

---

# **iCBD-Replication Documentation**

***Release 1.0.0***

**Luis Silva**

**Nov 19, 2018**



## ICBD REPLICATION MODULE

<b>1</b>	<b>API documentation</b>	<b>3</b>
1.1	icbdrep.ImageRepo module . . . . .	3
1.2	icbdrep.KeepAlive module . . . . .	4
1.3	icbdrep.MasterNode module . . . . .	5
1.4	icbdrep.NameServer module . . . . .	6
1.5	icbdrep.ReplicaNode module . . . . .	6
1.6	icbdrep.icbdrepd module . . . . .	8
1.7	lib.benchmarkinglib module . . . . .	8
1.8	lib.btrfslib module . . . . .	9
1.9	lib.compressionlib module . . . . .	10
1.10	lib.icbdSnapshot module . . . . .	12
1.11	lib.restapilib module . . . . .	13
1.12	lib.serializerslib module . . . . .	14
1.13	lib.sshlib module . . . . .	14
1.14	lib.utllib module . . . . .	14
1.15	exceptions.ImageRepoException module . . . . .	15
1.16	exceptions.ReplicasException module . . . . .	15
1.17	tests.benchLibTests module . . . . .	15
1.18	tests.pyroNSTests module . . . . .	16
1.19	tests.utilTests module . . . . .	16
1.20	Indices and tables . . . . .	16
	<b>Python Module Index</b>	<b>17</b>
	<b>Index</b>	<b>19</b>



This site covers iCBD-Replication usage & API documentation. For basic info on what iCBD-rep is, including its public changelog & how the project is maintained, please see the git repo.



## API DOCUMENTATION

We maintain a set of API documentation, autogenerated from the python source code's docstrings (which are typically very thorough.) and for the RESTfull API (TODO: FUTURE)

### 1.1 icbdrep.ImageRepo module

```
class icbdrep.ImageRepo.ImageRepo (config)
    Bases: object

    addImage (image_name: str)
        Add an image name to the repository And checks if in that directory are already present some snapshots

        Args: image_name: name of the image to be added

        Returns: None

        Raises: DirNotFoundException, BTRFSPathNotFoundException, ImageAlreadyExistsException

    addSnapshot (image_name: str, snap_number: str) → None
        Add a snapshot to a image

        Args: image_name: the name of the image to receive a snapshot snap_number: the snapshot

        Returns: None

        Raises: BTRFSSubvolumeNotFoundException, SnapshotAlreadyExistsException

    deleteImage (image_name: str) → None
        Deletes a given image from the repository

        Args: image_name: the name of the image to be deleted

        Returns: None

        Raises: ImageNotFoundException

    deleteSnapshot (image_name: str, snap_number: str) → lib.icbdSnapshot.icbdSnapshot
        Deletes a given snapshot of an image

        Args: image_name: the image to which the snapshot refers to snap_number: the snapshot number

        Returns: None

        Raises: SnapshotNotFoundException

    getImageList () → typing.List[str]
        Get the list of the VM images present in the repo

        Returns: a list of strings with the images names
```

**getImagepath** (*image\_name: str*) → str

Returns the path to the given image.

**Args:** image\_name: the name of the image

Returns: a string with the path to the image

Raises: ImageNotFoundException

**getLastSnapshot** (*image\_name: str*) → lib.icbdSnapshot.icbdSnapshot

Get the last snapshot from the given image.

**Args:** image\_name: name of the image

Returns: an obj icbdSnapshot

Raises: ImageNotFoundException

**getSnapshot** (*image\_name: str, snap\_number: str*) → lib.icbdSnapshot.icbdSnapshot

Gets a specific snapshot given its number and the image name

**Args:** image\_name: the name of the image snap\_number: the number of the snapshot

Returns: an icbdSnapshot object

Raises: SnapshotNotFoundException

**getSnapshotlist** (*image\_name: str*) → typing.List[lib.icbdSnapshot.icbdSnapshot]

Get the list of snapshots present in the repo for the given image. If there are no snapshots it returns a empty list.

**Args:** image\_name: The image name that contains the snapshots

Returns: a list with the snapshots present in the repo

Raises: ImageNotFoundException

**hasImage** (*image\_name: str*) → bool

Check if a given image name is present in the repository

**Args:** image\_name: the image name to be checked

Returns: True if present, otherwise False

**hasSnapshot** (*image\_name: str, snap\_number: str*) → bool

Check if a snapshot is present in the given image

**Args:** image\_name: the name of the image that should contain the snapshot snap\_number: the snapshot

Returns: True if the snapshot is present, otherwise False

## 1.2 icbdrep.KeepAlive module

**class** icbdrep.KeepAlive.**KeepAlive** (*interval=10, tries\_num=3*)

Bases: threading.Thread

**keepAlive** (*pyro\_bind: bool*) → None

Check a replica state and updates NS if needed.

