Xsan Reference

Storage > File Management

2006-05-23

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

Contents

Tables

Xsan Reference

Companion guide Xsan Programming Guide

Overview

This book describes the `fcntl`(2) requests that make up the Xsan API.

Functions by Task

The `fcntl`(2) requests are organized into the following sections:

- "Getting Information About Xsan" (page 7). This section describes the `fcntl`(2) request that gets version information about Xsan, including the version, build, and creation date of a kernel that includes Xsan, and the API version.
- "Managing Affinities" (page 8). This section describes the `fcntl`(2) requests for getting and setting a file's affinity.
- "Managing Space in Files" (page 8). This section describes the `fcntl`(2) requests for managing space in files. There is a request that allocates extent space in a file and a request that gets the list of the extent space that has been allocated for a file.
- "Working With Real-Time I/O" (page 8). This section describes `fcntl`(2) requests that manage real-time I/O. Real-time I/O can be enabled for a storage pool or for an individual file descriptor. When real-time I/O is enabled for a storage pool, you must also enable real-time I/O for a specific file descriptor. This section also includes requests that get the real-time I/O parameters of a storage pool and that disable real-time I/O for a file descriptor.
- "Managing Quotas" (page 8). This section describes `fcntl`(2) requests that allow you to get the current disk usage for a user or a group and to set the disk quota limits for a user or a group.

Getting Information About Xsan

This request described in this section gets Xsan version information, consisting of version, build, and creation date of a kernel that includes Xsan, and the API version.

`F_GETVERINFO` (page 13)
Gets Xsan version information.

Managing Affinities

The requests described in this section are for getting and setting the affinity for a file.

`F_GETAFFINITY` (page 10)
: Gets the affinity for a file.

`F_SETAFFINITY` (page 14)
: Sets the affinity for a file.

Managing Space in Files

The requests described in this section allow you to allocate extent space in and file and get the list of extent space that has already been allocated in a file.

`F_ALLOCEXTSPACE` (page 9)
: Allocates extent space in a file.

`F_GETEXTLIST` (page 11)
: Gets the list of extents for a file.

`F_LOADEXT` (page 13)
: Pre-loads a range of extents for a file.

Working With Real-Time I/O

The `fcntl`(2) requests described in this section manage real-time I/O. Real-time I/O can be enabled for a storage pool or a file descriptor by the `F_SETRTIO` request. When real-time I/O is enabled for a storage pool, you must also make a `F_ENABLERTIO` request for a specific file descriptor. This section also includes requests that get the real-time I/O parameters of a storage pool and that disable real-time I/O for a file descriptor.

`F_SETRTIO` (page 15)
: Enables the real-time I/O feature for a storage pool or a file.

`F_GETRTIO` (page 12)
: Gets the real-time I/O parameters for a storage pool.

`F_ENABLERTIO` (page 10)
: Enables real-time I/O for a file descriptor.

`F_DISABLERTIO` (page 9)
: Disables real-time I/O for a file descriptor.

Managing Quotas

These `fcntl` requests allow you to get the current disk usage for a user or a group and set the disk quota limits for a user or a group.

`F_GETQUOTA` (page 12)
: Gets the current quota usage and limits for a user or a group.

`F_SETQUOTA` (page 14)
: Sets quota limits for a user or a group.

Functions

F_ALLOCEXTSPACE

Allocates extent space in a file.

```
fcntl (
int fd,
AllocSpaceReqReply_t ReqReply);
```

Parameters

fd

File descriptor for the file for which extent space is to be allocated.

ReqReply

An `AllocSpaceReqReply_t` union consisting of an `AllocSpaceReq_t` structure `AllocSpaceReq_t Structure` (page 17) and an `AllocSpaceReply_t` structure `AllocSpaceReply_t Structure` (page 18). For details on the `AllocSpaceReqReply_t` union, see AllocSpaceReqReply_t Union (page 17).

Return Value

Zero indicates success; –1 indicates failure and `errno` is set to indicate the error. Possible values include `ENOSPC` (insufficient space in the file system to satisfy the request), `EINVAL` (invalid affinity), and `EEXISTS` (an allocation already exists that maps some or all of the specified offset and length).

Discussion

This request allocates in the file specified by `fd` a single extent of the specified size (`aq_size`) beginning at the next file system block boundary beyond the current end of file. The number of bytes allocated is rounded up to the file system block size.

You can also specify the offset (`ac_offset`) at which the allocation is to start, in which case, the request allocates space at the specified offset, rounded up to the file system block size. If any portion of <offset + size> has already been allocated, this request returns `EEXISTS`.

In both cases, the file size is updated if the allocation causes an increase in the current end of file.

You can also use this request to set the file's affinity. When a file's affinity is set, this and all future allocations will be made only from storage pools that have the matching affinity. If the file's affinity has already been set, using this request to set the file's affinity has no effect.

You can also use this request to cause extent information to be loaded into the file system client's extent mapping tables. Loading extent information improves performance by eliminating a subsequent trip to the metadata controller to retrieve extent information for the range mapped by this request.All byte sizes and offsets are rounded up to the nearest file system block size. On output, the `AllocSpaceReqReply_t` union contains the actual allocated size and offset.

Version Notes

Introduced in Xsan version 1.0.

F_DISABLERTIO

Disables real-time I/O for a file descriptor.

```
fcntl (
int fd,
F_DISABLERTIO);
```

Parameters

fd

File descriptor for the target file.

