# API Specification

## Software-Enabled Flash™

Version: 1.14

**SEF-API-01-14**

#### LEGAL DISCLAIMER

THIS DOCUMENT AND THE INFORMATION CONTAINED HEREIN IS PROVIDED ON AN "AS IS" BASIS. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, THE SOFTWARE-ENABLED FLASH PROJECT, THE LINUX FOUNDATION, AND THE CONTRIBUTORS TO THIS DOCUMENT HEREBY DISCLAIM ALL REPRESENTATIONS, WARRANTIES AND/OR COVENANTS, EITHER EXPRESS OR IMPLIED, STATUTORY OR AT COMMON LAW, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, VALIDITY, AND/OR NONINFRINGEMENT.

All product names, trademarks, registered trademarks, and/or servicemarks may be claimed as the property of their respective owners.

#### DEFINITIONS AND CLARIFICATIONS

Definition of capacity: we define a megabyte (MB) as 1,000,000 bytes, a gigabyte (GB) as 1,000,000,000 bytes and a terabyte (TB) as 1,000,000,000,000 bytes. A computer operating system, however, reports storage capacity using powers of 2 for the definition of 1GB $= 2^{30} =$ 1,073,741,824 bytes and therefore shows less storage capacity. Available storage capacity (including examples of various media files) will vary based on file size, formatting, settings, software and operating system, such as Microsoft Operating System and/or pre-installed software applications, or media content. Actual formatted capacity may vary.

KiB: A kibibyte (KiB) means $2^{10}$, or 1,024 bytes, a mebibyte (MiB) means $2^{20}$, or 1,048,576 bytes, and a gibibyte (GiB) means $2^{30}$, or 1,073,741,824 bytes.

Read and write speed may vary depending on the host device, read and write conditions, and file size.

#### TRADEMARKS

NVM Express is a trademark of NVM Express, Inc.

PCI Express and PCIe are trademarks or registered trademarks of PCI-SIG.

Linux is a registered trademark of Linus Torvalds in the U.S. and other countries.

All company names, product names and service names may be trademarks of their respective companies.

# LICENSES

## SPECIFICATION LICENSE

Specifications in the repository are subject to the Community Specification License available at https://github.com/SoftwareEnabledFlash/SEF-API, and subject to the notices and license acceptance criteria located in the NOTICES.md file contained therein.

## SOURCE CODE LICENSE

If source code is included in the specification itself, that code is subject to the BSD 3-Clause license unless otherwise designated. In the case of any conflict or confusion within this specification repository between the Community Specification License and the BSD 3-Clause or other designated license, the terms of the Community Specification License shall apply.

If source code is included in this repository, or for sample or reference code included in the specification itself, that code is subject to the BSD 3-Clause license unless otherwise marked.

In the case of any conflict or confusion within this specification repository between the Community Specification License and the designated source code license, the terms of the Community Specification License shall apply.

# Contents

# 1 Revision History

| Version | Date | Description of change(s) |
|---|---|---|
| 1.10 | 2020.08.17 | Initial version of the document |
| 1.11 | 2021.01.22 | API clarifications |
| 1.12 | 2021.08.01 | API modified to support new SEF command set |
| 1.13 | 2022.01.10 | API modified. Transferred to SoftwareEnabledFlash.org |
| 1.14 | 2022.01.10 | API modified. Changed API license. |

# 2 | Introduction

This specification describes the core components of the Software-Enabled Flash™ (SEF) application programming interface (API). It is platform-agnostic and is usable by any code that can use a C interface.

The SEF API provides a simple but powerful interface for developers that abstracts low-level flash details of low-level flash memory device mechanics in such a way that allows hosts to interact with flash memory devices as though they were simple performance-optimized read/write devices. Hosts can make use of the SEF API to implement a custom Flash Translation Layer (FTL) or build SEF native applications bypassing all file systems in accordance with their application-specific requirements.

