IME

Developer Guide

Version 1.4 | 96-00450-001 | Revision A7

Information in this document is subject to change without notice and does not represent a commitment on the part of DataDirect Networks, Inc. No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose other than the purchaser's personal use without the written permission of DataDirect Networks, Inc.

June 2020

Changes in this Revision

Section	Description
Section 4 on page 18	Refactored **`ime-ctl`**. Updated to equivalent long options.
Appendix A on page 28	Refactored **`ime-ctl`**. Added equivalent long options.
Appendix C on page 40	Added **`IM_BULKDATA_POOL_TINY_MIN_MAX`** to specify the quantity of tiny-sized buffers that IME should use. Added **`IM_CLIENT_JOB_TRACKING_ENABLED`** to control per-job monitoring feature. Deprecated **`IM_NETWORK_STACK`**. Use the configuration file option **`network_stack`** instead. Added **`IM_CLIENT_HANDSHAKE_ENABLED`** to control client-server handshake.

Preface

Audience

This guide is intended for application developers porting legacy applications or developing new applications for use with the DataDirect Networks (DDN) Infinite Memory Engine software (IME®).

The level of content presented assumes that the developer is familiar with hierarchical storage systems and has expert knowledge of Portable Operating System Interface (POSIX) or Message Passing Interface-Input/Output (MPI-IO).

About this Guide

This guide provides information about porting your existing application or developing new applications for use with IME. It includes an overview of IME; details about IME Client including the I/O interfaces, utilities, and Application Programming Interface (API); information about the development environment; descriptions of typical use cases; and examples of integration with 3rd party system tools.

For details about supported configurations, refer to the *IME Compatibility Guide*.

Conventions

Throughout the document, links are provided in the format (See Section X on page y.), which provide additional or related details about the topic.

Related Documentation

The following documents are additional sources of information for IME:

- IME Installation and Administration Guide
- IME Product Release Notes
- IME Compatibility Guide

The following documents describe the IME hardware platforms:

- IME240 Hardware Installation and Maintenance Guide
- IME140 Hardware Installation and Maintenance Guide
- IME14K Hardware Installation and Maintenance Guide for IME 1.0.0

The latest version of the documentation is available on the Customer Support Portal at:

https://community.ddn.com/login

Table of Contents

1. Getting Started

PREVIEW FEATURE:	Capabilities listed in this document as "preview features" are disabled in IME 1.3 and later by default. To enable any one of these features, contact your DDN Sales Representative or Field Support Engineer. Note that these "preview features" are not supported for production use and that hot-fixes will not be provided for issues found. For a complete list of "preview features", refer to the *IME Release Notes*.

This guide provides information about porting your existing application or developing new applications for use with Infinite Memory Engine (IME®). It includes the following sections:

- **IME Software Architecture** – Overview of the IME software architecture. (Section 2 on page 7.)
- **IME Client** – Description of IME Client, including I/O interfaces, file paths and URIs, and dockerized IME client. (Section 3 on page 11.)
- **Data Residency Control** – Explanation and examples of controlling data residency from the command line using IME utilities or from your application using IME Native API. (Section 4 on page 18.)
- **Integrating with Job Schedulers** – Definition of scripts for integration with job schedulers. (Section 5 on page 25.)
- **IME Command-line Utilities** – List of command-line utilities including IME Control, stand-alone, and general-use utilities. (Appendix A on page 28.)
- **IME Native API Reference** – List of IME Native API, which is a non-POSIX compliant, low-level I/O protocol that bypasses the kernel communicating directly with IME Client. (Appendix B on page 36.)
- **IME Environment Variables** – List of IME environment variables, generally used for IME Native API and MPI-IO enabled applications. (Appendix C on page 40.)

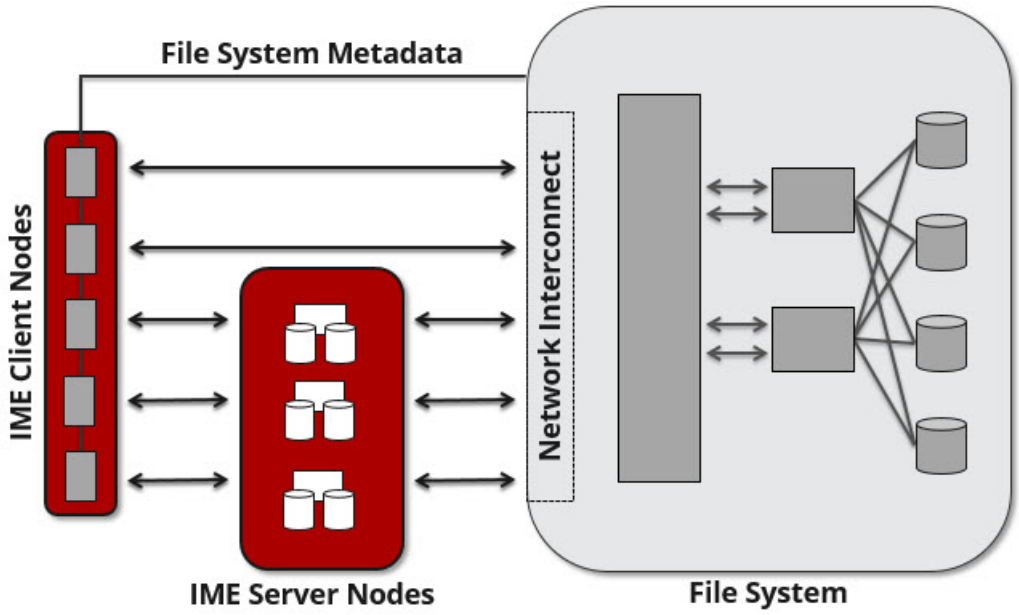
Get started developing applications for IME by following these steps:

1. Become familiar with the IME software architecture. (Section 2 on page 7.)
2. Determine the IME Client I/O interface that best meets your needs and set up the appropriate development environment. (Section 3 on page 11.)
3. Implement data residency control using IME CLI utilities or IME Native API. (Section 4 on page 18.)
4. Use IME scripts to integrate with job schedulers. (Section 5 on page 25.)

2. IME Software Architecture

Infinite Memory Engine (IME®) adds a fast data tier between the compute nodes and file system in a high-performance computing environment, as shown in Figure 1. This software-defined storage appliance transparently integrates non-volatile memory with existing file systems used for long-term data storage. IME enables the fastest possible ingest of data to the non-volatile memory by allowing the data to be stored in an unstructured, dynamically load balance-able manner. It then performs the orderly, transfer of data from the non-volatile memory to permanent storage. The result is accelerated I/O, reduced latency, consistent application performance, and greater operational and economic efficiency.

Figure 1. Infinite Memory Engine Cluster



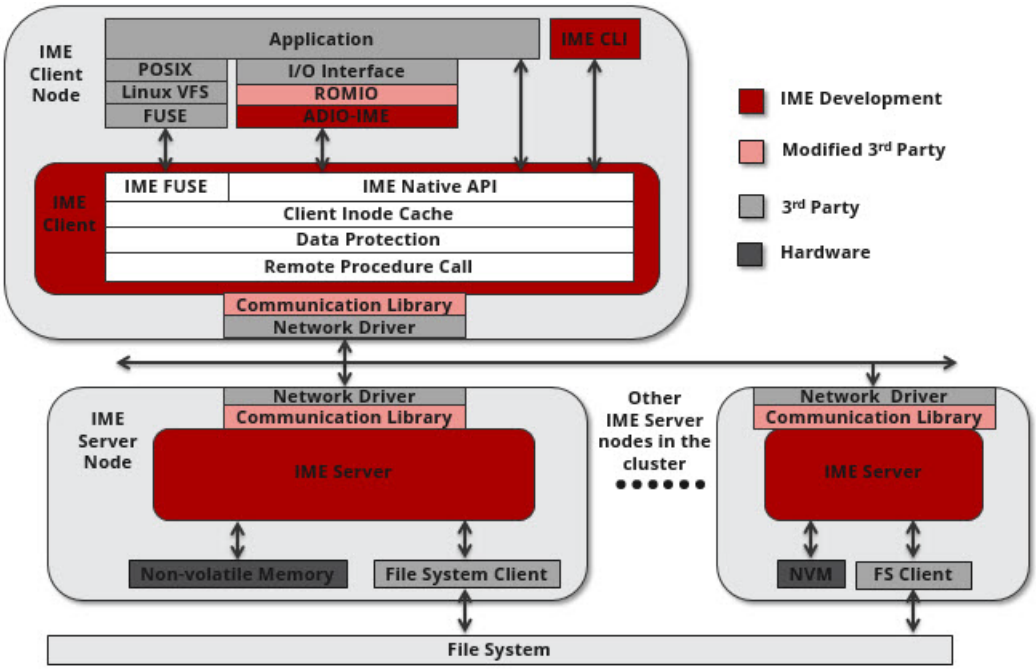
NOTE: For an overview of the IME cluster and a list of IME capabilities, refer to the *IME Installation and Administration Guide*

For details about supported configurations, including operating system versions, firmware versions, and network fabric interconnects, refer to the *IME Compatibility Guide*.

2.1 Client-Server Architecture

IME implements a client-server architecture, which consists of the IME Client and IME Server processes. Figure 2 shows the IME software architecture.

Figure 2. Infinite Memory Engine Software Architecture



NOTE: For a complete list of the I/O interfaces supported, refer to the *IME Compatibility Guide*.

The IME Server process (**`ime-server`**) resides on the server nodes, which constitute the new fast data tier. The server process handles requests from IME Client and the BFS, in addition to managing data resident in IME. IME is a non-deterministic write system, which means that if a server becomes saturated IME Client will automatically re-direct the data traffic to another server in the cluster. **`ime-server`** creates a tight coupling between the data devices and the traditional file system to leverage the scale-out capacity characteristics of these file systems.

IME Client is a library that manages connections and routes I/O requests to IME Server nodes. This library can be used directly (IME Native API), as part of a process that implements FUSE file system (POSIX), or as part of the MPI driver (MPI-IO). IME Client implements an advanced, non-blocking, low-level I/O protocol that allows applications to take full advantage of the performance of the underlying data devices. However, applications require no modification and behave as if they are accessing the file data in the same way that they would without IME.