**Args:** pyro\_bind: boolean True to use of the \_pyroBind or False to use the ping method

Returns: None



**run()**

The main method of the class. This is triggered in the thread.start() call

Returns: None

**stopKeepAlive()** → None

Stop the execution of the keep alive thread. This should be part of the shutdown process.

Returns: None

## 1.3 icbdrep.MasterNode module

**class** icbdrep.MasterNode.**MasterNode** (*node\_config, ns\_config, interactive\_mode\_flag: bool*)

Bases: threading.Thread

**addImage** (*image\_name: str, node: int*) → None

Add an image to the node repository

**Args:** image\_name: the name of the image to be added node: the node where the image will be added

Returns: Node

**delete\_snapshot** (*image\_name: str, snap\_number: str, node: int*) → None

Deletes a snapshot from a given image in a node.

**Args:** image\_name: the image name snap\_number: the snapshot number node: the node to do the deletion

Returns: None

**exeCommand** (*line: str*) → None

Receives a command line and interprets the content. Separating the various fields of the string into arguments, and calls the appropriated function.

**Args:** line: a line with the command to execute

Returns: None

**getReplicasFromNS** () -> (<class 'int'>, typing.Dict[int, Pyro4.core.Proxy])

Get a list of the replicas present in the system (Name Server) and saves them to the replicas proxy list

Returns: the number of found replicas

**interactiveMode** () → None

When in interactive mode, the server runs with a prompt, so that individual commands can be typed in

Returns: None

**listImages** (*node: int*) → None

List the collection of images available in a node.

**Args:** node: The node to list. (Master or one of the Replicas)

Returns: None

**listReplicas** () → None

List the replicas present in the system and prints to the console.

Returns: None

**listSnapshots** (*node: int, image\_name: str*) → None

List the collection of snapshots of a given image in a node.

**Args:** node: The node to list (Master or one of the replicas) image\_name: The image the snapshots refer to

Returns: None

**registerInNS** () → Pyro4.core.Daemon  
Register the server in the Name Server

Returns: the registered daemon

**run** ()  
The main method of the class. This is triggered in the thread.start() call

Returns: None

**send** (node: int, image\_name: str, snapshot\_number: str, blocking: bool, ssh: bool = False, compression: str = None) → None  
Send Command - Instructs the replica to listen for a transfer, and sends the snapshot in the btrfs path

**Args:** node: the number of the node image\_name: the name of the image snapshot\_number: the number of the image blocking: if the function should block

Returns: None

**stopMaster** () → None  
WARNING!! Don't use this! Only for testing and should be deprecated!

Returns: None

## 1.4 icbdrep.NameServer module

**class** icbdrep.NameServer.**NameServer** (config)  
Bases: threading.Thread

**run** ()  
The main method of the class. This is triggered in the thread.start() call

Returns: None

**stopNS** () → None  
This function closes both the broadcast and name servers. This is called in the shutdown procedure.

Returns: None

## 1.5 icbdrep.ReplicaNode module

**class** icbdrep.ReplicaNode.**ReplicaNode** (rep\_id: int, node\_config, ns\_config)  
Bases: object

**addImage** (image\_name: str) → bool  
Add an image to the node's repository

**Args:** image\_name: the name of the image to be added.

Returns: a boolean with the success of the operation

**deleteSnapshot** (image\_name: str, snap\_number: str) → lib.icbdSnapshot.icbdSnapshot  
Delete a snapshot from the repo and FS

**Args:** image\_name: the name of the image snap\_number: the number of the snapshot

Returns: the snapshot which as deleted

**getImagesList** () → typing.List[str]

Get the list of images present in the replica

Returns: a list of strings

**getLastSnapshot** (*image\_name: str*) → lib.icbdSnapshot.icbdSnapshot

Return the last snapshot of the given image.

**Args:** image\_name: the name of the image

Returns: an obj icbdSnapshot

**getName** () → str

Get the replica name

Returns: a string with the name

**getReplicaBtrfsAddress** () → typing.Tuple[str, int]

Return the IP and PORT address for the btrfs transfer.

Returns: A tuple with an IP and PORT

**getReplicaID** () → int

Get the replica ID number. This should be a integer that originates from the

Returns: the replica ID

**getSnapshotList** (*image\_name: str*) → typing.List[lib.icbdSnapshot.icbdSnapshot]

Return the list of snapshots stored in the repo for the given image name. Case there are no snapshots the list returned is empty. Case the image in args ins't in the repo return None.

**Args:** image\_name: Image name to get the snapshot list.

Returns: a list with the snapshots.

