Bynar Documentation

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Date | Reason for Change | Version |
| Michelle Zhong | 10/8/2019 | Outline the Document | 0.1 |
| Michelle Zhong | 10/9/2019 | Outline the Document Modules, fill in the API section, Config File section, start filling out the Backend Section | 0.2 |
| Michelle Zhong | 10/10/2019 | Reorganize Headers in API section, Fill out the Backend, add Database Schema, add Error Module, Host Information, Helper Library | 0.3 |
|  |  |  |  |

# Table of Contents

[Revision History 2](#_Toc21619192)

[Table of Contents 3](#_Toc21619193)

[API 5](#_Toc21619194)

[Introduction 5](#_Toc21619195)

[Messages 5](#_Toc21619196)

[Enums 5](#_Toc21619197)

[Structs 6](#_Toc21619198)

[Configuration Files 8](#_Toc21619199)

[Introduction 8](#_Toc21619200)

[List of Config Files 8](#_Toc21619201)

[Bynar JSON 8](#_Toc21619202)

[Ceph JSON 9](#_Toc21619203)

[Disk-Manager JSON 9](#_Toc21619204)

[Backend 9](#_Toc21619205)

[Introduction 9](#_Toc21619206)

[Backend Module 9](#_Toc21619207)

[Enums 9](#_Toc21619208)

[Interface 9](#_Toc21619209)

[Ceph 10](#_Toc21619210)

[Structs 10](#_Toc21619211)

[Helper Functions 14](#_Toc21619212)

[Database Schema 18](#_Toc21619213)

[Introduction 18](#_Toc21619214)

[Postgres 18](#_Toc21619215)

[Schema 19](#_Toc21619216)

[Database Logging 19](#_Toc21619217)

[Introduction 19](#_Toc21619218)

[Logging 19](#_Toc21619219)

[Helper Functions 19](#_Toc21619220)

[Introduction 19](#_Toc21619221)

[Error Module 19](#_Toc21619222)

[Type 19](#_Toc21619223)

[Enums 19](#_Toc21619224)

[Structs 21](#_Toc21619225)

[Host Information 21](#_Toc21619226)

[Enums 21](#_Toc21619227)

[Structs 21](#_Toc21619228)

[Helper Functions 22](#_Toc21619229)

[Helper Module 23](#_Toc21619230)

[Structs 23](#_Toc21619231)

[Helper Functions 23](#_Toc21619232)

[Client 26](#_Toc21619233)

[Introduction 26](#_Toc21619234)

[Client Interface 26](#_Toc21619235)

[Support Tickets 26](#_Toc21619236)

[Introduction 26](#_Toc21619237)

[JIRA Support 26](#_Toc21619238)

[Disk Manager 26](#_Toc21619239)

[Introduction 26](#_Toc21619240)

[Disk Manager 26](#_Toc21619241)

[Disk Testing 26](#_Toc21619242)

[Introduction 26](#_Toc21619243)

[State Machine 26](#_Toc21619244)

[Hardware Testing 26](#_Toc21619245)

[Introduction 26](#_Toc21619246)

[Hardware Tests 26](#_Toc21619247)

[Bynar 26](#_Toc21619248)

[Introduction 26](#_Toc21619249)

[Main Process 26](#_Toc21619250)

# API

## Introduction

This package uses Protobuf version 2 to create Messages that can be sent over the network. Protobuf is a fast and small protocol for serializing structs (or structured data). Serialized messages can be sent between Sockets, unpackaged, and read easily and quickly. The protobuf package automatically generates the rust code needed to create, modify, and destroy Messages as well as their attributes.

## Messages

### Enums

#### DiskType

The type of disk or device

##### Enum Values

|  |  |
| --- | --- |
| Name | Description |
| LOOPBACK | Special loopback device |
| LVM | Logical Volume Device |
| MDRAID | Linux software RAID |
| NVME | Non-Volatile Memory Express, a logical device interface |
| RAM | Ramdisk |
| ROTATIONAL | Regular rotational device |
| SOLID\_STATE | SSD |
| VIRTUAL | Virtual Disk |
| UNKNOWN | Unknown disk |

#### ResultType

A result value

##### Enum Values

|  |  |
| --- | --- |
| Name | Description |
| OK | ok |
| ERR | There is an error |

#### Op

An operation on a disk

##### Enum Values

|  |  |
| --- | --- |
| Name | Description |
| Add | Generic Add Disk command, returns an OpResult |
| AddPartition | Add a Partition Command, returns an OpResult |
| List | List the Disks, returns a list of Disks |
| Remove | Remove a Disk, returns an OpResult |
| SafeToRemove | Checks if a Disk is safe to remove, returns a bool |
| GetCreatedTickets | list created tickets, returns a list of created tickets |

#### DatacenterOp

Datacenter API’s, these all require server\_id as a parameter for the operation

##### Enum Values

|  |  |
| --- | --- |
| Name | Description |
| GetDc | Get ? Returns an OpStringResult |
| GetRack | Get the rack of a server, returns an OpStringResult |
| GetRow | Get the row of a server, returns an OpStringResult |
| GetElevation | Get the elevation of a server, returns an OpStringResult |

### Structs

#### Osd

A Ceph OSD object descriptor

##### Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| fsid | Option<String> | OSD File System ID, if one exists |
| id | u64 | OSD ID number |
| block\_device | String | Block Device of the OSD |
| journal | Option<String> | Name of the Journal if the OSD has one set |
| active | bool | Whether or not an OSD is active or a spare |
| used\_space | u64 | How much space in the OSD is currently used |
| total\_space | u64 | Total space in the OSD |

#### Partition

A single partition descriptor

##### Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| uuid | String | The id of the partition |
| first\_lba | u64 | The first logical block address of the partition |
| last\_lba | u64 | The last logical block address of the partition |
| flags | u64 | Flags associated with the partition |
| name | String | The name of the partition |

#### PartitionInfo

A list of Partitions

##### Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| partition | Vec<Partition> | List of partitions |

#### Disk

A disk object descriptor

##### Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| type | DiskType | The type of disk |
| dev\_path | String | ?? Device path? |
| partitions | PartitionInfo | Disk partitions |
| serial\_number | Option<String> | Disk serial number |

#### OpResult

A result of an Op message

##### Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| result | ResultType | Whether the result is ok or Error |
| error\_msg | Option<String> | Error message if there is an error |