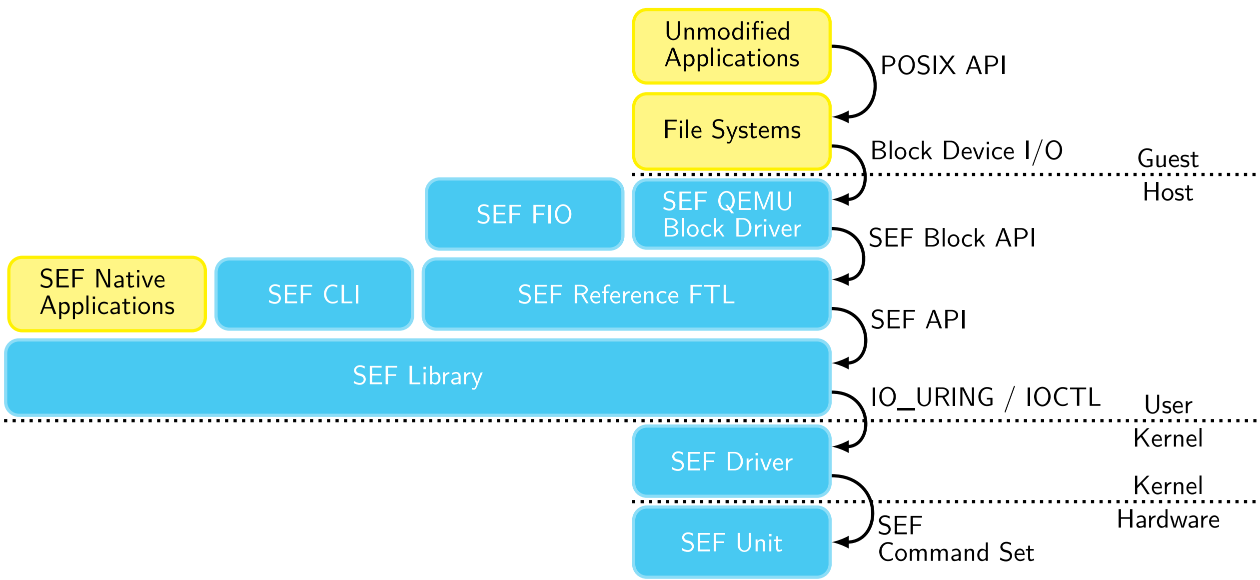
The SEF API interfaces with SEF Units. SEF Units are PCIe® based NVMe™ devices that implement the SEF specific extensions to the NVMe Base Specification. These extensions are separately defined as the SEF Command Set.

The SEF API addresses the following:

- Maintaining interface compatibility across flash memory generations
- Allowing host control over data placement to enable application-specific optimizations
- Providing mechanisms to enforce hardware isolation to support multi-tenancy and workload isolation
- Provides control over housekeeping functions to support predictable latency
- Reduces CPU cycles and host overhead via powerful API primitives
- Improves flash memory life and health via intelligent automatic resource allocation