2.2 Data Consistency Model

Users must maintain close-to-open consistency when multiple clients access the same files. This requirement guarantees that any other client will see the latest changes made by one client as soon as the client opens the file. A client must synchronize all file data and metadata changes when it closes a file and unconditionally retrieve a file's attributes when it opens a file, ignoring any information it may have cached about the file. IME implements an enhanced close-to-open consistency model, allowing IME to be lock free.

Unsynchronized data within an IME client's cache are not visible or retrievable by other clients. By default, files managed through the IME Native API are fsync'd within **`close()`**/**`ime_client_native2_close()`** in a blocking manner. Synchronizing on close ensures that IME clients will provide close-to-open consistency. Errors from fsync-on-close are returned in **`close()`**/ **`ime_client_native2_close()`**.

IME's client cache management should not affect distributed applications that perform shared file operations on write-only and read-only file descriptors. This holds true for write-only files where applications issue writes into adjacent (or overlapping), non-page aligned regions. IME provides byte-level granularity for all write operations - serializing input into any file region is not required. Overlapping writes originating from more than one client instance are applied in the order in which they are received.

Distributed applications performing read-write shared file operations must issue **`fsync()`** or a library equivalent before attempting to read data that may have been resident in another IME client's write-back cache. To assure that no data resides in the IME clients' write-back caches, for those IME clients that are part of the job context, synchronize the cached data belonging to your files and do not write new data to those files. If another job is reading or writing those same files, unexpected results may occur. Note IME Client will synchronize all cached data for the respective file when **`fsync()`** or a library equivalent is called. Applications that do not explicitly synchronize may see inconsistent file views.

Applications that do not wish to wait for **`ime_client_native2_fsync()`** to complete may synchronize asynchronously by calling **`ime_client_native2_async_fd()`** on their respective IME file descriptors. Enabling asynchronous operation on the file descriptor will cause **`ime_client_native2_close()`** to return without acknowledgment from the IME storage service. **`ime_client_native2_finalize()`** will cause all asynchronous file descriptors to be simultaneously synchronized and an error from any synchronization failure will be returned here.

For details about the IME Native API, Appendix B on page 36.

2.3 Simultaneous Access to File Data

In addition to accessing file data using IME, users are able to access their files directly through the BFS mount point. No guard exists to prevent this access. For read-only workloads applications, use either the IME or BFS mount point.

When writing to the IME service, applications and users should not expect the BFS to have intact files by default. IME will synchronize its storage with the BFS according to the internal scheduling and prioritization of **`ime-server`**, unless explicitly directed. **`ime-server`** uses an algorithm for determining BFS synchronization order, which accounts for both data residency time and volume of data in the file. The order is not a strict FIFO.

To guarantee an up-to-date file view of the BFS, request a blocking operation when synchronizing IME file data. For details about manual synchronization, see Section 4.2 on page 20. IME provides the means to view synchronization status using the IME CLI utilities. If unsynchronized data is present, do not attempt direct access through the BFS. For details about data residency, see Section 4 on page 18.

Users should not expect IME automatically to detect file changes done directly to the BFS if IME is accessing the same files. If applications write a file from IME and the BFS at the same time, unexpected results may occur. The application using IME will see the updates that it wrote to IME and may see updates made directly to the BFS if the IME application has not written the same file regions as the BFS application. The application using the BFS will not see any data resident in IME until that data is synchronized with the BFS. During synchronization, IME will assume that it has exclusive access to the BFS file and overwrite any changes that other users of that file made.

One general rule to keep in mind is that if IME has a metadata descriptor for a file chunk it will always read that file chunk from IME. The opposite is also true; any file region not represented by an IME metadata descriptor will be read from the BFS. File regions that are fetched from the BFS on behalf of application **`read()`** are not copied to the IME SSDs. Therefore, no metadata descriptors are made for them. Consider the scenario in which an application simultaneously executes random writes and sequential reads. For random writes, the write offset is generated randomly and is not sequential. As reads are occurring on a sequential offset, the application may come across a case where the write to IME has not occurred yet. In such cases, the data will be read from the BFS.

For additional details about parallel file system coherency, see Section 4.1 on page 19.

3. IME Client

IME Client is a library that manages connections and routes I/O requests to IME Server nodes. This library can be used directly (IME Native API), as part of a process that implements FUSE file system (POSIX), or as part of a driver (MPI-IO), as described in the following sections.

3.1 I/O Interface

3.1.1 POSIX

IME supports transparent access through a POSIX mount point, implementing one layer of abstraction between your application and IME Client. The functional chain

Application → POSIX mount point → IME Client → IME Server

can easily be implemented using FUSE (Filesystem in Userspace) via an IME FUSE client

Application → Kernel → IME FUSE → IME Client → IME Server

> **NOTE:** IME includes a FUSE client to present a POSIX mount point to the userspace. For details about the IME FUSE client, including how to start and stop the client, refer to the *IME Installation and Administration Guide*.

Because POSIX does not bypass the kernel, it is less performant than using MPI-IO or IME Native API. IME imposes no performance penalty for using one file in your parallel job instead of many files. However, FUSE imposes a scaling limitation for a single client.

IME applications that use POSIX run without special code modifications. Nonetheless, you must ensure that the IME I/O semantics work with your application. IME is compatible with the POSIX API with the relaxed constraint that IME does not give visibility into unsynchronized cache content of clients.

3.1.2 MPI-IO

MPI-IO provides a way to bypass the kernel, improving performance over POSIX. In addition, MPI-IO is more performant with small I/O files than POSIX. IME implements the functional chain

Application → MPI-IO → IME Client → IME Server

using a modified version of ROMIO, which is a high-performance, portable implementation of the MPI-IO specification.

Although MPI-IO applications recognize IME as a mount point without code modifications, you must build and run your application using an MPI tool chain that includes the IME shared library dependencies, which enable the IME client library to handle files designated by the IME file prefix. (See Section 3.3 on page 16.)

> **NOTE:** If you need an IME-compatible version of MPI, download the source code and compile it, as described in the *IME Installation and Administration Guide*.

> **NOTE:** When using MPI-IO or IME Native API from any MPI context, you must explicitly export **RDMAV_FORK_SAFE=1** and **RDMAV_HUGEPAGES_SAFE=1** in the **mpirun** command.
> Example: **mpirun -genv RDMAV_FORK_SAFE 1 -genv RDMAV_HUGEPAGES_SAFE 1**
> If these arguments are omitted, IME will report errors during initialization of the RPC layer.

3.1.3 IME Native API

IME Native API provides the best possible performance. This non-POSIX compliant, low-level I/O protocol bypasses the kernel, communicating directly with IME Client. IME Native API directly implements the functional chain

Application → IME Client → IME Server

To use the IME Native API I/O interfaces, you must link your applications with the DDN-supplied libraries, which enable the IME client library to handle files designated by the IME file prefix. (See Section 3.3 on page 16.)

Table 1 lists IME Native API functions that have equivalent POSIX I/O functions. For details about these API, see Appendix B.2 on page 36 or refer to the *Linux Programmer's Manual*.

Table 1. POSIX Equivalent IME Native API

IME Native API Function	Equivalent POSIX I/O Function
`ime_client_native2_close()`	`close()`
`ime_client_native2_dup2()`	`dup2()`
`ime_client_native2_fsync()`	`fsync()`
`ime_client_native2_lseek()`	`lseek()`
`ime_client_native2_open()`	`open()`
`ime_client_native2_pread()`	`pread()`
`ime_client_native2_pwrite()`	`pwrite()`
`ime_client_native2_read()`	`read()`
`ime_client_native2_stat()`	`stat()`
`ime_client_native2_ftruncate()`	`ftruncate()`
`ime_client_native2_unlink()`	`unlink()`
`ime_client_native2_write()`	`write()`

Table 2 lists IME Native API functions for asynchronous IO. For details about these API, see Appendix B.3 on page 37.

Table 2. IME Client Asynchronous Library

IME Native API Function
`ime_client_native2_aio_read()`
`ime_client_native2_aio_write()`

Table 3 lists IME Native API functions that do not have a POSIX equivalent. For details about these API, see Appendix B.4 on page 38.

Table 3. Non-POSIX Equivalent IME Native API

IME Native API Function
ime_client_native2_async_fd()
ime_client_native2_bfs_sync()
ime_client_native2_config_ctl()
ime_client_native2_data_release()
ime_client_native2_finalize()
ime_client_native2_frag_fstat()
ime_client_native2_get_ime_fsize()
ime_client_native2_init()
ime_client_native2_prestage()
ime_client_native2_prestage_blocking()
ime_client_native2_sync_ctl()

Example Application

Figure 3 gives an example application to synchronize data to the BFS using IME Native API.

Figure 3. Example Application

```
/*  #gcc -I /opt/ddn/ime/include -L /opt/ddn/ime/lib  -Wall -O2 -o \
ime_native_example-small ime_native_example-small.c -lim_client
#IM_CLIENT_CFG_FILE=/opt/ddn/ime/config/ime.conf ./ime_native_example-small  */
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>
#include <im_client_native2.h>

void my_exit(const char *msg, int ret)
{
    if (ret)
        fprintf(stderr, "%s : err=%s", msg, strerror(ret));
    ime_client_native2_finalize();
    exit(ret);
}

#define FILENAME "ime:///mnt/lustre_client/dir/testfile"
#define NUM_INTEGERS 255
#define IO_LENGTH (NUM_INTEGERS * sizeof(int))

int main(void)
{
    int fd;
    int buf[NUM_INTEGERS];
    ssize_t nbytes = 0, rc;

    /*  Initialize the IME native client.  */
    ime_client_native2_init();

    /*  Open a file via IME.  */
    fd = ime_client_native2_open(FILENAME, O_RDWR | O_CREAT, 0666);
    if (fd < 0)
        my_exit("open failed", errno);

    /*  Write a small amount of data to IME.  */
    while (nbytes < IO_LENGTH)
    {
        rc = ime_client_native2_write(fd, (const char *)buf, IO_LENGTH);
        if (rc < 0)
            my_exit("ime_client_native2_write failed", errno);
        nbytes += rc;
    }

    /*  Synchronize data to IME service.  */
    if (ime_client_native2_fsync(fd))
        my_exit("ime_client_native2_fsync failed", errno);

    /*  Initiate synchronization to the BFS. This operation is non-blocking.  */
    if (ime_client_native2_bfs_sync(fd))
        my_exit("ime_client_native2_bfs_sync failed", errno);
    if (ime_client_native2_close(fd))
        my_exit("ime_client_native2_close failed", errno);
    my_exit("done", 0);
}
```

3.2 Libraries

IME libraries are distributed with the RPMs, as listed in the *IME Installation and Administration Guide*.

