IBM Spectrum Scale
Version 4 Release 2.0

Data Management API Guide

GA76-0442-06

IBM Spectrum Scale
Version 4 Release 2.0

Data Management API Guide

GA76-0442-06

Note
Before using this information and the product it supports, read the information in "Notices" on page 47.

This edition applies to version 4 release 2 of the following products, and to all subsequent releases and modifications until otherwise indicated in new editions:
- IBM Spectrum Scale ordered through Passport Advantage® (product number 5725-Q01)
- IBM Spectrum Scale ordered through AAS/eConfig (product number 5641-GPF)
- IBM Spectrum Scale for Linux on z Systems (product number 5725-S28)

Significant changes or additions to the text and illustrations are indicated by a vertical line (|) to the left of the change.

IBM welcomes your comments; see the topic "How to send your comments" on page xii. When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures

Tables

About this information

This edition applies to IBM Spectrum Scale™ version 4.2 for AIX®, Linux, and Windows.

IBM Spectrum Scale is a file management infrastructure, based on IBM® General Parallel File System (GPFS™) technology, that provides unmatched performance and reliability with scalable access to critical file data.

To find out which version of IBM Spectrum Scale is running on a particular AIX node, enter:

```
lslpp -l gpfs\*
```

To find out which version of IBM Spectrum Scale is running on a particular Linux node, enter:

```
rpm -qa | grep gpfs
```

To find out which version of IBM Spectrum Scale is running on a particular Windows node, open the **Programs and Features** control panel. The IBM Spectrum Scale installed program name includes the version number.

Which IBM Spectrum Scale information unit provides the information you need?

The IBM Spectrum Scale library consists of the information units listed in Table 1.

To use these information units effectively, you must be familiar with IBM Spectrum Scale and the AIX, Linux, or Windows operating system, or all of them, depending on which operating systems are in use at your installation. Where necessary, these information units provide some background information relating to AIX, Linux, or Windows; however, more commonly they refer to the appropriate operating system documentation.

Note: Throughout this documentation, the term "Linux" refers to all supported distributions of Linux, unless otherwise specified.

Table 1. IBM Spectrum Scale library information units

Information unit	Type of information	Intended users
IBM Spectrum Scale: Administration and Programming Reference	This information unit explains how to do the following: • Use the commands, programming interfaces, and user exits unique to GPFS • Manage clusters, file systems, disks, and quotas • Export a GPFS file system using the Network File System (NFS) protocol	System administrators or programmers of GPFS systems

Table 1. IBM Spectrum Scale library information units (continued)

Information unit	Type of information	Intended users
IBM Spectrum Scale: Advanced Administration Guide	This information unit explains how to use the following advanced features of GPFS: • Accessing GPFS file systems from other GPFS clusters • Policy-based data management for GPFS • Creating and maintaining snapshots of GPFS file systems • Establishing disaster recovery for your GPFS cluster • Monitoring GPFS I/O performance with the **mmpmon** command • Miscellaneous advanced administration topics	System administrators or programmers seeking to understand and use the advanced features of GPFS
IBM Spectrum Scale: Concepts, Planning, and Installation Guide	This information unit provides information about the following topics: • Introducing GPFS • GPFS architecture • Planning concepts for GPFS • Installing GPFS • Migration, coexistence and compatibility • Applying maintenance • Configuration and tuning • Uninstalling GPFS	System administrators, analysts, installers, planners, and programmers of GPFS clusters who are very experienced with the operating systems on which each GPFS cluster is based

Table 1. IBM Spectrum Scale library information units (continued)

Information unit	Type of information	Intended users
IBM Spectrum Scale: Data Management API Guide	This information unit describes the Data Management Application Programming Interface (DMAPI) for GPFS. This implementation is based on The Open Group's System Management: Data Storage Management (XDSM) API Common Applications Environment (CAE) Specification C429, The Open Group, ISBN 1-85912-190-X specification. The implementation is compliant with the standard. Some optional features are not implemented. The XDSM DMAPI model is intended mainly for a single-node environment. Some of the key concepts, such as sessions, event delivery, and recovery, required enhancements for a multiple-node environment such as GPFS. Use this information if you intend to write application programs to do the following: • Monitor events associated with a GPFS file system or with an individual file • Manage and maintain GPFS file system data	Application programmers who are experienced with GPFS systems and familiar with the terminology and concepts in the XDSM standard
IBM Spectrum Scale: Problem Determination Guide	This information unit contains explanations of GPFS error messages and explains how to handle problems you may encounter with GPFS.	System administrators of GPFS systems who are experienced with the subsystems used to manage disks and who are familiar with the concepts presented in the *IBM Spectrum Scale: Concepts, Planning, and Installation Guide*

Prerequisite and related information

For updates to this information, see IBM Spectrum Scale in IBM Knowledge Center (www.ibm.com/support/knowledgecenter/STXKQY/ibmspectrumscale_welcome.html).

For the latest support information, see the IBM Spectrum Scale FAQ in IBM Knowledge Center (www.ibm.com/support/knowledgecenter/STXKQY/gpfsclustersfaq.html).

Conventions used in this information

Table 2 on page xii describes the typographic conventions used in this information. UNIX file name conventions are used throughout this information.

Note: Users of IBM Spectrum Scale for Windows must be aware that on Windows, UNIX-style file names need to be converted appropriately. For example, the GPFS cluster configuration data is stored in the `/var/mmfs/gen/mmsdrfs` file. On Windows, the UNIX namespace starts under the `%SystemDrive%\cygwin64` directory, so the GPFS cluster configuration data is stored in the `C:\cygwin64\var\mmfs\gen\mmsdrfs` file.

Table 2. Conventions

Convention	Usage
bold	**Bold** words or characters represent system elements that you must use literally, such as commands, flags, values, and selected menu options. Depending on the context, **bold** typeface sometimes represents path names, directories, or file names.
bold underlined	**bold underlined** keywords are defaults. These take effect if you do not specify a different keyword.
`constant width`	Examples and information that the system displays appear in `constant-width` typeface. Depending on the context, `constant-width` typeface sometimes represents path names, directories, or file names.
italic	*Italic* words or characters represent variable values that you must supply. *Italics* are also used for information unit titles, for the first use of a glossary term, and for general emphasis in text.
<*key*>	Angle brackets (less-than and greater-than) enclose the name of a key on the keyboard. For example, **<Enter>** refers to the key on your terminal or workstation that is labeled with the word *Enter*.

\	In command examples, a backslash indicates that the command or coding example continues on the next line. For example:
	`mkcondition -r IBM.FileSystem -e "PercentTotUsed > 90" \`
	`-E "PercentTotUsed < 85" -m p "FileSystem space used"`

{*item*}	Braces enclose a list from which you must choose an item in format and syntax descriptions.
[*item*]	Brackets enclose optional items in format and syntax descriptions.
<**Ctrl-***x*>	The notation <**Ctrl-***x*> indicates a control character sequence. For example, <**Ctrl-c**> means that you hold down the control key while pressing <**c**>.
item...	Ellipses indicate that you can repeat the preceding item one or more times.
|	In *synopsis* statements, vertical lines separate a list of choices. In other words, a vertical line means *Or*. In the left margin of the document, vertical lines indicate technical changes to the information.

How to send your comments

Your feedback is important in helping us to produce accurate, high-quality information. If you have any comments about this information or any other IBM Spectrum Scale documentation, send your comments to the following e-mail address:

mhvrcfs@us.ibm.com

Include the publication title and order number, and, if applicable, the specific location of the information about which you have comments (for example, a page number or a table number).

To contact the IBM Spectrum Scale development organization, send your comments to the following e-mail address:

gpfs@us.ibm.com

Summary of changes

This topic summarizes changes to the IBM Spectrum Scale licensed program and the IBM Spectrum Scale library. Within each information unit in the library, a vertical line (|) to the left of text and illustrations indicates technical changes or additions made to the previous edition of the information.

Summary of changes
for IBM Spectrum Scale version 4 release 2
as updated, November 2015

Changes to this release of the IBM Spectrum Scale licensed program and the IBM Spectrum Scale library include the following:

Cluster Configuration Repository (CCR): Backup and restore
You can backup and restore a cluster that has Cluster Configuration Repository (CCR) enabled. In the **mmsdrbackup** user exit, the type of backup that is created depends on the configuration of the cluster. If the Cluster Configuration Repository (CCR) is enabled, then a CCR backup is created. Otherwise, a mmsdrfs backup is created. In the **mmsdrrestore** command, if the configuration file is a Cluster Configuration Repository (CCR) backup file, then you must specify the `-a` option. All the nodes in the cluster are restored.

Changes in IBM Spectrum Scale for object storage

Object capabilities
Object capabilities describe the object protocol features that are configured in the IBM Spectrum Scale cluster such as unified file and object access, multi-region object deployment, and S3 API emulation. For more information, see the following topics:

- *Object capabilities* in *IBM Spectrum Scale: Concepts, Planning, and Installation Guide*
- *Managing object capabilities* in *IBM Spectrum Scale: Administration and Programming Reference*

Storage policies for object storage
Storage policies enable segmenting of the object storage within a single cluster for various use cases. Currently, OpenStack Swift supports storage polices that allow you to define the replication settings and location of objects in a cluster. IBM Spectrum Scale enhances storage policies to add compression and unified file and object access functions for object storage. For more information, see the following topics:

- *Storage policies for object storage* in *IBM Spectrum Scale: Concepts, Planning, and Installation Guide*
- *Mapping of storage policies to filesets* in *IBM Spectrum Scale: Administration and Programming Reference*
- *Administering storage policies for object storage* in *IBM Spectrum Scale: Administration and Programming Reference*

Multi-region object deployment
The main purpose of the object protocol is to enable the upload and download of object data. When clients have a fast connection to the cluster, the network delay is minimal. However, when client access to object data is over a WAN or a high-latency network, the network can introduce an unacceptable delay and affect quality-of-service metrics. To improve that response time, you can create a replica of the data in a cluster closer to the clients using the active-active multi-region replication support in OpenStack Swift. Multi-region can also be used to distribute the object load over several clusters to reduce contention in the file system. For more information, see the following topics:

- *Overview of multi-region object deployment* in *IBM Spectrum Scale: Concepts, Planning, and Installation Guide*
- *Planning for multi-region object deployment* in *IBM Spectrum Scale: Concepts, Planning, and Installation Guide*
- *Enabling multi-region object deployment initially* in *IBM Spectrum Scale: Concepts, Planning, and Installation Guide*
- *Adding a region in a multi-region object deployment* in *IBM Spectrum Scale: Administration and Programming Reference*
- *Administering a multi-region object deployment environment* in *IBM Spectrum Scale: Administration and Programming Reference*

Unified file and object access

Unified file and object access allows users to access the same data as an object and as a file. Data can be stored and retrieved through IBM Spectrum Scale for object storage or as files from POSIX, NFS, and SMB interfaces. For more information, see the following topics:

- *Unified file and object access overview* in *IBM Spectrum Scale: Concepts, Planning, and Installation Guide*
- *Planning for unified file and object access* in *IBM Spectrum Scale: Concepts, Planning, and Installation Guide*
- *Installing and using unified file and object access* in *IBM Spectrum Scale: Concepts, Planning, and Installation Guide*
- *Unified file and object access in IBM Spectrum Scale* in *IBM Spectrum Scale: Administration and Programming Reference*

S3 access control lists (ACLs) support

IBM Spectrum Scale for object storage supports S3 access control lists (ACLs) on buckets and objects. For more information, see *Managing OpenStack access control lists using S3 API emulation* in *IBM Spectrum Scale: Administration and Programming Reference*.

Changes in IBM Spectrum Scale for Linux on z Systems™

- Compression support
- AFM-based Async Disaster Recovery (AFM DR) support
- IBM Spectrum Protect™ Backup-Archive and Space Management client support
- Support for all editions:
  - Express®
  - Standard
  - Advanced (without encryption)

For more information about current requirements and limitations of IBM Spectrum Scale for Linux on z Systems, see *Q2.25* of IBM Spectrum Scale FAQ.

Change in AFM-based Async Disaster Recovery (AFM DR)

- Support for IBM Spectrum Scale for Linux on z Systems

File compression

With file compression, you can reclaim some of the storage space occupied by infrequently accessed files. Run the **mmchattr** command or the **mmapplypolicy** command to identify and compress a few files or many files. Run file compression synchronously or defer it. If you defer it, you can run the **mmrestripefile** or **mmrestripefs** to complete the compression. You can decompress files with the same commands used to compress files. When a compressed file is read it is decompressed on the fly and remains compressed on disk. When a compressed file is overwritten, the parts of the file that overlap with the changed data are decompressed on disk synchronously in the granularity of ten data blocks. File compression in this release is designed to

be used only for compressing cold data or write-once objects and files. Compressing other types of data can result in performance degradation. File compression uses the zlib data compression library and favors saving space over speed.

GUI servers

The IBM Spectrum Scale system provides a GUI that can be used for managing and monitoring the system. Any server that provides this GUI service is referred to as a GUI server. If you need GUI service in the system, designate at least two nodes as GUI servers in the cluster. A maximum of three nodes can be designated as GUI servers. For more information on installing IBM Spectrum Scale using the GUI, see *IBM Spectrum Scale: Concepts, Planning, and Installation Guide*.

IBM Spectrum Scale management GUI

The management GUI helps to manage and monitor the IBM Spectrum Scale system. You can perform the following tasks through management GUI:

- Monitoring the performance of the system based on various aspects
- Monitoring system health
- Managing file systems
- Creating filesets and snapshots
- Managing Objects and NFS and SMB data exports
- Creating administrative users and defining roles for the users
- Creating object users and defining roles for them
- Defining default, user, group, and fileset quotas
- Monitoring the capacity details at various levels such as file system, pools, filesets, users, and user groups

Hadoop Support for IBM Spectrum Scale

IBM Spectrum Scale has been extended to work seamlessly in the Hadoop ecosystem and is available through a feature called File Placement Optimizer (FPO). Storing your Hadoop data using FPO allows you to gain advanced functions and the high I/O performance required for many big data operations. FPO provides Hadoop compatibility extensions to replace HDFS in a Hadoop ecosystem, with no changes required to Hadoop applications.