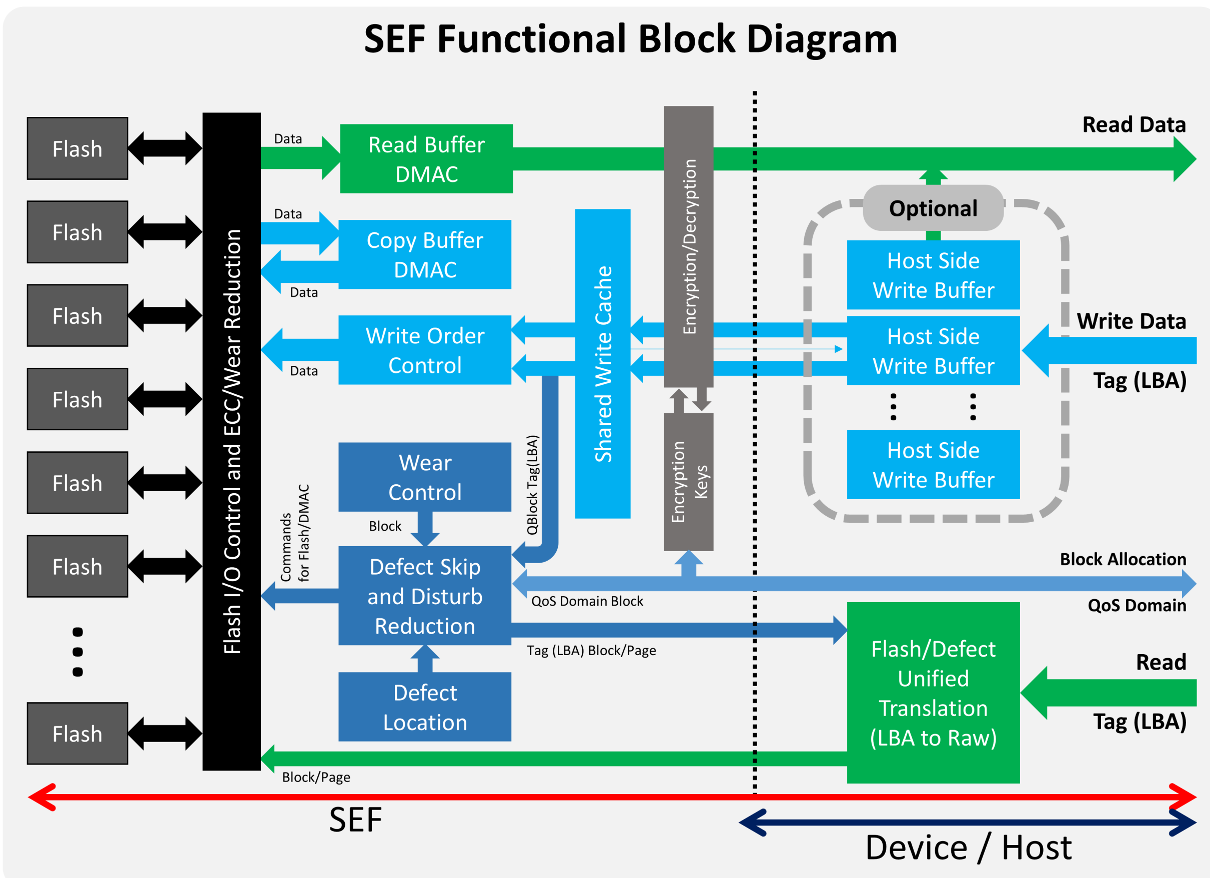
The SEF Library implements the SEF API as a linkable library module. The following figure 2.1 illustrates where the SEF Library is located in the context of a traditional SSD-like application stack. It shows multiple possible integrations of the SEF Library. In some cases the integration is direct like with a SEF enable FIO. Other cases require a host-define FTL exposed to applications as a virtual block device in a QEMU guest. Tooling can also use the SEF Library for configuring SEF units with a command line interface. In all cases, the SEF Library accepts admin and I/O requests and issues a set of commands to the SEF Unit. The SEF Unit then translates this down to an appropriate set of flash memory-level operations and returns status and data back up the stack.