When developing applications for use with IME ensure that you do the following:

- Include IME library directories in your path when building applications against IME.
- Use the library versions distributed with the IME RPMs and listed in the installation section of the *IME Installation and Administration Guide*.
- Link your IME native application to the **`libim_client.so`** library.
  For example: using **`-lim_client`**
- Use **`-I/opt/ddn/ime/include`** on the command line. The following is a typical setting:

  ```
  -I /opt/ddn/ime/include -L /opt/ddn/ime/lib -lim_client
  ```

To link against installed libraries in a given directory, use one of the following methods:

- Use libtool and specify the full pathname of the library.
- Use the **`-L/opt/ddn/ime/lib`** flag during linking and do at least one of the following:
  - Add **`/opt/ddn/ime/lib`** to **`LD_LIBRARY_PATH`** during execution.
    For important details, see LD_LIBRARY_PATH / rpath on page 16.
  - Add **`/opt/ddn/ime/lib`** to **`LD_RUN_PATH`** during linking.
  - Use the **`-Wl`**,**`-rpath -Wl`**, or **`-L/opt/ddn/ime/lib`** linker flag.
  - Have your system administrator add **`/opt/ddn/ime/lib`** to **`/etc/ld.so.conf`**.

Key Points

The following are key points with respect to the IME library RPMs:

- IME auto-detects the underlying network protocol and selects the correct network stack to use. Developers can change these defaults as follows:

  ```
  ime-server -n <stack>
  ime-fuse --network_stack=<stack>
  ```

- The preferred path to libcci and libfabric is encoded.
- The libisal RPM provided in the IME distribution provides a static library only and no longer needs to be installed. This new libisal RPM can be seen as a transitional package and will be removed in future IME versions. However, existing software already compiled against IME 1.3 or older (such as MPI) might still need the shared **`libisal.so`** file. In this case, recompile that software, do not update or un-install the older libisal RPM, or contact DDN for the old version.

IMPORTANT: For additional details about IME libraries, refer to the *IME Installation and Administration Guide*.

For additional fixed and known issues, refer to the latest *IME Release Notes*.

LD_LIBRARY_PATH / rpath

During compilation and linking of the IME project, the preferred path to libcci and libfabric is encoded into the IME libraries ime-net-libfabric and ime-net-cci, which are used by libim_common for network communication. It is important to note that, in general, the **LD_LIBRARY_PATH** environment variable has a higher path preference than the rpath encoded into the binaries.

System users and their application users that use the following are most likely affected:

- IME Native API applications
- MPI applications, using MPI-IO and linked against an MPI stack that supports IME-MPIIO

If users of such applications set an **LD_LIBRARY_PATH** and that environmental variable also points to libfabric or, more unlikely, to libcci, network communication between the IME Client nodes and IME Server nodes might fail. In this case, contact DDN for assistance.

3.3 File Paths and URIs

Table 4 shows the file path formats for accessing file data in IME.

Table 4. File Paths and URIs

	Relative	Absolute
IME client library	**ime://dir/myfile**	**ime:///fsmnt/bfs/dir/myfile**
Equivalent POSIX I/O function	**dir/myfile**	**/fsmnt/bfs/dir/myfile**
IME FUSE	n/a	**/imefusemnt/dir/myfile**

The IME file prefix designates file names for handling by the IME client library. IME supports relative and absolute paths, as follows:

- 3 slashes (**ime:///**) designate an absolute file rooted in the IME BFS top directory.
  For example: **ime:///fsmnt/bfs/dir/myfile**
- 2 slashes (**ime://**) designate a file relative to the IME BFS top directory, which is defined in the IME Configuration file.
  For example: **ime://dir/myfile**

  The root or fixed dirpath to which the relative path will be concatenated is the value of **mount_point** specified in the **ime_bfs** section of the IME Configuration file (**/etc/ddn/ime/ime.conf**).

NOTE: For details about **/etc/ddn/ime/ime.conf**, refer to the *IME Installation and Administration Guide*.

Example

Assume the value of **mount_point** specified in **/etc/ddn/ime/ime.conf** is **/fsmnt/bfs.**

In the case of **ime://dir/myfile**,

dir/myfile will be considered as a relative path in the BFS from the top directory **/fsmnt/bfs**.

Files not using the prefix are handled directly by the equivalent POSIX I/O function, as follows:

- 1 slash (**/**) designates an absolute path.
  For example: **/fsmnt/bfs/dir/myfile**
- no slashes designate a relative path.
  For example: **dir/myfile**

Key Points

- If a BFS is mounted, the IME file prefix (**ime://**) is not allowed with the FUSE path. The prefix is used for BFS path only.
- If a BFS is not mounted and FUSE is mounted, run the IME CLI utilities with the FUSE path.

3.4 Dockerized IME Client

IME supports the following uses of container with IME Client:

- IME Client inside container
- IME Client on host

3.4.1 IME Client Inside Container

To mount the IME FUSE client inside your container, install the binaries required for IME Client and the configuration files inside your container. For scripts to install and mount IME Client inside containers, refer to the **README.md** files located in the **scripts** folder.

3.4.2 IME Client On Host

To access "IME client mounted on host" inside container, mount IME client mountpoint inside container using docker volume feature, as shown by the following docker run parameter:

```
-v /<path_for_ime_client_on_host>:/<Path_inside_container>
```

4. Data Residency Control

PREVIEW FEATURE: Auto-prestage is a preview feature in IME 1.3 and later. To enable, contact your DDN Sales Representative or Field Support Engineer.

IME acts as a high-performance burst buffer and write-back cache. Buffering refers to an intermediate staging area for data while it transits to more distant storage locations, whereas caching is a staging area for data that may likely be re-used, perhaps extensively, before arriving at its final destination. In both cases, movement of data is defined by the data residency within IME.

It is important to understand the definitions of data movement used in this guide, in order to control data residency:

- **Read** – If the file data exists in IME, read from IME; otherwise, read from the BFS.
- **Write** – Write file data into IME. IME writes new or modified data, regardless of its original location, and marks it as **unsynchronized**.
- **Prestage** – Copy file data resident in the BFS into IME, prior to your job. IME marks pre-staged data as **synchronized**. By default, data read from the BFS is not prestaged in IME automatically.

PREVIEW FEATURE: When enabled, auto-prestage adds the capability to transparently promote file data from the BFS to IME in the case of cache miss.

- **Synchronize** – Copy file data resident in IME into the BFS, keeping the data in IME. IME marks data waiting for synchronization as **pending** and marks data as **synchronized** once the process is complete. Synchronized data can be removed safely from IME. ***Synchronizing*** refers to writing **unsynchronized** data to the BFS.
  - **Auto-Synchronize** – IME automatically copies file data resident in IME into the BFS, based on the policy set by the IME synchronization engine.
- **Purge** – Remove file data resident in IME, which may or may not be synchronized to the BFS. IME marks purged data as **deletable**, and it is then freed from IME. If synchronized, the data resident in the BFS remains intact. If not synchronized, the data will be lost.
  - **Auto-Purge** – IME automatically removes synchronized file data resident in IME to free up capacity, based on the policy set by the IME synchronization engine. IME ***only*** automatically purges data that has been synchronized already, as opposed to manual purge in which applications can purge any data.

NOTE: For a detailed discussion of the IME synchronization engine, refer to the *IME Installation and Administration Guide*.

Control data residency from the command line using IME CLI utilities or from your application using IME Native API, as summarized in Table 5.

Table 5. Data Residency Control Utilities and API

Data Residency Control	IME CLI Utility [note 1]	IME Native API Function [note 2]
Prestage (Section 4.1 on page 19)	**ime-prestage** **ime-ctl --prestage**	**ime_client_native2_prestage** **ime_client_native2_prestage_blocking**
Synchronize (Section 4.2 on page 20)	**ime-sync** **ime-ctl --sync**	**ime_client_native2_bfs_sync**
Purge (Section 4.3 on page 22)	**ime-release** **ime-ctl --purge**	**ime_client_native2_data_release**
File Pinning (Section 4.4 on page 23)	**ime-pin -M/-m** **ime-ctl --pin/--unpin**	**ime_client_native2_sync_ctl**

1. The **ime-ctl** utility is for use only on hosts that present the IME client mount to applications. For details about **ime-ctl**, see Appendix A.1 on page 31.
   For details about all other utilities, see Appendix A.2 on page 32.
2. For details about the IME Native API, see Appendix B on page 36.

4.1 Prestage

Prestage copies file data resident in the BFS into IME prior to your job, taking advantage of the IME device space as a cache. By pre-staging data in cache, a job can quickly retrieve and use the data it needs. Likewise, by reading large chunks of data from the BFS into IME, IME acts as a burst buffer, rapidly delivering smaller subsets of data to the application while reducing slower accesses to the BFS. The prestaged data is not removed from the BFS. It resides in IME and the BFS. To prestage data, use **ime-prestage** or **ime-ctl --prestage**.

Example

```
ime-prestage ime:///fsmnt/bfs/dir/myfile
or
ime-ctl --prestage /imefusemnt/dir/myfile
```

NOTE: By default, **ime-prestage** uses **/etc/ddn/ime/ime.conf** to establish communication with **ime-server**. To specify a different configuration file, use the **-c** option.

When the prestage command is issued for a file, IME will purge any existing synchronized IME file data prior to copying the file data resident in the BFS by default. To keep the existing synchronized IME file data on prestage, pass the **-K** option.