**ping** () → str

Responds to a ping request with "pong"

Returns: "pong"

**poisonPill** () → None

Shutdown message to the replica

Returns: None

**prepareReceive** (*image\_name: str, snap\_number: str*) → bool

This function should precede the receive() call. Checks if the node wants the image in question or if the snapshot is already present.

**Args:** image\_name: the name of the image snap\_number: the name of the snap

Returns: a bool that indicates if the replica will accept the receive

**receive** (*image\_name: str, snap\_number: str, compression: str = None*)

Receives a snapshot

Returns: None

## 1.6 icbdrep.icbdrepd module

## 1.7 lib.benchmarkinglib module

```
class lib.benchmarkinglib.Benchmark (name)
    Bases: object
    addRun (run: lib.benchmarkinglib.Run)
    get_name ()
    mean ()
    median ()
    stdev ()

class lib.benchmarkinglib.Run (interfaceName, runNumber=-1, imageName='default')
    Bases: object
    getBtrfsTransferBytes ()
        Returns:
    getBtrfsTransferPackets ()
        Returns:
    getBtrfsTransferRuntime ()
        Returns:
    getGlobalTransferRuntime ()
        Returns:
    getIcbbBootTransferBytes ()
        Returns:
    getIcbbBootTransferPackets ()
        Returns:
    getIcbbBootTransferRuntime ()
        Returns:
    getIscsiTargetTransferBytes ()
        Returns:
    getIscsiTargetTransferPackets ()
        Returns:
    getIscsiTargetTransferRuntime ()
        Returns:
    startTimmer (transferType)
        Start a timmer for one of the transfer counters.
        Args: transferType: the type of the transfer to start counting time
        Returns: call the appropriated function
    stopTimmer (transferType)
        Stop a timmer for one of the transfer counters.
        Args: transferType: the type of the transfer to start counting time
        Returns: call the appropriated function
```

```
class lib.benchmarkinglib.linuxNetworkTraffic
    Bases: object

    static getInterfaceStats (interfaceName)

        Args: interfaceName:

        Returns:
```

## 1.8 lib.btrfslib module

```
class lib.btrfslib.BtrfsFsCheck
    Bases: object

    static isBtrfsPath (path: str)
        Check if a given path is in fact present in a BTRFS tree

        !!Caution!! : This function does not takes into account the fact that the path might not be a valid one.

        Args: path: the path to be checked

        Returns: true if present, otherwise false

    static isBtrfsSubvolume (path: str)
        Check if the given path is a BTRFS subvolume / snapshot.

        Args: path: the path to be checked

        Returns: True if a subvolume, otherwise false

    static searchForSnapshots (path: str) → typing.List[str]
        Search the directory , and gets the snapshots that are already present

        Args: path: the directory to be searched

        Returns: a List with the name of the snapshot

class lib.btrfslib.BtrfsTool
    Bases: object

    static delete (path: str) → None
        Wrapper for the BTRFS Tools subvolume delete command.

        The method receives a path and calls the btrfs subvolume delete for that path.

        Args: path: the path to the subvolume to delete

        Returns: None

    static receive (dst_path: str, src_port: int, compression: str = None)
        Wrapper for the BTRFS Tools receive() command.

        This method opens a socket and listens for a connection Then receives a snapshot and redirect it to the
        stdin of the BTRFS receive

        Args: dst_path: the path of the image to place the snapshot src_port: the port to listening for the transfer

        Returns: None

    static send (src_path: str, dst_ip: str, dst_port: int, parent: str = None, compression: str = None)
        Wrapper for the BTRFS Tools send() command.

        This method is BLOCKING, it will wait for the conclusion of the send command. It uses regular sockets
        to send to an endpoint the data from the snapshot.
```

**Args:** `src_path`: the path of the snapshot to be send `dst_ip`: the IP of the destiny socket `dst_port`: the Port the destiny is listening

Returns: None

**static sendNonBlock** (*src\_path: str, dst\_ip: str, dst\_port: int, parent: str = None, compression: str = None*)

Wrapper for the BTRFS Tools `send()` command.

This method is NON BLOCKING, it will NOT wait for the conclusion of the send command. It uses regular sockets to send to an endpoint the data from the snapshot.

**Args:** `src_path`: the path of the snapshot to be send `dst_ip`: the IP of the destiny socket `dst_port`: the Port the destiny is listening

Returns: None

**static sendSSH** (*src\_path: str, dst\_ip: str, dst\_port: int, parent: str = None, compression: str = None*)

Wrapper for the BTRFS Tools `send()` command.

This method is BLOCKING, it will wait for the conclusion of the send command. It uses regular sockets to send to an endpoint the data from the snapshot.

**Args:** `src_path`: the path of the snapshot to be send `dst_ip`: the IP of the destiny socket `dst_port`: the Port the destiny is listening

Returns: None

**static setReadOnly** (*path: str, state: bool*) → None

Wrapper for the BTRFS Tools property set read only command.