Return Value
Zero indicates success; –1 indicates failure and `errno` is set to indicate the error.

Discussion
This request disables the real-time I/O for the specified file descriptor and releases bandwidth to the file system. Any subsequent I/O using the specified file descriptor will be gated even if the storage pool remains in real-time mode. Real-time I/O is also disabled and bandwidth released to the file system when `fd` is closed.

Version Notes
Introduced in Xsan version 1.0.

F_ENABLERTIO

Enables real-time I/O for a file descriptor.

```
fcntl (
int fd,
F_ENABLERTIO);
```

Parameters

fd

File descriptor for the target file.

Return Value
Zero indicates success; –1 indicates failure and `errno` is set to indicate the error.

Discussion
This request enables real-time I/O for the specified file descriptor, which gives the file descriptor full, ungated access to the SAN. File descriptors for which real-time I/O is enabled remain in real-time mode until real-time mode is explicitly disabled (by making an `F_DISABLERTIO` (page 9) request or the file descriptor is closed.

It is important to note that gating occurs on a file descriptor basis, not a file basis. It is therefore possible for multiple threads with multiple file descriptors to share a file, and for some of them to receive ungated (real-time) access, while the others have gated access.

You do not need to make this request if the file descriptor refers to a regular file and the file descriptor was specified in an `F_SETRTIO` (page 15) request.

Version Notes
Introduced in Xsan version 1.0.

F_GETAFFINITY

Gets the affinity for a file.

```
fcntl(
int fd,
F_GETAFFINITY,
GetAffinityReply_t reply);
```

Parameters

fd

File descriptor for the target file.

reply

A `GetAffinityReply_t` structure. For details, see GetAffinityReply_t Structure (page 19).

Return Value

Zero indicates success; –1 indicates failure and `errno` is set to indicate the error.

Discussion

This request gets the current affinity for the file specified by `fd`. When a file's affinity is set, all future allocations are made only from storage pools that have an affinity that matches the file's affinity. In this way, the affinity is used to direct space allocations from specific storage pools.

Version Notes

Introduced in Xsan version 1.0.

F_GETEXTLIST

Gets the list of extents for a file.

```
fcntl(
int fd,
F_GETEXTLIST,
GetExtListReqReply_t reqreply);
```

Parameters

fd

File descriptor for the target file.

reqreply

A `GetExtListReqReply_t` union consisting of a `GetExtListReq_t` structure `GetExtListReq_t Structure` (page 20) and a `GetExtListReply_t` structure `GetExtListReply_t Structure` (page 20). For details on the `GetExtListReqReply_t` union, see GetExtListReqReply_t Union (page 20).

Return Value

`ENOENT` indicates that all of the file's extents have been returned; –1 indicates failure and `errno` is set to indicate the error. Possible values include `EFAULT` (buffer is invalid) and `EINVAL` (invalid starting offset, which probably indicates that the extent list has changed).

Discussion

This iterative request returns as many extents as possible in a single request directly from the metadata controller but does not load the extents in the file system client's extent map.

The caller is responsible for allocating and freeing buffers used by this request. A file may have many extents, so the caller can iterate over the list, specifying a different starting offset in the `gq_startfrbase` field of the `GetExtListReq_t` `GetExtListReq_t Structure` (page 20) structure. When no more extents are available, this request returns `ENOENT`.

The caller must allocate the buffer for the reply data and initialize the `gq_buf` field of the `GetExtListReply_t` `GetExtListReply_t Structure` (page 20) structure to point to it. The buffer should be aligned on a minimum of an 8 byte boundary.

Version Notes
Introduced in Xsan version 1.0.

F_GETQUOTA

Gets the current quota usage and limits for a user or a group.

```
fcntl(
int fd,
F_GETQUOTA,
GetQuotaReqReply_t reqreply);
```

Parameters

fd

File descriptor for any file; usually a file descriptor for the root directory.

reqreply

A `GetQuotaReqReply_t` union consisting of a `GetQuotaReq_t` structure`GetQuotaReq_t Structure` (page 21) and a `GetQuotaReply_t` structure `GetQuotaReply_t Structure` (page 22). For details on the `GetQuotaReqReply_t` union, see GetQuotaReqReply_t Union (page 21).

Return Value
Zero indicates success; –1 indicates failure and `errno` is set to indicate the error. Possible values include `ENOTSUP` (the quota system is not enabled on the metadata controller, the file system client, or both), `EINVAL` (the `gq_type` field of the `GetQuotaReq_t` structure is not `QUOTA_TYPE_USER` or `QUOTA_TYPE_GROUP`), or `ENOENT` (the `gq_quotaname` field of the `GetQuotaReq_t` structure is not a valid user or group name).

Discussion
This request gets the current quota usage and limits for the specified user or group. The `fd` parameter may refer to any open file or directory that resides on the file system; it does not have to represent a file owned by the user or group whose quota values are being queried.

Version Notes
Introduced in Xsan version 1.0.

F_GETRTIO

Gets the real-time I/O parameters for a storage pool.

```
fcntl (
int fd,
RtQueryReqReply_t reqreply);
```

Parameters

fd

File descriptor for the target file, which can be any file in the file system.

reqreply

An `RtQueryReqReply_t` union consisting of an `RtReq_t` structure `RtReq_t Structure` (page 26) and an `RtQueryReply_t` structure `RtQueryReply_t Structure` (page 24). For details on the `RtQueryReqReply_t` union, see RtQueryReqReply_t Union (page 24).