Example

```
ime-prestage -K ime:///fsmnt/bfs/dir/myfile
or
ime-ctl --prestage --keep /imefusemnt/dir/myfile
```

By default, these utilities are non-blocking. Passing the **`--block`** option with **`ime-prestage`** or **`ime-ctl --prestage`** creates a blocking operation, which guarantees all data is prestaged before any other operation can begin.

Example

`ime-ctl --prestage /imefusemnt/dir/myfile`
queues up data to be copied from the BFS and allows other operations to begin in parallel, while
`ime-ctl --prestage --block /imefusemnt/dir/myfile`
waits for data to be copied before allowing other operations.

NOTE: To prestage data using IME Native API, use **`ime_client_native2_prestage`** or **`ime_client_native2_prestage_blocking`**. (See Appendix B.4 on page 38.)

Parallel File System Coherency

File data that is prestaged from the BFS to IME will not be refreshed if that file data in the BFS are modified, unless explicitly requested by the user. Such a scenario may occur when the user first requests file pre-staging and then modifies the BFS file data afterwards. Applications using IME will see the version of the file that was originally prestaged. If you have file data cached in IME but know that the BFS file data is more recent or otherwise preferred, reissue the prestage command or purge the cached contents from IME.

When the prestage command is reissued for a file, IME will purge any existing synchronized IME file data prior to issuing new read requests to the BFS for the more recent file data. If you choose to purge the cached contents, first close all open file descriptors on the IME clients, then purge the file data using **`ime-release`** or **`ime-ctl --purge`**. The second step causes IME to release all data belonging to the file without synchronizing to the BFS. After the purge step has completed, IME will default to the BFS for all future read or prestage requests. If such a workflow is desired, you will likely want to exempt that file data from being automatically synchronized (or "pin") using **`ime-pin`** or **`ime-ctl --pin`**.

In addition, IME includes a stale file detector thread that checks periodically if there are any stale files in IME. Stale files are defined as files that have been deleted directly from the BFS without IME's knowledge. Such files will be removed by this stale file detector thread.

4.2 Synchronize

Typically, parallel file systems achieve only a fraction of the potential bandwidth due to random access patterns caused by unaligned, fragmented, or otherwise disjoint payloads. IME enables greater alignment, coalescence, and assembling of payload data, before the data is copied to the BFS. Synchronization can be performed automatically or manually. Data that are synchronized remains in IME until automatically or manually purged.

NOTE: For a detailed discussion about automatic data migration, which includes auto-synchronization and auto-purge, and the IME synchronization engine configuration, refer to the *IME Installation and Administration Guide*.
You can disable automatic data migration for specific file data. For details about file pinning, see Section 4.4 on page 23.

Manual synchronization enable applications to define which files can be synchronized to the BFS. To manually synchronize specific files, use **ime-sync** or **ime-ctl --sync**.

Example

```
ime-sync ime:///fsmnt/bfs/dir/myfile
or
ime-ctl --sync /imefusemnt/dir/myfile
```

> **NOTE:** By default, **ime-sync** uses **/etc/ddn/ime/ime.conf** to establish communication with **ime-server**. To specify a different configuration file, use the **-c** option.

When **ime-sync** or **ime-ctl --sync** is issued, IME marks the unsynchronized file data as pending. After synchronization is complete, IME marks the file data as synchronized, unless there is a new write simultaneously occurring on the file. If there is an issue in writing to the BFS, such as no available space, IME reverts the fragments from pending to unsynchronized and pauses the synchronization. In this case, you must re-issue **ime-sync** or **ime-ctl --sync** after the BFS issue has been resolved.

View the fragment status of the data resident in IME using **ime-frag-stat** or **ime-ctl --frag-stat**. Note that the output is dynamic and might change at any time due to automatic data migration. After a restart of **ime-server**, IME persists the state of the file fragments, and **ime-frag-stat** or **ime-ctl --frag-stat** shows data that was synchronized before IME shutdown.

Comparing the file size reported by IME to the size reported by the BFS is not a reliable way to determine if the file data is synchronized completely to the BFS. To monitor synchronization completion, use **ime-frag-stat** or **ime-ctl --frag-stat** to show if there is unsynchronized or pending data.

By default, **ime-sync** and **ime-ctl --sync** are non-blocking. Passing the **--block** option with these utilities creates a blocking operation, which guarantees all data synchronizes before any other operation can begin.

To guarantee that the data synchronizes to the BFS before it is manually purged from IME, use the sequence of commands shown in the following example.

Example

```
ime-ctl --sync --block /imefusemnt/dir/myfile
ime-ctl --purge-synced /imefusemnt/dir/myfile
```

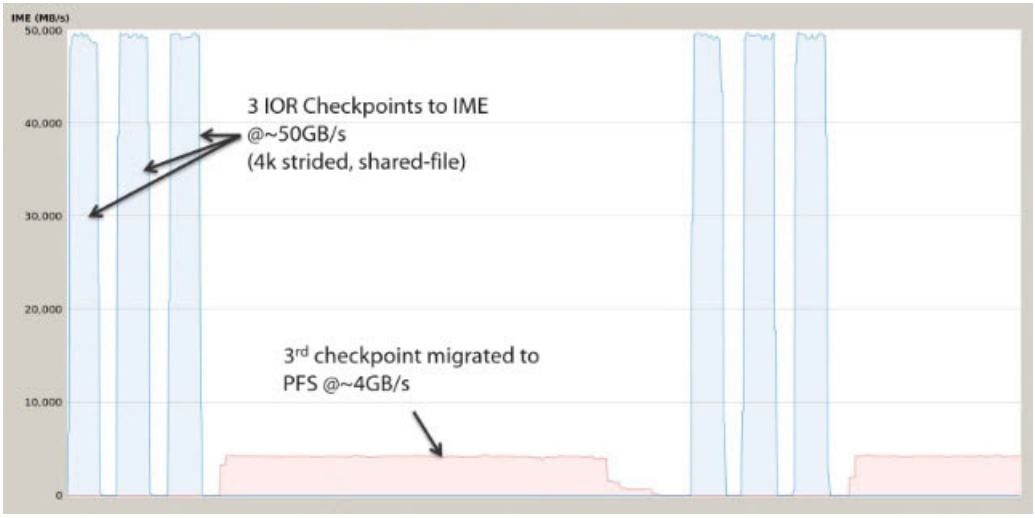
> **NOTE:** To synchronize data manually using IME Native API, use **ime_client_native2_bfs_sync**. (See Appendix B.4 on page 38.)

Checkpoint-Restart

Checkpoint-restart is a specialized case of synchronization. A checkpoint operation involves the rapid writing of most or all of a computer systems' memory to storage at points throughout the execution of very large and very long-running applications. If an application crashes for some reason during its execution, administrators may restart the application by reading the last checkpoint image from storage. This process is vital to high-performance computing systems due to the frequent failure rates of both hardware and software.

Figure 4 illustrates the relationship of checkpoints to IME versus checkpoints that are synchronize to the BFS.

Figure 4. Checkpoint Performance



IME is well suited to absorb the burst of rapid checkpoint writes. However, synchronizing data to the slower BFS requires time and bandwidth. The time required to write a full system checkpoint to the BFS is significantly greater than the time required to write to IME. To maximize the performance of your system, synchronize only the necessary data to the BFS. For details about explicitly managing (or "pinning") the file data, see Section 4.4 on page 23.

4.3 Purge

Pro-active management of buffer capacity is of crucial importance to ensure consistent system-wide performance. If the IME data tier becomes full, applications must wait for IME to synchronize data to the BFS, if necessary, and subsequently purge the data. Removing file data resident in IME can be performed automatically or manually. Automatic purge removes synchronized data only, while manual purge removes data regardless of the synchronization state by default.

NOTE: For a detailed discussion about automatic data migration, which includes auto-synchronization and auto-purge, and the IME synchronization engine configuration, refer to the *IME Installation and Administration Guide*.
You can disable automatic data migration for specific file data. For details about file pinning, see Section 4.4 on page 23.

In some cases, manually purging data from IME before it synchronizes to the BFS can increase system performance. Consider the scenario in which only 1 out of 10 large data sets written to IME every hour is needed to recover from system failure. In this case, the remaining 9 do not need to be synchronized and can be manually purged from IME.

To manually purge specific files, use **`ime-release`** or **`ime-ctl --purge`**. By default, these utilities purge file data from IME, regardless of its synchronization state. File data that is not synchronized to the BFS will be lost.

Example

```
ime-release ime:///fsmnt/bfs/dir/myfile
```
or
```
ime-ctl --purge /imefusemnt/dir/myfile
```

NOTE: By default, **`ime-release`** uses **`/etc/ddn/ime/ime.conf`** to establish communication with **`ime-server`**. To specify a different configuration file, use the **`-c`** option.

Purging data occurs asynchronously. When the command is issued, it sends the request to purge IME data and returns. It does not wait for the data to be purged. Therefore, the data may not be removed from IME immediately.

To purge only synchronized data, use **`ime-release -k`** or **`ime-ctl --purge-synced`**.

Example

`ime-ctl --purge /imefusemnt/dir/myfile` purges data regardless of its synchronization state, while **`ime-ctl --purge-synced /imefusemnt/dir/myfile`** purges only synchronized data.

To guarantee that the data synchronizes to the BFS before it is manually purged from IME, use the sequence of commands shown in the following example.

Example

```
ime-ctl --sync --block /imefusemnt/dir/myfile
ime-ctl --purge-synced /imefusemnt/dir/myfile
```

NOTE: To purge data from IME manually using IME Native API, use **`ime_client_native2_data_release`**. (See Appendix B.4 on page 38.)

4.4 File Pinning

Policies for the IME synchronization engine establish procedures to prevent IME from filling. These configuration settings determine the maximum percentage of ***used*** IME device space before IME automatically synchronizes data to the backing file system (BFS). If the configuration is set to use only a small percentage, you may want to explicitly manage (or "pin") the file data that IME automatically synchronizes to prevent unnecessary data movement. Consider the scenario in which only 1 out of 10 large data sets written to IME every hour is needed to recover from system failure. In this case, the remaining 9 do not need to be synchronized.

Similarly, these configuration settings determine the minimum percentage of ***free*** IME device space before IME automatically purges synchronized data. If the configuration is set to keep a large percentage free, you may want to explicitly manage (or "pin") the file data that IME automatically purges to prevent unwanted removal of data. Consider the scenario in which you prestage file data from the BFS, which will be marked as synchronized, and need to access the data often. In this case, the data should not be removed from IME.