You can deploy a IBM Spectrum Scale using FPO as a file system platform for big data analytics. The topics in this guide covers a variety of Hadoop deployment architectures, including IBM BigInsights®, Platform Symphony®, or with a Hadoop distribution from another vendor to work with IBM Spectrum Scale.

IBM Spectrum Scale offers two kinds of interfaces for Hadoop applications to access File System data. One is IBM Spectrum Scale connector, which aligns with Hadoop Compatible File System architecture and APIs. The other is HDFS protocol, which provides a HDFS compatible interfaces.

For more information, see the following sections in the *IBM Spectrum Scale: Advanced Administration Guide*:

- *Hadoop support for IBM Spectrum Scale*
- *Configuring FPO*
- *Hadoop connector*
- *HDFS protocol*

IBM Spectrum Scale installation GUI

You can use the installation GUI to install the IBM Spectrum Scale system. For more information on how to launch the GUI installer, see the *Installing IBM Spectrum Scale using the graphical user interface (GUI)* section in *IBM Spectrum Scale: Concepts, Planning, and Installation Guide*.

Performance Monitoring Tool using the Installation Kit

The usage statement and optional arguments have changed during the installation of the toolkit. The new usage statement with options is as follows:

```
spectrumscale config perfmon [-h] [-l] [-r {on,off}]
```

For more information, see *IBM Spectrum Scale: Concepts, Planning, and Installation Guide*.

Protocols cluster disaster recovery (DR)
You can use the **mmcesdr** command to perform DR setup, failover, failback, backup, and restore actions. Protocols cluster DR uses the capabilities of Active File Management based Async Disaster Recovery (AFM DR) to provide a solution that allows an IBM Spectrum Scale cluster to fail over to another cluster and fail back, and backup and restore the protocol configuration information in cases where a secondary cluster is not available. For more information, see *Protocols cluster disaster recovery* in *IBM Spectrum Scale: Advanced Administration Guide*.

Quality of Service for I/O operations (QoS)
You can use the QoS capability to prevent I/O-intensive, long-running GPFS commands, called *maintenance commands*, from dominating file system performance and significantly delaying normal tasks that also compete for I/O resources. Determine the maximum capacity of your file system in I/O operations per second (IOPS) with the new **mmlsqos** command. With the new **mmchqos** command, assign a smaller share of IOPS to the QoS **maintenance** class, which includes all the maintenance commands. Maintenance command instances that are running at the same time compete for the IOPS allocated to the **maintenance** class, and are not allowed to exceed that limit.

Security mode for new clusters
Starting with IBM Spectrum Scale V4.2, the default security mode for new clusters is AUTHONLY. The **mmcrcluster** command sets the security mode to AUTHONLY when it creates the cluster and automatically generates a public/private key pair for authenticating the cluster. In the AUTHONLY security mode, the sending and receiving nodes authenticate each other with a TLS handshake and then close the TLS connection. Communication continues in the clear. The nodes do not encrypt transmitted data and do not check data integrity.

In IBM Spectrum Scale V4.1 or earlier, the default security mode is EMPTY. If you update a cluster from IBM Spectrum Scale V4.1 to V4.2 or later by running `mmchconfig release=LATEST`, the command checks the security mode. If the mode is EMPTY, the command issues a warning message but does not change the security mode of the cluster.

Snapshots
You can display information about a global snapshot without displaying information about fileset snapshots with the same name. You can display information about a fileset snapshot without displaying information about other snapshots that have the same name but are snapshots of other filesets.

spectrumscale Options
The **spectrumscale** command options for installing GPFS and deploying protocols have changed to remove **config enable** and to add **config perf**. For more information, see *IBM Spectrum Scale: Concepts, Planning, and Installation Guide*.

New options have been added to **spectrumscale setup** and **spectrumscale deploy** to disable prompting for the encryption/decryption secret. Note that if **spectrumscale setup --storesecret** is used, passwords will not be secure. New properties have been added to **spectrumscale cofig object** for setting password data instead of doing so through **enable object**. For more information, see *IBM Spectrum Scale: Administration and Programming Reference*.

The **spectrumscale** options for managing share ACLs have been added. For more information, see *IBM Spectrum Scale: Administration and Programming Reference*.

ssh and scp wrapper scripts
Starting with IBM Spectrum Scale V4.2, a cluster can be configured to use **ssh** and **scp** wrappers. The wrappers allow GPFS to run on clusters where remote root login through **ssh** is disabled. For more information, see the help topic "Running IBM Spectrum Scale without remote root login" in the *IBM Spectrum Scale: Administration and Programming Reference*.

Documented commands, structures, and subroutines
The following lists the modifications to the documented commands, structures, and subroutines:

New commands

The following commands are new:

- **mmcallhome**
- **mmcesdr**
- **mmchqos**
- **mmlsqos**

New structures

There are no new structures.

New subroutines

There are no new subroutines.

Changed commands

The following commands were changed:

- **mmadddisk**
- **mmaddnode**
- **mmapplypolicy**
- **mmauth**
- **mmbackup**
- **mmces**
- **mmchattr**
- **mmchcluster**
- **mmchconfig**
- **mmchdisk**
- **mmcheckquota**
- **mmchnode**
- **mmcrcluster**
- **mmdefragfs**
- **mmdeldisk**
- **mmdelfileset**
- **mmdelsnapshot**
- **mmdf**
- **mmfileid**
- **mmfsck**
- **mmlsattr**
- **mmlscluster**
- **mmlsconfig**
- **mmlssnapshot**
- **mmnfs**
- **mmobj**
- **mmperfmon**
- **mmprotocoltrace**
- **mmremotefs**
- **mmrestripefile**
- **mmrestripefs**
- **mmrpldisk**
- **mmsdrbackup**

- **mmsdrrestore**
- **mmsmb**
- **mmuserauth**
- **spectrumscale**

Changed structures
There are no changed structures.

Changed subroutines
There are no changed subroutines.

Deleted commands
There are no deleted commands.

Deleted structures
There are no deleted structures.

Deleted subroutines
There are no deleted subroutines.

Messages
The following lists the new, changed, and deleted messages:

New messages
6027-2354, 6027-2355, 6027-2356, 6027-2357, 6027-2358, 6027-2359, 6027-2360, 6027-2361, 6027-2362, 6027-3913, 6027-3914, 6027-3107, 6027-4016, 6027-3317, 6027-3318, 6027-3319, 6027-3320, 6027-3405, 6027-3406, 6027-3582, 6027-3583, 6027-3584, 6027-3585, 6027-3586, 6027-3587, 6027-3588, 6027-3589, 6027-3590, 6027-3591, 6027-3592, 6027-3593

Changed messages
6027-2299, 6027-887, 6027-888

Deleted messages
None.

Chapter 1. Overview of IBM Spectrum Scale Data Management API for GPFS

The Data Management Application Programming Interface (DMAPI) for GPFS allows you to monitor events associated with a GPFS file system or with an individual file. You can also manage and maintain file system data.

See the IBM Spectrum Scale FAQ in IBM Knowledge Center (www.ibm.com/support/knowledgecenter/STXKQY/gpfsclustersfaq.html) for the current limitations of DMAPI-managed file systems.

Note: Tivoli® Storage Manager for Space Management client (Hierarchical Storage Management) for GPFS file systems is not available for Windows.

DMAPI for GPFS is compliant with the Open Group's XDSM Standard and includes these features:

GPFS-specific DMAPI events

There are three GPFS-specific DMAPI events: events implemented in DMAPI for GPFS, optional events that are not implemented in DMAPI for GPFS, and GPFS-specific attribute events that are not part of the DMAPI standard.

For more information, see:

Events implemented in DMAPI for GPFS

These are the events, as defined in the *System Management: Data Storage Management (XDSM) API* Common Applications Environment (CAE) Specification C429, The Open Group, ISBN 1-85912-190-X, implemented in DMAPI for GPFS:

File system administration events

- mount
- preunmount
- unmount
- nospace

Namespace events

- create, postcreate
- remove, postremove
- rename, postrename
- symlink, postsymlink
- link, postlink

Data events

- read
- write
- truncate

Metadata events

- attribute
- destroy
- close

Pseudo event

- user event

GPFS guarantees that asynchronous events are delivered, except when the GPFS daemon fails. Events are enqueued to the session before the corresponding file operation completes. For further information on failures, see Chapter 5, "Failure and recovery of IBM Spectrum Scale Data Management API for GPFS," on page 41.

Optional events that are not implemented in DMAPI for GPFS

The following optional events, as defined in the *System Management: Data Storage Management (XDSM) API* Common Applications Environment (CAE) Specification C429, The Open Group, ISBN 1-85912-190-X, are **not** implemented in DMAPI for GPFS:

File system administration event

- debut

Metadata event

- cancel

GPFS-specific attribute events that are not part of the DMAPI standard

GPFS generates the following attribute events for DMAPI that are specific to GPFS and not part of the DMAPI standard:

- Pre-permission change
- Post-permission change

For additional information, refer to "GPFS-specific DMAPI events" on page 38.

DMAPI functions

All mandatory DMAPI functions and most optional functions that are defined in the *System Management: Data Storage Management (XDSM) API* Common Applications Environment (CAE) Specification C429, The Open Group, ISBN 1-85912-190-X, are implemented in DMAPI for GPFS.

For C declarations of all the functions implemented in DMAPI for GPFS, refer to the **dmapi.h** file located in the **/usr/lpp/mmfs/include** directory.

For changes and restrictions on functions in DMAPI for GPFS, see "Usage restrictions on DMAPI functions" on page 22, and "Semantic changes to DMAPI functions" on page 37.

Mandatory functions implemented in DMAPI for GPFS

These mandatory functions, as defined in the *System Management: Data Storage Management (XDSM) API* Common Applications Environment (CAE) Specification C429, The Open Group, ISBN 1-85912-190-X, are implemented in DMAPI for GPFS.

For C declarations of all the mandatory functions implemented in DMAPI for GPFS, refer to the **dmapi.h** file located in the **/usr/lpp/mmfs/include** directory. However, for a quick description of the mandatory functions and their applications, refer to the following set of functions:

dm_create_session
Create a new session.

dm_create_userevent
Create a pseudo-event message for a user.

dm_destroy_session
Destroy an existing session.

dm_fd_to_handle
Create a file handle using a file descriptor.

dm_find_eventmsg
Return the message for an event.

dm_get_allocinfo
Get a file's current allocation information.

dm_get_bulkattr
Get bulk attributes of a file system.

dm_get_config
Get specific data on DMAPI implementation.

dm_get_config_events
List all events supported by the DMAPI implementation.

dm_get_dirattrs
Return a directory's bulk attributes.

dm_get_eventlist
Return a list of an object's enabled events.

dm_get_events
Return the next available event messages.

dm_get_fileattr
Get file attributes.

dm_get_mountinfo
Return details from a mount event.

dm_get_region
Get a file's managed regions.

dm_getall_disp
For a given session, return the disposition of all file system's events.

dm_getall_sessions
Return all extant sessions.

dm_getall_tokens
Return a session's outstanding tokens.

dm_handle_cmp
Compare file handles.

dm_handle_free
Free a handle's storage.

dm_handle_hash
Hash the contents of a handle.

dm_handle_is_valid
Check a handle's validity.

dm_handle_to_fshandle
Return the file system handle associated with an object handle.

dm_handle_to_path
Return a path name from a file system handle.

dm_init_attrloc
Initialize a bulk attribute location offset.

dm_init_service
Initialization processing that is implementation-specific.

dm_move_event
Move an event from one session to another.

dm_path_to_fshandle
Create a file system handle using a path name.

dm_path_to_handle
Create a file handle using a path name.

dm_query_right
Determine an object's access rights.

dm_query_session
Query a session.

dm_read_invis
Read a file without using DMAPI events.

dm_release_right
Release an object's access rights.

dm_request_right
Request an object's access rights.

dm_respond_event
Issue a response to an event.

dm_send_msg
Send a message to a session.

dm_set_disp
For a given session, set the disposition of all file system's events.

dm_set_eventlist
For a given object, set the list of events to be enabled.

dm_set_fileattr
Set a file's time stamps, ownership and mode.

dm_set_region
Set a file's managed regions.

dm_write_invis
Write to a file without using DMAPI events.

Optional functions implemented in DMAPI for GPFS

These optional functions, as defined in the *System Management: Data Storage Management (XDSM) API* Common Applications Environment (CAE) Specification C429, The Open Group, ISBN 1-85912-190-X, are implemented in DMAPI for GPFS.

For C declarations of these optional functions implemented in DMAPI for GPFS, refer to the **dmapi.h** file located in the **/usr/lpp/mmfs/include** directory. However, for a quick description of the optional functions and their applications, refer to the following set of functions:

dm_downgrade_right
Change an exclusive access right to a shared access right.

dm_get_bulkall
Return a file system's bulk data management attributes.

dm_get_dmattr
Return a data management attribute.

dm_getall_dmattr
Return all data management attributes of a file.

dm_handle_to_fsid
Get a file system ID using its handle.

dm_handle_to_igen
Get inode generation count using a handle.

dm_handle_to_ino
Get inode from a handle.

dm_make_handle
Create a DMAPI object handle.

dm_make_fshandle
Create a DMAPI file system handle.

dm_punch_hole
Make a hole in a file.

dm_probe_hole
Calculate the rounded result of the area where it is assumed that a hole is to be punched.

dm_remove_dmattr
Delete a data management attribute.

dm_set_dmattr
Define or update a data management attribute.

dm_set_return_on_destroy
Indicate a DM attribute to return with destroy events.

dm_sync_by_handle
Synchronize the in-memory state of a file with the physical medium.

dm_upgrade_right
Change a currently held access right to be exclusive.

Optional functions that are not implemented in DMAPI for GPFS

There are optional functions that are not implemented in DMAPI for GPFS.

The following optional functions, as defined in the *System Management: Data Storage Management (XDSM) API* Common Applications Environment (CAE) Specification C429, The Open Group, ISBN 1-85912-190-X, are **not** implemented in DMAPI for GPFS:

dm_clear_inherit
Reset the inherit-on-create status of an attribute.

dm_create_by_handle
Define a file system object using a DM handle.

dm_getall_inherit
Return a file system's inheritable attributes.

dm_mkdir_by_handle
Define a directory object using a handle.

dm_obj_ref_hold
Put a hold on a file system object.

dm_obj_ref_query
Determine if there is a hold on a file system object.

dm_obj_ref_rele
Release the hold on a file system object.

dm_pending
Notify FS of slow DM application processing.

dm_set_inherit
Indicate that an attribute is inheritable.

dm_symlink_by_handle
Define a symbolic link using a DM handle.