Return Value
Zero indicates success; –1 indicates failure and `errno` is set to indicate the error.

Discussion
This request returns the real-time parameters for a storage pool. The parameters include the stripe group number, the real-time I/O limit in I/O operations per second, the amount of real-time I/O currently committed to file system clients, the number of non-real-time I/O operations per second a file system client is most likely to obtain when requesting non-real-time I/O, and the number of file system clients that have outstanding non-real-time I/O tokens.

Version Notes
Introduced in Xsan version 1.0.

F_GETVERINFO

Gets Xsan version information.

```
fcntl (
int fd,
F_GETVERINFO,
VerInfoReply_t reply);
```

Parameters

fd

File descriptor for any file.

reply

A `VerInfoReply_t` structure. On output, the version, build, and creation date of a kernel that includes Xsan, and the API version. For details, see VerInfoReply_t Structure (page 29).

Return Value
Zero indicates success; –1 indicates failure and `errno` is set to indicate the error.

Discussion
This request returns the version, build, and creation date of a kernel that includes Xsan, as well as the version of this API, in a `VerInfoReply_t` structure.

Call `statfs`(2) to determine whether a volume is an Xsan volume. For Xsan folumes, `statfs`(2) returns `acfs` in the `f_fstypename` field.

Version Notes
Introduced in Xsan version 1.0.

F_LOADEXT

Pre-loads a range of extents for a file.

```
fcntl(
int fd,
F_LOADEXT,
LoadExtReq_t req);
```

Parameters

fd

File descriptor for the target file.

req

A `LoadExtReq_t` `LoadExtReq_t Structure` (page 22) structure.

Return Value
Zero indicates success; –1 indicates failure and `errno` is set to indicate the error.

Discussion
This request pre-loads the specified extent information and is used to increase performance when the file is first accessed. Any holes in the file are preserved.

This request does not allocate any space.

Version Notes
Introduced in Xsan version 1.0.

F_SETAFFINITY

Sets the affinity for a file.

```
fcntl(
int fd,
F_SETAFFINITY,
SetAffinityReq_t req);
```

Parameters

fd

File descriptor for the target file.

req

A `SetAffinityReq_t` structure. For details, see SetAffinityReq_t Structure (page 27).

Return Value
Zero indicates success; –1 indicates failure and `errno` is set to indicate the error.

Discussion
This request sets the affinity for a file. When a file's affinity is set, all future allocations are made only from storage pools that have an affinity that matches the file's affinity.

Version Notes
Introduced in Xsan version 1.0.

F_SETQUOTA

Sets quota limits for a user or a group.

```
fcntl(
int fd,
F_SETQUOTA,
SetQuotaReq_t req);
```

Parameters

fd

File descriptor for any file; usually a file descriptor for the root directory.

req

A `SetQuotaReq_t` structure. For details, see SetQuotaReq_t Structure (page 27).

Return Value

Zero indicates success; –1 indicates failure and `errno` is set to indicate the error. Possible values include `ENOTSUP` (the quota system is not enabled on the metadata controller or the file system client, or both), `EINVAL` (the `sq_type` field of the `SetQuotaReq_t` structure is not `QUOTA_TYPE_USER` or `QUOTA_TYPE_GROUP`), `ENOENT` (the `sq_quotaname` field of the `SetQuotaReq_t` structure is not a valid user or group name), or `EIO` (the `sq_softlimit` field of the `SetQuotaReq_t` structure is greater than `sq_hardlimit` field of that structure, or an internal error occurred).

Discussion

This request sets the quota limit for the specified user or group. The `fd` parameter may refer to any open file or directory that resides on the file system; it does not have to represent a file owned by the user or group whose quota values are being set.

The quota system automatically rounds up the specified hard and soft limits to the nearest file system block size.

Version Notes

Introduced in Xsan version 1.0.

F_SETRTIO

Enables the real-time I/O feature for a storage pool or a file.

```
fcntl (
int fd;
RtReqRepy_t reqrep);
```

Parameters

fd

File descriptor for the target file, in which case, the request affects the target file only, or file descriptor for the root directory, in which case, the request affects all files in the storage pool.

reqreply

An `RtReqReply_t` union consisting of an `RtReq_t` structure `RtReq_t Structure` (page 26) and an `RtReply_t` structure `RtReply_t Structure` (page 26). For details on the `RtReqReply_t` union, see RtReqReply_t Union (page 25).

Return Value

Zero indicates success; –1 indicates failure and `errno` is set to indicate the error.

Discussion
This request makes the real-time I/O feature available for files in a storage pool (if `fd` refers to the root directory) or enables real-time I/O for a file (if `fd` refers to a regular file). Storage pools can have different access characteristics and can be used for different types of files (for example, audio and video files), so real-time I/O is set on a storage pool basis, as opposed to a file system basis.

The caller specifies the desired number of I/O operations per second or the number of megabytes per second for the storage pool or file in the `rq_rtios` field or the `rq_rtmb` field of the `RtReq_t` structure `RtReq_t Structure` (page 26). The desired number of I/O operations per second can be calculated by dividing megabytes per second by the file system block size, stripe breadth, and stripe depth. For example, given an 8 disk storage pool with a stripe breadth of 16 and a file system block size of 4KB, a requested rate of 50 MB per second would be (50MB per second * 1024KB) / (8 * 16 * 4KB) = 100 I/O operations per second.