To exempt specific files from automatic data migration, use **`ime-pin`** or **`ime-ctl --pin`**. The **`--pin`** option sets a flag to not auto-synchronize and auto-purge data for this file.

Example

```
ime-pin ime:///fsmnt/bfs/dir/myfile
or
ime-ctl --pin /imefusemnt/dir/myfile
```

Note that you can manually synchronize or purge this file as described in Section 4.2 on page 20 and Section 4.3 on page 22, respectively.

Key Points

- By default, **`ime-pin`** uses **`/etc/ddn/ime/ime.conf`** to establish communication with **`ime-server`**. To specify a different configuration file, use the **`-c`** option.
- The default option for **`ime-pin`** is **`-M`** ("pin the file").
- If a file is marked as pinned, this setting will not persist after a reboot.
- For additional details about file pinning, refer to the *IME Installation and Administration Guide*.

NOTE: To pin files using IME Native API, use **`ime_client_native2_sync_ctl`**. (See Appendix B.4 on page 38.)

---

5. Integrating with Job Schedulers

IME integrates with job schedulers at the user-level by job chaining or at the administrator-level using prologue and epilogue scripts. In both cases, the integration is expressed by scripts that implement native IME command line tools. The software distribution of IME includes ready-to-use examples, which can also be adapted to an administrator's own tool set.

Considering that IME is by essence a hierarchical storage, the definition of a data policy is the important part of the job scheduler integration. For a given job, the administrator must determine what data should be prestaged prior to the execution of the job itself and what specific actions should be done for the data produced at the end of the job.

The scripts provided with IME propose a policy in five keywords:

- **IN** – Prestage the files in IME prior to job execution. This keyword is interpreted by the **`ime-stage-in.sh`** script only.
- **SYNC** – Copy the files to the BFS and retain in IME after job execution, which is equivalent to calling **`ime-sync`** on the file. (Default behavior) This keyword is interpreted by the **`ime-stage-out.sh`** script only.
- **PURGE** – Copy the files to the BFS and remove the remaining version in IME after job execution. This keyword is interpreted by the **`ime-stage-out.sh`** script only.
- **RETAIN** – Keep the files in IME without explicit action, letting the automated synchronization engine of IME manage the status of the file. This keyword is interpreted by the **`ime-stage-out.sh`** script only.
- **DISCARD** – Remove the files from IME and the BFS after job execution, which means that no remaining copies of the file will exist. This is typically useful for temporary or intermediate data. This keyword is interpreted by the **`ime-stage-out.sh`** script only.

Figure 5 shows a policy example.

Figure 5. policy_example.ime

```
IN file1 file2 dir_input
SYNC file3
PURGE file4 file5
RETAIN file6
DISCARD file7 file8 file9
```

The following sections define the job scheduler scripts. Depending on the existence of an IME fuse mount point, the scripts will use the fastest possible method to perform stage-in and stage-out.

5.1 Data Stage-in Script

`/opt/ddn/ime/bin/ime-stage-in.sh [options] <files and directories>`

This script is a driver for stage-in capabilities of IME, which prepares IME for job execution. It prestages all files and directories passed as arguments. When calling the script, you must specify the list of files and directories in a space-separated format. The process is blocking; thus it will return once all data have been successfully staged in IME.

Options include:

`-c <file>`, **`--config-file <file>`**	Set absolute path to IME Configuration file. Default is **`/etc/ddn/ime/ime.conf`**.
`-f <file>`, **`--policy-file <file>`**	Set temporal locality of data policy file.
`-h`, **`--help`**	Display help.
`-v`, **`--verbose`**	Display verbose output.

Examples

```
./ime-stage-in.sh input_dir
```

```
./ime-stage-in.sh input_file1 input_file2 input_dir
```

5.2 Data Stage-out Script

`/opt/ddn/ime/bin/ime-stage-out.sh [options] <files and directories>`

This script is a driver for stage-out capabilities of IME, which defines the specific actions that should be done for the data produced at the end of the job. It will discard, synchronize, or purge all files and directories passed as arguments. When calling the script, you must specify the list of files and directories in a space-separated format. The process is blocking; thus it will return once all data have been successfully handled according to the requested action.

Options include:

`-c <file>`, **`--config-file <file>`**	Set absolute path to IME Configuration file. Default is **`/etc/ddn/ime/ime.conf`**.
`-d`, **`--DISCARD`**	Specify list of space-separated files or directories to remove from IME and the BFS after job execution.
`-f <file>`, **`--policy-file <file>`**	Set temporal locality of data policy file.
`-h`, **`--help`**	Display help.
`-p`, **`--PURGE`**	Specify list of space-separated files or directories to copy to the BFS and remove the remaining version in IME after job execution.
`-s`, **`--SYNC`**	Specify list of space-separated files or directories to copy to the BFS and keep within IME after job execution.
`-v`, **`--verbose`**	Display verbose output.

Examples

```
./ime-stage-out.sh input_dir
```

```
./ime-stage-out.sh input_file1 input_file2 input_dir
```

5.3 Traffic Monitor Script

`/opt/ddn/ime/sbin/ime-traffic-monitor.sh [options] -l <nodes>`

This script tracks the amount of I/O traffic sent to IME for the specified nodes. When calling the script, you must specify the list of nodes in a comma-separated format. The nodes involved in a job can be extract from the job scheduler environment:

- In PBS/Torque, **`$PBS_NODEFILE`** contains the nodes assigned to a job.
- In Slurm, **`$SLURM_JOB_NODELIST`** contains similar information.

Typically, this script is used to report the traffic that occurred during a job. At the beginning of the job, call this script with the **`-c`** option to reset the traffic count to zero and start monitoring the traffic. At the end of the job, the numbers gathered represent the total I/O activity emitted by the specified nodes during the lifetime of the job.

Options include:

`-c`, **`--clean`**	Reset traffic count to **`0`**.
`-d`, **`--display`**	Display current traffic.
`-f <file>`, **`--file <file>`**	Set path and name of the output file in batch mode. Default is **`/tmp/ime/ime_traffic.log`**.
`-i`, **`--interactive`**	Interactive mode. Refresh every second and display updated traffic information. If this option is not included, the script runs in batch mode.
`-l`, **`--list <list>`**	Set list of nodes in comma-separated format.
`-h`, **`--help`**	Display help.
`-v`, **`--verbose`**	Display verbose output.

Example

```
./ime-traffic-monitor.sh -i -l imeclient1,imeclient2,imeclient3
```

Figure 6 shows example output for the interactive mode. The bandwidth will be updated dynamically depending on the connection speed between node.

Figure 6. Example Output in Interactive Mode

```
Monitored IME Traffic for node
std0801,std0802,std0803,std0805,std0806,std0807,std0808,std0809,std0810,std0811,std0812,std0813,std0
814,std0815,std0816 in Byte and MiB

_____________________________________________________________________________
std0801: WRITE 7081214304 (6753 MiB) READ 0 (0 MiB)
std0802: WRITE 344 (0 MiB) READ 0 (0 MiB)
std0803: WRITE 344 (0 MiB) READ 0 (0 MiB)
std0805: WRITE 344 (0 MiB) READ 0 (0 MiB)
std0806: WRITE 0 (0 MiB) READ 0 (0 MiB)
std0807: WRITE 344 (0 MiB) READ 0 (0 MiB)
std0808: WRITE 344 (0 MiB) READ 0 (0 MiB)
std0809: WRITE 344 (0 MiB) READ 0 (0 MiB)
std0810: WRITE 344 (0 MiB) READ 0 (0 MiB)
std0811: WRITE 688 (0 MiB) READ 0 (0 MiB)
std0812: WRITE 344 (0 MiB) READ 0 (0 MiB)
std0813: WRITE 344 (0 MiB) READ 0 (0 MiB)
std0814: WRITE 344 (0 MiB) READ 0 (0 MiB)
std0815: WRITE 344 (0 MiB) READ 0 (0 MiB)
std0816: WRITE 344 (0 MiB) READ 0 (0 MiB)
```

IMPORTANT: This script only reports the IME traffic that went through a fuse mount point. It does not capture the IME MPI-IO traffic.

Appendix A IME Command Line Utilities

IME provides a set of command line interface (CLI) utilities to control the movement of data between IME and the BFS and to manipulate files resident in the IME data tier, defined as follows:

- **IME Control** (**`ime-ctl`**) – Specifically designed for use on hosts that present the IME client mount to applications, such as clients that use FUSE. **`ime-ctl`** can only be used with the IME client mount point or FUSE path. By passing various options, it performs functions equivalent to the stand-alone utilities. This utility has the same functionality as its equivalent stand-alone utilities, yet its performance is better due to its use of already established server connections. (Appendix A.1 on page 31.)
- **Stand-alone** – Functionality equivalent to **`ime-ctl`** but designed for use on hosts that do not present the IME client mount to applications. These utilities provide flexibility by not requiring a POSIX mount point at the cost of lower performance. (Section A.2 on page 32.)
- **General-use** – General functionality and can be used on all hosts, regardless of whether the IME client mount is presented. If IME client mount is present, the **`-c`** option is not needed. These utilities do not have an **`ime-ctl`** equivalent.(Section A.3 on page 34.)

The following sections give detailed descriptions of each utility, including all available options, as listed in Table 6 and Table 7.

NOTE: For the latest information, refer to the corresponding man page or use the **`-h`** option at the command line.

Table 6. IME CLI Utilities: IME Control versus Stand-alone Utilities

IME CLI Utilities		Description
IME Control note[1]	**Stand-alone**	
`ime-ctl --frag-stat`	**`ime-frag-stat`** (Appendix A.2.1 on page 32)	Show file fragment status for IME file data.
`ime-ctl --pin`	**`ime-pin -M`** (Appendix A.2.2 on page 32)	Disable automatic data migration of IME file data.
`ime-ctl --unpin`	**`ime-pin -m`** (Appendix A.2.2 on page 32)	Enable automatic data migration of IME file data.
`ime-ctl --prestage`	**`ime-prestage`** (Appendix A.2.3 on page 33)	Prestage file data from the BFS to IME.
`ime-ctl --purge`	**`ime-release`** (Appendix A.2.4 on page 33)	Purge file data resident in IME, regardless of synchronization state.
`ime-ctl --sync`	**`ime-sync`** (Appendix A.2.5 on page 33)	Synchronize file data from IME to the BFS keeping the data in IME.