#### OpBoolResult

A boolean result of an Op message

##### Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| result | ResultType | Whether Ok or Error |
| value | Option<bool> | A value is set if OK |
| error\_msg | Option<String> | Error message is set if there is an Error |

#### OpStringResult

A String result of an Op message

##### Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| result | ResultType | Whether Ok or Error |
| value | Option<String> | A value is set if OK |
| error\_msg | Option<String> | Error message is set if there is an Error |

#### JiraInfo

A Jira Ticket information descriptor

##### Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| ticket\_id | String | Ticket number |
| server\_name | String | Name of the JIRA server |

#### OpJiraTicketsResult

A Jira ticket result

##### Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| result | ResultType | Whether Ok or Error |
| value | Option<String> | A value is set if OK |
| error\_msg | Option<String> | Error message is set if there is an Error |

#### DatacenterOperation

A Datacenter operation message

##### Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| Op\_type | DatacenterOp | The type of operation to be performed |
| server\_id | String | The ID of the server to be operated on |

#### Operation

A service operation that can be performed

##### Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| Op\_type | Op | The operation type |
| disk | Option<String> | The disk name, used for an Add or Remove |
| simulate | Option<bool> | Whether the operation is a simulation, used for Add, Remove, and SafeToRemove |
| partition\_start | Option<u64> | Optional field for AddPartition, start of a partition |
| partition\_end | Option<u64> | Optional field for AddPartition, end of a partition |
| partition\_name | Option<String> | Optional field for AddPartition, partition name |
| osd\_id | Option<u64> | Optional Ceph related field, the id of an OSD |
| replica\_set | Vector<String> | Host:/dev/disk strings list for gluster replica sets |

# Configuration Files

## Introduction

Bynar uses a set of configuration files to configure different settings. Bynar uses JSON as the format for its configuration files, as JSON files are easily parsed, serialized, and deserialized using the Rust serde and serde-json crates.

## List of Config Files

### Bynar JSON

This config file, bynar.json, is used to configure several different settings, including a Slack webhook, JIRA support, Redfish access, Vault password access, and Postgres database access

|  |  |  |
| --- | --- | --- |
| Name | Description | Example Value |
| proxy | Proxy web server? | “https://my.proxy” |
| manager\_host | The host ip of the bynar disk manager | “localhost” |
| manager\_port | The port of the Bynar disk manager | 5555 |
| slack\_webhook | Slack webhook to access Slack API | "<https://hooks.slack.com/services/ID>" |
| slack\_channel | Slack channel to post messages to | “#my-channel" |
| slack\_botname | Name of the Bot to post messages under | "my-bot" |
| jira\_user | JIRA username to create tickets under | “test\_user” |
| jira\_password | JIRA password | “user\_pass” |
| jira\_host | JIRA host to create tickets under | “https://tickets.jira.com” |
| jira\_issue\_type | JIRA issue type name to create tickets of | “3” |
| jira\_priority | JIRA priority value of tickets created | “4” |
| jira\_project\_id | JIRA project id to create tickets under | “MyProject” |
| jira\_ticket\_assignee | User created JIRA tickets are assigned to | “assignee\_username” |
| redfish\_ip | IP address of a Redfish instance | “localhost” |
| redfish\_username | Username to access Redfish instance | “redfish\_user” |
| redfish\_password | Password to access Redfish | “redfish\_pass” |
| redfish\_port | Port of the Redfish instance | 4443 |
| vault\_endpoint | Hashicorp vault endpoint | “https://my\_vault.com” |
| vault\_token | Hashicorp vault token to access the vault | “token\_234464562” |
| database | List of Database parameters |  |
| database:username | Username to access database with | “postgres” |
| database:password | Password to access database with | “” |
| database:port | Port of the database | 5432 |
| database:dbname | Name of the database | “bynar” |
| database:endpoint | Database endpoint | “some.endpoint” |

### Ceph JSON

This config file, ceph.json, is used to tell Bynar where the ceph.conf file is, what user to use when running Ceph commands, and what journal devices are known?

|  |  |  |
| --- | --- | --- |
| Name | Description | Example Value |
| config\_file | The path to the ceph.conf file | “/etc/ceph/ceph.conf” |
| user\_id | User to use when running Ceph commands | “admin” |
| journal\_devices | Journal device list |  |
| journal\_devices:device | Path of the device | “/dev/sda” |
| journal\_devices:partition\_id | Partition ID number | 1 |

### Disk-Manager JSON

This config file, disk-manager.json is used to tell Bynar what the backend storage system is

|  |  |  |
| --- | --- | --- |
| Name | Description | Example Value |
| backend | The backend type of the storage system | "ceph” |

# Backend

## Introduction

Different distributed storage clusters have different ways of adding and removing disks, the backend module seeks to create an interface to the different backends

## Backend Module

A Generic Module for interfacing with different storage backends

### Enums

#### BackendType

##### Enum Values

|  |  |
| --- | --- |
| Name | Description |
| Ceph | Ceph is the backend type |
| Gluster | GlusterFS is the backend type |

##### Trait Implementations

###### FromStr

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Inputs | Description | Outputs |
| from\_str | s: &str | Converts a string to a BackendType. Return Ok(BackendType) if successful or an Error if the string is not a BackendType | BynarResult<BackendType> |

###### Clone, Debug, Deserialize

### Interface

#### Backend

##### Trait Function Definition

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Inputs | Description | Outputs |
| add\_disk | device: &Path  id: Option<u64>  simulate: bool | Add a disk at path *device*, *id* an optional OSD id for Ceph clusters to ensure the OSD is set to that id, if *simulate* is passed no action is taken. Returns Ok(()) if successful or an Error if one occurs | BynarResult<()> |
| remove\_disk | device: &Path  simulate: bool | Remove a disk at path *device* from a cluster. If *simulate* is passed no action is taken. Returns Ok(()) if successful or an Error if one occurs | BynarResult<()> |
| safe\_to\_remove | device: &Path  simulate: bool | Check if safe to remove a disk from a cluster at path *device*. If *simulate* passed then return true. Returns Ok(true) if successful and safe, Ok(false) if successful and not safe to remove, or an Error if one occurs | BynarResult<bool> |

##### Public Functions

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Inputs | Description | Outputs |
| load\_backend | backend\_type: &BackendType  config\_dir: Option<&Path> | Given a BackendType, *backend\_type,* and a config file directory from *config\_dir*, return Ok(Backend) if successful or Error if one occurs. | BynarResult<Box<dyn Backend>> |