If the desired amount of bandwidth is not be available at the time of the request, in which case the `F_SETRTIO` request will successfully but with less than the requested amount. If that amount of bandwidth is not acceptable, you should disable real-time I/O mode for the storage pool or file and make the request again. Alternatively, you can set the `RT_MUST` flag in the `flags` field of the `RtReq_t` structure `RtReq_t Structure` (page 26) when making an `F_SETRTIO` request. When the `RT_MUST` flag is set, the `F_SETRTIO` request fails when the requested amount of bandwidth is not available. You can repeat the request until the desired amount of bandwidth is available.

You can reduce the number of I/O operations per second or the number of megabytes per second by setting the `RT_CLEAR` flag, setting the `rq_rtios` field or the `rq_rtmb` field to the amount to be subtracted, and making this request again.

If the file descriptor is for the root directory of the file system (`/`), upon successful return from this request the parameters specified by this request apply to all files on the specified storage pool that are opened and put into real-time I/O mode by an `F_ENABLERTIO` (page 10) request.

To remove a storage pool from real-time I/O mode, specify zero in the `rq_rtios` field or the `rq_rtmb` field of the `RtReq_t` structure, set the `RT_CLEAR` flag, and make this request again. Any file descriptors that are in real-time mode return to non-real-time I/O mode but are not gated. Note that closing the file descriptor for the root directory does not make real-time I/O mode unavailable on the storage pool.

If the file descriptor is for a regular file, upon successful return from this request, the file descriptor is in real-time I/O mode, and no additional requests are required. Extent structure mapping is loaded by default. You can disable extent structure mapping by setting `RT_NOLOAD` in the `rq_flags` field of the `RtReq_t` structure. After this request, existing file descriptors on the storage pool are gated and any new file descriptors that are opened are gated. Making an `F_SETRTIO` request for a regular file (as opposed to an `F_ENABLERTIO` request) allows non-cooperating applications to request different amounts of real-time I/O on the same storage pool.

If the same file is shared in real-time and non-real-time mode, the caller must use the `fcntl`(2) system call to differentiate between real-time and non-real-time accesses. This is because file descriptor flags are not exported down to the file system. The call to identify the real-time file descriptor is `fcntl(fd, F_SETFL, O_SYNC)`.

Explicitly or implicitly closing the file descriptor disables real-time I/O mode and causes the bandwidth allocated for real-time I/O to be returned to the system. However, if a file has multiple file descriptors, real-time I/O is not disabled until the last file descriptor is closed. When real-time I/O mode is disabled and no other file descriptor on the storage pool is in real-time I/O mode, other file descriptors that are open are no longer gated.

If the system running the metadata controller software is rebooted or is reset, file system clients attempt to renegotiate the real-time I/O requirements for storage pools and individual files with the metadata controller. Renegotiation may introduce a period of instability. The order in which requests are processed during the recovery period may make it impossible to guarantee the same bandwidth as prior to the reboot or reset. If the same amount of real-time bandwidth cannot be obtained during recovery processing, the next access to a file descriptor that is real-time mode will fail and an event will be logged in the system log.

The file system cannot distinguish between the use of `O_SYNC` for identifying real-time file descriptors and for specifying synchronous writes. Therefore, when a file is opened `O_SYNC` or if `O_SYNC` is set on the file via `fcntl`(2), all writes are synchronous and all I/O performed on the file is non-gated.

Version Notes
Introduced in Xsan version 1.0.

Data Types

AllocSpaceReqReply_t Union

Union used to allocate extent space in a file.

```
typedef union _allocspacereqreply {
AllocSpaceReq_t req;
AllocSpaceReply_t reply
} AllocSpaceReqReply_t;
```

Fields

`req`

An `AllocSpaceReq_t` structure containing information describing how the extent space is to be allocated. For details, see AllocSpaceReq_t Structure (page 17).

`reply`

An `AllocSpaceReply_t` structure. On output, this structure contains information describing the results of the allocation. For details, see AllocSpaceReply_t Structure (page 18).

Discussion
The `AllocSpaceReqReply_t` union is used as a parameter to the `F_ALLOCEXTSPACE` (page 9) request.

AllocSpaceReq_t Structure

Structure used to request the allocation of extent space in a file.

```
typedef struct _AllocSpaceReq {
uint64_t aq_size;
uint64_t aq_offset;
uint64_t aq_affinitykey;
uint32_t aq_flags;
    #define ALLOC_OFFSET 0x01
    #define ALLOC_LOAD_EXT 0x02
    #define ALLOC_STRIPE_ALIGN 0x04
    #define ALLOC_AFFINITY 0x08
    #define ALLOC_KEEPSIZE 0x10
    #define ALLOC_PERFECTFIT 0x20
uint32_t aq_pad1;
} AllocSpaceReq_t;
```

Fields

`aq_size`

Number of bytes to allocate; must not be zero. If not a multiple of the file system block size, the number is rounded up to the nearest file system block size.

`aq_offset`

If `ALLOC_OFFSET` is set, then `aq_size` bytes (rounded up to the nearest file system block size) will be allocated starting at the file byte offset `aq_offset` (rounded up to the nearest file system block size). If `ALLOC_OFFSET` is not set, the value of `aq_offset` is ignored when allocating space and the space is allocated at the end of the file.