Figure 2.1: SEF Library Application Stack



The SEF Unit handles functionality including super block allocation, identifying and working around defective blocks, low-level flash memory I/O, scheduling, prioritization and other device-level concerns. The host layer in turn is responsible for implementing its own data placement strategy (including devising an appropriate logical-to-physical address mapping) as well as coordinating housekeeping functions such as wear leveling, garbage collection, and responding to asynchronous event notifications. The following figure 2.2 provides a detailed view of data flow within a SEF Unit.

Figure 2.2: SEF Block Diagram




Built in conjunction with the SEF API is a SEF SDK that provides a starting point for host application development. The SEF SDK includes quick start guides for tooling to setup SEF devices for use, sample drivers, libraries, supporting documentation, and an implementation of a fully functional Reference FTL that can be extended or modified as appropriate. It is documented separately.

# 3 Definitions and Acronyms

Table 3.1: Definitions and Acronyms

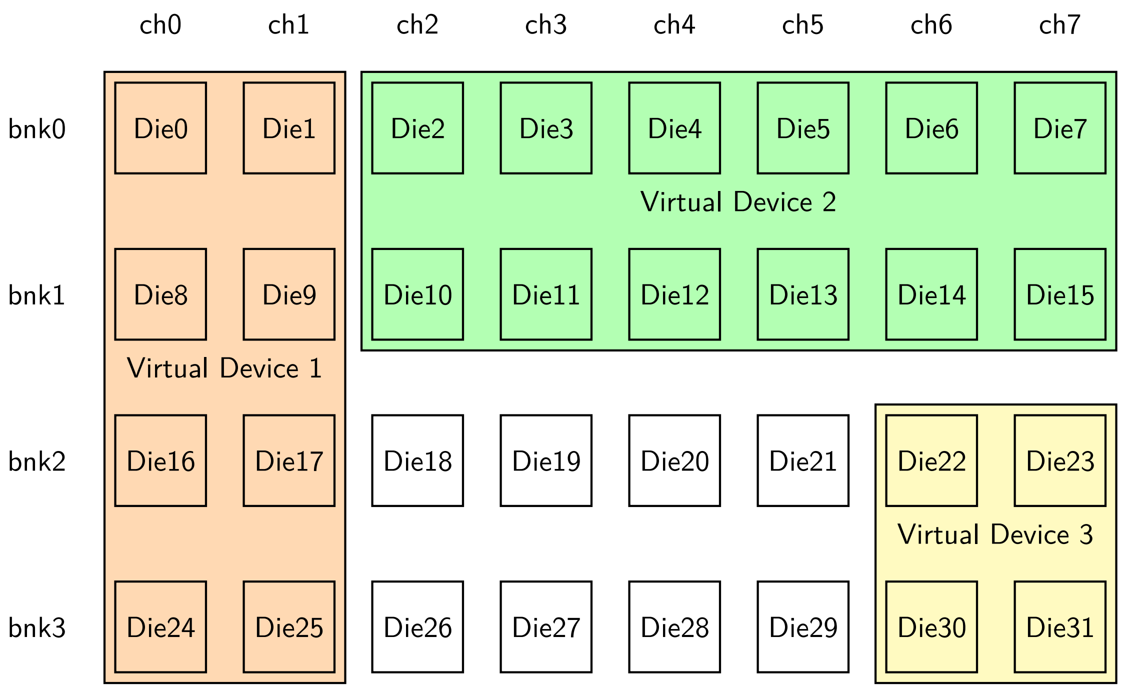
| Terms/Acronyms | Definition |
|---|---|
| Software-Enabled Flash™ (SEF) | A flash memory-based storage hardware platform that is driven by software. Pronounced as ess ē ef. |
| SEF Unit | A PCIe® flash memory storage device. Contains one or more flash memory dies and provides flash memory service functions. The SEF Unit command set consists of a subset of the NVMe™ command set with extensions. |
| Flash Translation Layer (FTL) | A mapping of Logical Block Addresses (LBA) to flash memory addresses providing a block based API on top of a flash memory API. |
| Virtual Device (VD) | A set of flash memory dies. A Virtual Device occupies one or more flash memory dies and provides one or more QoS domains and wear leveling service between QoS domains. Flash memory dies can only be assigned to one virtual device; they are never shared between virtual devices. Virtual devices provide true hardware-based isolation.<br>*Refer to Chapter 5 for more information.* |
| Pseudo Single-Level Cell (pSLC) | SEF devices may optionally support programming flash memory as if it's SLC for increased endurance. |

| | |
|---|---|
| QoS Domain (QD) | A logical construct exposed to the host and enumerated as a SEF Unit node. QoS domains are created within a single virtual device, and draw super blocks from a common pool within the virtual device. Many QoS domains may be created within a single virtual device. QoS domains provide software-based isolation, impose quotas on capacity, and are comprised of a set of super blocks within a virtual device. Super blocks are not shared between QoS domains. Read/write commands are issued to a specific QoS domain.<br>*Refer to Chapter 6 for more information.* |
| Super Block | A set of flash memory blocks spanning all of the dies in a virtual device. All flash memory blocks in a super block can be programmed and read in parallel.<br>*Refer to Chapter 8 for more information.* |
| Logical Block Address (LBA) | Represents one component of an optional user-visible addressing interface implemented by an FTL. |
| ADU | Atomic data unit. A SEF-defined internal representation of abstract storage that is the minimum read/write quantum (analogous to the block size of a traditional block device). A SEF Unit may support multiple ADU sizes and the ADU size is specified when creating a QoS domain. The minimum ADU size is 4096 bytes. |
| User Address | Eight bytes of arbitrary metadata that is stored with an ADU. For block storage applications, this is typically the LBA. However the SEF Unit makes no assumptions about the format of this data for non-block storage applications. |
| Placement ID | A placement ID is used when writing data to a QoS domain. It's used to group data of similar lifetime together. ADUs written with the same placement ID are stored in the same super blocks. |
| Root Pointer | Provides a bootstrapping mechanism to retrieve metadata from a QoS domain. |