GPFS-specific DMAPI functions

There are several GPFS-specific DMAPI functions that are not part of the DMAPI open standard.

The GPFS-specific functions are listed and described in "Definitions for GPFS-specific DMAPI functions" on page 24.

DMAPI configuration attributes

The *System Management: Data Storage Management (XDSM) API* Common Applications Environment (CAE) Specification C429, The Open Group, ISBN 1-85912-190-X defines a set of configuration attributes to be exported by each DMAPI implementation. These attributes specify which optional features are supported and give bounds on various resources.

The Data Management (DM) application can query the attribute values using the function **dm_get_config**. It can also query which events are supported, using the function **dm_get_config_events**.

The functions **dm_get_config** and **dm_get_config_events** receive a file handle from input arguments *hanp* and *hlen*. In GPFS, both functions ignore the handle, as the configuration is not dependent on the specific file or file system. This enables the DM application to query the configuration during initialization, when file handles may not yet be available.

Note: To guarantee that the most current values are being used, the DM application should always query the configuration at runtime by using **dm_get_config**.

Table 3 shows the attribute values that are used in DMAPI for GPFS:

Table 3. DMAPI configuration attributes

Name	Value
DM_CONFIG_BULKALL	1
DM_CONFIG_CREATE_BY_HANDLE	0
DM_CONFIG_DTIME_OVERLOAD	1
DM_CONFIG_LEGACY	1
DM_CONFIG_LOCK_UPGRADE	1

Table 3. DMAPI configuration attributes (continued)

Name	Value
DM_CONFIG_MAX_ATTR_ON_DESTROY	1022
DM_CONFIG_MAX_ATTRIBUTE_SIZE	1022
DM_CONFIG_MAX_HANDLE_SIZE	32
DM_CONFIG_MAX_MANAGED_REGIONS	32
DM_CONFIG_MAX_MESSAGE_DATA	4096
DM_CONFIG_OBJ_REF	0
DM_CONFIG_PENDING	0
DM_CONFIG_PERS_ATTRIBUTE	1
DM_CONFIG_PERS_EVENTS	1
DM_CONFIG_PERS_INHERIT_ATTRIBS	0
DM_CONFIG_PERS_MANAGED_REGIONS	1
DM_CONFIG_PUNCH_HOLE	1
DM_CONFIG_TOTAL_ATTRIBUTE_SPACE	7168
DM_CONFIG_WILL_RETRY	0

Attribute value **DM_CONFIG_TOTAL_ATTRIBUTE_SPACE** is per file. The entire space is available for opaque attributes. Non-opaque attributes (event list and managed regions) use separate space.

DMAPI restrictions for GPFS

All DMAPI APIs must be called from nodes that are in the cluster where the file system is created. DMAPI APIs may **not** be invoked from a remote cluster.

In addition to the DMAPI API restrictions, GPFS places the following restrictions on the use of file system snapshots when you have DMAPI enabled:

- Snapshots cannot coexist with file systems using GPFS 3.1 or earlier.
- GPFS 3.2 and later permits snapshots with DMAPI-enabled file systems. However, GPFS places the following restrictions on DMAPI access to the snapshot files:
  - The DM server may read files in a snapshot using **dm_read_invis**.
  - The DM server is not allowed to modify or delete the file using **dm_write_invis** or **dm_punch_hole**.
  - The DM server is not allowed to establish a managed region on the file.
  - Snapshot creation or deletion does not generate DMAPI namespace events.
  - Snapshots of a file are not managed regardless of the state of the original file and they do not inherit the DMAPI attributes of the original file.

Chapter 2. Concepts of IBM Spectrum Scale Data Management API for GPFS

The XDSM standard is intended mainly for a single-node environment. Some of the key concepts in the standard such as sessions, event delivery, mount and unmount, and failure and recovery, are not well defined for a multiple-node environment such as GPFS.

For a list of restrictions and coexistence considerations, see "Usage restrictions on DMAPI functions" on page 22.

All DMAPI APIs must be called from nodes that are in the cluster where the file system is created.

Key concepts of DMAPI for GPFS include these areas:

Sessions

In GPFS, a session is associated only with the node on which the session was created. This node is known as the *session node*.

Events are generated at any node where the file system is mounted. The node on which a given event is generated is called the *source node* of that event. The event is delivered to a session queue on the session node.

There are restrictions as to which DMAPI functions can and cannot be called from a node other than the session node. In general, functions that change the state of a session or event can only be called on the session node. For example, the maximum number of DMAPI sessions that can be created on a node is 4000. See "Usage restrictions on DMAPI functions" on page 22 for details.

Session ids are unique over time within a GPFS cluster. When an existing session is assumed, using **dm_create_session**, the new session id returned is the same as the old session id.

A session fails when the GPFS daemon fails on the session node. Unless this is a total failure of GPFS on all nodes, the session is recoverable. The DM application is expected to assume the old session, possibly on another node. This will trigger the reconstruction of the session queue. All pending synchronous events from surviving nodes are resubmitted to the recovered session queue. Such events will have the same token id as before the failure, except mount events. Asynchronous events, on the other hand, are lost when the session fails. See Chapter 5, "Failure and recovery of IBM Spectrum Scale Data Management API for GPFS," on page 41 for information on failure and recovery.

Events

Events arrive on a session queue from any of the nodes in the GPFS cluster.

The source node of the event is identified by the **ev_nodeid** field in the header of each event message in the structure **dm_eventmsg**. The identification is the GPFS cluster data node number, which is attribute **node_number** in the **mmsdrfs2** file for a PSSP node or **mmsdrfs** file for any other type of node.

Data Management events are generated only if the following two conditions are true:

1. The event is enabled.
2. It has a disposition.

A file operation will fail with the **EIO** error if there is no disposition for an event that is enabled and would otherwise be generated.

A list of enabled events can be associated individually with a file and globally with an entire file system. The XDSM standard leaves undefined the situation where the individual and the global event lists are in conflict. In GPFS, such conflicts are resolved by always using the individual event list, if it exists.

Note: The XDSM standard does not provide the means to remove the individual event list of a file. Thus, there is no way to enable or disable an event for an entire file system without explicitly changing each conflicting individual event list.

In GPFS, event lists are persistent.

Event dispositions are specified per file system and are not persistent. They must be set explicitly after the session is created.

Event generation mechanisms have limited capacity. In case resources are exceeded, new file operations will wait indefinitely for free resources.

File operations wait indefinitely for a response from synchronous events. The **dmapiEventTimeout** configuration attribute on the **mmchconfig** command, can be used to set a timeout on events that originate from NFS file operations. This is necessary because NFS servers have a limited number of threads that cannot be blocked for long periods of time. Refer to "GPFS configuration attributes for DMAPI" on page 18 and "Support for NFS" on page 14.

The XDSM standard permits asynchronous events to be discarded at any time. In GPFS, asynchronous events are guaranteed when the system runs normally, but may be lost during abnormal conditions, such as failure of GPFS on the session node. Asynchronous events are delivered in a timely manner. That is, an asynchronous event is enqueued to the session before the corresponding file operation completes.

Figure 1 on page 11, shows the flow of a typical synchronous event in a multiple-node GPFS environment. The numbered arrows in the figure correspond to the following steps:

1. The user application on the source node performs a file operation on a GPFS file. The file operation thread generates a synchronous event and blocks, waiting for a response.
2. GPFS on the source node sends the event to GPFS on the session node, according to the disposition for that event. The event is enqueued to the session queue on the session node.
3. The Data Management application on the session node receives the event (using **dm_get_events**) and handles it.
4. The Data Management application on the session node responds to the event (using **dm_respond_event**).
5. GPFS on the session node sends the response to GPFS on the source node.

6. GPFS on the source node passes the response to the file operation thread and unblocks it. The file operation continues.

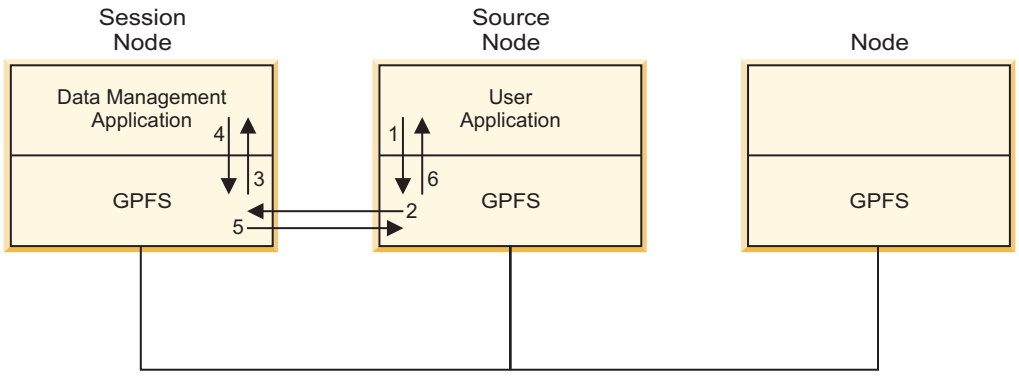

Figure 1. Flow of a typical synchronous event in a multiple-node GPFS environment

Reliable DMAPI destroy events

A metadata destroy event is generated when the operating system has destroyed an object. This type of event is different from a remove event, which is a namespace event and is not related to the destruction of an object. A reliable destroy event supports synchronous destroy events in the same way that other synchronous events do. When a synchronous event is generated, a user process is suspended in the kernel; it will be suspended until a DM application issues an explicit response to the event. The DM application at the session that supports the reliable destroy event must be capable of handling the synchronous destroy event. In other words, it must respond to the **DM_EVENT_DESTROY** event with **DM_RESPOND_EVENT**. Otherwise, the event will wait forever at the session node for a response. Based on this, it is recommended that the cluster not be made up of nodes that are running back-level code and new code, because the destroy event is not reliable in a mixed environment.

Mount and unmount

The XDSM standard implicitly assumes that there is a single mount, pre-unmount and unmount event per file system. In GPFS, a separate mount event is generated by each mount operation on each node. Similarly, if the pre-unmount and unmount events are enabled, they are generated by each unmount operation on each node. Thus, there may be multiple such events for the same file system.

To provide additional information to the DM application, the mode field in the respective event message structures (**me_mode** for mount, and **ne_mode** for pre-unmount and unmount) has a new flag, **DM_LOCAL_MOUNT**, which is not defined in the standard. When the flag is set, the mount or unmount operation is local to the session node. In addition, the new field **ev_nodeid** in the header of the event message can be used to identify the source node where the mount or unmount operation was invoked. The identification is the GPFS cluster data node number, which is attribute **node_number** in the **mmsdrfs2** file for a PSSP node or **mmsdrfs** file for any other type of node.

The mount event is sent to multiple sessions that have a disposition for it. If there is no disposition for the mount event, the mount operation fails with an **EIO** error.

There is no practical way to designate the *last* unmount, since there is no serialization of all mount and unmount operations of each file system. Receiving an unmount event with the value 0 in the **ne_retcode** field is no indication that there will be no further events from the file system.

An unmount initiated internally by the GPFS daemon, due to file system forced unmount or daemon shutdown, will not generate any events. Consequently, there need not be a match between the number of mount events and the number of pre-unmount or unmount events for a given file system.

The **dmapiMountTimeout** attribute on the **mmchconfig** command enables blocking the mount operation for a limited time until some session has set the mount disposition. This helps GPFS and the DM application synchronize during initialization. See "GPFS configuration attributes for DMAPI" on page 18 and "Initializing the Data Management application" on page 20.

Mount events are enqueued on the session queue ahead of any other events. This gives mount events a higher priority, which improves the response time for mount events when the queue is very busy.

If the **DM_UNMOUNT_FORCE** flag is set in the pre-unmount event message, the response of the DM application to the pre-unmount event is ignored, and the forced unmount proceeds. If the **DM_LOCAL_MOUNT** flag is also set, the forced unmount will result in the loss of all access rights of the given file system that are associated with any local session.

If the unmount is not forced (the **DM_UNMOUNT_FORCE** flag is not set), and the **DM_LOCAL_MOUNT** flag is set, the DM application is expected to release all access rights on files of the given file system associated with any local session. If any access rights remain held after the **DM_RESP_CONTINUE** response is given, the unmount will fail with **EBUSY**. This is because access rights render the file system busy, similar to other locks on files.

The function **dm_get_mountinfo** can be called from any node, even if the file system is not mounted on that node. The **dm_mount_event** structure returned by the **dm_get_mountinfo** function provides the following enhanced information. The **me_mode** field contains two new flags, **DM_LOCAL_MOUNT** and **DM_REMOTE_MOUNT**. At least one of the two flags is always set. When both flags are set simultaneously, it is an indication that the file system is mounted on the local node, as well as one or more other (remote) nodes. When only **DM_LOCAL_MOUNT** is set, it is an indication that the file system is mounted on the local node but not on any other node. When only **DM_REMOTE_MOUNT** is set, it is an indication that the file system is mounted on some remote node, but not on the local node.

In the latter case (only **DM_REMOTE_MOUNT** is set), the fields **me_roothandle** and **me_handle2** (the mount point handle) in the **dm_mount_event** structure are set to **DM_INVALID_HANDLE**. Also in this case, the **me_name1** field (the mount point path) is taken from the stanza in the file **/etc/filesystems** on one of the remote nodes (with the use of GPFS cluster data, the stanzas on all nodes are identical).

The enhanced information provided by the **dm_get_mountinfo** function can be useful during the processing of mount and pre-unmount events. For example, before responding to a mount event from a remote (non-session) node, **dm_get_mountinfo** could be invoked to find out whether the file system is already mounted locally at the session node, and if not, initiate a local mount. On receiving a pre-unmount event from the local session node, it is possible to find out whether the file system is still mounted elsewhere, and if so, fail the local unmount or delay the response until after all remote nodes have unmounted the file system.

Note: The **DM_REMOTE_MOUNT** flag is redundant in the **dm_mount_event** structure obtained from the mount event (as opposed to the **dm_get_mountinfo** function).

Tokens and access rights

A DMAPI token is an identifier of an outstanding event (a synchronous event that the DM application has received and is currently handling). The token is unique over time in the cluster. The token becomes invalid when the event receives a response.

