context that represents the parent snap
- **p_name** – the parent image name
- **p_snapname** – the parent image snapshot name
- **c_ioctx** – the child context that represents the new clone
- **c_name** – the clone (child) name
- **features** (*int*) – bitmask of features to enable; if set, must include layering
- **order** (*int*) – the image is split into (2**order) byte objects

Raises: **TypeError**

Raises: **InvalidArgument**

Raises: **ImageExists**

Raises: **FunctionNotSupported**

Raises: **ArgumentOutOfRange**

create(*ioctx*, *name*, *size*, *order=None*, *old_format=True*, *features=0*, *stripe_unit=0*, *stripe_count=0*)

Create an rbd image.

Parameters:
- **ioctx** (**rados.Ioctx**) – the context in which to create the image
- **name** (*str*) – what the image is called
- **size** (*int*) – how big the image is in bytes
- **order** (*int*) – the image is split into (2**order) byte objects
- **old_format** (*bool*) – whether to create an old-style image that is accessible by old clients, but can't use more advanced features like layering.
- **features** (*int*) – bitmask of features to enable
- **stripe_unit** (*int*) – stripe unit in bytes (default 0 for object size)
- **stripe_count** (*int*) – objects to stripe over before looping

Raises: **ImageExists**

Raises: **TypeError**

Raises: **InvalidArgument**

Raises: **FunctionNotSupported**

list(*ioctx*)

List image names.

Parameters: **ioctx** (**rados.Ioctx**) – determines which RADOS pool is read

Returns: list – a list of image names

remove(*ioctx*, *name*)

Delete an RBD image. This may take a long time, since it does not return until every object that comprises the image has been deleted. Note that all snapshots must be deleted before the image can be removed. If there are snapshots left, **ImageHasSnapshots** is raised. If the image is still open, or the watch from a crashed client has not expired,

ImageBusy is raised.

Parameters:
- **ioctx** (**rados.Ioctx**) – determines which RADOS pool the image is in
- **name** (*str*) – the name of the image to remove

Raises: **ImageNotFound**, **ImageBusy**, **ImageHasSnapshots**

rename(*ioctx*, *src*, *dest*)

Rename an RBD image.

Parameters:
- **ioctx** (**rados.Ioctx**) – determines which RADOS pool the image is in
- **src** (*str*) – the current name of the image
- **dest** (*str*) – the new name of the image

Raises: **ImageNotFound**, **ImageExists**

version()

Get the version number of the `librbd` C library.

Returns: a tuple of `(major, minor, extra)` components of the librbd version

class rbd.**Image**(*ioctx*, *name*, *snapshot=None*, *read_only=False*)

This class represents an RBD image. It is used to perform I/O on the image and interact with snapshots.

Note: Any method of this class may raise **ImageNotFound** if the image has been deleted.

break_lock(*client*, *cookie*)

Release a lock held by another rados client.

close()

Release the resources used by this image object.

After this is called, this object should not be used.

copy(*dest_ioctx*, *dest_name*)

Copy the image to another location.

Parameters:
- **dest_ioctx** (**rados.Ioctx**) – determines which pool to copy into
- **dest_name** (*str*) – the name of the copy

Raises: **ImageExists**

create_snap(*name*)

Create a snapshot of the image.

Parameters: **name** (*str*) – the name of the snapshot

Raises: **ImageExists**

diff_iterate(*offset*, *length*, *from_snapshot*, *iterate_cb*)

Iterate over the changed extents of an image.

This will call iterate_cb with three arguments:

(offset, length, exists)

where the changed extent starts at offset bytes, continues for length bytes, and is full of data (if exists is True) or zeroes (if exists is False).

If from_snapshot is None, it is interpreted as the beginning of time and this generates all allocated extents.

The end version is whatever is currently selected (via set_snap) for the image.

Raises **InvalidArgument** if from_snapshot is after the currently set snapshot.

Raises **ImageNotFound** if from_snapshot is not the name of a snapshot of the image.

Parameters:
- **offset** (*int*) – start offset in bytes
- **length** (*int*) – size of region to report on, in bytes
- **from_snapshot** (*str or None*) – starting snapshot name, or None

- **iterate_cb** (*function acception arguments for offset, length, and exists*) – function to call for each extent

Raises: **InvalidArgument**, **IOError**, **ImageNotFound**

discard(*offset*, *length*)

Trim the range from the image. It will be logically filled with zeroes.

flatten()

Flatten clone image (copy all blocks from parent to child)

flush()

Block until all writes are fully flushed if caching is enabled.

invalidate_cache()

Drop any cached data for the image.

is_protected_snap(*name*)

Find out whether a snapshot is protected from deletion.