`aq_affinitykey`

Affinity identifier specifying the affinity for the allocation; valid only if `ALLOC_AFFINITY` is set in `aq_flags`. The affinity identifier forces the space to be allocated exclusively from storage pools with a matching affinity.

`aq_flags`

Control flags. For details, see `Space Allocation Control Flags` (page 29).

Discussion

The `AllocSpaceReq_t` structure is used to request the allocation of extent space in a file. This structure is a member of the `AllocSpaceReqReply_t` union `AllocSpaceReqReply_t Union` (page 17) that is passed as a parameter to the `F_ALLOCEXTSPACE` (page 9) request.

AllocSpaceReply_t Structure

Structure used for receiving replies from requests to allocate extent space.

```
typedef struct _AllocSpaceReply {
uint64_t ar_size;
uint64_t ar_offset;
} AllocSpaceReply_t;
```

Fields

`ar_size`

The amount of extent space that was allocated in bytes and rounded up, if necessary to the nearest file system block size.

`ar_offset`

The file byte offset at which the extent space was allocated. If `ALLOC_OFFSET` was set in the `aq_flags` field of the `AllocSpaceReq_t` structure `AllocSpaceReq_t Structure` (page 17), the space was allocated starting at the specified file byte offset. If `ALLOC_OFFSET` was not set, the space was allocated at the end of the file.

Discussion
The `AllocSpaceReply_t` structure is used to receive the results of allocating extent space in a file. This structure is a member of the `AllocSpaceReqReply_t` union `AllocSpaceReqReply_t Union` (page 17) that is passed as a parameter to the `F_ALLOCEXTSPACE` (page 9) request.

CvExternalExtent_t Structure

Structure used to describe an extent of a file.

```
typedef struct _cvexternalextent {
uint64_t ex_frbase;
uint64_t ex_base;
uint64_t ex_end;
uint32_t ex_sg;
uint32_t ex_depth;
} CvExternalExtent_t;
```

Fields

`ex_frbase`
: File-relative starting byte offset. This offset can be anywhere in the extent; it does not have to correspond exactly to the starting offset of an extent. Any extent that contains `ex_frbase` is returned.

`ex_base`
: File system starting byte offset.

`ex_end`
: File system ending byte, inclusive. This byte offset is inclusive and specifies the last valid byte in the extent, so it is not a multiple of the file system block size. To get the starting offset of the next extent, add one to `ex_end`.

`ex_sg`
: Storage pool ordinal.

`ex_depth`
: Depth of the storage pool for this extent. Storage pools can grow and shrink dynamically, so the depth of the extent may not match the current depth of the storage pool.

Discussion
The `CvExternalExtent_t` structure is used to receive information the describes an extent of a file when an `F_GETEXTLIST` (page 11) request is made.

GetAffinityReply_t Structure

Structure used to receive the affinity for a file.

```
typedef struct getaffinityreply {
uint64_t ar_affinity;
} GetAffinityReply_t;
```

Fields

`ar_affinity`
: The file's current affinity.

Discussion
The `GetAffinityReply_t` structure is used as a parameter to the `F_GETAFFINITY` (page 10) request to receive the affinity for a file.

GetExtListReqReply_t Union

Union used to get the list of extents for a file.

```
typedef union _getextlistreqrep {
GetExtListReq_t req;
GetExtListReply_t reply;
} GetExtListReqReply_t;
```

Fields

`req`

A `GetExtListReq_t` structure specifying the file whose extents are to be obtained. For details, see GetExtListReq_t Structure (page 20).

`reply`

A `GetExtListReply_t` structure. On output, this structure contains the results of the request. For details, see GetExtListReply_t Structure (page 20).

Discussion

The `GetExtListReqReply_t` union is used as a parameter to the `F_GETEXTLIST` (page 11) request.

GetExtListReq_t Structure

Structure used to request the list of extents for a file.

```
typedef struct _getextlistreq {
uint64_t gq_startfrbase;
uint32_t gq_numbufs;
uint32_t gq_pad1;
void *qb_buf;
} GetExtListReq_t;
```

Fields

`gq_startfrbase`

Starting file-relative byte offset.

`gq_numbufs`

Number of `CvExternalExtent_t` structures the buffer pointed to by `qb_buf` can accommodate. For information on this structure, see CvExternalExtent_t Structure (page 19).

`gq_pad1`

Pad.

`gq_buf`

User-allocated buffer where `F_GETEXTLIST` (page 11) is to place an array of `CvExternalExtent_t` structures.

Discussion

The `GetExtListReq_t` structure is used to request the list of extents in a file. This structure is a member of the `GetExtListReqReply_t` union `GetExtListReqReply_t Union` (page 20) that is passed as a parameter to the `F_GETEXTLIST` (page 11) request.

GetExtListReply_t Structure

Structure used to receive the list of extents for a file.

```
typedef struct _getextlistreply {
uint32_t gr_numreturned;
uint32_t gr_pad;
} GetExtListReply_t;
```

Fields

`gr_numreturned`

Number of `CvExternalExtent_t` `CvExternalExtent_t Structure` (page 19) structures in the buffer pointed to by the `gq_buf` field in the `GetExtListReq_t` structure.

`gr_pad`

Pad.

Discussion

The `GetExtListReply_t` structure is used to receive the list of extent space for a file. This structure is a member of the `GetExtListReqReply_t Union` (page 20) union that is passed as a parameter to the `F_GETEXTLIST` (page 11) request.