The main purpose of tokens is to convey access rights in DMAPI functions. Access rights are associated with a specific event token. A function requiring access rights to some file may present an event token that has the proper access rights.

DMAPI functions can also be invoked using **DM_NO_TOKEN**, in which case sufficient access protection is provided for the duration of the operation. This is semantically equivalent to holding an access right, but no access right on the file is actually acquired.

In GPFS, when an event is received, its token has no associated access rights.

DM access rights are implemented in GPFS using an internal lock on the file. Access rights can be acquired, changed, queried, and released only at the session node. This is an implementation restriction caused by the GPFS locking mechanisms.

In GPFS, it is not possible to set an access right on an entire file system from the file system handle. Thus, DMAPI function calls that reference a file system, using a file system handle, are not allowed to present a token and must specify **DM_NO_TOKEN**. For the same reason, functions that acquire or change access rights are not allowed to present a file system handle.

Holding access rights renders the corresponding file system busy at the session node, preventing normal (non-forced) unmount. This behavior is similar to that of other locks on files. When receiving a pre-unmount event, the DM application is expected to release all access rights before responding. Otherwise, the unmount operation will fail with an **EBUSY** error.

All access rights associated with an event token are released when the response is given. There is no transfer of access rights from DMAPI to the file operation thread. The file operation will acquire any necessary locks after receiving the response of the event.

Parallelism in Data Management applications

Given the multiple-node environment of GPFS, it is desirable to exploit parallelism in the Data Management application as well.

This can be accomplished in several ways:

- On a given session node, multiple DM application threads can access the same file in parallel, using the same session. There is no limit on the number of threads that can invoke DMAPI functions simultaneously on each node.
- Multiple sessions, each with event dispositions for a different file system, can be created on separate nodes. Thus, files in different file systems can be accessed independently and simultaneously, from different session nodes.
- Dispositions for events of the same file system can be partitioned among multiple sessions, each on a different node. This distributes the management of one file system among several session nodes.
- Although GPFS routes all events to a single session node, data movement may occur on multiple nodes. The function calls **dm_read_invis**, **dm_write_invis**, **dm_probe_hole**, and **dm_punch_hole** are honored from a root process on another node, provided it presents a session ID for an established session on the session node.

  A DM application may create a *worker process*, which exists on any node within the GPFS cluster. This worker process can move data to or from GPFS using the **dm_read_invis** and **dm_write_invis** functions. The worker processes must adhere to these guidelines:

  1. They must run as root.
  2. They must present a valid session ID that was obtained on the session node.
  3. All writes to the same file which are done in parallel must be done in multiples of the file system block size, to allow correct management of disk blocks on the writes.

4. No DMAPI calls other than **dm_read_invis**, **dm_write_invis**, **dm_probe_hole**, and **dm_punch_hole** may be issued on nodes other than the session node. This means that any rights required on a file must be obtained within the session on the session node, prior to the data movement.
5. There is no persistent state on the nodes hosting the worker process. It is the responsibility of the DM application to recover any failure which results from the failure of GPFS or the data movement process.

Data Management attributes

Data Management attributes can be associated with any individual file. There are opaque and non-opaque attributes.

An opaque attribute has a unique name, and a byte string value which is not interpreted by the DMAPI implementation. Non-opaque attributes, such as managed regions and event lists, are used internally by the DMAPI implementation.

DM attributes are persistent. They are kept in a hidden file in the file system.

GPFS provides two *quick access* single-bit opaque DM attributes for each file, stored directly in the inode. These attributes are accessible through regular DMAPI functions, by specifying the reserved attribute names **_GPFSQA1** and **_GPFSQA2** (where **_GPF** is a reserved prefix). The attribute data must be a single byte with contents 0 or 1.

Support for NFS

A DM application could be slow in handling events. NFS servers have a limited number of threads which must not all be blocked simultaneously for extended periods of time. GPFS provides a mechanism to guarantee progress of NFS file operations that generate data events without blocking the server threads indefinitely.

The mechanism uses a timeout on synchronous events. Initially the NFS server thread is blocked on the event. When the timeout expires, the thread unblocks and the file operation fails with an **ENOTREADY** error code. The event itself continues to exist and will eventually be handled. When a response for the event arrives at the source node it is saved. NFS is expected to periodically retry the operation. The retry will either find the response which has arrived between retries, or cause the operation to fail again with **ENOTREADY**. After repeated retries, the operation is eventually expected to succeed.

The interval is configurable using the **dmapiEventTimeout** configuration attribute on the **mmchconfig** command. See “GPFS configuration attributes for DMAPI” on page 18. The default is no timeout.

The timeout mechanism is activated only for data events (read, write, truncate), and only when the file operation comes from NFS.

Quota

GPFS supports user quota. When **dm_punch_hole** is invoked, the file owner's quota is adjusted by the disk space that is freed. The quota is also adjusted when **dm_write_invis** is invoked and additional disk space is consumed.

Since **dm_write_invis** runs with root credentials, it will never fail due to insufficient quota. However, it is possible that the quota of the file owner will be exceeded as a result of the invisible write. In that case the owner will not be able to perform further file operations that consume quota.

Memory mapped files

In GPFS, a read event or a write event will be generated (if enabled) at the time the memory mapping of a file is established.

No events will be generated during actual mapped access, regardless of the setting of the event list or the managed regions. Access to the file with regular file operations, while the file is memory mapped, will generate events, if such events are enabled.

To protect the integrity of memory mapped access, the DM application is not permitted to punch a hole in a file while the file is memory mapped. If the DM application calls **dm_punch_hole** while the file is memory mapped, the error code **EBUSY** will be returned.

Chapter 3. Administration of IBM Spectrum Scale Data Management API for GPFS

To set up the DMAPI for GPFS, install the DMAPI files that are included in the GPFS installation package, and then choose the configuration attributes for DMAPI with the **mmchconfig** command. For each file system that you want DMAPI access, enable DMAPI with the **-z** flag of the **mmcrfs** or **mmchfs** command.

All DMAPI APIs must be called from nodes that are in the cluster where the file system is created. DMAPI APIs may **not** be invoked from a remote cluster. The GPFS daemon and each DMAPI application must be synchronized to prevent failures.

Administration of DMAPI for GPFS includes:

Required files for implementation of Data Management applications

The installation image for GPFS contains the required files for implementation of Data Management applications.

For more information about installation, see the *IBM Spectrum Scale: Concepts, Planning, and Installation Guide*.

The required files are:

dmapi.h
: The header file that contains the C declarations of the DMAPI functions.

  This header file must be included in the source files of the DM application.

  The file is installed in directory: **/usr/lpp/mmfs/include**.

dmapi_types.h
: The header file that contains the C declarations of the data types for the DMAPI functions and event messages.

  The header file **dmapi.h** includes this header file.

  The file is installed in directory: **/usr/lpp/mmfs/include**.

libdmapi.a
: The library that contains the DMAPI functions.

  The library **libdmapi.a** consists of a single shared object, which is built with auto-import of the system calls that are listed in the export file **dmapi.exp**.

  The file is installed in directory: **/usr/lpp/mmfs/lib**.

dmapi.exp
: The export file that contains the DMAPI system call names.

  The file **dmapi.exp** needs to be explicitly used only if the DM application is to be explicitly built with static binding, using the binder options **-bnso -bI:dmapi.exp**.

The file is installed in directory: **/usr/lpp/mmfs/lib**.

dmapicalls, dmapicalls64
Module loaded during processing of the DMAPI functions.

The module is installed in directory: **/usr/lpp/mmfs/bin**.

Notes:

- On Linux nodes running DMAPI, the required files **libdmapi.a**, **dmapi.exp**, **dmapicalls**, and **dmapicalls64** are replaced by **libdmapi.so**.
- If you are compiling with a non-IBM compiler on AIX nodes, you must compile DMAPI applications with -D_AIX.

GPFS configuration attributes for DMAPI

GPFS uses several attributes for DMAPI that define various timeout intervals. These attributes can be changed with the **mmchconfig** command.

The DMAPI configuration attributes are:

dmapiDataEventRetry
Controls how GPFS handles the data event when it is enabled again right after this event is handled by the DMAPI application. Valid values are:

-1 Specifies that GPFS will always regenerate the event as long as it is enabled. This value should only be used when the DMAPI application recalls and migrates the same file in parallel by many processes at the same time.

0 Specifies to never regenerate the event. This value should not be used if a file could be migrated and recalled at the same time.

Positive Number
Specifies how many times the data event should be retried. The default is 2, which should be enough to cover most DMAPI applications. Unless a special situation occurs, you can increase this to a larger number or even set this to **-1** to always regenerate the events. Unless you perform careful testing, IBM recommends that you never change the default setting.

dmapiEventTimeout
Controls the blocking of file operation threads of NFS, while in the kernel waiting for the handling of a DMAPI synchronous event. The parameter value is the maximum time, in milliseconds, the thread will block. When this time expires, the file operation returns **ENOTREADY**, and the event continues asynchronously. The NFS server is expected to repeatedly retry the operation, which eventually will find the response of the original event and continue. This mechanism applies only to read, write, and truncate events, and only when such events come from NFS server threads.

The timeout value is given in milliseconds. The value 0 indicates immediate timeout (fully asynchronous event). A value greater than or equal to 86400000 (which is 24 hours) is considered 'infinity' (no timeout, fully synchronous event). The default value is 86400000. See also “Support for NFS” on page 14.

dmapiFileHandleSize
Controls the size of file handles generated by GPFS. For clusters created with GPFS 3.2 or later, the default DMAPI file handle size is 32 bytes. For clusters created prior to GPFS 3.2, the default DMAPI file handle size is 16 bytes. After all of the nodes in the cluster are upgraded to the latest GPFS release and you have also run the **mmchconfig release=LATEST** command, then you can change the file handle size to 32 bytes by issuing the command: **mmchconfig dmapiFileHandleSize=32**.

Note: To change the DMAPI file handle size, GPFS must be stopped on all nodes in the cluster.

dmapiMountEvent
Controls the generation of the **mount**, **preunmount**, and **unmount** events. Valid values are:

all Specifies that **mount**, **preunmount**, and **unmount** events are generated on each node. This is the default behavior.

LocalNode
Specifies that **mount**, **preunmount**, and **unmount** events are generated only if the node is a session node.

SessionNode
Specifies that **mount**, **preunmount**, and **unmount** events are generated on each node and are delivered to the session node, but the session node will respond with **DM_RESP_CONTINUE** to the event node without delivering the event to the DMAPI application, unless the event is originated from the **SessionNode** itself.

dmapiMountTimeout
Controls the blocking of mount operations, waiting for a disposition for the mount event to be set. This timeout is activated at most once on each node, by the first mount of a file system which has DMAPI enabled, and only if there has never before been a mount disposition. Any mount operation on this node that starts while the timeout period is active will wait for the mount disposition. The parameter value is the maximum time, in seconds, that the mount operation will wait for a disposition. When this time expires and there still is no disposition for the mount event, the mount operation fails, returning the **EIO** error.

The timeout value is given in full seconds. The value 0 indicates immediate timeout (immediate failure of the mount operation). A value greater than or equal to 86400 (which is 24 hours) is considered 'infinity' (no timeout, indefinite blocking until there is a disposition). The default value is 60. See also “Mount and unmount” on page 11 and “Initializing the Data Management application” on page 20.

dmapiSessionFailureTimeout
Controls the blocking of file operation threads, while in the kernel, waiting for the handling of a DMAPI synchronous event that is enqueued on a session that has suffered a failure. The parameter value is the maximum time, in seconds, the thread will wait for the recovery of the failed session. When this time expires and the session has not yet recovered, the event is aborted and the file operation fails, returning the **EIO** error.

The timeout value is given in full seconds. The value 0 indicates immediate timeout (immediate failure of the file operation). A value greater than or equal to 86400 (which is 24 hours) is considered 'infinity' (no timeout, indefinite blocking until the session recovers). The default value is 0. See also Chapter 5, “Failure and recovery of IBM Spectrum Scale Data Management API for GPFS,” on page 41 for details on session failure and recovery.

For more information about the **mmchconfig** command, see the *IBM Spectrum Scale: Administration and Programming Reference*.

Enabling DMAPI for a file system

DMAPI must be enabled individually for each file system.

DMAPI can be enabled for a file system when the file system is created, using the **-z yes** option on the **mmcrfs** command. The default is **-z no**. The setting can be changed when the file system is not mounted anywhere, using the **-z yes | no** option on the **mmchfs** command. The setting is persistent.

The current setting can be queried using the **-z** option on the **mmlsfs** command.

While DMAPI is disabled for a given file system, no events are generated by file operations of that file system. Any DMAPI function calls referencing that file system fail with an **EPERM** error.

When **mmchfs -z no** is used to disable DMAPI, existing event lists, extended attributes, and managed regions in the given file system remain defined, but will be ignored until DMAPI is re-enabled. The command **mmchfs -z no** should be used with caution, since punched holes, if any, are no longer protected by managed regions.

If the file system was created with a release of GPFS earlier than GPFS 1.3, the file system descriptor must be upgraded before attempting to enable DMAPI. The upgrade is done using the **-V** option on the **mmchfs** command.

For more information about GPFS commands, see the *IBM Spectrum Scale: Administration and Programming Reference*.

Initializing the Data Management application

All DMAPI APIs must be called from nodes that are in the cluster where the file system is created. DMAPI APIs may **not** be invoked from a remote cluster.

During initialization of GPFS, it is necessary to synchronize the GPFS daemon and the DM application to prevent mount operations from failing. There are two mechanisms to accomplish this:

1. The shell script **gpfsready** invoked by the GPFS daemon during initialization.
2. A timeout interval, allowing mount operations to wait for a disposition to be set for the mount event.

During GPFS initialization, the daemon invokes the shell script **gpfsready**, located in directory **/var/mmfs/etc**. This occurs as the file systems are starting to be mounted. The shell script can be modified to start or restart the DM application. Upon return from this script, a session should have been created and a disposition set for the mount event. Otherwise, mount operations may fail due to a lack of disposition.

In a multiple-node environment such as GPFS, usually only a small subset of the nodes are session nodes, having DM applications running locally. On a node that is not a session node, the **gpfsready** script can be modified to synchronize between the local GPFS daemon and a remote DM application. This will prevent mount from failing on any node.