## Ceph

The Ceph backend implementation

### Structs

#### CephBackend

This is a public struct object defining a Ceph cluster

##### Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| cluster\_handle | Rados | A handle to the ceph librados |
| config | CephConfig | Handle for the Ceph Configuration File |
| version | CephVersion | The Ceph Version |

##### Implementation

|  |
| --- |
| Function Definition |
| new(config\_dir: Option<&Path>) -> BynarResult<()>  DESCRIPTION: Create a new CephBackend  PARAMETERS: config\_dir – the directory of the ceph.json file or NONE if in the .config directory of the HOME directory  RETURNS: Ok(CephBackend) on success, Error otherwise  IMPLEMENTATION: Get the ceph.json file from the config\_dir parameter. If successful, create the CephConfig object from the ceph.json file. Using the CephConfig object, connect to the specified Ceph instance using the specified user id to get the librados handle. Using the Rados handle, get the Ceph version string and convert it into a CephVersion object. If all steps are successful return a new CephBackend object with the CephConfig, Rados handle, and CephVersion. |
| add\_bluestore\_osd(&self, dev\_path:&Path, id:Option<u64>, simulate: bool) -> BynarResult<()>  DESCRIPTION: Add a bluestore OSD to the Ceph Cluster  PARAMETERS: dev\_path – the device path of the OSD  id– the OSD id of the OSD to add  simulate – if passed skip execution of the function  RETURNS: Ok(()) on success, Error otherwise  IMPLEMENTATION: Find a journal device that has enough free space? Create a new osd and get its osd\_id (if id is not NONE then the new osd id should match id. Create an osd\_fsid, and use it, the osd id, the device path, and the journal to create an lvm. Create a mount point path for the drive if necessary. Write the osd fsid to a file. Resolve the created lvm name to a true device path and chown it so ceph can use it. Symlink the lvm device name to the mount point’s /block, and if a journal device with enough space was found, symlink the journal to the mount point’s /block.wal and change the permissions so ceph can use it. Write activate monmap out by getting the map, and creating a file activate.monmap. Lookup the ceph user id and change all the permissions on the created files so ceph can use them. Create a ceph authorization entry, get the keyring created and save it. Format the osd with the osd filesystem. Use the ceph bluestore tool, and add the osd to the crush. Enable the osd, and then initialize the osd. If all steps are successful return (), else it error’d out somewhere. |
| add\_filestore\_osd(&self, dev\_path:&Path, id:Option<u64>, simulate:bool) -> BynarResult<()>  DESCRIPTION: Add a new /dev/ path as an osd, with xfs, for Jewel or earlier  PARAMETERS: dev\_path – the device path of the OSD  id– the OSD id of the OSD to add  simulate – if passed skip execution of the function  RETURNS: Ok(()) on success, Error otherwise  IMPLEMENTATION: Format the drive with the Xfs filesystem. Prove the drive by getting the device info and checking if it has a filesystem id. Create a new osd and get its id, which should be the same as the input id if one was input. Create the mount point path and mount the drive. Select a journal with enough space (if there is one, can be None). Format the osd with the osd filesystem. Create a ceph authorization entry, get the authorization key and save the keyring. Add the osd to the crush, add the osd to the fstab, then init the osd. If all steps are successful return (), else it error’d out somewhere. |
| change\_permissions(&self, paths: &[&Path], perms: &Passwd) -> BynarResult<()>  DESCRIPTION: change permissions of many files at once  PARAMETERS: paths – the paths of the files to change the permissions of  perms – the group and owner permissions to change the file permissions to  RETURNS: Ok(()) on success, Error otherwise  IMPLEMENTATION: loop through the paths and chown each path to the input permission values. If all steps are successful return (), else it error’d out somewhere. |
| create\_lvm(&self, osd\_fsid: &uuid::Uuid, new\_osd\_id: u64, dev\_path: &Path, journal\_device: Option<&JournalDevice>) -> BynarResult<(PathBuf, u64)>  DESCRIPTION: Create the lvm device and return the path and size of it  PARAMETERS: osd\_fsid – the osd filesystem id  new\_osd\_id – the id of the osd  dev\_path – the path to the device of the osd  journal\_device – an optional journal device ? Dunno what it’s used for...  RETURNS: Ok(PathToLvm,Size) on success, Error otherwise  IMPLEMENTATION: probe the device for its udev info. create a volume group name, and logical volume name, and use them to create the logical volume device name. Initialize a new LVM, and scan it. create the volume group on the LVM, then add the device path to the volume group by extending it and writing. create a linear logical volume in the volume group, create its tags. If all steps are successful return the path to the lvm device name and the volume group size, else it error’d out somewhere. |
| create\_lvm\_tags(&self, lv: &LogicalVolume<\_,\_>, lv\_dev\_name: &Path, osd\_fsid: &uuid::Uuid, new\_osd\_id:u64, info:&block\_utils::Device, journal\_device:Option<&JournalDevice)->BynarResult<()>  DESCRIPTION: Add the lvm tags that ceph requires to identify the osd  PARAMETERS: lv – the logical volume  lv\_dev\_name – the path to the logical volume device  osd\_fsid – the osd filesystem id  new\_osd\_id – the id of the osd  info – the device info  journal\_device – an optional journal device ? Dunno what it’s used for...  RETURNS: Ok(()) on success, Error otherwise  IMPLEMENTATION: create the lvm tags. If there is a journal device input, add a tag for the wal\_device and add the wal\_uuid. Once all tags are created add them to the logical volume. If all steps are successful return (), else it error’d out somewhere. |
| remove\_bluestore\_osd(&self, dev\_path:&Path, simulate:bool) -> BynarResult<()>  DESCRIPTION: Remove a bluestore OSD to the Ceph Cluster  PARAMETERS: dev\_path – the device path of the OSD  simulate – if passed skip execution of the function  RETURNS: Ok(()) on success, Error otherwise  IMPLEMENTATION: Initialize an lvm and scan it for volume groups and LVM metadata. Get the volume group that the device is associated with, if it cannot find the volume group, check if it is a filestore and if so fall back. otherwise, open the volume group and list all logical volumes in the volume group. List the tags to get the osd id and osd fsid. Set the osd as out, remove it from the crush, delete the authorization key, stop the osd, and remove it. Then, wipe the disk. remove all the logical volumes associated with the volume group, remove the volume group, and remove the physical volume and erase the physical volume. Then disable the osd. If all steps are successful return (), else it error’d out somewhere. |
| remove\_filestore\_osd(&self, dev\_path: &Path, simulate: bool) -> BynarResult<()>  DESCRIPTION: Remove a bluestore OSD to the Ceph Cluster  PARAMETERS: dev\_path – the device path of the OSD  simulate – if passed skip execution of the function  RETURNS: Ok(()) on success, Error otherwise  IMPLEMENTATION: get the mountpoint of the dev path and get the osd\_id. Set the osd as out, remove it from the crush, delete the osd auth key, and remove the osd. Then, wipe the disk by erasing the block device. If all steps are successful return (), else it error’d out somewhere. |
| resolve\_lvm\_device(&self, lv\_dev\_name: &Path) -> BynarResult<PathBuf>  DESCRIPTION: Resolve the lvm device name to an absolute path, since the lvm device name is a symlink, so it needs to be resolved to an absolute path to do anything with it.  PARAMETERS: lv\_dev\_name – the device name of the lvm  RETURNS: Ok(Lvm Absolute Path) on success, Error otherwise  IMPLEMENTATION: read the symlink. If it is a relative path, get its parent and the relative path to its parent, and canonicalize it, which returns the canonical, absolute form of a path with all intermediate components normalized and symbolic links resolved. If all steps are successful return the absolute path, else it error’d out somewhere. |
| select\_journal(&self) -> BynarResult<Option<JournalDevice>>  DESCRIPTION: Find a journal device that has enough free space if there is one  PARAMETERS:  RETURNS: Ok(Some(JournalDevice)) or Ok(None) on success, Error otherwise  IMPLEMENTATION: get the journal size from the Rados config. Convert it from MB to bytes. Get the journal devices from the ceph.json and sort them by the number of partitions. Iterate over the journal devices and remove the devices that are too small, and take the first journal with enough space. If all steps are successful, return Ok(Some(JournalWithEnoughSpace)) or Ok(None) if there are no journals with enough space, else it error’d out somewhere. |