GetQuotaReqReply_t Union

Union used to get quota information for a user or group.

```
typedef union _getquotareqreply {
GetQuotaReq_t req;
GetQuotaReply_t reply;
} GetQuotaReqReply_t
```

Fields

`req`

A `GetQuotaReq_t` structure specifying the user or group for which quota information is to be obtained. For details, see GetQuotaReq_t Structure (page 21).

`reply`

A `GetQuotaReply_t` structure. On output, this structure contains the results of the request. For details, see GetQuotaReply_t Structure (page 22).

Discussion

The `GetQuotaReqReply_t` union is used as a parameter to the `F_GETQUOTA` (page 12) request.

GetQuotaReq_t Structure

Structure used to request quota information for a user or a group.

```
#define MAX_QUOTA_NAME_LENGTH 256
typedef struct _getquotareq {
    uint32_t gq_type;
        #define QUOTA_TYPE_USER(uint32_t)'U'
        #define QUOTA_TYPE_GROUP(uint32_t)'G'
    uint32_t gq_pad;
    char gq_quotaname[MAX_QUOTA_NAME_LENGTH];
} GetQuotaReq_t;
```

Fields

`gq_type`

Type of quota (user or group) to get; must be `QUOTA_TYPE_USER` or `QUOTA_TYPE_GROUP`.

`gq_pad`
Pad.

`gq_quotaname`
The null-terminated name of the user or group whose quota is being obtained.

Discussion
The `GetQuotaReq_t` structure is used to request quota information for a user or a group. This structure is a member of the `GetQuotaReqReply_t` union `GetQuotaReqReply_t Union` (page 21) that is passed as a parameter to the `F_GETQUOTA` (page 12) request.

GetQuotaReply_t Structure

Structure used to receive quota information for a user or group.

```
typedef struct _getquotareply {
uint64_t gr_hardlimit;
uint64_t gr_softlimit;
uint64_t gr_cursize;
uint32_t gr_timelimit;
uint32_t gr_timeout;
} GetQuotaReply_t;
```

Fields
`gr_hardlimit`
The hard quota limit in bytes, rounded up to the nearest file system block.

`gr_softlimit`
The soft quota limit in bytes, rounded up to the nearest file system block.

`gr_cursize`
The current usage in bytes, rounded up to the nearest file system block.

`gr_timelimit`
When the soft limit is exceeded, `gr_timelimit` contains the amount of time (in minutes) before the soft limit will be treated as a hard limit.

`gr_timeout`
If the soft quota has been exceeded, `gr_timeout` contains the time in seconds since January 1, 1970 at which the soft limit will be treated as a hard limit.

Discussion
The `GetQuotaReply_t` structure is used to receive quota information for a user or a group. This structure is a member of the `GetQuotaReqReply_t` union `GetQuotaReqReply_t Union` (page 21) that is passed as a parameter to the `F_GETQUOTA` (page 12) request.

LoadExtReq_t Structure

Structure used to specify the range of extents that is to be pre-loaded.

```
typedef struct _LoadExtReq {
uint64_t lq_size;
uint64_t lq_offset;
uint32_t lq_pad1;
uint32_t lq_pad2;
} LoadExtReq_t;
```

Fields

`lq_size`
: The amount of extent space in bytes that is to be pre-loaded and rounded up, if necessary to the nearest file system block size.

`lq_offset`
: The starting file byte offset of the range of extents that is to be pre-loaded.

`lq_pad1`
: Pad.

`lq_pad2`
: Pad.

Discussion

The `LoadExtReq_t` structure is used to specify the starting byte offset and the size of the extent space that is to be pre-loaded. This structure is passed as a parameter to the `F_LOADEXT` (page 13) request.

Real-Time I/O Control Flags

The real-time I/O control flags are used when requesting real-time I/O for a storage pool or file descriptor.

Discussion

The real-time I/O control flags are specified in the `rq_flags` field of the `RtReq_t` structure `RtReq_t Structure` (page 26) when making an `F_SETRTIO` (page 15) request. The following flags are available:

Table 1 Real-time I/O control flags

`RT_IO`	Indicates that the `rr_rtio` field of the `RtReq_t` structure `RtReq_t Structure` (page 26) contains valid data and that the request for real-time I/O is being made in real-time I/O operations per second.
`RT_MB`	Indicates that the `rr_rtmb` field of the `RtReq_t` structure `RtReq_t Structure` (page 26) contains is valid data and that the request for real-time I/O is being made in megabytes per second.
`RT_CLEAR`	Disables real-time I/O for the specified storage pool. For each `F_SETRTIO` request, `RT_CLEAR` or `RT_SET` must be set.
`RT_SET`	Enables real-time I/O for the specified storage pool or file descriptor. For each `F_SETRTIO` request, `RT_CLEAR` or `RT_SET` must be set.
`RT_MUST`	Causes the request for real-time I/O to fail if the requested amount of bandwidth cannot be satisfied. Using this flag prevents the metadata controller from returning a lesser amount than was requested.
`RT_SEQ`	Reserved for future use.

RT_NOGATE	Puts the specified file descriptor into ungated mode if RT_SET is also set. In this mode, I/O using this file descriptor is not in real-time I/O mode, yet does not consume non-real-time I/O bandwidth and is not gated. This flag is typically set for files that are accessed infrequently, such as index files, where the caller takes full responsibility to ensure that enough bandwidth is available. When RT_NOGATE is set, the rq_rtios and rq_rtmb fields of the RtReq_t structure should not contain valid data. If RT_NOGATE and RT_CLEAR are set, the file descriptor is removed from this mode.
RT_NOLOAD	Prevents the loading of extent structure mapping.
RT_ABSOLUTE	Indicates that the request is an absolute request and not one of a series of multiple incremental requests.