A sample shell script **gpfsready.sample** is installed in directory **/usr/lpp/mmfs/samples**.

If no mount disposition has ever been set in the cluster, the first external mount of a DMAPI-enabled file system on each node will activate a timeout interval on that node. Any mount operation on that node that starts during the timeout interval will wait for the mount disposition until the timeout expires. The timeout interval is configurable using the **dmapiMountTimeout** configuration attribute on the **mmchconfig** command (the interval can even be made infinite). A message is displayed at the beginning of the wait. If there is still no disposition for the mount event when the timeout expires, the mount operation will fail with an **EIO** error code. See "GPFS configuration attributes for DMAPI" on page 18 for more information on **dmapiMountTimeout**.

Chapter 4. Specifications of enhancements for IBM Spectrum Scale Data Management API for GPFS

DMAPI for GPFS provides numerous enhancements in data structures and functions.

These enhancements are provided mainly by the multiple-node environment. Some data structures have additional fields. Many functions have usage restrictions, changes in semantics, and additional error codes. The enhancements are in these areas:

Enhancements to data structures

This is a description of GPFS enhancements to data structures defined in the XDSM standard.

For complete C declarations of all the data structures that are used in DMAPI for GPFS, refer to the **dmapi_types.h** file located in the **/usr/lpp/mmfs/include** directory.

- All file offsets and sizes in DMAPI data structures are 64 bits long.
- Names or path names that are passed in event messages are character strings, terminated by a null character. The length of the name buffer, as specified in the **dm_vardata_t** structure, includes the null character.
- The **dm_region_t** structure has a new 4-byte field, **rg_opaque**. The DMAPI implementation does not interpret **rg_opaque**. The DM application can use this field to store additional information within the managed region.
- The **dt_change** field in the **dm_stat** structure is not implemented in the inode. The value will change each time it is returned by the **dm_get_fileattr** function.
- The **dt_dtime** field in the **dm_stat** structure is overloaded on the **dt_ctime** field.
- The **dm_eventmsg** structure has a 4 byte field, **ev_nodeid** that uniquely identifies the node that generated the event. The id is the GPFS cluster data node number, which is attribute **node_number** in the **mmsdrfs2** file for a PSSP node or **mmsdrfs** file for any other type of node.
- The **ne_mode** field in the **dm_namesp_event** structure has an additional flag, **DM_LOCAL_MOUNT**. For the events preunmount and unmount when this flag is set, the unmount operation is local to the session node. See "Mount and unmount" on page 11. The **me_mode** field in the **dm_mount_event** structure has two additional flags; **DM_LOCAL_MOUNT**, and **DM_REMOTE_MOUNT**. See "Mount and unmount" on page 11.
- There are two 'quick access' single-bit opaque DM attributes for each file, stored directly in the inode. See "Data Management attributes" on page 14.
- The data type **dm_eventset_t** is implemented as a bit map, containing one bit for each event that is defined in DMAPI. The bit is set if, and only if, the event is present.

  Variables of type **dm_eventset_t** should be manipulated only using special macros. The XDSM standard provides a basic set of such macros. GPFS provides a number of additional macros. The names of all such macros begin with the prefix **DMEV_**.

  This is the list of additional macros that are provided in DMAPI for GPFS:

DMEV_ALL(eset)
Add all events to **eset**

DMEV_ISZERO(eset)
Check if **eset** is empty

DMEV_ISALL(eset)
Check if **eset** contains all events

DMEV_ADD(eset1, eset2)
Add to **eset2** all events in **eset1**

DMEV_REM(eset1, eset2)
Remove from **eset2** all events in **eset1**

DMEV_RES(eset1, eset2)
Restrict **eset2** by **eset1**

DMEV_ISEQ(eset1, eset2)
Check if **eset1** and **eset2** are equal

DMEV_ISDISJ(eset1, eset2)
Check if **eset1** and **eset2** are disjoint

DMEV_ISSUB(eset2)
Check if **eset1** is a subset of **eset2**

DMEV_NORM(eset)
Normalize the internal format of **eset**, clearing all unused bits

- DMAPI for GPFS provides a set of macros for comparison of token ids (value of type **dm_token_t**).

  DM_TOKEN_EQ (x,y)
  Check if **x** and **y** are the same

  DM_TOKEN_NE (x,y)
  Check if **x** and **y** are different

  DM_TOKEN_LT (x,y)
  Check if **x** is less than **y**

  DM_TOKEN_GT (x,y)
  Check if **x** is greater than **y**

  DM_TOKEN_LE (x,y)
  Check if **x** is less than or equal to **y**

  DM_TOKEN_GE (x,y)
  Check if **x** is greater than or equal to **y**

Usage restrictions on DMAPI functions

There are usage restrictions on the DMAPI for GPFS functions.

- The maximum number of DMAPI sessions that can be created on a node is 4000.
- Root credentials are a prerequisite for invoking any DMAPI function, otherwise the function fails with an **EPERM** error code.
- DMAPI functions are unable to run if the GPFS kernel extension is not loaded, or if the runtime module **dmapicalls** is not installed. An **ENOSYS** error code is returned in this case.
- Invoking a DMAPI function that is not implemented in GPFS results in returning the **ENOSYS** error code.
- DMAPI functions will fail, with the **ENOTREADY** error code, if the local GPFS daemon is not running.
- DMAPI functions will fail, with the **EPERM** error code, if DMAPI is disabled for the file system that is referenced by the file handle argument.

- DMAPI functions cannot access GPFS reserved files, such as quota files, inode allocation maps, and so forth. The **EBADF** error code is returned in this case.
- GPFS does not support access rights on entire file systems (as opposed to individual files). Hence, DMAPI function calls that reference a file system (with a file system handle) cannot present a token, and must use **DM_NO_TOKEN**. Functions affected by this restriction are:
  - **dm_set_eventlist**
  - **dm_get_eventlist**
  - **dm_set_disp**
  - **dm_get_mountinfo**
  - **dm_set_return_on_destroy**
  - **dm_get_bulkattr**
  - **dm_get_bulkall**

  If a token is presented, these functions fail with the **EINVAL** error code.
- DMAPI functions that acquire, change, query, or release access rights, must not present a file system handle. These functions are:
  - **dm_request_right**
  - **dm_upgrade_right**
  - **dm_downgrade_right**
  - **dm_release_right**
  - **dm_query_right**

  If a file system handle is presented, these functions fail with the **EINVAL** error code.
- The function **dm_request_right**, when invoked without wait (the *flags* argument has a value of 0), will almost always fail with the **EAGAIN** error. A GPFS implementation constraint prevents this function from completing successfully without wait, even if it is known that the requested access right is available. The **DM_RR_WAIT** flag must always be used. If the access right is available, there will be no noticeable delay.
- DMAPI function calls that reference a specific token, either as input or as output, can be made only on the session node. Otherwise, the call fails with the **EINVAL** error code.
- DMAPI function calls that reference an individual file by handle must be made on the session node. The corresponding file system must be mounted on the session node. The call fails with **EINVAL** if it is not on the session node, and with **EBADF** if the file system is not mounted.
- DMAPI function calls that reference a file system by handle (as opposed to an individual file) can be made on any node, not just the session node. The relevant functions are:
  - **dm_set_eventlist**
  - **dm_get_eventlist**
  - **dm_set_disp**
  - **dm_get_mountinfo**
  - **dm_set_return_on_destroy**
  - **dm_get_bulkattr**
  - **dm_get_bulkall**

  For **dm_get_bulkattr** and **dm_get_bulkall**, the system file must be mounted on the node that is making the call. For the other functions, the file system must be mounted on some node, but not necessarily on the node that is making the call. As specified previously, all such function calls must use **DM_NO_TOKEN**. The function fails with the **EBADF** error code if the file system is not mounted as required.
- The function **dm_punch_hole** will fail with the **EBUSY** error code if the file to be punched is currently memory-mapped.

- The function **dm_move_event** can only be used when the source session and the target session are on the same node. The function must be called on the session node. Otherwise, the function fails with the **EINVAL** error code.
- The function **dm_create_session**, when providing an existing session id in the argument *oldsid*, can only be called on the session node, except after session node failure. Otherwise, the call will return the **EINVAL** error code.
- The function **dm_destroy_session** can only be called on the session node, otherwise the call will fail with the **EINVAL** error code.
- The function **dm_set_fileattr** cannot change the file size. If the **dm_at_size** bit in the attribute mask is set, the call fails with the **EINVAL** error code.
- DMAPI functions that reference an event with a token fail with the **ESRCH** error code, if the event is not in an outstanding state. This is related to session recovery. See Chapter 5, "Failure and recovery of IBM Spectrum Scale Data Management API for GPFS," on page 41 for details on session failure and recovery.

For additional information about:
- Semantic changes to the DMAPI for GPFS functions, see "Semantic changes to DMAPI functions" on page 37.
- C declarations of all functions in DMAPI for GPFS, refer to the **dmapi.h** file located in the **/usr/lpp/mmfs/include** directory.

Definitions for GPFS-specific DMAPI functions

The GPFS-specific DMAPI functions are not part of the DMAPI open standard.

You can use the following GPFS-specific DMAPI functions to work with file system snapshots:
- "dm_handle_to_snap" on page 25
- "dm_make_xhandle" on page 26

You can use the following GPFS-specific DMAPI functions to make asynchronous updates to attributes, managed regions, and event lists on files:
- "dm_remove_dmattr_nosync" on page 28
- "dm_set_dmattr_nosync" on page 30
- "dm_set_eventlist_nosync" on page 32
- "dm_set_region_nosync" on page 34

You can use the following GPFS-specific DMAPI function to make the previously listed asynchronous updates persistent by flushing them to disk:
- "dm_sync_dmattr_by_handle" on page 36

dm_handle_to_snap

Extracts a snapshot ID from a handle.

Synopsis

```
int dm_handle_to_snap(
    void          *hanp,          /* IN */
    size_t        hlen,           /* IN */
    dm_snap_t     *isnapp         /* OUT */
);
```

Description

Use the **dm_handle_to_snap** function to extract a snapshot ID from a handle. **dm_handle_to_snap()** is a GPFS-specific DMAPI function. It is not part of the open standard.

Parameters

void *****_hanp_ **(IN)**
A pointer to an opaque DM handle previously returned by DMAPI.

size_t _hlen_ **(IN)**
The length of the handle in bytes.

dm_snap_t *****_isnapp_ **(OUT)**
A pointer to the snapshot ID.

Return values

Zero is returned on success. On error, -1 is returned, and the global *errno* is set to one of the following values:

[EBADF]
The file handle does not refer to an existing or accessible object.

[EFAULT]
The system detected an invalid address in attempting to use an argument.

[EINVAL]
The argument *token* is not a valid token.

[ENOMEM]
DMAPI could not obtain the required resources to complete the call.

[ENOSYS]
Function is not supported by the DM implementation.

[EPERM]
The caller does not hold the appropriate privilege.

See also

"dm_make_xhandle" on page 26

dm_make_xhandle

Converts a file system ID, inode number, inode generation count, and snapshot ID into a handle.

Synopsis

```
int dm_make_xhandle(
    dm_fsid_t               *fsidp,         /* IN */
    dm_ino_t                *inop,          /* IN */
    dm_igen_t               *igenp,         /* IN */
    dm_snap_t               *isnapp,        /* IN */
    void                    **hanpp,        /* OUT */
    size_t                  *hlenp          /* OUT */
);
```

Description

Use the **dm_make_xhandle()** function to convert a file system ID, inode number, inode generation count, and snapshot ID into a handle. **dm_make_xhandle()** is a GPFS-specific DMAPI function. It is not part of the open standard.

Parameters

dm_fsid_t ******fsidp* **(IN)**
The file system ID.

dm_ino_t ******inop* **(IN)**
The inode number.

dm_igen_t ******igenp* **(IN)**
The inode generation count.

dm_snap_t ******isnapp* **(IN)**
The snapshot ID.

void *******hanpp* **(OUT)**
A DMAPI initialized pointer that identifies a region of memory containing an opaque DM handle. The caller is responsible for freeing the allocated memory.

size_t ******hlenp* **(OUT)**
The length of the handle in bytes.

Return values

Zero is returned on success. On error, -1 is returned, and the global *errno* is set to one of the following values:

[EBADF]
The file handle does not refer to an existing or accessible object.

[EFAULT]
The system detected an invalid address in attempting to use an argument.

[EINVAL]
The argument *token* is not a valid token.

[ENOMEM]
DMAPI could not obtain the required resources to complete the call.

[ENOSYS]
Function is not supported by the DM implementation.

[EPERM]
The caller does not hold the appropriate privilege.

See also

"dm_handle_to_snap" on page 25

dm_remove_dmattr_nosync

Asynchronously removes the specified attribute.

Synopsis

```
int dm_remove_dmattr_nosync(
    dm_sessid_t      sid,
    void            *hanp,
    size_t           hlen,
    dm_token_t       token,
    int              setdtime,
    dm_attrname_t   *attrnamep
);
```

Description

Use the **dm_remove_dmattr_nosync** function to asynchronously remove the attribute specified by *attrname*.

dm_remove_dmattr_nosync is a GPFS-specific DMAPI function; it is not part of the open standard. It has the same purpose, parameters, and return values as the standard DMAPI **dm_remove_dmattr** function, except that the update that it performs is not persistent until some other activity on that file (or on other files in the file system) happens to flush it to disk. To be certain that your update is made persistent, use one of the following functions:

- Standard DMAPI **dm_sync_by_handle** function, which flushes the file data and attributes
- GPFS-specific **dm_sync_dmattr_by_handle** function, which flushes only the attributes.

Parameters

dm_sessid_t *sid* **(IN)**
: The identifier for the session of interest.

void ***hanp*** **(IN)**
: The handle for the file for which the attributes should be removed.

size_t *hlen* **(IN)**
: The length of the handle in bytes.

dm_token_t ***token*** **(IN)**
: The token referencing the access right for the handle. The access right must be **DM_RIGHT_EXCL**, or the token **DM_NO_TOKEN** may be used and the interface acquires the appropriate rights.

int *setdtime* **(IN)**
: If *setdtime* is non-zero, updates the file's attribute time stamp.

dm_attrname_t ***attrnamep*** **(IN)**
: The attribute to be removed.