##### Trait Implementation

###### Backend

|  |
| --- |
| Trait Function Definition |
| add\_disk(&self, device: &Path, id: Option<u64>, simulate: bool) -> BynarResult<()>  DESCRIPTION: Add a disk to the Cluster  PARAMETERS: device – the device path of the disk to add  id – an optional id to give the osd  simulate – if passed, skip the evaluation of this function  RETURNS: Ok(()) on success, Error otherwise  IMPLEMENTATION: If the Ceph Version is >= Luminous, then run add\_bluestore\_osd. Otherwise, run add\_filestore\_osd. If all steps are successful return (), else it error’d out somewhere. |
| remove\_disk(&self, device:&Path, simulate: bool) -> BynarResult<()>  DESCRIPTION: remove a disk from the Cluster  PARAMETERS: device – the device path of the disk to add  simulate – if passed skip execution of the function  RETURNS: Ok(()) on success, Error otherwise  IMPLEMENTATION: check if the Ceph Version is >= Luminous. If so, run remove\_bluestore\_osd. Otherwise, run remove\_filestore\_osd. If all steps are successful return (), else it error’d out somewhere. |
| safe\_to\_remove(&self, \_device:&Path, \_simulate:bool) -> BynarResult<()>  DESCRIPTION: check if a disk is safe to remove from the cluster  PARAMETERS: device – the unused device path of the disk to remove  simulate – if passed skip execution of the function  RETURNS: Ok(True) or Ok(False)on success, Error otherwise  IMPLEMENTATION: Create a DiagMap and run an exhaustive check. If all steps are successful, then return true if the Status is Safe, return false if the Status is NonSafe or Unknown, otherwise the function error’d out somewhere. |

#### JournalDevice

A Journal Device

##### Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| device | PathBuf | The device name? Device path??? |
| partition\_id | Option<u32> | The id of the partition |
| partition\_uuid | Option<uuid::Uuid> | The user? Unique? id of the partition |
| num\_partitions | Option<usize> | The number of partitions in the Journal Device |

##### Implementation

|  |
| --- |
| Function Definition |
| update\_num\_partitions(&mut self) -> BynarResult<()>  DESCRIPTION: Discover the number of partitions on the device and update the num\_partitions field  PARAMETERS:  RETURNS: Ok(CephBackend) on success, Error otherwise  IMPLEMENTATION: use GPT to get the number of partitions from the partition table, and update the num\_partitions field. If all steps are successful, then return (), else the function error’d out somewhere |

##### Trait Implementation

###### Display

|  |
| --- |
| Trait Function Definition |
| fmt(&self, f: &mut fmt::Formatter) -> fmt::Result  DESCRIPTION: format the journal device for printing as a string/displaying as a string  PARAMETERS: f: some formatter  RETURNS: Ok(()) on success, fmt::Result error type otherwise  IMPLEMENTATION: if there is a partition\_id, display the device and the id, otherwise just display the device. |

###### Clone, Debug, Deserialize, PartialEq

#### CephConfig

The ceph configuration object descriptor

##### Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| config\_file | String | The location of the ceph.conf file |
| user\_id | String | The cephx user to connect to the Ceph service with |
| journal\_devices | Option<Vec<JournalDevice>> | The /dev/xxx devices to use for journal partitions. Bynar will create new partitions on these devices as needed if no journal\_partition\_id is given. |