1. For details about **`ime-ctl`**, see Appendix A.1 on page 31.

Table 7. IME CLI Utilities: General-use

IME CLI Utilities General-use	Description
ime-cat (Appendix A.3.1 **on page** 34)	Concatenate a file resident in IME and print on standard output.
ime-file-to-fid (Appendix A.3.2 **on page** 34)	Print IME FID for a file specified in one or more IME file paths.
ime-lsfiles (Appendix A.3.3 **on page** 34)	List all files resident in IME at a specific path.
ime-rm (Appendix A.3.4 **on page** 34)	Delete file data resident in IME and its BFS file.
ime-stat (Appendix A.3.5 **on page** 35)	Show file statistics for one or more files resident in IME.

Common Options

Unless noted otherwise, the following options are available with all CLI utilities:

-c <file> Set absolute path to IME Configuration file. Default is **/etc/ddn/ime/ime.conf**. (Not valid with **ime-ctl**.)

-h, **--help** Display help. Use **-hh** for additional detail.

-L <level> Set detail level of data logged. (Not valid with **ime-ctl**.)
Valid values are
- **0** = Trace (Most detail)
- **1** = Debug
- **2** = Information
- **3** = Warning (Default)
- **4** = Error
- **5** = Fatal (Least detail)

-l <file> Set absolute path to log file, which will include a log of events. (Not valid with **ime-ctl**.)

-v, **--version** Display version and then exit.

Key Points

The following are key points to know about the IME CLI utilities:

- Unless noted otherwise, IME CLI utilities are called from the client-side.
- Unless noted otherwise, IME CLI utilities support use of wildcards for POSIX pathnames and those prefixed by the IME file prefix (**`ime:///`** or **`ime://`**). (See Section 3.3 on page 16.)
- For **`ime-ctl`**, the specified **`<path>`** must be the absolute path to the file on the client mount. It does not support relative paths. For example: **`/imefusemnt/dir/myfile`**
- When using the IME file prefix, the specified **`<path>`** can be the relative path or absolute path to the file. (See Section 3.3 on page 16.)
- Unless noted otherwise, the specified **`<path>`** must be the path to the file in IME.
- If a BFS is mounted, the IME file prefix (**`ime://`**) is not allowed with the FUSE path. The prefix is used for BFS path only.
- If a BFS is not mounted and FUSE is mounted, run the IME CLI utilities with the FUSE path.
- Unless noted otherwise, IME CLI utilities will resolve a symbolic link to retrieve the file to which it points and will then use the same file for further operations. The utility will work on the link only if the file it points to is present on the BFS.
- If the **`-R`** or **`-r`** option is supported, file data can include all files in a specified directory and its subdirectories recursively.
- Unless noted otherwise, IME CLI utilities are non-blocking (by default). The **`-b`** option enables blocking mode.
- The **`-L`** and **`-l`** options for the IME CLI utilities override the detail level of data logged and the absolute path to the log file at run time, respectively.
- The **`--server-log-level`** and **`--client-log-level`** options for **`ime-ctl`** override the detail level of data logged at run time.
- If an IME file that contains all zeros is copied to a new file within IME, **`ime-stat/stat()`** displays a size of **`0`** for the copied file.

IMPORTANT:	Due to the operating system and the underlying network libraries and data devices being used, there are limits to how many copies of the IME CLI utilities can run simultaneously. This number will vary across IME clusters. If a utility reports that it cannot start up due to lack of system resources, decrease the level of parallelism. In general, limit the number to ***eight*** utility commands simultaneously running on a single IME client.

A.1 IME Control Utility (ime-ctl)

The **ime-ctl** utility is a general-purpose, ioctl-based client tool and is specifically designed for use on hosts that present the IME client mount to applications, such as clients that use FUSE. The utility performs various operations on IME file data at the specified path, as described by the supported options:

```
ime-ctl [commands] [options] <path>
```

This utility has the same functionality as its equivalent stand-alone utilities, yet its performance is better due to its use of already established server connections.

For additional details, see Key Points on page 30 and Common Options on page 29.

One of the following commands is required:

--purge-stale <FID> <mount_point>	Purge stale file data from IME.
--get-short-fid, **--get-long-fid**	Obtain the IME FID for the specified file.
--prestage	Prestage file data from the BFS to IME. IME will purge any existing synchronized IME file data and then prestage data from the BFS file.
--purge-synced	Purge only synchronized IME file data. Unsynchronized data will be retained and only synchronized data will be released.
--pin	Disable automatic data migration of file data (“pin the file”). If a file is marked as pinned, this setting will not persist after a reboot.
--unpin	Enable automatic data migration of file data (“un-pin the file”).
--purge	Purge file data from IME, regardless of synchronization state.
--sync	Synchronize file data from IME to the BFS keeping the data in IME.
--frag-stat	Show file fragment status for IME file data.
--unmap	Send unmap IOs on all devices for all freed blocks that need to be unmapped. Unmaps are sent even if there is read/write activity on the devices. May temporarily slow down the devices.
--unmap-threshold	Change threshold to trigger non-idle unmap commands based on the percentage of modified device blocks, which is counted individually per device including commit log device. Valid values are **0** to **100** percent. Default is **90%**. If the value is negative, the trigger is disabled.

Additional command options for use in data residency and data discovery include

--block	Block on prestage or synchronize completion. For use with **--prestage** or **--sync**. Default is non-blocking.
-K, **--keep**	Keep existing synchronized IME file data on prestage. For use with **--prestage**.
-R, --recursive	Run command on files in subdirectories recursively.

Additional commands to change the log level include

--client-log-level=<level>	Set detail level of data logged to the client log. Default is **3**.
--server-log-level=<level>	Set detail level of data logged to the server log file. Default is **3**.

Additional output options include

-H, **--human**	Convert byte values to human-readable format.
--short	Display short output mode.
-V, **--verbose**	Display verbose output.

IMPORTANT: You must have root privileges to use the **--purge-stale** option.

A.2 Stand-alone Utilities

The stand-alone utilities have functionality equivalent to **`ime-ctl`** but are designed for use on hosts that do not present the IME client mount to applications. These utilities provide flexibility by not requiring a POSIX mount point at the cost of lower performance.

For additional details, see Key Points on page 30 and Common Options on page 29.

A.2.1 ime-frag-stat

`ime-frag-stat [options] <path>`

Show file fragment status for the specified file resident in IME.

Functionality equivalent to **`ime-ctl --frag-stat`** but is used on hosts that do not present IME client mount to applications.

File fragments may be in one of four states:

- **Dirty** – New or modified data in IME without an exact copy on the BFS.
- **Pending** – Dirty data awaiting synchronization to the BFS.
- **Clean** – Data in IME has an exact copy on the BFS. It has either been synchronized or prestaged.
- **Deletable** – Data that may be freed from IME. Data can be manually marked as deletable regardless of its state.

Options include the Common Options on page 29 and the following:

`-r` Show file fragment status for all files in the specified directory and its subdirectories recursively.

A.2.2 ime-pin

`ime-pin {-M|-m} [options] <path>`

Disable or enable automatic data migration for specified file data resident in IME.

Functionality equivalent to **`ime-ctl --pin`** and **`ime-ctl --unpin`** but is used on hosts that do not present IME client mount to applications.

Commands include the following:

`-M` Disable automatic data migration of file data (“pin the file”).
If no option is passed, defaults to **`-M`**.
If a file is marked as pinned, this setting will not persist after a reboot.

`-m` Enable automatic data migration of file data (“un-pin the file”).

Additional options include the Common Options on page 29 and the following:

`-r` Disable or enable automatic data migration for all files in the specified directory and its subdirectories recursively.

A.2.3 ime-prestage

`ime-prestage [options] <path>`

Prestage specified file data from the BFS to IME.

Functionality equivalent to **`ime-ctl --prestage`** but is used on hosts that do not present IME client mount to applications.

`<path>` must be path on the BFS, as it is used to bring file data from the BFS to IME. By default, this operation is non-blocking.

Options include the Common Options on page 29 and the following:

- **`-b`** Block until the prestage operation is complete. Default is non-blocking.
- **`-K`** Keep existing synchronized IME file data on prestage.
- **`-r`** Prestage all files in the specified directory and its subdirectories recursively.
- **`-V`** Display progress when blocking mode (**`-b`**) is specified.

> **NOTE:** IME will purge any existing synchronized IME file data and then prestage the data from the BFS file. To keep the existing synchronized data on prestage, pass the **`-K`** option.

A.2.4 ime-release

`ime-release [options] <path>`

Purge the specified file data resident in IME.

Functionality equivalent to **`ime-ctl --purge`** but is used on hosts that do not present IME client mount to applications.

By default, **`ime-release`** removes all specified file data from IME regardless of its synchronization state. If synchronized, the file content resident in the BFS remains intact. File data that is not synchronized to the BFS will be lost.

Options include the Common Options on page 29 and the following:

- **`-k`** Purge only synchronized IME file data. Unsynchronized data will be retained and only synchronized data will be released.
- **`-r`** Purge all files in the specified directory and its subdirectories recursively.

A.2.5 ime-sync

`ime-sync [options] <path>`

Synchronize specified file data from IME to the BFS keeping the data in IME.

Functionality equivalent to **`ime-ctl --sync`** but is used on hosts that do not present IME client mount to applications. By default, this operation is non-blocking.

Options include the Common Options on page 29 and the following:

- **`-b`** Block until the synchronization operation is complete. Default is non-blocking.
- **`-r`** Synchronize all files in the specified directory and its subdirectories recursively.
- **`-V`** Display progress when blocking mode (**`-b`**) is specified.

A.3 General-Use Utilities

Use the utilities listed in this section on all hosts, regardless of whether IME client mount is presented. If IME client mount is present, the **-c** option is not needed. These utilities do not have an **ime-ctl** equivalent. For additional details, see Key Points on page 30 and Common Options on page 29.