Return values

Zero is returned on success. On error, -1 is returned, and the global *errno* is set to one of the following values:

[EACCES]
: The access right referenced by the token for the handle is not **DM_RIGHT_EXCL**.

[EBADF]
: The file handle does not refer to an existing or accessible object.

[EFAULT]
: The system detected an invalid address in attempting to use an argument.

[EINVAL]
The argument *token* is not a valid token.

[EINVAL]
The session is not valid.

[EIO] I/O error resulted in failure of operation.

[ENOSYS]
The DMAPI implementation does not support this optional function.

[EPERM]
The caller does not hold the appropriate privilege.

[EROFS]
The operation is not allowed on a read-only file system.

See also

"dm_set_dmattr_nosync" on page 30, "dm_sync_dmattr_by_handle" on page 36

dm_set_dmattr_nosync

Asynchronously creates or replaces the value of the named attribute with the specified data.

Synopsis

```
int dm_set_dmattr_nosync(
    dm_sessid_t     sid,
    void           *hanp,
    size_t          hlen,
    dm_token_t      token,
    dm_attrname_t  *attrnamep,
    int             setdtime,
    size_t          buflen,
    void           *bufp
);
```

Description

Use the **dm_set_dmattr_nosync** function to asynchronously create or replace the value of the named attribute with the specified data.

dm_set_dmattr_nosync is a GPFS-specific DMAPI function; it is not part of the open standard. It has the same purpose, parameters, and return values as the standard DMAPI **dm_set_dmattr** function, except that the update that it performs is not persistent until some other activity on that file (or on other files in the file system) happens to flush it to disk. To be certain that your update is made persistent, use one of the following functions:

- Standard DMAPI **dm_sync_by_handle** function, which flushes the file data and attributes
- GPFS-specific **dm_sync_dmattr_by_handle** function, which flushes only the attributes.

Parameters

dm_sessid_t *sid* **(IN)**
: The identifier for the session of interest.

void ***hanp*** **(IN)**
: The handle for the file for which the attributes should be created or replaced.

size_t *hlen* **(IN)**
: The length of the handle in bytes.

dm_token_t ***token*** **(IN)**
: The token referencing the access right for the handle. The access right must be **DM_RIGHT_EXCL**, or the token **DM_NO_TOKEN** may be used and the interface acquires the appropriate rights.

dm_attrname_t ***attrnamep*** **(IN)**
: The attribute to be created or replaced.

int *setdtime* **(IN)**
: If *setdtime* is non-zero, updates the file's attribute time stamp.

size_t *buflen* **(IN)**
: The size of the buffer in bytes.

void ***bufp*** **(IN)**
: The buffer containing the attribute data.

Return values

Zero is returned on success. On error, -1 is returned, and the global *errno* is set to one of the following values:

[E2BIG]
The attribute value exceeds one of the implementation defined storage limits.

[E2BIG]
buflen is larger than the implementation defined limit. The limit can be determined by calling the **dm_get_config()** function.

[EACCES]
The access right referenced by the token for the handle is not **DM_RIGHT_EXCL**.

[EBADF]
The file handle does not refer to an existing or accessible object.

[EFAULT]
The system detected an invalid address in attempting to use an argument.

[EIO] An attempt to write the new or updated attribute resulted in an I/O error.

[EINVAL]
The argument *token* is not a valid token.

[EINVAL]
The session is not valid.

[ENOMEM]
The DMAPI could not acquire the required resources to complete the call.

[ENOSPC]
An attempt to write the new or updated attribute resulted in an error due to no free space being available on the device.

[ENOSYS]
The DMAPI implementation does not support this optional function.

[EPERM]
The caller does not hold the appropriate privilege.

[EROFS]
The operation is not allowed on a read-only file system.

See also

"dm_remove_dmattr_nosync" on page 28, "dm_sync_dmattr_by_handle" on page 36

dm_set_eventlist_nosync

Asynchronously sets the list of events to be enabled for an object.

Synopsis

```
int dm_set_eventlist_nosync(
    dm_sessid_t      sid,
    void            *hanp,
    size_t           hlen,
    dm_token_t       token,
    dm_eventset_t   *eventsetp,
    u_int            maxevent
);
```

Description

Use the **dm_set_eventlist_nosync** function to asynchronously set the list of events to be enabled for an object.

dm_set_eventlist_nosync is a GPFS-specific DMAPI function; it is not part of the open standard. It has the same purpose, parameters, and return values as the standard DMAPI **dm_set_eventlist** function, except that the update that it performs is not persistent until some other activity on that file (or on other files in the file system) happens to flush it to disk. To be certain that your update is made persistent, use one of the following functions:

- Standard DMAPI **dm_sync_by_handle** function, which flushes the file data and attributes
- GPFS-specific **dm_sync_dmattr_by_handle** function, which flushes only the attributes.

Parameters

dm_sessid_t *sid* **(IN)**
: The identifier for the session of interest.

void ***hanp*** **(IN)**
: The handle for the object. The handle can be either the system handle or a file handle.

size_t *hlen* **(IN)**
: The length of the handle in bytes.

dm_token_t ***token*** **(IN)**
: The token referencing the access right for the handle. The access right must be **DM_RIGHT_EXCL**, or the token **DM_NO_TOKEN** may be used and the interface acquires the appropriate rights.

dm_eventset_t ***eventsetp*** **(IN)**
: The list of events to be enabled for the object.

u_int *maxevent* **(IN)**
: The number of events to be checked for dispositions in the event set. The events from 0 to *maxevent***-1** are examined.

Return values

Zero is returned on success. On error, -1 is returned, and the global *errno* is set to one of the following values:

[EACCES]
: The access right referenced by the token for the handle is not **DM_RIGHT_EXCL**.

[EBADF]
: The file handle does not refer to an existing or accessible object.

[EFAULT]
The system detected an invalid address in attempting to use an argument.

[EINVAL]
The argument *token* is not a valid token.

[EINVAL]
The session is not valid.

[EINVAL]
Tried to set event on a global handle.

[ENOMEM]
The DMAPI could not acquire the required resources to complete the call.

[ENXIO]
The implementation of the DMAPI does not support enabling event delivery on the specified handle.

[EPERM]
The caller does not hold the appropriate privilege.

[EROFS]
The operation is not allowed on a read-only file system.

See also

"dm_sync_dmattr_by_handle" on page 36

dm_set_region_nosync

Asynchronously replaces the set of managed regions for a file.

Synopsis

```
int dm_set_region_nosync(
    dm_sessid_t     sid,
    void           *hanp,
    size_t          hlen,
    dm_token_t      token,
    u_int           nelem,
    dm_region_t    *regbufp,
    dm_boolean_t   *exactflagp
);
```

Description

Use the **dm_set_region_nosync** function to asynchronously replace the set of managed regions for a file.

dm_set_region_nosync is a GPFS-specific DMAPI function; it is not part of the open standard. It has the same purpose, parameters, and return values as the standard DMAPI **dm_set_region** function, except that the update that it performs is not persistent until some other activity on that file (or on other files in the file system) happens to flush it to disk. To be certain that your update is made persistent, use one of the following functions:

- Standard DMAPI **dm_sync_by_handle** function, which flushes the file data and attributes
- GPFS-specific **dm_sync_dmattr_by_handle** function, which flushes only the attributes.

Parameters

dm_sessid_t *sid* **(IN)**
: The identifier for the session of interest.

void ***hanp*** **(IN)**
: The handle for the regular file to be affected.

size_t *hlen* **(IN)**
: The length of the handle in bytes.

dm_token_t ***token*** **(IN)**
: The token referencing the access right for the handle. The access right must be **DM_RIGHT_EXCL**, or the token **DM_NO_TOKEN** may be used and the interface acquires the appropriate rights.

u_int *nelem* **(IN)**
: The number of input regions in *regbufp*. If *nelem* is 0, then all existing managed regions are cleared.

dm_region_t ***regbufp*** **(IN)**
: A pointer to the structure defining the regions to be set. May be NULL if *nelem* is zero.

dm_boolean_t ***exactflagp*** **(OUT)**
: If **DM_TRUE**, the file system did not alter the requested managed region set.

  Valid values for the *rg_flags* field of the region structure are created by OR'ing together one or more of the following values:

  DM_REGION_READ
  : Enable synchronous event for read operations that overlap this managed region.

  DM_REGION_WRITE
  : Enable synchronous event for write operations that overlap this managed region.

  DM_REGION_TRUNCATE
  : Enable synchronous event for truncate operations that overlap this managed region.

DM_REGION_NOEVENT
Do not generate any events for this managed region.

Return values

Zero is returned on success. On error, -1 is returned, and the global *errno* is set to one of the following values:

[E2BIG]
The number of regions specified by nelem exceeded the implementation capacity.

[EACCES]
The access right referenced by the token for the handle is not **DM_RIGHT_EXCL**.

[EBADF]
The file handle does not refer to an existing or accessible object.

[EFAULT]
The system detected an invalid address in attempting to use an argument.

[EINVAL]
The argument *token* is not a valid token.

[EINVAL]
The file handle does not refer to a regular file.

[EINVAL]
The regions passed in are not valid because they overlap or some other problem.

[EINVAL]
The session is not valid.

[EIO] An I/O error resulted in failure of operation.

[ENOMEM]
The DMAPI could not acquire the required resources to complete the call.

[EPERM]
The caller does not hold the appropriate privilege.

[EROFS]
The operation is not allowed on a read-only file system.

See also

"dm_sync_dmattr_by_handle" on page 36

dm_sync_dmattr_by_handle

Synchronizes one or more files' in-memory attributes with those on the physical medium.

Synopsis

```
int m_sync_dmattr_by_handle(
    dm_sessid_t    sid,
    void          *hanp,
    size_t         hlen,
    dm_token_t     token
);
```

Description

Use the **dm_sync_dmattr_by_handle** function to synchronize one or more files' in-memory attributes with those on the physical medium.

dm_sync_dmattr_by_handle is a GPFS-specific DMAPI function; it is not part of the open standard. It has the same purpose, parameters, and return values as the standard DMAPI **dm_sync_by_handle** function, except that it flushes only the attributes, not the file data.

Like **dm_sync_by_handle**, **dm_sync_dmattr_by_handle** commits all previously unsynchronized updates for that node, not just the updates for one file. Therefore, if you update a list of files and call **dm_sync_dmattr_by_handle** on the last file, the attribute updates to all of the files in the list are made persistent.

Parameters

dm_sessid_t *sid* **(IN)**
: The identifier for the session of interest.

void ***hanp*** **(IN)**
: The handle for the file whose attributes are to be synchronized.

size_t *hlen* **(IN)**
: The length of the handle in bytes.

dm_token_t ***token*** **(IN)**
: The token referencing the access right for the handle. The access right must be **DM_RIGHT_EXCL**, or the token **DM_NO_TOKEN** may be used and the interface acquires the appropriate rights.

Return values

Zero is returned on success. On error, -1 is returned, and the global *errno* is set to one of the following values:

[EACCES]
: The access right referenced by the token for the handle is not **DM_RIGHT_EXCL**.

[EBADF]
: The file handle does not refer to an existing or accessible object.

[EFAULT]
: The system detected an invalid address in attempting to use an argument.

[EINVAL]
: The argument *token* is not a valid token.

[ENOMEM]
: The DMAPI could not acquire the required resources to complete the call.

[ENOSYS]
The DMAPI implementation does not support this optional function.

[EPERM]
The caller does not hold the appropriate privilege.

See also

"dm_remove_dmattr_nosync" on page 28, "dm_set_dmattr_nosync" on page 30, "dm_set_eventlist_nosync" on page 32, and "dm_set_region_nosync" on page 34

Semantic changes to DMAPI functions

There are semantic changes to functions in DMAPI for GPFS. These changes are entailed mostly by the multiple-node environment.

For a list of additional error codes that are used in DMAPI for GPFS, see "Additional error codes returned by DMAPI functions" on page 39. For C declarations of all the DMAPI for GPFS functions, refer to the **dmapi.h** file located in the **/usr/lpp/mmfs/include** directory.

- The following DMAPI functions can be invoked on any node, not just the session node, as long as the session exists on some node in the GPFS cluster.
  - **dm_getall_disp**
  - **dm_query_session**
  - **dm_send_msg**
- DMAPI functions that reference a file system, as opposed to an individual file, can be made on any node, not just the session node. Being able to call certain functions on any node has advantages. The DM application can establish event monitoring when receiving a mount event from any node. Also, a distributed DM application can change event lists and dispositions of any file system from any node.
  - **dm_set_eventlist**
  - **dm_get_eventlist**
  - **dm_set_disp**
  - **dm_get_mount_info**
  - **dm_set_return_on_destroy**
  - **dm_get_bulkattr**
  - **dm_get_bulkall**
- The following functions, that construct a handle from its components, do not check if the resulting handle references a valid file. Validity is checked when the handle is presented in function calls that actually reference the file.
  - **dm_make_handle**
  - **dm_make_fshandle**
  - **dm_make_xhandle**
- The following data movement functions may be invoked on any node within the GPFS cluster, provided they are run as root and present a session ID for an established session on the session node. For guidelines on how to perform data movement from multiple nodes, see "Parallelism in Data Management applications" on page 13.
  - **dm_read_invis**
  - **dm_write_invis**
  - **dm_probe_hole**
  - **dm_punch_hole**

- The following functions that extract components of the handle, do not check whether the specified handle references a valid file. Validity is checked when the handle is presented in function calls that actually reference the file.
  - **dm_handle_to_fsid**
  - **dm_handle_to_igen**
  - **dm_handle_to_ino**
  - **dm_handle_to_snap**
- **dm_handle_to_fshandle** converts a file handle to a file system handle without checking the validity of either handle.
- **dm_handle_is_valid** does not check if the handle references a valid file. It verifies only that the internal format of the handle is correct.
- **dm_init_attrloc** ignores all of its arguments, except the output argument *locp*. In DMAPI for GPFS, the location pointer is initialized to a constant. Validation of the session, token, and handle arguments is done by the bulk access functions.
- When **dm_query_session** is called on a node other than the session node, it returns only the first eight bytes of the session information string.
- **dm_create_session** can be used to move an existing session to another node, if the current session node has failed. The call must be made on the new session node. See Chapter 5, "Failure and recovery of IBM Spectrum Scale Data Management API for GPFS," on page 41 for details on session node failure and recovery.
- Assuming an existing session, using **dm_create_session** does not change the session id. If the argument *sessinfop* is **NULL**, the session information string is not changed.
- The argument *maxevent* in the functions **dm_set_disp** and **dm_set_eventlist** is ignored. In GPFS the set of events is implemented as a bitmap, containing a bit for each possible event.
- The value pointed to by the argument *nelemp*, on return from the functions **dm_get_eventlist** and **dm_get_config_events**, is always **DM_EVENT_MAX-1**. The argument *nelem* in these functions is ignored.
- The *dt_nevents* field in the **dm_stat_t** structure, which is returned by the **dm_get_fileattr** and **dm_get_bulkall** functions, has a value of **DM_EVENT_MAX-1** when the file has a file-system–wide event enabled by calling the **dm_set_eventlist** function. The value will always be **3** when there is no file-system–wide event enabled. A value of **3** indicates that there could be a managed region enabled for the specific file, which might have enabled a maximum of three events: READ, WRITE, and TRUNCATE.
- The functions **dm_get_config** and **dm_get_config_events** ignore the arguments *hanp* and *hlen*. This is because the configuration is not dependent on the specific file or file system.
- The function **dm_set_disp**, when called with the global handle, ignores any events in the event set being presented, except the mount event. When **dm_set_disp** is called with a file system handle, it ignores the mount event.
- The function **dm_handle_hash**, when called with an individual file handle, returns the inode number of the file. When **dm_handle_hash** is called with a file system handle, it returns the value 0.
- The function **dm_get_mountinfo** returns two additional flags in the **me_mode** field in the **dm_mount_event** structure. The flags are **DM_MOUNT_LOCAL** and **DM_MOUNT_REMOTE**. See "Mount and unmount" on page 11 for details.