Parameters: **name** (*str*) – the snapshot to check
Returns: bool - whether the snapshot is protected
Raises: **IOError**, **ImageNotFound**

list_children()

List children of the currently set snapshot (set via set_snap()).

Returns: list - a list of (pool name, image name) tuples

list_lockers()

List clients that have locked the image and information about the lock.

Returns: dict - contains the following keys:

- `tag` - the tag associated with the lock (every additional locker must use the same tag)
- `exclusive` - boolean indicating whether the lock is exclusive or shared
- `lockers` - a list of (client, cookie, address) tuples

list_snaps()

Iterate over the snapshots of an image.

Returns: `SnapIterator`

lock_exclusive(*cookie*)

Take an exclusive lock on the image.

Raises: **ImageBusy** if a different client or cookie locked it **ImageExists** if the same client and cookie locked it

lock_shared(*cookie*, *tag*)

Take a shared lock on the image. The tag must match that of the existing lockers, if any.

Raises: **ImageBusy** if a different client or cookie locked it **ImageExists** if the same client and cookie locked it

protect_snap(*name*)

Mark a snapshot as protected. This means it can't be deleted until it is unprotected.

Parameters: **name** (*str*) – the snapshot to protect
Raises: **IOError**, **ImageNotFound**

read(*offset*, *length*)

Read data from the image. Raises **InvalidArgument** if part of the range specified is outside the image.

Parameters:
- **offset** (*int*) – the offset to start reading at
- **length** (*int*) – how many bytes to read

Returns: str - the data read

Raises: **InvalidArgument**, **IOError**

remove_snap(*name*)

Delete a snapshot of the image.

Parameters: **name** (*str*) – the name of the snapshot

Raises: **IOError**, **ImageBusy**

resize(*size*)

Change the size of the image.

Parameters: **size** (*int*) – the new size of the image

rollback_to_snap(*name*)

Revert the image to its contents at a snapshot. This is a potentially expensive operation, since it rolls back each object individually.

Parameters: **name** (*str*) – the snapshot to rollback to

Raises: **IOError**

set_snap(*name*)

Set the snapshot to read from. Writes will raise ReadOnlyImage while a snapshot is set. Pass None to unset the snapshot (reads come from the current image) , and allow writing again.

Parameters: **name** (*str or None*) – the snapshot to read from, or None to unset the snapshot

size()

Get the size of the image. If open to a snapshot, returns the size of that snapshot.

Returns: the size of the image in bytes

stat()

Get information about the image. Currently parent pool and parent name are always -1 and ‘’.

Returns: dict - contains the following keys:
- `size` (int) - the size of the image in bytes
- `obj_size` (int) - the size of each object that comprises the image
- `num_objs` (int) - the number of objects in the image
- `order` (int) - log_2(object_size)
- `block_name_prefix` (str) - the prefix of the RADOS objects used to store the image
- `parent_pool` (int) - deprecated
- `parent_name` (str) - deprecated

See also **`format()`** and **`features()`**.

stripe_count()

Returns the stripe count used for the image.

stripe_unit()

Returns the stripe unit used for the image.

unlock(*cookie*)

Release a lock on the image that was locked by this rados client.

unprotect_snap(*name*)

Mark a snapshot unprotected. This allows it to be deleted if it was protected.

Parameters: **name** (*str*) – the snapshot to unprotect

Raises: **IOError**, **ImageNotFound**

write(*data*, *offset*)

Write data to the image. Raises **InvalidArgument** if part of the write would fall outside the image.

Parameters:	• **data** (*str*) – the data to be written • **offset** (*int*) – where to start writing data
Returns:	int - the number of bytes written
Raises:	**IncompleteWriteError**, **LogicError**, **InvalidArgument**, **IOError**

class rbd.**SnapIterator**(*image*)

Iterator over snapshot info for an image.

Yields a dictionary containing information about a snapshot.

Keys are:

- id (int) - numeric identifier of the snapshot
- size (int) - size of the image at the time of snapshot (in bytes)
- name (str) - name of the snapshot