This method sets the the read only property for the given subvolume in the path.

**Args:** `path`: the path to the subvolume `state`: a boolean of the state of the read only

Returns: None

## 1.9 lib.compressionlib module

**class** `lib.compressionlib.compressionLib`

Bases: `object`

**static checkCompression** (*compression: str*) → bool

Check if the given compression algorithm is available to use in the lib.

**Args:** `compression`: A string with the algorithm to check

Returns: A bool representing the availability of the chosen algo.

**class** `lib.compressionlib.g_snappy`

Bases: `object`

**static compressStream** (*in\_stream, out\_stream, blocksize=65536*) → None

Uses the Google snappy compress function to compress a stream of bytes.

Takes an incoming file-like object and an outgoing file-like object, reads data from “`in_stream`”, compresses it, and writes it to “`out_stream`”. “`in_stream`” should support the read method, and “`out_stream`” should support the write method.

**Args:** `in_stream`: a stream of bytes `out_stream`: a compressed stream `blocksize`: [optional] the size used for the buffer in bytes

Returns: None

**static compress\_native** (*in\_stream, out\_stream, blocksize=65536*) → None

Wrapper for the snappy native stream compression

**Args:** *in\_stream*: a stream of bytes *out\_stream*: a compressed stream *blocksize*: [optional] the size used for the buffer in bytes

Returns:

**static decompressStream** (*in\_stream, out\_stream, blocksize=65536*) → None

Uses the Google snappy decompress function to handle a compressed stream.

Takes an incoming file-like object and an outgoing file-like object, reads data from “*in\_stream*”, decompresses it, and writes it to “*out\_stream*”. “*in\_stream*” should support the read method, and “*out\_stream*” should support the write method.

**Args:** *in\_stream*: a compressed stream *out\_stream*: the original stream of bytes *blocksize*: [optional] the size used for the buffer in bytes

Returns:None

**static decompress\_native** (*in\_stream, out\_stream, blocksize=65536*) → None

Wrapper for the snappy native stream decompression

**Args:** *in\_stream*: a compressed stream *out\_stream*: the original stream of bytes *blocksize*: [optional] the size used for the buffer in bytes

Returns:

**class** lib.compressionlib.lz4

Bases: object

**static compressStream** (*in\_stream, out\_stream*) → None

Uses the lz4 compress function to compress a stream of bytes

Takes an incoming file-like object and an outgoing file-like object, reads data from “*in\_stream*”, compresses it, and writes it to “*out\_stream*”. “*in\_stream*” should support the read method, and “*out\_stream*” should support the write method.

**Args:** *in\_stream*: a bytes input stream to be compressed *out\_stream*: the compressed stream

Returns: None

**static decompressStream** (*in\_stream, out\_stream*) → None

Uses the lz4 decompress function to decompress a stream of bytes

Takes an incoming file-like object and an outgoing file-like object, reads data from “*in\_stream*”, decompresses it, and writes it to “*out\_stream*”. “*in\_stream*” should support the read method, and “*out\_stream*” should support the write method.

**Args:** *in\_stream*: a compressed stream *out\_stream*: the original bytes

Returns: None

**class** lib.compressionlib.z\_lib

Bases: object

**static compress2** (*in\_stream, out\_stream*)

!!!IN TESTING!!! !!DONT USE THIS!!

**Args:** *in\_stream*: *out\_stream*:

Returns:

**static compressStream** (*in\_stream*, *out\_stream*, *blocksize=32768*) → None

Uses the zlib compress function to compress a stream of bytes.

Takes an incoming file-like object and an outgoing file-like object, reads data from “in\_stream”, compresses it, and writes it to “out\_stream”. “in\_stream” should support the read method, and “out\_stream” should support the write method.

**Args:** in\_stream: a stream of bytes out\_stream: a compressed stream blocksize: [optional] the size used for the buffer in bytes

Returns: None

**static decompress2** (*in\_stream*, *out\_stream*)

!!IN TESTING!! !!DONT USE THIS!!

**Args:** in\_stream: out\_stream:

Returns:

**static decompressStream** (*in\_stream*, *out\_stream*, *blocksize=32768*) → None

Uses the zlib decompress function to handle a compressed stream.

Takes an incoming file-like object and an outgoing file-like object, reads data from “in\_stream”, decompresses it, and writes it to “out\_stream”. “in\_stream” should support the read method, and “out\_stream” should support the write method.

**Args:** in\_stream: a compressed stream out\_stream: the original stream of bytes blocksize: [optional] the size used for the buffer in bytes

Returns: None

## 1.10 lib.icbdSnapshot module

**class** lib.icbdSnapshot.**icbdSnapshot** (*mount\_point: str*, *image\_name: str*, *snapshot\_number: str*, *icbd\_boot\_package\_path: str*, *iscsi\_target\_folder: str*)

Bases: object

**getICBDBootPackagePath** ()

Get a string with the full path to the iCBD Boot Package of the Image.