GPFS-specific DMAPI events

The GPFS-specific events are not part of the DMAPI open standard. You can use these GPFS events to filter out events that are not critical to file management and to prevent system overloads from trivial information.

The DMAPI standard specifies that the system must generate ATTRIBUTE events each time the "changed time" (**ctime**) attribute for a file changes. For systems that write files in parallel, like GPFS, this generates

ATTRIBUTE events from every node writing to the file. Consequently, it is easy for ATTRIBUTE events to overwhelm a data management server. However, the only **ctime** changes that are critical to GPFS are changes to either the permissions or ACLs of a file. In most cases, GPFS can ignore other **ctime** changes.

To distinguish file permission and ACL changes from other **ctime** updates, the following DMAPI metadata attribute events allow GPFS to filter **ctime** updates. Using these events, DM servers are able to track file permission changes without overwhelming the system with irrelevant ATTRIBUTE events. However, these events are not part of the CAE Specification C429 open standard and they were implemented specifically for GPFS 3.2 systems. Systems using GPFS 3.1 (or earlier versions) cannot enable or generate these events.

Metadata Events

DM_EVENT_PREPERMCHANGE
Pre-permission change event. Event is triggered before file permission change.

DM_EVENT_POSTPERMCHANGE
Post-permission change event. Event is triggered after file permission change.

Notes:

1. All nodes on your system must be running GPFS 3.2 or later. Mixed clusters and clusters with previous versions of GPFS will experience unexpected results if you enable these events.
2. If you only want to track permission and ACL changes, turn off the **DM_EVENT_ATTRIBUTE** and turn on both the **DM_EVENT_PREPERMCHANGE** and **DM_EVENT_POSTPERMCHANGE** events.

Additional error codes returned by DMAPI functions

DMAPI for GPFS uses additional error codes, not specified in the XDSM standard, for most DMAPI functions.

For C declarations of all the DMAPI for GPFS functions, refer to the **dmapi.h** file located in the **/usr/lpp/mmfs/include** directory.

For **all DMAPI functions**, these error codes are used:

ENOSYS
The GPFS kernel extension is not loaded, or the runtime module **dmapicalls** is not installed.

ENOSYS
An attempt has been made to invoke a DMAPI function that is not implemented in GPFS.

ENOTREADY
The local GPFS daemon is not running or is initializing.

ENOMEM
DMAPI could not acquire the required resources to complete the call. **ENOMEM** is defined in the XDSM standard for some DMAPI functions, but not for all.

ESTALE
An error has occurred which does not fit any other error code specified for this function.

For **DMAPI functions that provide a file handle as an input argument**, these error codes are used:

EINVAL
The format of the file handle is not valid.

This error is returned without attempting to locate any object that is referenced by the handle. The **EINVAL** error code is to be distinguished from the **EBADF** error code, which, as specified in the XDSM standard, indicates that the object does not exist or is inaccessible. Thus, GPFS provides a refinement, distinguishing between format and access errors related to handles.

EPERM
DMAPI is disabled for the file system that is referenced by the file handle.

For **DMAPI functions that provide a token as an input argument**, these error codes are used:

ESRCH
The event referenced by the token is not in outstanding state.

This is to be distinguished from the **EINVAL** error code, which is returned when the token itself is not valid. **ESRCH** is defined in the XDSM standard for some DMAPI functions, but not for all relevant functions. In GPFS, the **ESRCH** error code occurs mostly after recovery from session failure. See "Event recovery" on page 43 for details.

For these **specific DMAPI functions**, the error code listed is used:

Table 4. Specific DMAPI functions and associated error codes.

Name of function	Error codes and descriptions
dm_downgrade_right() **dm_upgrade_right()**	**EINVAL** - The session or token is not valid.
dm_get_region()	**EPERM** - The caller does not hold the appropriate privilege.
dm_init_service()	**EFAULT** - The system detected an invalid address in attempting to use an argument.
dm_move_event() **dm_respond_event()**	**EINVAL** - The token is not valid.
dm_punch_hole()	**EBUSY** - The file is currently memory mapped.
dm_probe_hole() **dm_punch_hole()**	**EINVAL** - The argument *len* is too large, and will overflow if cast into **offset_t**. **EINVAL** - The argument *off* is negative.
dm_write_invis()	**EINVAL** - The argument *flags* is not valid.
dm_read_invis() **dm_write_invis()**	**EINVAL** - The argument *len* is too large, and will overflow if placed into the **uio_resid** field in the structure **uio**. **EINVAL** - The argument *off* is negative.
dm_sync_by_handle()	**EROFS** - The operation is not allowed on a read-only file system.
dm_find_eventmsg() **dm_get_bulkall()** **dm_get_bulkattr()** **dm_get_dirattrs()** **dm_get_events()** **dm_get_mountinfo()** **dm_getall_disp()** **dm_getall_dmattr()** **dm_handle_to_path()**	**EINVAL** - The argument *buflen* is too large; it must be smaller than **INT_MAX**.
dm_get_alloc_info() **dm_getall_sessions()** **dm_getall_tokens()**	**EINVAL** - The argument *nelem* is too large; DMAPI cannot acquire sufficient resources.

Chapter 5. Failure and recovery of IBM Spectrum Scale Data Management API for GPFS

Failure and recovery of DMAPI applications in the multiple-node GPFS environment is different than in a single-node environment.

The failure model in XDSM is intended for a single-node environment. In this model, there are two types of failures:

DM application failure
: The DM application has failed, but the file system works normally. Recovery entails restarting the DM application, which then continues handling events. Unless the DM application recovers, events may remain pending indefinitely.

Total system failure
: The file system has failed. All non-persistent DMAPI resources are lost. The DM application itself may or may not have failed. Sessions are not persistent, so recovery of events is not necessary. The file system cleans its state when it is restarted. There is no involvement of the DM application in such cleanup.

The simplistic XDSM failure model is inadequate for GPFS. In a multiple-node environment, GPFS can fail on one node, but survive on other nodes. This type of failure is called *single-node failure* (*or partial system failure*). GPFS is built to survive and recover from single-node failures, without meaningfully affecting file access on surviving nodes.

Designers of Data Management applications for GPFS must comply with the enhanced DMAPI failure model, in order to support recoverability of GPFS. These areas are addressed:

Single-node failure

In DMAPI for GPFS, single-node failure means that DMAPI resources are lost on the failing node, but not on any other node.

The most common single-node failure is when the local GPFS daemon fails. This renders any GPFS file system at that node inaccessible. Another possible single-node failure is file system forced unmount. When just an individual file system is forced unmounted on some node, its resources are lost, but the sessions on that node, if any, survive.

Single-node failure has a different effect when it occurs on a session node or on a source node:

session node failure
: When the GPFS daemon fails, all session queues are lost, as well as all nonpersistent local file system resources, particularly DM access rights. The DM application may or may not have failed. The missing resources may in turn cause DMAPI function calls to fail with errors such as **ENOTREADY** or **ESRCH**.

Events generated at other source nodes remain pending despite any failure at the session node. Moreover, client threads remain blocked on such events.

source node failure
Events generated by that node are obsolete. If such events have already been enqueued at the session node, the DM application will process them, even though this may be redundant since no client is waiting for the response.

According to the XDSM standard, sessions are not persistent. This is inadequate for GPFS. Sessions must be persistent to the extent of enabling recovery from single-node failures. This is in compliance with a basic GPFS premise that single-node failures do not affect file access on surviving nodes. Consequently, after session node failure, the session queue and the events on it must be reconstructed, possibly on another node.

Session recovery is triggered by the actions of the DM application. The scenario depends on whether or not the DM application itself has failed.

If the DM application has failed, it must be restarted, possibly on another node, and assume the old session by id. This will trigger reconstruction of the session queue and the events on it, using backup information replicated on surviving nodes. The DM application may then continue handling events. The session id is never changed when a session is assumed.

If the DM application itself survives, it will notice that the session has failed by getting certain error codes from DMAPI function calls (**ENOTREADY**, **ESRCH**). The application could then be moved to another node and recover the session queue and events on it. Alternatively, the application could wait for the GPFS daemon to recover. There is also a possibility that the daemon will recover before the DM application even notices the failure. In these cases, session reconstruction is triggered when the DM application invokes the first DMAPI function after daemon recovery.

Session failure and recovery

A session fails when the GPFS daemon of the session node fails.

Session failure results in the loss of all DM access rights associated with events on the queue, and all the tokens become invalid. After the session has recovered, any previously outstanding synchronous events return to the initial (non-outstanding) state, and must be received again.

Session failure may also result in partial loss of the session information string. In such case, GPFS will be able to restore only the first eight characters of the session string. It is suggested to not have the DM application be dependent on more than eight characters of the session string.

In extreme situations, failure may also result in the loss of event dispositions for some file system. This happens only if the GPFS daemon fails simultaneously on all nodes where the file system was mounted. When the file system is remounted, a mount event will be generated, at which point the dispositions could be reestablished by the DM application.

During session failure, events originating from surviving nodes remain pending, and client threads remain blocked on such events. It is therefore essential that the DM application assume the old session and continue processing the pending events. To prevent indefinite blocking of clients, a mechanism has been implemented whereby pending events will be aborted and corresponding file operations failed with the **EIO** error if the failed session is not recovered within a specified time-out interval. The interval is configurable using the **dmapiSessionFailureTimeout** attribute on the **mmchconfig** command. See “GPFS configuration attributes for DMAPI” on page 18. The default is immediate timeout.

GPFS keeps the state of a failed session for 24 hours, during which the session should be assumed. When this time has elapsed, and the session has not been assumed, the session is discarded. An attempt to assume a session after it has been discarded will fail.

Event recovery

Synchronous events are recoverable after session failure.

The state of synchronous events is maintained both at the source node and at the session node. When the old session is assumed, pending synchronous events are resubmitted by surviving source nodes.

All the events originating from the session node itself are lost during session failure, including user events generated by the DM application. All file operations on the session node fail with the **ESTALE** error code.

When a session fails, all of its tokens become obsolete. After recovery, the **dm_getall_tokens** function returns an empty list of tokens, and it is therefore impossible to identify events that were outstanding when the failure occurred. All recovered events return to the initial non-received state, and must be explicitly received again. The token id of a recovered event is the same as prior to the failure (except for the mount event).

If the token of a recovered event is presented in any DMAPI function before the event is explicitly received again, the call will fail with the **ESRCH** error code. The **ESRCH** error indicates that the event exists, but is not in the outstanding state. This is to be distinguished from the **EINVAL** error code, which indicates that the token id itself is not valid (there is no event).

The semantics of the **ESRCH** error code in GPFS are different from the XDSM standard. This is entailed by the enhanced failure model. The DM application may not notice that the GPFS daemon has failed and recovered, and may attempt to use a token it has received prior to the failure. For example, it may try to respond to the event. The **ESRCH** error code tells the DM application that it must receive the event again, before it can continue using the token. Any access rights associated with the token prior to the failure are lost. See "Loss of access rights."

When a mount event is resubmitted to a session during session recovery, it will have a different token id than before the failure. This is an exception to the normal behavior, since all other recovered events have the same token id as before. The DM application thus cannot distinguish between recovered and new mount events. This should not be a problem, since the DM application must in any case be able to handle multiple mount events for the same file system.

Unmount events will not be resubmitted after session recovery. All such events are lost. This should not be a problem, since the event cannot affect the unmount operation, which has already been completed by the time the event was generated. In other words, despite being synchronous, semantically the unmount event resembles an asynchronous post event.

Loss of access rights

When the GPFS daemon fails on the session node, all file systems on the node are forced unmounted. As a result, all DM access rights associated with any local session are lost.

After daemon recovery, when the old sessions are assumed and the events are resubmitted, there is no way of identifying events that were already being handled prior to the failure (outstanding events), nor is there a guarantee that objects have not been accessed or modified after the access rights were lost. The DM application must be able to recover consistently without depending on persistent access rights. For example, it could keep its own state of events in progress, or process events idempotently.

Similarly, when a specific file system is forced unmounted at the session node, all DM access rights associated with the file system are lost, although the events themselves prevail on the session queue. After the file system is remounted, DMAPI calls using existing tokens may fail due to insufficient access rights. Also, there is no guarantee that objects have not been accessed or modified after the access rights were lost.

DODeferred deletions

The asynchronous recovery code supports deferred deletions if there are no external mounts at the time of recovery.

Once a node successfully generates a mount event for an external mount, the **sgmgr** node will start deferred deletions if it is needed. Any internal mounts would bypass deferred deletions if the file system is DMAPI enabled.