A.3.1 ime-cat

RECOMMENDED:	Use the Linux alternative **cat** on the **ime-fuse** mount point instead of **ime-cat**.

ime-cat [options] <path>

Concatenate a file resident in IME or the BFS and print to standard output. Provides similar functionality to Linux command **cat()** for data resident in IME. Options include the Common Options on page 29.

A.3.2 ime-file-to-fid

ime-file-to-fid [options] <path>

Print 128-byte IME FID for a file specified in one or more IME file paths. **ime-file-to-fid** does not support wildcard characters. All path arguments must be absolute or relative IME paths. Options include the Common Options on page 29.

A.3.3 ime-lsfiles

RECOMMENDED:	Use the Linux alternative **ls** on the **ime-fuse** mount point instead of **ime-lsfiles**.

ime-lsfiles [options] <path>

List all files resident in IME for the specified path. Provides similar functionality to Linux command **ls()** for data resident in IME. Options include the Common Options on page 29.

A.3.4 ime-rm

RECOMMENDED:	Use the Linux alternative **rm** on the **ime-fuse** mount point instead of **ime-rm**.

ime-rm [options] <path>

Delete the specified file resident in IME and its BFS file. If the specified file is a symbolic link, **ime-rm** will delete the symbolic link only. The file to which the symbolic link points remains intact unless it is deleted explicitly. If **<path>** is a symbolic link pointing to a directory, **ime-rm -r** deletes all directory contents and the symbolic link.

Options include the Common Options on page 29 and the following:

-r Delete all files, directories, and symbolic links in the specified directory and its subdirectories recursively.

A.3.5 ime-stat

RECOMMENDED:	Use the Linux alternative **`stat`** and **`ime-frag-stat`** on the **`ime-fuse`** mount point instead of **`ime-stat`**.

`ime-stat [options] <path>...`

Show file statistics for one or more files resident in IME. Provides similar functionality to Linux command **`stat()`** for data resident in IME. Symbolic links are not de-referenced by default. If **`ime-stat`** is run on a directory with **`-r`** and without **`-f`** option, it traverses its sub-directory even if it is a symbolic link. However for symbolic links pointing to file, it displays an error message to use the **`-f`** option.

Options include the Common Options on page 29 and the following:

`-f` Show file statistics for the target file referenced by a symbolic link.
`-r` Show file statistics for all files in the specified directory and its subdirectories recursively.

Appendix B IME Native API Reference

The following sections provide brief descriptions of the IME Native API. For the latest information, refer to the **`ime-native-api`** man page.

> **NOTE:** On failure, the IME Native API functions return **`-1`** and **`errno`** is set to indicate the error.

B.1 Initialization and Finalization

ime_client_native2_init

void ime_client_native2_init(void)

Prepare the IME native client interface for use. You must initialize the interface with a call to this function prior to any other IME Native API call.

ime_client_native2_finalize

int ime_client_native2_finalize(void)

Destroy the IME native client interface.

B.2 POSIX Equivalent Operations

ime_client_native2_close

int ime_client_native2_close(int fd)

IME Native implementation of **`close()`**

ime_client_native2_dup2

int ime_client_native2_dup2(int oldfd, int newfd)

IME Native implementation of **`dup2()`**

ime_client_native2_fsync

int ime_client_native2_fsync(int fd)

IME Native implementation of **`fsync()`**

ime_client_native2_lseek

off_t ime_client_native2_lseek(int fd, off_t offset, int whence)

IME Native implementation of **`lseek()`**

ime_client_native2_open

int ime_client_native2_open(const char *user_path, int flags, mode_t mode)

IME Native implementation of **`open()`**

ime_client_native2_pread

ssize_t ime_client_native2_pread(int fd, char *buf, size_t count, off_t offset)

IME Native implementation of **`pread()`**

ime_client_native2_preadv

```
ssize_t ime_client_native2_preadv(int fd, const struct iovec *iov, int iovcnt,
                                  off_t offset)
```

IME Native implementation of **preadv()**. This implementation is based on the IME **aio_write** implementation.

ime_client_native2_pwrite

ssize_t ime_client_native2_pwrite(int fd, const char *buf, size_t count, off_t offset)

IME Native implementation of **pwrite()**

ime_client_native2_pwritev

```
ssize_t ime_client_native2_pwritev(int fd, const struct iovec *iov, int iovcnt,
                                   off_t offset)
```

IME Native implementation of **pwritev()**. This implementation is based on the IME **aio_write** implementation.

ime_client_native2_read

ssize_t ime_client_native2_read(int fd, char *buf, size_t count)

IME Native implementation of **read()**

ime_client_native2_stat

int ime_client_native2_stat(const char **user_path, struct stat *buf)

IME Native implementation of **stat()**

ime_client_native2_ftruncate

int ime_client_native2_ftruncate(int fd, off_t off)

IME Native implementation of **ftruncate()**

ime_client_native2_unlink

int ime_client_native2_unlink(const char *pathname)

IME Native implementation of **unlink()**

ime_client_native2_write

ssize_t ime_client_native2_write(int fd, const char *buf, size_t count)

IME native implementation of **write()**

B.3 IME Client Asynchronous Library

The **ime_client_native2_aio_read** / **ime_client_native2_aio_write** functions queue the I/O requests described by the control block structure. They are the asynchronous analogs of **ime_client_native2_preadv** / **ime_client_native2_pwritev**, but the same arguments are passed in the control block. These functions are "asynchronous" in that the calls return as soon as the requests have been enqueued, and notification of I/O completion is signaled via call back to the user-specified function in the control block.

The completion function would be called only if I/O enqueuing call returns success, as indicated by a value of 0. Note that the completion function may be called from a different thread and can even be called earlier than I/O enqueuing call returns. The control block must not be changed while the operation is in progress.

The completion callback has the following signature:

```
void  complete_cb(struct ime_aiocb *aiocb, int err, ssize_t bytes);
```

where

aiocb is the same pointer to the control block that was used for enqueuing.
err is the error code of operation (**0** if successful, non-zero error code otherwise).
bytes are actual number of bytes read or written as result of I/O (only valid when err=0).

The following structure is used to submit IO to the functions:

```
struct ime_aiocb
{
  int          fd;
  int          iovcnt;
  uint32_t     flags;
  struct iovec *iov;
  off_t        file_offset;
  void         *complete_cb)(struct ime_aiocb *aiocb, int err, ssize_t bytes);
  intptr_t     user_context;
};
```

Fields of this **struct** are as follows:

fd	File descriptor to which to read or write.
iovcnt	Number of elements in the sgl.
flags	Not used. Must be initialize to **0**.
iov	Scatter gather elements.
file_offset	File seek offset.
complete_cb	Function pointer to call upon completion of IO.
user_context	Handle for the IO submitter to use, which can be used to reference its data structures.

ime_client_native2_aio_read

int ime_client_native2_aio_read(struct ime_aiocb *aiocb)

Submit a read from the IME client asynchronous library. Returns 0 if the IO was successfully submitted.

ime_client_native2_aio_write

int ime_client_native2_aio_write(struct ime_aiocb *aiocb)

Submit a write to the IME client asynchronous library. Returns 0 if the IO was successfully submitted.

B.4 Non-POSIX Equivalent Operations

ime_client_native2_async_fd

int ime_client_native2_async_fd(int fd, int async_on)

Set or unset an asynchronous file descriptor. Asynchronous file descriptors will not block in **ime_client_native2_close()** waiting for the synchronization of cached data to the IME storage service. All remaining unsynchronized data is synchronized in **ime_client_native2_finalize()**.

ime_client_native2_bfs_sync

int ime_client_native2_bfs_sync(int fd, bool blocking)

Synchronize file data from IME to the BFS keeping the data in IME. This operation is blocking/non-blocking based on the value passed. The IME service copies the file content in parallel, using large chunk sizes when available.

ime_client_native2_config_ctl

int ime_client_native2_config_ctl(int ratio, uint32_t type)

Change the ratios that control auto-synchronization and auto-purge.

ime_client_native2_data_release

int ime_client_native2_data_release(int fd, int clean_data_only)

Purge file data in IME. Data will be purged regardless of its synchronization state, unless **`clean_data_only`** is passed. File data that is not synchronized to the BFS will be lost.

ime_client_native2_frag_fstat

int ime_client_native2_frag_fstat(int fd, size_t *nbytes_dirty, size_t *nbytes_syncing, size_t *nbytes_clean, size_t *nbytes_deletable)

Determine fragment status for IME file data. Returns the number of bytes in the dirty (unsynchronized), syncing (pending), clean (synchronized), and deletable states. This function determines if a synchronization or prestage operation has completed.

ime_client_native2_get_ime_fsize

ssize_t ime_client_native2_get_ime_fsize(int fd)

Get the file size of a file resident in IME. Note that this file size may have been affected by application calls to **`truncate()`** or **`ftruncate()`** and may not be consistent with the size reported by the BFS.

ime_client_native2_prestage

int ime_client_native2_prestage(int fd, bool keep_clean_data)

Prestage file data from the BFS to IME prior to your job. This operation is non-blocking. Similar to **`ime_client_native2_bfs_sync`**, prestage activities are parallelized across the IME data tier to achieve a high-performance interaction between IME and the BFS.

ime_client_native2_prestage_blocking

int ime_client_native2_prestage_blocking(int fd, bool keep_clean_data)

Prestage file data from the BFS to IME prior to your job. Similar to **`ime_client_native2_prestage`**, except the call is blocking.

ime_client_native2_sync_ctl

int ime_client_native2_sync_ctl(int fd, int disable_auto_sync)

If auto-synchronization is enabled, exempt the file data from being automatically synchronized to the BFS when capacity reaches the **`sync_threshold`**.

Appendix C IME Environment Variables

The following sections provide brief descriptions of the IME environment variables that are generally used for IME Native API and MPI-IO enabled applications. For the latest information, refer to the **`ime-environment-variables`** man page.

> **IMPORTANT:** Command line parameters take precedence over environment variables with the same meaning.

IM_BULKDATA_POOL_LARGE_MIN_MAX=<num>

Specify the quantity of large-sized buffers that IME should use.