Returns: a string with the path

**getISCSITarget** ()

Get a string with the path to the iSCSI target for this snapshot.

Returns: a string with the path

**getImagePath** () → str

Get a string with the formatted path, but without the snapshot number. This should be used as a destiny path

Returns: a string with the path in the format {/mountpoint/imagename}

**getMountpointPath** () → str

Get a string with only the mount point of the snapshot

Returns: the mountpoint

**getPath** () → str

Get a string with the full path of the snapshot, including the mountpoint and image name. Format: {mount-point/imagename/snapshotnumber}



Returns: a string with the path

## 1.11 lib.restapilib module

**class** lib.restapilib.**RestAPI** (*port: int = 5009*)

Bases: object

iCBD-Replication Rest API Class

This instantiate the micro-framework Flask to provide a simple HTTP API for interacting with the system.

Note that every communication with this API uses JSON files. Responses are in JSON and an example can be found in the documentation of each method.

**api** = <flask\_restful.Api object>

**app** = <Flask 'lib.restapilib'>

**deleteImageVersion** (*replica, imi, version*)

Delete a version of an iMI in a Replica Json response example:

Endpoint path : <IP>:<Port>/api/replicas/<replica>/imis/<imi>/versions/<version>/delete/

Returns:

**listImageVersionsByReplica** (*replica, imi*)

List the version of an iMI that are present in a Replica Json response example:

Endpoint path : <IP>:<Port>/api/replicas/<replica>/imis/<imi>/versions

Returns:

**listImagesByReplica** (*replica*)

List all iMIS present in a replica. Json response example:

Endpoint path : <IP>:<Port>/api/replicas/<replica>/imis

Returns:

**listReplicas** ()

List all the replicas registered in the system. Json response example:

Endpoint path : <IP>:<Port>/api/replicas

Returns:

**listSystemImages** ()

List all the iMIs present in the Master Node This will list all iMIs available to be transfered to any replica.

Json response example:

Endpoint path : <IP>:<Port>/api/master/imis

Returns:

**listSystemImagesVersions** (*imi*)

List all the versions of an iMIs present in the Master Node

Json response example:

Endpoint path : <IP>:<Port>/api/master/imis/<imi>/versions

Returns:

**root ()**

Default root route endpoint. Mainly for testing

Endpoint path : <IP>:<Port>/api

Returns: a simple test string

**sendImageVersionToReplica ()**

List all the versions of an iMIs present in the Master Node

Json response example:

Endpoint path : <IP>:<Port>/api/master/send?imi={imi}&version={version}&replica={replica}

Returns:

**subscribeImage (replica, imi)**

Replica subscribe to a iMI Json response example:

Endpoint path : <IP>:<Port>/api/replicas/<replica>/imis/subscribe/<imi>

Returns:

**unsubscribeImage (replica, imi)**

Replica unsubscribe to a iMI Json response example:

Endpoint path : <IP>:<Port>/api/replicas/<replica>/imis/unsubscribe/<imi>

Returns:

## 1.12 lib.serializerslib module

**class lib.serializerslib.icbdSnapshotSerializer**

Bases: object

**static icbdSnapshot\_class\_to\_dict** (*obj: lib.icbdSnapshot.icbdSnapshot*)

**static icbdSnapshot\_dict\_to\_class** (*class\_name, dict*)

## 1.13 lib.sshlib module

**class lib.sshlib.sshTunnel (host, local\_port, remote\_port)**

Bases: object

**createTunnel** (*host, local\_port, remote\_port*)

## 1.14 lib.utllib module

**class lib.utllib.icbdUtil**

Bases: object

**logHeading** (*string*)

Big header for logger –[ “string” ]—————

**Args:** string: a string to be placed inside the big header

Returns: the string encapsulated in the header

**prettyfy** (*obj*)

Return pretty representation of obj. Useful for debugging.

**Args:** obj: the object to prettyfy

Returns: a pretty representation of obj

## 1.15 exceptions.ImageRepoException module

**exception** exceptions.ImageRepoException.BTRFSPathNotFoundException (*message*)

Bases: Exception

Raise when a BTRFS Path is not in the File System

**exception** exceptions.ImageRepoException.BTRFSSubvolumeNotFoundException (*message*)

Bases: Exception

Raise when a BTRFS Subvolume is not in the File System

**exception** exceptions.ImageRepoException.DirNotFoundException (*message*)

Bases: Exception

Raise when a Directory is not in the File System

**exception** exceptions.ImageRepoException.ImageAlreadyExistsException (*message*)

Bases: Exception

Raise when a Images already is present in the repo

**exception** exceptions.ImageRepoException.ImageNotFoundException (*message*)

Bases: Exception

Raise when a Images is not found

**exception** exceptions.ImageRepoException.SnapshotAlreadyExistsException (*message*)

Bases: Exception

Raise when a Snapshot already is present in the repo

**exception** exceptions.ImageRepoException.SnapshotNotFoundException (*message*)

Bases: Exception

Raise when a Snapshot is not found

## 1.16 exceptions.ReplicasException module

**exception** exceptions.ReplicasException.ReplicaNotFoundException (*message*)

Bases: Exception

Raise when a replica is not found

## 1.17 tests.benchLibTests module

tests.benchLibTests.dummyFunc ()