RtQueryReqReply_t Union

Union used to get the real-time parameters for a storage pool.

```
typedef union rtqueryreqrep {
RtReq_t req;
RtQueryReply_t reply;
} RtQueryReqReply_t;
```

Fields

`req`

An `RtReq_t` structure specifying the storage pool whose real-time parameters are to be obtained. You must also set the `rq_flags` field of this structure to `RT_GET`. For details, see RtReq_t Structure (page 26).

`reply`

An `RtQueryReply_t` structure. On output, this structure contains the results of the request. For details, see RtQueryReply_t Structure (page 24).

Discussion

The `RtQueryReqReply_t` union is used as a parameter to the `F_GETRTIO` (page 12) request.

RtQueryReply_t Structure

Structure used to receive the results of requesting the real-time parameters for a storage pool.

```
typedef struct _rtqueryreply {
uint32_t rrq_sgid;
uint32_t rrq_state;
    #define RT_STATE_NONREALTIME 0x01
    #define RT_STATE_REALTIME 0x02
    #define RT_STATE_REQUEST 0x04
    #define RT_STATE_TIMEOUT 0x10
int32_t rrq_limit;
int32_t rrq_cur;
int32_t rrq_nrtio_hint;
uint32_t rrq_nrtio_clients;
} RtQueryReply_t;
```

Fields

`rrq_sgid`
: Storage pool ordinal.

`rrq_state`
: Flags indicating the storage pool's state with respect to real-time I/O. For details, see `Real-Time I/O Status Flags` (page 28).

`rrq_limit`
: Configured real-time I/O limit in I/O operations per second.

`rrq_cur`
: Current amount of real-time I/O committed to file system clients in I/O operations per second.

`rrq_nrtio_hint`
: Amount of non-real-time I/O in operations per second a file system client is most likely to obtain when requesting a non-real-time I/O token.

`rrq_nrtio_hint`
: Number of file system clients with outstanding non-real-time I/O tokens.

Discussion

The `RtQueryReply_t` structure is used to receive the results of requesting the real-time I/O parameters for a storage pool. This structure is a member of the `RtQueryReqReply_t` union `RtQueryReqReply_t Union` (page 24) that is passed as a parameter to the `F_GETRTIO` (page 12) request.

RtReqReply_t Union

Union used to request real-time I/O on a storage pool or file descriptor.

```
typedef union rtreqrep {
RtReq_t req;
RtReply_t reply;
} RtReqReply_t;
```

Fields

`req`
: An `RtReq_t` structure describing how real-time I/O is to be set. For details, see RtReq_t Structure (page 26).

`reply`
: An `RtReply_t` structure. On output, this structure contains the results of the request. For details, see RtReply_t Structure (page 26).

Discussion

The `RtReqReply_t` union is used as a parameter to the `F_SETRTIO` (page 15) request.

RtReq_t Structure

Structure used to request real-time I/O on a storage pool or file descriptor.

```
typedef struct _rtreq {
union {
    uint32_t ru_rtios;
    uint32_t ru_rtmb;
} rq_un;
#define rq_rtios rq_un.ru_rtios
#define rq_rtmb rq_un.ru_rtmb
uint32_t rq_flags;
    #define RT_IO 0x01
    #define RT_MB 0x02
    #define RT_CLEAR 0x04
    #define RT_SET 0x08
    #define RT_MUST 0x10
    #define RT_SEQ 0x20
    #define RT_GET 0x40
    #define RT_NOGATE 0x80
    #define RT_NOLOAD 0x100
    #define RT_ABSOLUTE 0x200
uint32_t rq_sgid;
uint32_t rq_pad;
} RtReq_t;
```

Fields

`rq_rtios`
: Requested real-time I/O operations per sec. Valid only if `RT_IO` is set in the `rq_flags` field of this structure. If `RT_IO` is set, `RT_MB` cannot be set in the `rq_flags` field.

`rq_rtmb`
: Requested megabytes per sec. Valid only if `RT_MB` is set in the `rq_flags` field of this structure; If `RT_MB` is set, `RT_IO` cannot be set in the `rq_flags` field.

`rq_flags`
: Control flags. For details, see Real-Time I/O Control Flags (page 23).

`rq_sgid`
: Storage pool ordinal that identifies this storage pool.

`rq_pad`
: Pad.

Discussion

The `RtReq_t` structure is used to request the real-time I/O parameters. This structure is a member of the `RtReqReply_t` union `RtReqReply_t Union` (page 25) that is passed as a parameter to the `F_SETRTIO` (page 15) request.

RtReply_t Structure

Structure used to receive replies from requests for real-time I/O.

```
typedef struct _rtreply {
union {
    int32_t ru_rtios;
    int32_t ru_rtmb;
} rr_un;
#define rr_rtio rr_un.ru_rtois
#define rr_rtmb rr_un.ru_rtmb
uint32_t rr_flags;
    #define RT_IO 0x01
    #define RT_MB 0x02
} RtReply_t;
```

Fields

`rr_rtio`

Allowed real-time I/O operations per second. This field is valid only if the `RT_IO` flag is set in the `rr_flags` field. This value may be less than or greater than the amount requested if the `RT_MUST` flag was not set in the `rq_flags` field when the request for real-time I/O was made. For information on these flags, see Real-Time I/O Control Flags (page 23).