IM_BULKDATA_POOL_MEDIUM_MIN_MAX=<num>

Specify the quantity of medium-sized buffers that IME should use.

IM_BULKDATA_POOL_SMALL_MIN_MAX=<num>

Specify the quantity of small-sized buffers that IME should use.

IM_BULKDATA_POOL_STAGE_MIN_MAX=<num>

Specify the quantity of stage in/out buffers that IME should use.

IM_BULKDATA_POOL_TINY_MIN_MAX=<num>

Specify the quantity of tiny-sized buffers that IME should use.

IM_BULKDATA_POOL_TYPICAL_MIN_MAX=<num>

Specify the quantity of typical-sized buffers that IME should use.

IM_CLIENT_CATCH_SIGSEGV=[0|1]

Specify if IME is allowed to set its custom handler for SIGSEGV signal that improves the level of crash diagnostics. Default is **0** (not allowed).

IM_CLIENT_CFG_FILE=<string>

Specify the absolute path to IME Configuration file, which contains the configuration of the IME Server and IME Client nodes. This file is required to boot **`ime-server`** and is used by IME Client nodes to contact **`ime-server`**. Paths specified by command line parameters take precedence over this environment variable.

> **NOTE:** For a detailed description of the IME Configuration file, refer to the *IME Installation and Administration Guide*.

IM_CLIENT_COMMAND_TIMEOUT=<integer>

Specify integer value in seconds for time during which commands to the IME Server node will be retried in case of connection or storage space problems. Default is **`1800`** (30 min).

IM_CLIENT_DATA_PLACEMENT_TYPE=<string>

Set the server data placement policy. Valid values are **`DETERMINISTIC`** (default) and **`NONDETERMINISTIC`**. The non-deterministic mode allows for adaptive load balancing across an IME Server node pool.

IM_CLIENT_DEBUG_LEVEL=<level>

Set detail level of data logged to the client log. Valid values are

0 = Trace (Most detail)
1 = Debug
2 = Information
3 = Warning (Default)
4 = Error
5 = Fatal (Least detail)

Levels specified by command line parameters take precedence over this environment variable.

IM_CLIENT_DISABLE_RA=[0|1]

Controls the read ahead mechanism of IME Client. Default is **0**, which enables read ahead. To disable read ahead, set to **1**.

IM_CLIENT_ENTRY_TIMEOUT=<float>

Specify a floating value for file entry cache entry timeout. Default is **0.0**.

IM_CLIENT_HANDSHAKE_ENABLED=[0|1]

Disable or enable client-server handshake. Default is **1**.

IM_CLIENT_INIT_TIMEOUT=<integer>

Specify an integer value in seconds for the maximum time allowed to establish connections with IME Server nodes. Default is **3600**.

IM_CLIENT_JOB_TRACKING_ENABLED=[0|1]

For use with IME Native API, enable or disable Slurm Job ID tracking. Default is **0**.

IM_CLIENT_LOG_FILE=<string>

Set absolute path to client log file, which is used to save a log of events. Paths specified by command line parameters take precedence over this environment variable.

IM_CLIENT_NET_IF=<device0>-<protocol>,<device1>-<protocol>,...

Specify the network devices and communication protocols for the IME Client nodes to use.

For IB and Intel OPA, use **ib*x*-verbs**
For Ethernet, use **eth*x*-tcp**

For single-rail, **IM_CLIENT_NET_IF** specifies the name of one network device.
For example: **IM_CLIENT_NET_IF=ib0-verbs**

For multi-rail, **IM_CLIENT_NET_IF** specifies a comma-separated list of devices.
For example: **IM_CLIENT_NET_IF=ib0-verbs,ib1-verbs**

For IME applications that use the IME FUSE Client, the option

--network_device=<device0>-<protocol>,<device1>-<protocol>,...

takes precedence over **IM_CLIENT_NET_IF**. In this case, typically, the network devices and communication protocols are specified in **/etc/ddn/ime/ime-fuse.conf**.

For additional details, refer to the *IME Installation and Administration Guide*.

IM_CLIENT_MIN_CONNECTIONS=[0|1]

Enable or disable support for minimum RPC connections. Default is **0**.

IM_CLIENT_NO_BFS_MOUNT=[0|1]

Specify if IME can access the BFS mount point on the client. This environment variable can be used for certain operations such as locks. For MPI-IO and IME native API, the default is **0** (BFS mount point is not used). For **`ime-fuse`**, the default is auto-detection. If the BFS mount point is passed as an option, the BFS mount point is used. If it is not passed, it will not be used. Explicitly setting **`IM_CLIENT_NO_BFS_MOUNT`** overrides that automatic decision.

IM_CLIENT_PGEOM=<N>+<K>

Enable erasure coding with a geometry of **<K>** redundant blocks per **<N>** data blocks.

Applications that use MPI-IO or IME Native API can override the **`def_pgeom`** setting in **`/etc/ddn/ime/ime.conf`** by passing the **`IM_CLIENT_PGEOM`** environment variable. Applications that use the IME FUSE client can override the **`def_pgeom`** setting and **`IM_CLIENT_PGEOM`** environment variable by passing the **`--pgeom=<N>+<K>`** option. For a detailed description of using erasure coding for data protection, refer to the *IME Installation and Administration Guide*.

IM_CLIENT_RD_BUFFERS=<integer>

Specify the maximum number of buffers for read caching. Values must be in the range of 32MB (minimum) to 8192MB (maximum). Default is **32**.

IM_CLIENT_WR_BUFFERS=<integer>

Specify the maximum number of buffers for write-back caching. Values must be in the range of 32MB (minimum) to 8192MB (maximum). Default is **32**.

IM_COREDUMP_DIR=<string>

Specify a destination directory for core dumps. For root user, default is the root of the BFS. For all other users, default is the current path, where the binary was executed.

The **`IM_COREDUMP_DIR`** environment variable is mainly for client purposes. If the directory was not specified using the **`ime-server --coredump-dir`** option, the directory specified by the environment variable is used. For additional details, refer to the *IME Installation and Administration Guide*.

IM_LOG_MAX_LINES_PER_FILE=<num>

Specify the number of lines per log file before log rotation is initiated.

IM_LOG_ROTATION_DISABLE_LEVEL=<level>

Specify the log level that disables log rotation. Default is **`IM_LOG_DBG`** (**1**) in order to avoid rotating out debug data.

IM_MONITOR_FILE=<string>

Enable IME monitoring with the output to specified file. If the path contains "**`%p`**" it will be substituted by PID.

IM_NETWORK_STACK=[CCI|OFI] (DEPRECATED: Use configuration option instead.)

IM_RPC_CLIENT_TIMEOUT_SECONDS=<num>

Set the timeout in seconds IME will wait before considering an RPC request as failed (only relevant for client-server communications). A value of **0** disables the timeout. Default is **0** (disabled).

IM_RPC_CM_MIN_RETRY_SECONDS=<num>

Define the minimum number of seconds between consecutive connection attempts. Default is **5**.

IM_RPC_CM_MAX_RETRY_SECONDS=<num>

Define the maximum number of seconds between consecutive connection attempts. Default is **640**.

IM_RPC_CM_TIMEOUT_SECONDS=<num>

Set the timeout IME will wait for a response following a connection request or shutdown. If a timeout is detected during a connection establishment or shutdown, the connection state is reverted back to disconnected and a connection request is re-sent. The value for the timeout must be greater than the network timeout. Default is **120**.

IM_RPC_CONN_NO_ACTIVITY_TIMEOUT_SECONDS=<num>

Set timeout without any activity (including keep-alive probes) in seconds after the connection is considered "dead" and the RPC initiates a tear-down. Must be equal to greater than twice the value of **IM_RPC_KEEPALIVE_TIMEOUT_SECONDS**. Default is **360**.

IM_RPC_CONNS_CHECKER_SECONDS=<num>

Set time between two consecutive executions of the connection checker, which is responsible for detecting connection timeouts. Default is **360**.

IM_RPC_KEEPALIVE_TIMEOUT_SECONDS=<num>

Set idle time in seconds before sending a keepalive probe. Default is **120**.

IM_RPC_SERVER_TIMEOUT_SECONDS=<num>

Set the timeout in seconds IME will wait before considering an RPC request as failed (only relevant for server-server communications). Value of **0** disables the timeout. Default is **300**.

IM_SERVER_EJECT_NODE_ON_NET_DEV_FAILURE=[0|1]

Specify the behavior of the IME Server node when a network interface used by the node goes down or enters a critical error state.

If set to **1** or undeclared (Default), the node will be ejected if any network device fails. For example:

If a node has 4 network devices and any 1 network device fails, it will trigger node ejection.

If set to **0**, node ejection will occur only when all network devices fail. For example:

If a node has 4 network devices and any 1 network device fails, it will not trigger node ejection. In this case, the server continues to run if any network device is available. All devices must fail before the node will be ejected.

In both cases, a warning message will be included in the IME logs to indicate that a network device failed.

IMPORTANT: Node ejection for network device failures is only supported in CCI.

IM_SPATIAL_READ_OBJ_POOL_SIZE=<num>

Specify (in MB) the maximum memory footprint for objects used to speedup read operations on small IO blocks.

IM_STOP_ON_EJECTED = <string>

Specify if an IME Server node is allowed to continue if it was marked as ejected at the last shutdown.
Valid settings include

yes or **YES** = Node will continue to run without crashing, but it will not participate in any IME-related communications.

no or **NO** = Node will not boot and stop at bootup once it sees itself marked as failed by the IME cluster.

Contacting DDN Support

If you have questions or require assistance, contact DDN Support:

Web

Support Portal	https://community.ddn.com/login
Portal Assistance	webportal.support@ddn.com

Telephone

DDN Worldwide Directory	https://www.ddn.com/support/global-services-overview/

Email

Support Email	support@ddn.com

Bulletins

Support Bulletins	http://www.ddn.com/support/technical-support-bulletins
End-of-Life Notices	http://www.ddn.com/support/end-of-life-notices
Bulletin Subscription Request	support-tsb@ddn.com

Corporate Headquarters · 9351 Deering Avenue, Chatsworth, CA 91311 · ddn.com · +1.818.700.4000 · info@ddn.com