##### Trait Implementation

###### Deserialize, Debug

### Helper Functions

|  |
| --- |
| Helper Function Definition |
| choose\_ceph\_config(config\_dir: Option<&Path>) -> BynarResult<PathBuf>  DESCRIPTION: get the path of the ceph.json file.  PARAMETERS: config\_dir – an optional path to the configuration directory  RETURNS: Ok(ceph.json path) on success, Error otherwise  IMPLEMENTATION: check if a config\_dir was provided. If so, check the directory for a ceph.json file. If a config\_dir is not provided, check in the Home directory under the .config directory for the ceph.json file. If the function was successful return Ok(ceph.json path) else the function error’d somewhere. |
| get\_osd\_id\_from\_path(path: &Path) -> BynarResult<u64>  DESCRIPTION: A fallback function to get the osd id from the mount path. Note, is not 100% accurate but will work for most cases unless the disk is mounted in the wrong location or is missing the osd id in the path name  PARAMETERS: path – the mount path  RETURNS: Ok(osd id) on success, Error otherwise  IMPLEMENTATION: get the last part of the path (file or directory name). If successful, split the name by ‘-’, and the osd-id SHOULD be the second item in the list created by the split. If the function was successful return Ok(osd\_id) else the function error’d somewhere. |
| get\_osd\_id(path: &Path, simulate: bool) -> BynarResult<u64>  DESCRIPTION: Get the osd id from the whoami file in the osd mount directory  PARAMETERS: path – the osd mount directory  RETURNS: Ok(osd id) on success, Error otherwise  IMPLEMENTATION: make the path to the whoami file, and read the whoami file. Contained in the whoami file should be the osd\_id, so convert that into a u64 and return it. if the function is successful return Ok(osd\_id), else the function error’d somewhere |
| save\_keyring(osd\_id: u64, key: &str, uid: Option<u32>, gid: Option<u32>, simulate: bool) -> BynarResult<()>  DESCRIPTION: save a Ceph authentication key to a keyring file (Note: as of now it also overwrites the keyring file every time....)  PARAMETERS: osd\_id – the osd id  key – the authentication key  uid – the uid of the user who will own the keyring file  gid – the gid of the group that will own the keyring file  simulate – if passed, skip the execution of the function  RETURNS: Ok(()) on success, Error otherwise  IMPLEMENTATION: convert the uid and guid into Uid and Gid types. Get the path to the base directory and check if it exists. If so, create the keyring file and write the key to the created file, and chown it to the uid and gid. If successful, return Ok(()), otherwise the function error’d out. |
| add\_osd\_to\_fstab(device\_info: &block\_utils::Device, osd\_id: u64, simulate: bool) -> BynarResult<()>  DESCRIPTION: add the osd to the file systems table (fstab)  PARAMETERS: device\_info: device information gathered from udev  osd\_id – the osd id  simulate – if passed, skip the execution of the function  RETURNS: Ok(()) on success, Error otherwise  IMPLEMENTATION: get the default value of the fstab (default path is /etc/fstab). Create an entry for the fstab, filling in the attributes: the device id for the fs\_spec, the mount point of the osd, the filesystem type, and the mount options, the dump, and fsck\_order. Add the entry to the fstab. If the function is successful, return Ok(()), else the function error’d somewhere |
| partition\_in\_use(partition\_uuid: &uuid::Uuid) -> BynarResult<bool>  DESCRIPTION: Look through all the /var/lib/ceph/osd/ directories and check if there is a matching partition id to the input id.  PARAMETERS: partition\_uuid – the uid of the partition to check  RETURNS: Ok(partition in use or not) on success, Error otherwise  IMPLEMENTATION: for each osd in the system, get the journal symlink and do a sanity check on the journal symlink. Get the metadata of the symlink and do another sanity check. resolve the symlink path to get the device and probe it. Get the partition uid from the device and compare to the input path. If the same, then return Ok(true), if not the same return Ok(false), otherwise it error’d |
| systemctl\_disable(osd\_id: u64, osd\_uuid: &uuid::Uuid, simulate: bool) -> BynarResult<()>  DESCRIPTION: run the systemctl disable command on an osd  PARAMETERS: osd\_id – the id of the osd  osd\_uuid – the user id? Of the osd  simulate – if passed, skip the execution of the function  RETURNS: Ok(()) on success, Error otherwise  IMPLEMENTATION: Create the command arguments, and create a new Command to run the systemctl command. If the command is successful, return Ok(()), else it error’d |
| systemctl\_enable(osd\_id: u64, osd\_uuid: &uuid::Uuid, simulate: bool) -> BynarResult<()>  DESCRIPTION: run the systemctl enable command on an osd  PARAMETERS: osd\_id – the id of the osd  osd\_uuid – the user id? Of the osd  simulate – if passed, skip the execution of the function  RETURNS: Ok(()) on success, Error otherwise  IMPLEMENTATION: Create the command arguments, and create a new Command to run the systemctl command. If the command is successful, return Ok(()), else it error’d |
| systemctl\_stop(osd\_id: u64, simulate: bool) -> BynarResult<()>  DESCRIPTION: run the systemctl disable command on an osd  PARAMETERS: osd\_id – the id of the osd  simulate – if passed, skip the execution of the function  RETURNS: Ok(()) on success, Error otherwise  IMPLEMENTATION: Create the command arguments, and create a new Command to run the systemctl command. If the command is successful, return Ok(()), else it error’d |
| setup\_osd\_init(osd\_id: u64, osd\_uuid: &uuid::Uuid, simulate: bool) -> BynarResult<()>  DESCRIPTION: initialize (start) the osd after having prepared the osd (it should be down and in) and be up and in once the function is run successfully  PARAMETERS: osd\_id – the id of the osd  simulate – if passed, skip the execution of the function  RETURNS: Ok(()) on success, Error otherwise  IMPLEMENTATION: check which daemon is running on the system to use the correct command. If the daemon is Systemd, use the systemctl command to start the osd and check the output. If the daemon is Upstart, then use the start command to start the osd and check the output. If the function is successful Ok(()) is returned, otherwise it error’d out somewhere. |
| settle\_udev() -> BynarResult<()>  DESCRIPTION: wait for udevd to create device nodes for all detected devices  PARAMETERS: NONE  RETURNS: Ok(()) on success, Error otherwise  IMPLEMENTATION: run the udevadm command with the argument “settle”. If successful, return Ok(()), else error. |
| ceph\_mkfs(osd\_id: u64, journal: Option<&JournalDevice>, bluestore: bool, monmap: Option<&Path>, osd\_data: Option<&Path>, osd\_uuid: Option<&uuid::Uuid>, user\_id: Option<&str>, group\_id: Option<&str>, simulate: bool) -> BynarResult<()>  DESCRIPTION: Run ceph-osd –mkfs and return the osd UUID  PARAMETERS: osd\_id – the id of the osd  journal – a JournalDevice if it is used by the OSD  bluestore – whether the OSD is a bluestore or filestore  monmap – optional path to the monmap  osd\_data – optional path to the osd data directory  osd\_uuid – optional user id of the osd?  user\_id – the optional user id permissions of the OSD  group\_id - the optional group id permissions of the OSD  simulate – if passed, skip the execution of the function  RETURNS: Ok(()) on success, Error otherwise  IMPLEMENTATION: create the arguments to the ceph-osd –mkfs command. Add more arguments depending on the contents of the input, and run the ceph-osd command. If successful, return Ok(()), else it error’d |
| ceph\_bluestore\_tool(device: &Path, mount\_path: &Path, simulate: bool) -> BynarResult<()>  DESCRIPTION: Prime a bluestore osd, generating the content for an osd data directory that can start up a bluestore osd  PARAMETERS: device – the path to the osd device  mount\_path – the mount path of the osd  simulate – if passed, skip the execution of the function  RETURNS: Ok(()) on success, Error otherwise  IMPLEMENTATION: create the arguments of the ceph-bluestore-tool command. Run the command. If the command is successful, return Ok(()), else it error’d |
| create\_journal(name: &str, size: u64, path: &Path) -> BynarResult<(u32, uuid::Uuid)>  DESCRIPTION: create a new ceph journal on a given device with the name and size in bytes  PARAMETERS: name – the name pf the ceph journal  size – the size of the journal in bytes  path – the path of the journal  RETURNS: Ok((partition id, partition group id)) on success, Error otherwise  IMPLEMENTATION: open the GPT (GUID partition table) in writable mode and inspect the path in the GPT. Add a new partition to the GPT of type CEPH JOURNAL and write it to the disk. update the partition cache and read it back into the GPT, then check if the partition was added to the GPT. If everything runs successfully return Ok(partition id, partition guid), otherwise it error’d |
| enough\_free\_space(device: &Path, size: u64) -> BynarResult<bool>  DESCRIPTION: Check if there is enough free space on the disk to fit a partition size request  PARAMETERS: device – the path to the osd device  size – the size of the partition request  RETURNS: Ok(is there enough space?) on success, Error otherwise  IMPLEMENTATION: open the GPT and check the device path. Find the free sectors on the dish, and for each pair of free sectors, check if there is enough space (if the length of the free sector > the input size). If the function is successful, return Ok(true) if there is a sector with enough space, Ok(False) if there is no sector with enough space, otherwise there was an error |
| evaluate\_journal(journal: &JournalDevice, journal\_size: u64) -> BynarResult<JournalDevice>  DESCRIPTION: Attempt to discover if there is a device in the journal, create journal partition if needed, and return a path to use for the journal  PARAMETERS: journal – the journal to evaluate  journal\_size – the size of the journal partition  RETURNS: Ok(path to journal) on success, Error otherwise  IMPLEMENTATION: If the journal has a partition id, and a device, check if the partition exists and whether its in use by another osd. We can check using the GPT table, looping over the partitions to find the requested partition id, and check all the other osd’s for this partition id. If it is in use or there is no journal partition, create a new partition for the journal and update the number of partitions. If successful, return Ok(JournalDevice) with the updated partition values, otherwise it error’d somwhere. |
| remove\_unused\_journals(journals: &[JournalDevice]) -> BynarResult<()>  DESCRIPTION: Checks all osd drives on the system against the journals and delets all unused partitions. Note: unused  PARAMETERS: journals – the list of journals  RETURNS: Ok(()) on success, Error otherwise  IMPLEMENTATION: For each journal in the list, open the GPT and check the disk at the journal device. get all of the partitions on the disk, and check if each partition is in use. If not, mark it as unused and save and update the partitions, and write all changes to the disk. If successful, return Ok(()), otherwise it error’d somewhere |
| is\_filestore(dev\_path: &Path) -> BynarResult<bool>  DESCRIPTION: Checks if the specified OSD is a filestore  PARAMETERS: dev\_path – the device path of the osd  RETURNS: Ok(is a filestore?) on success, Error otherwise  IMPLEMENTATION: Get the mount point from the device path. If there isn’t a mountpoint, create a temporary osd mount point and mount the device. Add type to the path and check if the path exists. If so, check if the contents of the file contain “filestore”. If the function is successful and “filestore” type is found, return Ok(true), if successful and “filestore” is NOT found, return Ok(false), else it error’d |
| update\_partition\_cache(device: &Path) -> BynarResult<()>  DESCRIPTION: Linux specific ioctl to update the partition table cache  PARAMETERS: device – the device path  RETURNS: Ok(()) on success, Error otherwise  IMPLEMENTATION: Open the device and run blkrrpart. If successful return Ok(()), else it error’d |