tests.benchLibTests.main ()

tests.benchLibTests.startCompleteRun ()

## 1.18 tests.pyroNSTests module

```
class tests.pyroNSTests.NamingTrasher (nsuri, number)
    Bases: threading.Thread

    list ()

    listprefix ()

    listregex ()

    lookup ()

    register ()

    remove ()

    run ()

tests.pyroNSTests.main ()
tests.pyroNSTests.randomname ()
```

## 1.19 tests.utilTests module

```
class tests.utilTests.TestMount (methodName='runTest')
    Bases: unittest.case.TestCase

    Our basic test class

    isBTRFS (path, assertVal)

    isSubvolume (path, assertVal)

    test_isBtrfsSet ()

    test_isSubvolumeSet ()
```

## 1.20 Indices and tables

- [genindex](#)
- [modindex](#)
- [search](#)

## PYTHON MODULE INDEX

### e

`exceptions.ImageRepoException`, [15](#)  
`exceptions.ReplicasException`, [15](#)

### i

`icbdrep.ImageRepo`, [3](#)  
`icbdrep.KeepAlive`, [4](#)  
`icbdrep.MasterNode`, [5](#)  
`icbdrep.NameServer`, [6](#)  
`icbdrep.ReplicaNode`, [6](#)

### l

`lib.benchmarkinglib`, [8](#)  
`lib.btrfslib`, [9](#)  
`lib.compressionlib`, [10](#)  
`lib.icbdSnapshot`, [12](#)  
`lib.restapilib`, [13](#)  
`lib.serializerslib`, [14](#)  
`lib.sshlib`, [14](#)  
`lib.utillib`, [14](#)

### t

`tests.benchLibTests`, [15](#)  
`tests.pyroNSTests`, [16](#)  
`tests.utilTests`, [16](#)



## A

addImage() (icbdrep.ImageRepo.ImageRepo method), 3  
 addImage() (icbdrep.MasterNode.MasterNode method), 5  
 addImage() (icbdrep.ReplicaNode.ReplicaNode method), 6  
 addRun() (lib.benchmarkinglib.Benchmark method), 8  
 addSnapshot() (icbdrep.ImageRepo.ImageRepo method), 3  
 api (lib.restapilib.RestAPI attribute), 13  
 app (lib.restapilib.RestAPI attribute), 13

## B

Benchmark (class in lib.benchmarkinglib), 8  
 BtrfsFsCheck (class in lib.btrfslib), 9  
 BTRFSPathNotFoundException, 15  
 BTRFSSubvolumeNotFoundException, 15  
 BtrfsTool (class in lib.btrfslib), 9

## C

checkCompression() (lib.compressionlib.compressionLib static method), 10  
 compress2() (lib.compressionlib.z\_lib static method), 11  
 compress\_native() (lib.compressionlib.g\_snappy static method), 11  
 compressionLib (class in lib.compressionlib), 10  
 compressStream() (lib.compressionlib.g\_snappy static method), 10  
 compressStream() (lib.compressionlib.lz4 static method), 11  
 compressStream() (lib.compressionlib.z\_lib static method), 11  
 createTunnel() (lib.sshlib.sshTunnel method), 14

## D

decompress2() (lib.compressionlib.z\_lib static method), 12  
 decompress\_native() (lib.compressionlib.g\_snappy static method), 11  
 decompressStream() (lib.compressionlib.g\_snappy static method), 11

decompressStream() (lib.compressionlib.lz4 static method), 11  
 decompressStream() (lib.compressionlib.z\_lib static method), 12  
 delete() (lib.btrfslib.BtrfsTool static method), 9  
 delete\_snapshot() (icbdrep.MasterNode.MasterNode method), 5  
 deleteImage() (icbdrep.ImageRepo.ImageRepo method), 3  
 deleteImageVersion() (lib.restapilib.RestAPI method), 13  
 deleteSnapshot() (icbdrep.ImageRepo.ImageRepo method), 3  
 deleteSnapshot() (icbdrep.ReplicaNode.ReplicaNode method), 6  
 DirNotFoundException, 15  
 dummyFunc() (in module tests.benchLibTests), 15