# 4 | SEF Unit

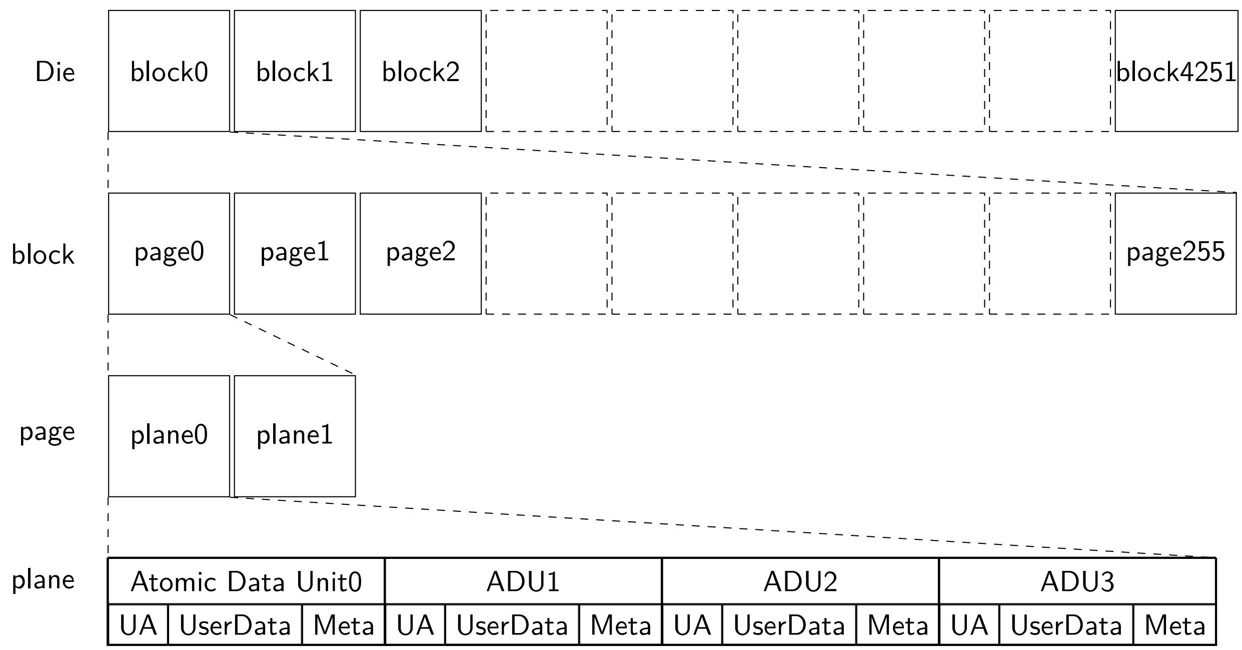
A SEF Unit is a set of dies and the associated control logic and (optional) DRAM. User-defined lists of dies form a many-to-one mapping of dies to a virtual device. A virtual device represents physical isolation with the number of virtual devices limited by the number of dies in the SEF Unit. Figure 4.1 shows an example of three virtual devices overlaid on an 8 x 4 SEF Unit with eight dies left unallocated.

Figure 4.1: SEF Unit Geometry



As shown in Figure 4.2, a die is a set of blocks. The blocks are the erase unit for a SEF Unit and consist of a set of pages. A page spans the die planes and is the programming unit. A plane is made up of atomic data units (ADUs). An ADU is the read/write unit holding both user data and metadata. Metadata consists of a user-defined tag data (UA) and a configured number of user supplied metadata (MD) bytes.

Figure 4.2: Die Geometry



# 5 | Virtual Devices

A Virtual Device encompasses one or more flash memory dies, providing the user the ability to utilize the hardware isolation of separate dies. Dies are not shared across separate virtual devices. I/O operations on one Virtual Device will not compete for die time with other virtual devices. There may be a minimal amount of latency caused by contention between virtual devices due to any internal controller bottlenecks or flash memory channel conflicts for virtual devices that share flash memory channels.

When virtual devices are created, several parameters are specified to define the characteristics of each virtual device. Virtual devices are created by using the `SEFCreateVirtualDevices()`. function. The size of each virtual device is user-configurable and dependent on the resources available. Each virtual device must be given a unique ID.

Because virtual devices represent hardware isolation, the SEF Unit will not wear level across the dies in different virtual devices. It is expected that virtual devices will be created when a SEF Unit is first set up and their geometry not subsequently altered.

## 5.1 Creation-time Parameters

**virtualDeviceID**: an identifier that will later be used to specify the created virtual device. This identifier must be unique across the entire SEF Unit. The maximum allowed ID is the number of dies in the SEF Unit.

**dieList**: Lists the dies that will be owned by the created virtual device.

**superBlockSize**: The number of dies that define a super block. It must evenly divide into the number of dies defined for the virtual device.

Once a virtual device is created it can be further configured with `SEFSetVirtualDeviceSuspendConfig()` and `SEFSetNumberOfPSLCSuperBlocks()`.