`rr_rtmb`

Allowed real-time megabytes per second. Valid only if `RT_MB` is set in the `rr_flags` field. This value may be less than or greater than the amount requested if the `RT_MUST` flag was not set in the `rq_flags` field when the request for real-time I/O was made. For information on these flags, see Real-Time I/O Control Flags (page 23).

`rr_flags`

Flag indicating that the request for real-time I/O was granted in real-time operations per second (`RT_IO`) or in real-time megabytes per second (`RT_MB`).

Discussion

The `RtReply_t` structure is used to receive the results of requesting real-time I/O for a storage group or a file. This structure is a member of the `RtReqReply_t` union `RtReqReply_t Union` (page 25) that is passed as a parameter to the `F_SETRTIO` (page 15) request.

SetAffinityReq_t Structure

Structure used to set the affinity for a file.

```
typedef struct setaffinityreq {
uint64_t sq_affinity;
} SetAffinityReq_t;
```

Fields

`sq_affinity`

The affinity that is to be set.

Discussion

The `SetAffinityReq_t` structure is used to set the affinity for a file and is passed as a parameter to the `F_SETAFFINITY` (page 14) request.

SetQuotaReq_t Structure

Structure used to set quota limitations for a user or a group.

```
#define MAX_QUOTA_NAME_LENGTH 256
typedef struct setquotareq {
    uint32_t sq_quotaname[MAX_QUOTA_NAME_LENGTH];
    uint32_t sq_type;
        #define QUOTA_TYPE_USER(uint32_t)'U'
        #define QUOTA_TYPE_GROUP(uint32_t)'G'
    uint32_t sq_timelimit;
    uint64_t sq_hardlimit;
    uint64_t sq_softlimit;
} SetQuotaReq_t;
```

Fields

`sq_quotaname`
: Name of the user or group whose quota limits are being set.

`sq_type`
: Type of name specified by `sq_quotaname`; must be `QUOTA_TYPE_USER` or `QUOTA_TYPE_GROUP`.

`sq_timelimit`
: Soft quota grace period, in minutes.

`sq_hardlimit`
: Hard limit in bytes.

`sq_softlimit`
: Soft limit in bytes.

Discussion

The `SetQuotaReq_t` structure is used to set quota limitations for a user or a group. It is a passed as a parameter to the `F_SETQUOTA` (page 14) request.

Real-Time I/O Status Flags

The real-time I/O status flags indicate the state of real-time I/O with respect to a storage pool.

Discussion

The real-time I/O status flags are returned in the `rrq_state` field of the `RtQueryReply_t` structure `RtQueryReply_t Structure` (page 24) that is returned b `F_GETRTIO` (page 12). The following values are possible:

Table 2 Real-time I/O status flags

`RT_STATE_NONREALTIME`	Indicates that the storage pool is not in real-time I/O mode.
`RT_STATE_REALTIME`	Indicates that the storage pool is in real-time I/O mode.
`RT_STATE_REQUEST`	Indicates that real-time I/O mode for the storage pool has been requested but the storage pool is not yet in real-time I/O mode.
`RT_STATE_TIMEOUT`	Indicates that real-time I/O mode for the storage pool has been requested but that the request has timed out.

Space Allocation Control Flags

The space allocation control flags are used when allocating extent space in a file.

Discussion

The space allocation control flags are set in the `aq_flags` field of the `AllocSpaceReq_t` structure `AllocSpaceReq_t Structure` (page 17) that is passed to `F_ALLOCEXTSPACE` (page 9). The following space allocation control flags are available:

Table 3

`ALLOC_OFFSET`	Indicates that the `aq_offset` field of the `AllocSpaceReq_t` structure is valid.
`ALLOC_LOAD_EXT`	Causes extent structure mapping to be loaded on the file system client and added to the client file system.
`ALLOC_STRIPE_ALIGN`	Causes space to be allocated starting at a storage pool boundary.
`ALLOC_AFFINITY`	Indicates that the `aq_affinitykey` field of the `AllocSpaceReq_t` structure is valid.

VerInfoReply_t Structure

Structure used to receive Xsan version information.

```
#define CVVERINFO_MAX 64
typedef struct _verinfo_reply {
    char vr_version[CVVERINFO_MAX];
    char vr_build[CVVERINFO_MAX];
    char vr_creationdate[CVVERINFO_MAX];
    uint32_t vr_apiversion;
    uint32_t vr_pad1;
} VerInfoReply_t;
```

Fields

`vr_version`
: String containing the release and build number, such as `#!@$ Xsan Client Revision 2.5.0 Build 26`.

`vr_build`
: String containing platform information, such as `#!@$ Built for Darwin 7.3.0`.

`vr_creationdate`
: String containing the creation date, such as `#!@$ Created on Tues Apr 6 14:15:53 PST 2004`.

`vr_apiversion`
: Version of this API.

`vr_pad1`
: Pad.

Discussion

The `VerInfoReply_t` structure is provided as a parameter to the `F_GETVERINFO` (page 13) request in order to receive the results of that request.

Document Revision History

This table describes the changes to *Xsan Reference*.

Date	Notes
2006-05-23	First publication of this content as a separate document.

Document Revision History

Index