## E

exceptions.ImageRepoException (module), 15  
 exceptions.ReplicasException (module), 15  
 exeCommand() (icbdrep.MasterNode.MasterNode method), 5

## G

g\_snappy (class in lib.compressionlib), 10  
 get\_name() (lib.benchmarkinglib.Benchmark method), 8  
 getBtrfsTransferBytes() (lib.benchmarkinglib.Run method), 8  
 getBtrfsTransferPackets() (lib.benchmarkinglib.Run method), 8  
 getBtrfsTransferRuntime() (lib.benchmarkinglib.Run method), 8  
 getGlobalTransferRuntime() (lib.benchmarkinglib.Run method), 8  
 getICBDBootPackagePath() (lib.icbdSnapshot.icbdSnapshot method), 12  
 getIcldBootTransferBytes() (lib.benchmarkinglib.Run method), 8  
 getIcldBootTransferPackets() (lib.benchmarkinglib.Run method), 8

`getIcldBootTransferRuntime()` (`lib.benchmarkinglib.Run` method), 8

`getImageList()` (`icbdrep.ImageRepo.ImageRepo` method), 3

`getImagepath()` (`icbdrep.ImageRepo.ImageRepo` method), 3

`getImagePath()` (`lib.icbdSnapshot.icbdSnapshot` method), 12

`getImagesList()` (`icbdrep.ReplicaNode.ReplicaNode` method), 6

`getInterfaceStats()` (`lib.benchmarkinglib.linuxNetworkTraffic` static method), 9

`getISCSITarget()` (`lib.icbdSnapshot.icbdSnapshot` method), 12

`getIscsiTargetTransferBytes()` (`lib.benchmarkinglib.Run` method), 8

`getIscsiTargetTransferPackets()` (`lib.benchmarkinglib.Run` method), 8

`getIscsiTargetTransferRuntime()` (`lib.benchmarkinglib.Run` method), 8

`getLastSnapshot()` (`icbdrep.ImageRepo.ImageRepo` method), 4

`getLastSnapshot()` (`icbdrep.ReplicaNode.ReplicaNode` method), 7

`getMountpointPath()` (`lib.icbdSnapshot.icbdSnapshot` method), 12

`getName()` (`icbdrep.ReplicaNode.ReplicaNode` method), 7

`getPath()` (`lib.icbdSnapshot.icbdSnapshot` method), 12

`getReplicaBtrfsAddress()` (`icbdrep.ReplicaNode.ReplicaNode` method), 7

`getReplicaID()` (`icbdrep.ReplicaNode.ReplicaNode` method), 7

`getReplicasFromNS()` (`icbdrep.MasterNode.MasterNode` method), 5

`getSnapshot()` (`icbdrep.ImageRepo.ImageRepo` method), 4

`getSnapshotlist()` (`icbdrep.ImageRepo.ImageRepo` method), 4

`getSnapshotList()` (`icbdrep.ReplicaNode.ReplicaNode` method), 7

## H

`hasImage()` (`icbdrep.ImageRepo.ImageRepo` method), 4

`hasSnapshot()` (`icbdrep.ImageRepo.ImageRepo` method), 4

## I

`icbdrep.ImageRepo` (module), 3

`icbdrep.KeepAlive` (module), 4

`icbdrep.MasterNode` (module), 5

`icbdrep.NameServer` (module), 6

`icbdrep.ReplicaNode` (module), 6

`icbdSnapshot` (class in `lib.icbdSnapshot`), 12

`icbdSnapshot_class_to_dict()` (`lib.serializerslib.icbdSnapshotSerializer` static method), 14

`icbdSnapshot_dict_to_class()` (`lib.serializerslib.icbdSnapshotSerializer` static method), 14