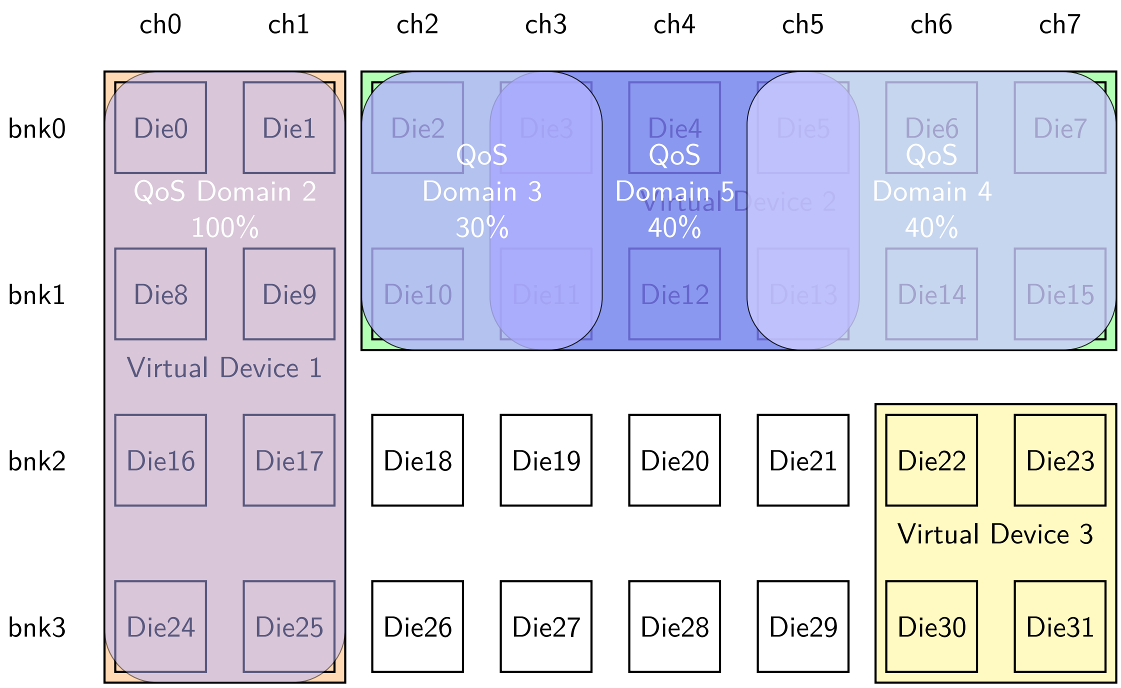
# 6 | QoS Domains

A QoS domain is the mechanism used to access data within a SEF Unit. QoS domains are created within a virtual device, and it is possible to have multiple QoS domains sharing a single virtual device. When multiple QoS domains share a virtual device, they will draw from a common pool of super blocks. However a super block is never shared between QoS domains and so data for QoS domains will never be intermingled in a super block. When QoS domains share a virtual device, there is no hardware isolation between them, so die-time conflicts are possible. The scheduling and prioritization features of SEF are used to order I/O for shared virtual devices and to resolve these die-time conflicts (e.g., software-defined isolation/quality of service).

When a QoS domain is created, several parameters are specified to define the characteristics of the QoS domain, which will be discussed below. Upon successful creation of a QoS domain, a device node will be created in the operating system namespace corresponding to the newly created QoS domain. It has a capacity and quota. The capacity is storage reserved in the virtual device for use by the QoS domain. The quota is how much total storage can be assigned to the QoS domain. Initially the quota is set to the capacity, but both can be changed later using `SEFSetQoSDomainCapacity()`. At boot time the SEF Unit driver will create device nodes for all QoS domains previously defined for the device. Device nodes for QoS domains may be used to enumerate existing QoS domains as well as to restrict access to/enforce ownership of a QoS domain. All user data access commands are issued against a QoS domain. Typically, a QoS domain will be used by a single application or Flash Translation Layer/block driver/key value driver.

An example of how the virtual devices of a SEF Unit could be divided into QoS domains is shown in the following figure 6.1. A QoS domain is a logical construct that defines a capacity taken from its virtual device's capacity. It also defines a quota that may exceed the capacity of the virtual device as shown with QoS domains three through four. A SEF Unit can have at most 65534 QoS domains defined. The actual limit depends on the specific hardware.

Figure 6.1: QoS Domain Example




Allocated super blocks are owned by only one QoS domain at a time and are never shared. Super blocks are allocated from a shared pool allowing for host-managed thin provisioning. A QoS domain can allocate super blocks until it hits its quota or the free pool is exhausted.

## 6.1 Creation-time Parameters

**vdHandle**: the handle to the virtual device the QoS domain will be created in.

**QoSDomainID**: an identifier that will later be used to specify the created QoS domain. This identifier must be unique across the entire SEF Unit. Valid IDs start at 1 and must be less than or equal to `maxQoSDomains` returned by `SEFGetInformation()`.

**flashCapacity**: the number of 4KiB ADUs reserved for the QoS domain. It supplies a capacity and a quota. The capacity is subtracted from the available ADUs from the virtual device so must be less than the currently available ADUs.