# Database Schema

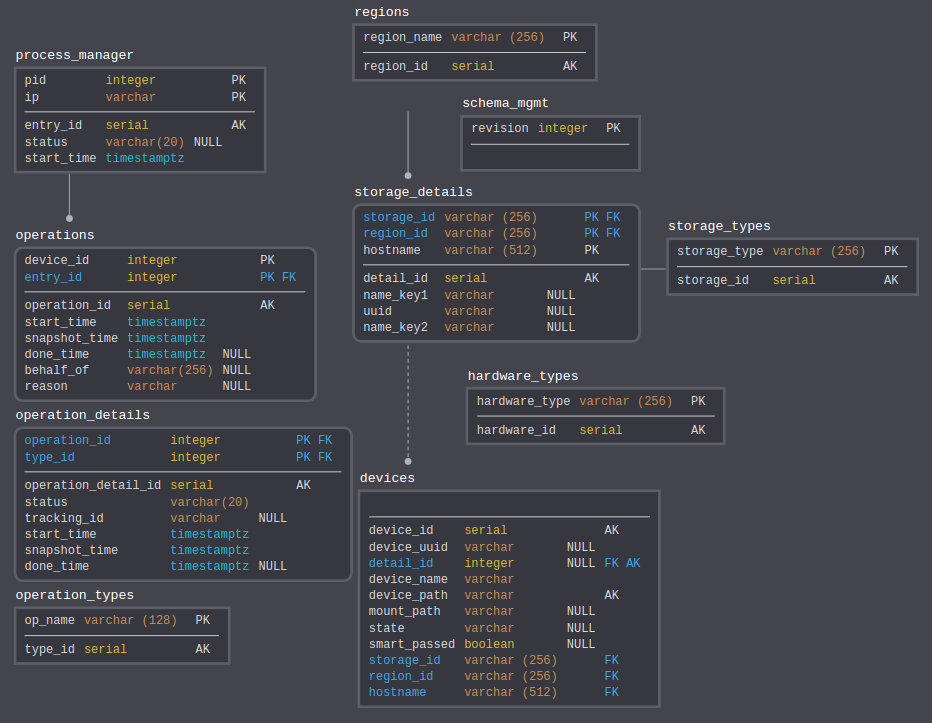
## Introduction

Bynar should have a database to log changes, errors, and other noteworthy messages. Currently Bynar only supports Postgres

## Postgres

In the dbschema folder, there is a bynar\_stats.sql file. You will need to import this into your Postgres Bynar Database. To import, you can run \i <path to file> from inside the psql prompt, or copy paste.