`icbdSnapshotSerializer` (class in `lib.serializerslib`), 14

`icbdUtil` (class in `lib.utllib`), 14

`ImageAlreadyExistsException`, 15

`ImageNotFoundException`, 15

`ImageRepo` (class in `icbdrep.ImageRepo`), 3

`interactiveMode()` (`icbdrep.MasterNode.MasterNode` method), 5

`isBTRFS()` (`tests.utilTests.TestMount` method), 16

`isBtrfsPath()` (`lib.btrfslib.BtrfsFsCheck` static method), 9

`isBtrfsSubvolume()` (`lib.btrfslib.BtrfsFsCheck` static method), 9

`isSubvolume()` (`tests.utilTests.TestMount` method), 16

## K

`KeepAlive` (class in `icbdrep.KeepAlive`), 4

`keepAlive()` (`icbdrep.KeepAlive.KeepAlive` method), 4

## L

`lib.benchmarkinglib` (module), 8

`lib.btrfslib` (module), 9

`lib.compressionlib` (module), 10

`lib.icbdSnapshot` (module), 12

`lib.restapilib` (module), 13

`lib.serializerslib` (module), 14

`lib.sshlib` (module), 14

`lib.utllib` (module), 14

`linuxNetworkTraffic` (class in `lib.benchmarkinglib`), 8

`list()` (`tests.pyroNSTests.NamingTrasher` method), 16

`listImages()` (`icbdrep.MasterNode.MasterNode` method), 5

`listImagesByReplica()` (`lib.restapilib.RestAPI` method), 13

`listImageVersionsByReplica()` (`lib.restapilib.RestAPI` method), 13

`listprefix()` (`tests.pyroNSTests.NamingTrasher` method), 16

`listregex()` (`tests.pyroNSTests.NamingTrasher` method), 16

`listReplicas()` (`icbdrep.MasterNode.MasterNode` method), 5

`listReplicas()` (`lib.restapilib.RestAPI` method), 13

`listSnapshots()` (`icbdrep.MasterNode.MasterNode` method), 5

`listSystemImages()` (`lib.restapilib.RestAPI` method), 13

`listSystemImagesVersions()` (`lib.restapilib.RestAPI` method), 13

`logHeading()` (`lib.utllib.icbdUtil` method), 14



lookup() (tests.pyroNSTests.NamingTrasher method), 16  
 lz4 (class in lib.compressionlib), 11

## M

main() (in module tests.benchLibTests), 15  
 main() (in module tests.pyroNSTests), 16  
 MasterNode (class in icbdrep.MasterNode), 5  
 mean() (lib.benchmarkinglib.Benchmark method), 8  
 median() (lib.benchmarkinglib.Benchmark method), 8

## N

NameServer (class in icbdrep.NameServer), 6  
 NamingTrasher (class in tests.pyroNSTests), 16

## P

ping() (icbdrep.ReplicaNode.ReplicaNode method), 7  
 poisonPill() (icbdrep.ReplicaNode.ReplicaNode method), 7  
 prepareReceive() (icbdrep.ReplicaNode.ReplicaNode method), 7  
 prettify() (lib.utillib.icbdUtil method), 14

## R

randomname() (in module tests.pyroNSTests), 16  
 receive() (icbdrep.ReplicaNode.ReplicaNode method), 7  
 receive() (lib.btrfslib.BtrfsTool static method), 9  
 register() (tests.pyroNSTests.NamingTrasher method), 16  
 registerInNS() (icbdrep.MasterNode.MasterNode method), 6  
 remove() (tests.pyroNSTests.NamingTrasher method), 16  
 ReplicaNode (class in icbdrep.ReplicaNode), 6  
 ReplicaNotFoundException, 15  
 RestAPI (class in lib.restapilib), 13  
 root() (lib.restapilib.RestAPI method), 13  
 Run (class in lib.benchmarkinglib), 8  
 run() (icbdrep.KeepAlive.KeepAlive method), 4  
 run() (icbdrep.MasterNode.MasterNode method), 6  
 run() (icbdrep.NameServer.NameServer method), 6  
 run() (tests.pyroNSTests.NamingTrasher method), 16

## S

searchForSnapshots() (lib.btrfslib.BtrfsFsCheck static method), 9  
 send() (icbdrep.MasterNode.MasterNode method), 6  
 send() (lib.btrfslib.BtrfsTool static method), 9  
 sendImageVersionToReplica() (lib.restapilib.RestAPI method), 14  
 sendNonBlock() (lib.btrfslib.BtrfsTool static method), 10  
 sendSSH() (lib.btrfslib.BtrfsTool static method), 10  
 setReadOnly() (lib.btrfslib.BtrfsTool static method), 10  
 SnapshotAlreadyExistsException, 15  
 SnapshotNotFoundException, 15  
 sshTunnel (class in lib.sshlib), 14

startCompleteRun() (in module tests.benchLibTests), 15  
 startTimmer() (lib.benchmarkinglib.Run method), 8  
 stdev() (lib.benchmarkinglib.Benchmark method), 8  
 stopKeepAlive() (icbdrep.KeepAlive.KeepAlive method), 5  
 stopMaster() (icbdrep.MasterNode.MasterNode method), 6  
 stopNS() (icbdrep.NameServer.NameServer method), 6  
 stopTimmer() (lib.benchmarkinglib.Run method), 8  
 subscribeImage() (lib.restapilib.RestAPI method), 14

## T

test\_isBtrfsSet() (tests.utilTests.TestMount method), 16  
 test\_isSubvolumeSet() (tests.utilTests.TestMount method), 16  
 TestMount (class in tests.utilTests), 16  
 tests.benchLibTests (module), 15  
 tests.pyroNSTests (module), 16  
 tests.utilTests (module), 16

## U

unsubscribeImage() (lib.restapilib.RestAPI method), 14

## Z

z\_lib (class in lib.compressionlib), 11