**pSLCFlashCapacity**: the number of 4KiB pSLC ADUs reserved for the QoS domain. It supplies a capacity and a quota. The capacity is subtracted from the available pSLC ADUs from the virtual device so must be less than the currently available pSLC ADUs.

**ADUIndex**: this is the index into the ADUSize[] array in `SEFInfo` returned by `SEFGetInformation()` to select the data and metadata sizes of an ADU.

**api**: this field specifies the API to be used for this QoS domain. Currently only the super block API is supported.

**defectStrategy**: Specifies how defective ADUs are handled by the QoS domain. The choices are Perfect, Packed or Fragmented. The Perfect strategy hides defective ADUs through overprovisioning and mapping. Capacity is reserved, and ADUs are remapped to provide static and consistent flash memory addresses with contiguous ADU offsets. Packed also hides defective ADUs presenting consistent flash memory addresses with contiguous ADU offsets, but the size of super blocks will shrink as the device wears. With the Fragmented strategy, the client is exposed to the device's defect management. ADU offsets are non-contiguous, and super blocks will shrink in size as the device wears. Refer to Chapter 10 for more details.

**recovery**: Specifies the error recovery strategy for this QoS domain.

**encryption**: specifies the key the QoS domain is to be encrypted with.

**numPlacementIDs**: specifies the number of separate, simultaneously opened super blocks that may be used by the QoS domain in auto allocation mode. It does not affect the number of manually opened super blocks, which instead depends on the device itself.

**maxOpenSuperBlocks**: this is the maximum number super blocks that can be open in a QoS domain. If less than numPlacementIDs it will be set to numPlacementIDs+2. This affects resource and memory usage in the device.

**defaultReadQueue**: specifies the default read queue to use for read I/O operations. This can be optionally overridden when submitting I/O to a QoS domain. Read queues are defined by the virtual device and shared by the QoS domains defined in the same virtual device.

**weights**: Specifies the default weights for erase and program.

# 7 | Super Pages

A Super Page is the optimal unit for physical read and write. It consists of the same hardware page from each die in the super block. When data is read or written, the super page construct allows the data to be striped across the dies to achieve the maximum performance by involving each die of the virtual device in parallel.

The size of a super page is not static but the maximum size is defined by the geometry of the virtual device. The size of a super page may be further constricted by setting a super block size. Super pages are read and written in integer multiples of ADUs. Super pages are grouped into super blocks. The number of super pages contained in a super block is a static number defined by the specific generation of flash memory die being used in the device.

# 8 Super Blocks

Super blocks are the main units of allocation used within the SEF API. By default, super blocks span all the dies within a single virtual device. However, the size of the super block may be changed prior to creating any QoS Domains. The number of super pages in a super block is fixed and is the same as the number of pages in a flash die. The size of a super block, however, is dependent on the configuration of the virtual device that it resides in. A super block can only be a member of a single QoS domain at any point in time. A super block can only be assigned to a different QoS domain after it has been released.

When an erase or allocation occurs within a QoS domain, it is performed in units of super blocks.