### Schema



# Database Logging

## Introduction

Most database logging functions are in the in\_progress.rs file

### Logging

# Helper Functions

## Introduction

There are a couple of functions and types that are needed across most of the Bynar program. These include the Error Type, host information, and various connection and requests.

## Error Module

The error module provides the error type for the Bynar program. Various error types are imported and generalized as a BynarResult Error

### Type

#### BynarResult<T>

This is the generic Bynar Errortype, a Result type of type <T, BynarError>

### Enums

#### PwdBError

##### Enum Values

|  |  |
| --- | --- |
| Name | Description |
| PwdError(PwdError) | An error from the pwd library |

##### Trait Implementations

###### Display

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Inputs | Description | Outputs |
| fmt | f: &mut fmt::Formatter | Given a PwBError, display the error msg | fmt::Result |

###### Debug

#### BynarError

##### Enum Values

|  |  |
| --- | --- |
| Name | Description |
| BlkidError(BlkidError) | A blkid command error |
| BlockUtilsError(BlockUtilsError) | A block\_utils library error |
| Error(String) | A generic String error |
| GojiError(GojiError) | A Gojira library error |
| HardwareError(HardwareError) | A hardware error |
| IoError(IOError) | A std::io error |
| LvmError(LvmError) | An lvm error |
| NixError(NixError) | A nix library error |
| ParseIntError(ParseIntError) | A parseint error (integer parser) |
| PostgresError(PostgresError) | A postgres command error |
| ProtobufError(ProtobufError) | A protobuf serializer error |
| PwdError(PwdBError) | A pwd error |
| R2d2Error(R2d2Error) | An R2d2 error |
| RadosError(RadosError) | A Ceph rados error |
| ReqwestError(ReqwestError) | A reqwest library error |
| SerdeJsonError(SerdeJsonError) | A serde json library error |
| SlackError(SlackError) | A Slack error |
| UuidError(UuidError) | A uuid error |
| VaultError(VaultError) | A vault error |
| ZmqError(ZmqError) | A zmq library error |

##### Implementation

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Inputs | Description | Outputs |
| new | err: String | Create a new BynarError with a String message | BynarError |
| to\_string | self | Convert a BynarError into a String representation | String |

##### Trait Implementations

###### Display

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Inputs | Description | Outputs |
| fmt | f: &mut fmt::Formatter | Given a Bynar, display the error msg | fmt::Result |

###### From<PwdError>

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Inputs | Description | Outputs |
| from | err: PwdError | Given a PwdError, create a BynarError | BynarError |

###### From<String>

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Inputs | Description | Outputs |
| from | err: String | Given a String, create a BynarError | BynarError |

###### From<’a str>

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Inputs | Description | Outputs |
| from | err: &str | Given a &str, create a BynarError | BynarError |

###### Debug, de::Error

### Structs

#### HardwareError

##### Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| error | String | The error |
| name | String | The name of the error |
| location | Option<String> | The location? Of the error |
| location\_format | Option<String> | Uh, the format?????? |
| serial\_number | Option<String> | Serial number of whatever is having issues? |

##### Trait Implementations

###### Display

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Inputs | Description | Outputs |
| fmt | f: &mut fmt::Formatter | Given a HardwareError, display the error msg | fmt::Result |

###### Debug

## Host Information

Gather information about the current host

### Enums

#### StorageTypeEnum

The type of distributed storage

##### Enum Values

|  |  |
| --- | --- |
| Name | Description |
| Ceph | Ceph storage type |
| Scaleio | Scaleio storage type |
| Gluster | Gluster storage type |
| Hitachi | Hitachi storage type |

##### Trait Implementations

###### Display

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Inputs | Description | Outputs |
| fmt | f: &mut fmt::Formatter | Given a StorageTypeEnum, display the storage type | fmt::Result |

###### Debug

### Structs

#### Host

##### Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| hostname | String | The host name |
| ip | IpAddr | The ip address |
| region | String | The region |
| kernel | String | The kernel type |
| server\_type | String | The server type |
| serial\_number | String | The serial number |
| machine\_architecture | String | The machine architecture |
| scsi\_info | Vec<block\_utils::ScsiInfo> | The scsi information |
| storage\_type | StorageTypeEnum | The storage type |
| array\_name | Option<String> | The array name |
| pool\_name | Option<String> | The pool name |

##### Implementation

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Inputs | Description | Outputs |
| new | N/A | Create a new Host | BynarResult<Host> |

##### Trait Implementations

###### Debug

### Helper Functions

|  |
| --- |
| Helper Function Definition |
| get\_default\_iface() -> BynarResult<String>  DESCRIPTION: get the default interface  PARAMETERS: None  RETURNS: Ok(default interface) on success, Error otherwise  IMPLEMENTATION: open the /proc/net/route file. For each line, try to find the default gateway “00000000” and return the interface. If successfule, return Ok(default interface) else error |
| get\_ip() -> BynarResult<IpAddr>  DESCRIPTION: Find the IP address on the default interface  PARAMETERS: None  RETURNS: Ok(ip address) on success, Error otherwise  IMPLEMENTATION: get all interfaces as well as the default interface. filter all interfaces to get the default. In the interface, loop through the ip addresses until an ipv4 address is found and return it. If successful, return the ipv4 address, else error. |
| get\_region\_from\_hostname(hostname: &str) -> BynarResult<String>  DESCRIPTION: Get the region from the hostname  PARAMETERS: hostname – the hostname  RETURNS: Ok(region) on success, Error otherwise  IMPLEMENTATION: Production hostnames are usually in the format name-regionpart1-regionpart2-\*, so split the hostname by ‘-’, skip the first sub string and combine the region parts. If successful, either return Ok(region) if totally successful, Ok(“test-region”) if the hostname is not regular, or error if it fails. |
| get\_storage\_type() -> BynarResult<StorageTypeEnum>  DESCRIPTION: get the storage type used on this system  PARAMETERS: None  RETURNS: Ok(storage type) on success, Error otherwise  IMPLEMENTATION: for now, it just returns Ceph.... |
| server\_type() -> BynarResult<String>  DESCRIPTION: Find the server type  PARAMETERS: None  RETURNS: Ok(server type) on success, Error otherwise  IMPLEMENTATION: Go to /sys/class/dmi/id/product\_name and read the file. If successful return the file contents as Ok(server type), else error |
| server\_serial() -> BynarResult<String>  DESCRIPTION: get the server serial number  PARAMETERS: None  RETURNS: Ok(server serial number) on success, Error otherwise  IMPLEMENTATION: for now, it just tries the easy way, which is reading the /sys/class/dmi/id/product\_serial file for the number. If successful returns Ok(server serial number), otherwise error |

## Helper Module

Public functions and structures that can be used outside of the library.

