# MDADM MANUAL

Welcome to the MDADM Library documentation! This library provides a simple yet efficient interface for working with JBOD (Just a Bunch Of Disks) storage architecture. It offers various functionalities to interact with the storage system, including caching for improved performance and remote JBOD server connections. The library has been designed to be user-friendly and easy to integrate into your projects.

## Table of Contents:



This manual has been designed to provide comprehensive guidance on using the MDADM Library. To make the most of this manual, follow the Table of Contents and explore each function in detail. By understanding the usage, parameters, and return values of each function, you'll be able to seamlessly integrate the mdadm library into your projects.

For the best experience, ensure you follow the guidelines and constraints mentioned for each function to prevent errors or issues.

```
int mdadm_mount()
```

### Description:

The mdadm_mount() function is used to mount the JBOD drives. This function ensures that the drives are only mounted once and properly initializes the system.

### Return values:

- **1**: The function returns **1** when the JBOD drives are successfully mounted.
- **-1**: The function returns **-1** if the drives are already mounted.

```
int mdadm_unmount()
```

**Description:**

The mdadm_unmount() function is used to unmount the JBOD drives. This function ensures that the drives are unmounted properly and the system is updated accordingly.

**Return values:**

- **1**: The function returns **1** when the JBOD drives are successfully unmounted.
- **-1**: The function returns **-1** if the drives are not currently mounted.

```
int mdadm_read(uint32_t addr, uint32_t len, uint8_t *buf)
```

### Description:

The mdadm_read() function is used to read data from the JBOD storage system. It reads len bytes into the buf buffer, starting at the linear address addr.

### Parameters:

- uint32_t addr: The starting linear address to read data from.
- uint32_t len: The number of bytes to read.
- uint8_t *buf: The buffer to store the read data.

### Return values:

- **len**: The function returns **len** when the data is read successfully.
- **-1**: The function returns **-1** in case of any error, such as:
    - Drives not mounted.
    - len being larger than 1,024 bytes.
    - Out-of-bound linear address.
    - Invalid buffer.

### Usage:

```
uint32_t addr = 5000;
uint32_t len = 512;
uint8_t buf[512];

int result = mdadm_read(addr, len, buf);
if (result == len) {
    printf("Data read successfully.\n");
} else {
    printf("Error: Failed to read data.\n");
}
```

```
int mdadm_write(uint32_t addr, uint32_t len, uint8_t *buf)
```

**Description:**

The mdadm_write() function is used to write data to the JBOD storage system. It writes len bytes from the buf buffer, starting at the linear address addr.

**Parameters:**

- uint32_t addr: The starting linear address to write data to.
- uint32_t len: The number of bytes to write.
- const uint8_t *buf: The buffer containing the data to write.

**Return values:**

- **len**: The function returns **len** when the data is written successfully.
- **-1**: The function returns **-1** in case of any error, such as:
    - Drives not mounted.
    - len being larger than 1,024 bytes.
    - Out-of-bound linear address.
    - Invalid buffer.

**Usage:**

```
uint32_t addr = 5000;
uint32_t len = 512;
const uint8_t buf[512] = { ... };

int result = mdadm_write(addr, len, buf);
if (result == len) {
    printf("Data written successfully.\n");
} else {
    printf("Error: Failed to write data.\n");
}
```

# `int cache_create(int num_entries)`

### Description:

The cache_create() function sets up a cache with a specified number of entries, managing the allocation and initialization process. Use this function to prepare a cache for use in your project, ensuring efficient data access and improved performance.

### Parameters:

- int num_entries: The number of cache entries to allocate.

### Return values:

- **1**: The function returns **1** when the cache is created successfully.
- **-1**: The function returns **-1** in case of any error, such as:
    - Cache already exists.
    - num_entries is less than 2 or greater than 4,096.

### Usage:

```
int num_entries = 100;

int result = cache_create(num_entries);
if (result == 1) {
    printf("Cache created successfully.\n");
} else {
    printf("Error: Failed to create cache.\n");
}
```

# int cache_destroy()

### Description:

The cache_destroy() function safely frees the cache memory and resets it to its initial state. Remember to call cache_create() before using this function. Calling cache_destroy() without creating the cache first or calling it twice in a row without an intervening cache_create() will result in an error.

### Return values:

- **1**: The function returns **1** when the cache is destroyed successfully.
- **-1**: The function returns **-1** in case of any error, such as:
    - Cache does not exist.

### Usage:

```
int result = cache_destroy();
if (result == 1) {
    printf("Cache destroyed successfully.\n");
} else {
    printf("Error: Failed to destroy cache.\n");
}
```

```
bool jbod_connect(const char *ip, uint16_t port)
```

### Description:

The jbod_connect() function is used to establish a connection to a JBOD server using the given IP address and port number.

### Parameters:

- const char *ip: The IP address of the JBOD server to connect to.
- uint16_t port: The port number of the JBOD server to connect to.

### Return values:

- **true**: The function returns **true** when the connection is established successfully.
- **false**: The function returns **false** in case of any error, such as:
    - Invalid IP address.
    - Error in socket creation.
    - Error in socket connection.

# Void jbod_disconnect()

### Description:

The jbod_disconnect() function is used to close the connection to the JBOD server that was established using the jbod_connect() function.

### Return values:

This function does not return any values.