DM application failure

If only the DM application fails, the session itself remains active, events remain pending, and client threads remain blocked waiting for a response. New events will continue to arrive at the session queue.

Note: GPFS is unable to detect that the DM application has failed.

The failed DM application must be recovered on the same node, and continue handling the events. Since no DMAPI resources are lost in this case, there is little purpose in moving the DM application to another node. Assuming an existing session on another node is not permitted in GPFS, except after session node failure.

If the DM application fails simultaneously with the session node, the **gpfsready** shell script can be used to restart the DM application on the failed node. See “Initializing the Data Management application” on page 20. In the case of simultaneous failures, the DM application can also be moved to another node and assume the failed session there. See “Single-node failure” on page 41.

Accessibility features for IBM Spectrum Scale

Accessibility features help users who have a disability, such as restricted mobility or limited vision, to use information technology products successfully.

Accessibility features

The following list includes the major accessibility features in IBM Spectrum Scale:

- Keyboard-only operation
- Interfaces that are commonly used by screen readers
- Keys that are discernible by touch but do not activate just by touching them
- Industry-standard devices for ports and connectors
- The attachment of alternative input and output devices

IBM Knowledge Center, and its related publications, are accessibility-enabled. The accessibility features are described in IBM Knowledge Center (www.ibm.com/support/knowledgecenter).

Keyboard navigation

This product uses standard Microsoft Windows navigation keys.

IBM and accessibility

See the IBM Human Ability and Accessibility Center (www.ibm.com/able) for more information about the commitment that IBM has to accessibility.

Notices

This information was developed for products and services that are offered in the USA.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Dept. H6MA/Building 707
Mail Station P300
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

Portions of this code are derived from IBM Corp. Sample Programs.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and trademark information at www.ibm.com/legal/copytrade.shtml.

Intel is a trademark of Intel Corporation or its subsidiaries in the United States and other countries.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of the Open Group in the United States and other countries.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

IBM Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, See IBM’s Privacy Policy at http://www.ibm.com/privacy and IBM’s Online Privacy Statement at http://www.ibm.com/privacy/details the section entitled “Cookies, Web Beacons and Other Technologies” and the “IBM Software Products and Software-as-a-Service Privacy Statement” at http://www.ibm.com/software/info/product-privacy.

Glossary

This glossary provides terms and definitions for IBM Spectrum Scale.

The following cross-references are used in this glossary:

- *See* refers you from a nonpreferred term to the preferred term or from an abbreviation to the spelled-out form.
- *See also* refers you to a related or contrasting term.

For other terms and definitions, see the IBM Terminology website (www.ibm.com/software/globalization/terminology) (opens in new window).

B

block utilization
The measurement of the percentage of used subblocks per allocated blocks.

C

cluster
A loosely-coupled collection of independent systems (nodes) organized into a network for the purpose of sharing resources and communicating with each other. See also *GPFS cluster*.

cluster configuration data
The configuration data that is stored on the cluster configuration servers.

cluster manager
The node that monitors node status using disk leases, detects failures, drives recovery, and selects file system managers. The cluster manager must be a quorum node. The selection of the cluster manager node favors the quorum-manager node with the lowest node number among the nodes that are operating at that particular time.

Note: The cluster manager role is not moved to another node when a node with a lower node number becomes active.

control data structures
Data structures needed to manage file data and metadata cached in memory. Control data structures include hash tables and link pointers for finding cached data; lock states and tokens to implement distributed locking; and various flags and sequence numbers to keep track of updates to the cached data.

D

Data Management Application Program Interface (DMAPI)
The interface defined by the Open Group's XDSM standard as described in the publication *System Management: Data Storage Management (XDSM) API Common Application Environment (CAE) Specification C429*, The Open Group ISBN 1-85912-190-X.

deadman switch timer
A kernel timer that works on a node that has lost its disk lease and has outstanding I/O requests. This timer ensures that the node cannot complete the outstanding I/O requests (which would risk causing file system corruption), by causing a panic in the kernel.

dependent fileset
A fileset that shares the inode space of an existing independent fileset.

disk descriptor
A definition of the type of data that the disk contains and the failure group to which this disk belongs. See also *failure group*.

disk leasing
A method for controlling access to storage devices from multiple host systems. Any host that wants to access a storage device configured to use disk leasing registers for a lease; in the event of a perceived failure, a host system can deny access, preventing I/O operations with the storage device until the preempted system has reregistered.

disposition
The session to which a data management event is delivered. An individual disposition is set for each type of event from each file system.

domain
A logical grouping of resources in a network for the purpose of common management and administration.

E

ECKD™
See *extended count key data (ECKD)*.

ECKD device
See *extended count key data device (ECKD device)*.

encryption key
A mathematical value that allows components to verify that they are in communication with the expected server. Encryption keys are based on a public or private key pair that is created during the installation process. See also *file encryption key*, *master encryption key*.

extended count key data (ECKD)
An extension of the count-key-data (CKD) architecture. It includes additional commands that can be used to improve performance.

extended count key data device (ECKD device)
A disk storage device that has a data transfer rate faster than some processors can utilize and that is connected to the processor through use of a speed matching buffer. A specialized channel program is needed to communicate with such a device. See also *fixed-block architecture disk device*.

F

failback
Cluster recovery from failover following repair. See also *failover*.

failover
(1) The assumption of file system duties by another node when a node fails. (2) The process of transferring all control of the ESS to a single cluster in the ESS when the other clusters in the ESS fails. See also *cluster*. (3) The routing of all transactions to a second controller when the first controller fails. See also *cluster*.

failure group
A collection of disks that share common access paths or adapter connection, and could all become unavailable through a single hardware failure.

FEK See *file encryption key*.

fileset A hierarchical grouping of files managed as a unit for balancing workload across a cluster. See also *dependent fileset*, *independent fileset*.

fileset snapshot
A snapshot of an independent fileset plus all dependent filesets.

file clone
A writable snapshot of an individual file.

file encryption key (FEK)
A key used to encrypt sectors of an individual file. See also *encryption key*.

file-management policy
A set of rules defined in a policy file that GPFS uses to manage file migration and file deletion. See also *policy*.

file-placement policy
A set of rules defined in a policy file that GPFS uses to manage the initial placement of a newly created file. See also *policy*.

file system descriptor
A data structure containing key information about a file system. This information includes the disks assigned to the file system (*stripe group*), the current state of the file system, and pointers to key files such as quota files and log files.

file system descriptor quorum
The number of disks needed in order to write the file system descriptor correctly.

file system manager
The provider of services for all the nodes using a single file system. A file system manager processes changes to the state or description of the file system, controls the regions of disks that are allocated to each node, and controls token management and quota management.

fixed-block architecture disk device (FBA disk device)
A disk device that stores data in blocks of fixed size. These blocks are addressed by block number relative to the beginning of the file. See also *extended count key data device*.

fragment
The space allocated for an amount of data

too small to require a full block. A fragment consists of one or more subblocks.

G

global snapshot
A snapshot of an entire GPFS file system.

GPFS cluster
A cluster of nodes defined as being available for use by GPFS file systems.

GPFS portability layer
The interface module that each installation must build for its specific hardware platform and Linux distribution.

GPFS recovery log
A file that contains a record of metadata activity, and exists for each node of a cluster. In the event of a node failure, the recovery log for the failed node is replayed, restoring the file system to a consistent state and allowing other nodes to continue working.

I

ill-placed file
A file assigned to one storage pool, but having some or all of its data in a different storage pool.

ill-replicated file
A file with contents that are not correctly replicated according to the desired setting for that file. This situation occurs in the interval between a change in the file's replication settings or suspending one of its disks, and the restripe of the file.

independent fileset
A fileset that has its own inode space.

indirect block
A block containing pointers to other blocks.

inode The internal structure that describes the individual files in the file system. There is one inode for each file.

inode space
A collection of inode number ranges reserved for an independent fileset, which enables more efficient per-fileset functions.

ISKLM
IBM Security Key Lifecycle Manager. For GPFS encryption, the ISKLM is used as an RKM server to store MEKs.

J

journaled file system (JFS)
A technology designed for high-throughput server environments, which are important for running intranet and other high-performance e-business file servers.

junction
A special directory entry that connects a name in a directory of one fileset to the root directory of another fileset.

K

kernel The part of an operating system that contains programs for such tasks as input/output, management and control of hardware, and the scheduling of user tasks.

M

master encryption key (MEK)
A key used to encrypt other keys. See also *encryption key*.

MEK See *master encryption key*.

metadata
Data structures that contain information that is needed to access file data. Metadata includes inodes, indirect blocks, and directories. Metadata is not accessible to user applications.

metanode
The one node per open file that is responsible for maintaining file metadata integrity. In most cases, the node that has had the file open for the longest period of continuous time is the metanode.

mirroring
The process of writing the same data to multiple disks at the same time. The mirroring of data protects it against data loss within the database or within the recovery log.

multi-tailed
A disk connected to multiple nodes.

N

namespace
Space reserved by a file system to contain the names of its objects.

Network File System (NFS)
A protocol, developed by Sun Microsystems, Incorporated, that allows any host in a network to gain access to another host or netgroup and their file directories.

Network Shared Disk (NSD)
A component for cluster-wide disk naming and access.

NSD volume ID
A unique 16 digit hex number that is used to identify and access all NSDs.

node An individual operating-system image within a cluster. Depending on the way in which the computer system is partitioned, it may contain one or more nodes.

node descriptor
A definition that indicates how GPFS uses a node. Possible functions include: manager node, client node, quorum node, and nonquorum node.

node number
A number that is generated and maintained by GPFS as the cluster is created, and as nodes are added to or deleted from the cluster.

node quorum
The minimum number of nodes that must be running in order for the daemon to start.

node quorum with tiebreaker disks
A form of quorum that allows GPFS to run with as little as one quorum node available, as long as there is access to a majority of the quorum disks.

non-quorum node
A node in a cluster that is not counted for the purposes of quorum determination.

P

policy A list of file-placement, service-class, and encryption rules that define characteristics and placement of files. Several policies can be defined within the configuration, but only one policy set is active at one time.

policy rule
A programming statement within a policy that defines a specific action to be performed.

pool A group of resources with similar characteristics and attributes.

portability
The ability of a programming language to compile successfully on different operating systems without requiring changes to the source code.

primary GPFS cluster configuration server
In a GPFS cluster, the node chosen to maintain the GPFS cluster configuration data.

private IP address
A IP address used to communicate on a private network.

public IP address
A IP address used to communicate on a public network.

Q

quorum node
A node in the cluster that is counted to determine whether a quorum exists.

quota The amount of disk space and number of inodes assigned as upper limits for a specified user, group of users, or fileset.

quota management
The allocation of disk blocks to the other nodes writing to the file system, and comparison of the allocated space to quota limits at regular intervals.

R

Redundant Array of Independent Disks (RAID)
A collection of two or more disk physical drives that present to the host an image of one or more logical disk drives. In the event of a single physical device failure, the data can be read or regenerated from the other disk drives in the array due to data redundancy.

recovery
The process of restoring access to file system data when a failure has occurred. Recovery can involve reconstructing data or providing alternative routing through a different server.

remote key management server (RKM server)
A server that is used to store master encryption keys.

replication
The process of maintaining a defined set of data in more than one location. Replication involves copying designated changes for one location (a source) to another (a target), and synchronizing the data in both locations.

RKM server
See *remote key management server*.

rule A list of conditions and actions that are triggered when certain conditions are met. Conditions include attributes about an object (file name, type or extension, dates, owner, and groups), the requesting client, and the container name associated with the object.

S

SAN-attached
Disks that are physically attached to all nodes in the cluster using Serial Storage Architecture (SSA) connections or using Fibre Channel switches.

Scale Out Backup and Restore (SOBAR)
A specialized mechanism for data protection against disaster only for GPFS file systems that are managed by Tivoli Storage Manager (TSM) Hierarchical Storage Management (HSM).

secondary GPFS cluster configuration server
In a GPFS cluster, the node chosen to maintain the GPFS cluster configuration data in the event that the primary GPFS cluster configuration server fails or becomes unavailable.

Secure Hash Algorithm digest (SHA digest)
A character string used to identify a GPFS security key.

session failure
The loss of all resources of a data management session due to the failure of the daemon on the session node.

session node
The node on which a data management session was created.

Small Computer System Interface (SCSI)
An ANSI-standard electronic interface that allows personal computers to communicate with peripheral hardware, such as disk drives, tape drives, CD-ROM drives, printers, and scanners faster and more flexibly than previous interfaces.

snapshot
An exact copy of changed data in the active files and directories of a file system or fileset at a single point in time. See also *fileset snapshot*, *global snapshot*.

source node
The node on which a data management event is generated.

stand-alone client
The node in a one-node cluster.

storage area network (SAN)
A dedicated storage network tailored to a specific environment, combining servers, storage products, networking products, software, and services.

storage pool
A grouping of storage space consisting of volumes, logical unit numbers (LUNs), or addresses that share a common set of administrative characteristics.

stripe group
The set of disks comprising the storage assigned to a file system.

striping
A storage process in which information is split into blocks (a fixed amount of data) and the blocks are written to (or read from) a series of disks in parallel.

subblock
The smallest unit of data accessible in an I/O operation, equal to one thirty-second of a data block.

system storage pool
A storage pool containing file system control structures, reserved files, directories, symbolic links, special devices, as well as the metadata associated with regular files, including indirect blocks and extended attributes The **system storage pool** can also contain user data.

T

token management
A system for controlling file access in which each application performing a read or write operation is granted some form of access to a specific block of file data.

Token management provides data consistency and controls conflicts. Token management has two components: the token management server, and the token management function.

token management function
A component of token management that requests tokens from the token management server. The token management function is located on each cluster node.

token management server
A component of token management that controls tokens relating to the operation of the file system. The token management server is located at the file system manager node.

twin-tailed
A disk connected to two nodes.

U

user storage pool
A storage pool containing the blocks of data that make up user files.

V

VFS See *virtual file system*.

virtual file system (VFS)
A remote file system that has been mounted so that it is accessible to the local user.

virtual node (vnode)
The structure that contains information about a file system object in a virtual file system (VFS).

Index

A

C

D

E

F

G

I

M

N

P

Q

R

S

T

U

X

Product Number: 5725-Q01
5641-GPF
5725-S28

Printed in USA

GA76-0442-06