### Structs

#### ConfigSettings

##### Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| hostname | String | The host name |
| ip | IpAddr | The ip address |
| region | String | The region |
| kernel | String | The kernel type |
| server\_type | String | The server type |
| serial\_number | String | The serial number |
| machine\_architecture | String | The machine architecture |
| scsi\_info | Vec<block\_utils::ScsiInfo> | The scsi information |
| storage\_type | StorageTypeEnum | The storage type |
| array\_name | Option<String> | The array name |
| pool\_name | Option<String> | The pool name |

##### Trait Implementations

###### Clone, Debug, Deserialize

#### DBConfig

##### Attributes

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| username | String | Database username |
| password | Option<String> | Database password |
| port | u16 | Port to connect to database with |
| endpoint | String | Database endpoint |
| dbname | String | Database name |

##### Trait Implementations

###### Clone, Debug, Deserialize

### Helper Functions

|  |
| --- |
| Helper Function Definition |
| load\_config<T>(config\_dir: &Path, name: &str) -> BynarResult<T>  DESCRIPTION: load a config file that is deserializable  PARAMETERS: config\_dir – the directory of the config file  name – name of the file to deserialize  RETURNS: Ok(deserialized structure) on success, Error otherwise  IMPLEMENTATION: create the path to the file, and check if it exists. Read the file and deserialize it into the struct. If successfule, return Ok(deserialized struct) otherwise error out |
| connect(host: &str, port: &str, server\_publickey: &str) -> BynarResult<Socket>  DESCRIPTION: connect to the input host:port ip and securing with the server public key  PARAMETERS: host – the host ip address  port – the port to connect to  server\_publickey – the public key of the server used to secure the socket  RETURNS: Ok(connected socket) on success, Error otherwise  IMPLEMENTATION: create a new zmq REQ socket. create a curveKeyPair to secure the socket. set the keys in the socket and connect using tcp to the host:port ip address. If successful, return Ok(REQ socket), otherwise error out. |
| get\_vault\_token(endpoint: &str, token: &str, hostname: &str) -> BynarResult<String>  DESCRIPTION: get the vault secret from the Hashicorp Vault  PARAMETERS: endpoint – the hashicorp endpoint  token – token to access the vault with  hostname – name of the host to get the secret of  RETURNS: Ok(vault secret??) on success, Error otherwise  IMPLEMENTATION: Connect to the Vault with VaultClient, and get the secret. If successful return Ok(Vault secret) else error |
| add\_disk\_request(s: &mut Socket, path: &Path, id: Option<u64>, simulate: bool) -> BynarResult<()>  DESCRIPTION: send a request to add a disk to a cluster  PARAMETERS: s – the socket to send and receive messages from  path – the path of the disk to add to the cluster  id – the osd id of the disk to add  simulate – if passed, skip evaluation  RETURNS: Ok(()) on success, Error otherwise  IMPLEMENTATION: Create the Operation message. Convert the message into bytes and send it from the socket and wait for a response. Parse the Operation result for OK or ERROR. If successful, return Ok(()), otherwise something failed. |
| list\_disks\_request(s: &mut Socket) -> BynarResult<Vec<Disk>>  DESCRIPTION: send a request to get a list of disks from a cluster  PARAMETERS: s – the socket to send and receive messages from  RETURNS: Ok(disk list) on success, Error otherwise  IMPLEMENTATION: Create the Operation message. Convert the message into bytes and send it from the socket and wait for a response. Parse the Operation result for the list of disks. If successful, return Ok(disk list), otherwise something failed. |
| safe\_to\_remove\_request(s: &mut Socket, path: &Path) -> BynarResult<bool>  DESCRIPTION: send a request to a cluster to ask if a disk is safe to remove  PARAMETERS: s – the socket to send messages from  path – the path of the disk to check if removable  RETURNS: Ok(is safe to remove?) on success, Error otherwise  IMPLEMENTATION: Create the Operation message. Convert the message into bytes and send it from the socket and wait for a response. Parse the Operation result for whether the disk is safe to remove. If successful, return Ok(true) if safe to remove, Ok(false) if the disk is not safe to remove, otherwise something failed so error out. |
| remove\_disk\_request(s: &mut Socket, path: &Path, id: Option<u64>, simulate: bool) -> BynarResult<()>  DESCRIPTION: send a request to remove a disk from a cluster  PARAMETERS: s – the socket to send messages from  path – the path of the disk to add to the cluster  id – the osd id of the disk to add  simulate – if passed, skip evaluation  RETURNS: Ok(()) on success, Error otherwise  IMPLEMENTATION: Create the Operation message. Convert the message into bytes and send it from the socket and wait for a response. Parse the Operation result for OK or ERROR. If successful, return Ok(()), otherwise something failed. |
| get\_jira\_tickets(s: &mut Socket) -> BynarResult<()>  DESCRIPTION: send a request to get Jira tickets  PARAMETERS: s – the socket to send messages from  RETURNS: Ok(()) on success, Error otherwise  IMPLEMENTATION: Create the Operation message. Convert the message into bytes and send it from the socket and wait for a response. Parse the Operation result for OK or ERROR. If Ok get the tickets and print the ticket information. If successful, return Ok(()), otherwise something failed. |

# Client

## Introduction

### Client Interface

# Support Tickets

## Introduction

### JIRA Support

# Disk Manager

## Introduction

### Disk Manager

# Disk Testing

## Introduction

### State Machine

# Hardware Testing

## Introduction

### Hardware Tests

# Bynar

## Introduction

### Main Process