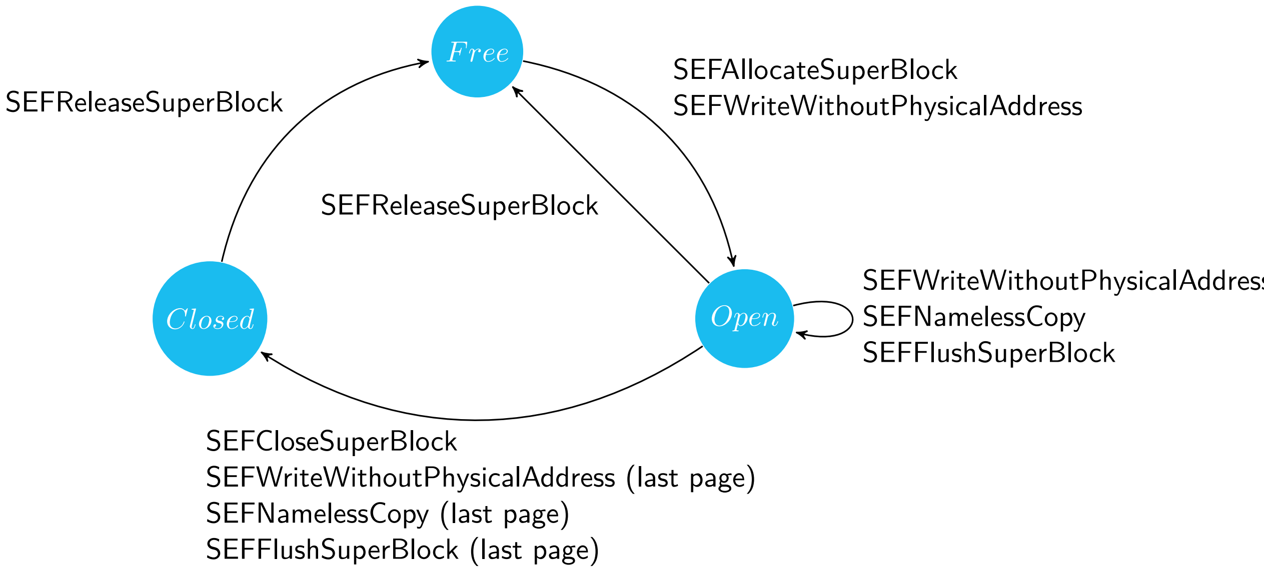
## 8.1 Super Block Management Commands

Super block management commands consist of three functions: `Allocate` , `Close` and `Release`. Super block data commands consist of the commands `Write` and `Copy`. Each command affects state conditions of the super block. Figure 8.1 shows the state transitions regarding super blocks.

Super blocks are allocated either explicitly by the `Allocate` command, or implicitly by the `Write` command. When the reserved flash memory address `SEFAutoAllocate` is specified in a `Write` command, SEF will check if a super block has been allocated for the corresponding placement ID; if not and the QoS domain has not exceeded its capacity limit, a new super block will automatically be allocated and assigned to the placement ID. When a `Write` command with the reserved flash memory address extends past the end of the current automatically opened super block, a new super block will be allocated (assuming the capacity limit is not exceeded) once the current super block is filled.

The host does not need to erase super blocks. When the defect strategy is packed or fragmented, the apparent size of the super block may shrink after it is erased. This affects `SEFWriteWithoutPhysicalAddress()`, `SEFGetSuperBlockInfo()` and `SEFAllocateSuperBlock()`. The number of available ADUs may also shrink as the super block is programmed.

Figure 8.1: Super Block State Transitions




### Free State

`Free` is the initial state for super blocks. `Free` super blocks belong to the free pool owned by a Virtual Device.

`Closed` super blocks transit to `Free` upon the Release command.

### Open State

This is the state of super blocks in the middle of being programmed. `Free` super blocks transit to `Open` by either the Allocate or Write Without Physical Address command.

There are two sub-states of the `Open` states:

- *Open for Write Without Physical Address*: A super block dedicated to `SEFWriteWithoutPhysicalAddress()`. The super block transits to this state via a Nameless Write command without explicit super block ID. The number of super blocks that can exist in this state is determined by the `placementID` parameter at the time of creation for a QoS domain.
- *Open by Erase*: A super block opened by the super block management command `SEFAllocateSuperBlock()`. This super block can be used as a destination for Nameless copy and by Nameless Write by specifying an explicit super block ID.

### Closed

This is the state of super blocks which retain effective data after all Super Pages have been programmed. `Open` super blocks transit to `Closed` by either a Nameless Write command, a Nameless Copy command, an explicit Close command, an explicit Flush command or a device-initiated automatic flush or close.

# 9 | Delayed Writes

Writes to a SEF Unit complete before the final flash memory addresses have been assigned. The addresses returned by `SEFWriteWithoutPhysicalAddress()` are preliminary flash addresses. When a defect is encountered, some portion of the write will be relocated to different flash addresses. When this occurs, a notification is sent mapping the preliminary flash addresses to the final flash memory address. There is no direct notification sent when the preliminary flash memory address is the final flash memory address. A close notification is only sent after all the flash addresses in a super block are final. For specific addresses, it can be inferred by utilizing buffer release notifications. A write buffer supplied to the device must remain valid until the data is committed to flash memory. By default, the library copies the write data to a library controlled buffer. The write call can set the `kSefIoFlagNotifyBufferRelease` flag indicating the the user supplied buffer can be used directly. The caller must keep the buffer valid until notified the buffer can be released. When a buffer release notification is sent, the flash addresses for that portion of the buffer are final.

# 10 Addressing

The physical address of an ADU is assigned by the SEF Unit and returned after the data has been written to a QoS domain. The returned addresses must be supplied when reading the data back from a QoS domain. Because the layout of a flash memory address depends on the type of a SEF Unit, flash memory addresses should be treated as opaque. When debugging, it can be useful to know their structure. They consist of a QoS domain ID, super block ID and an ADU offset as shown in Figure 10.1.

Figure 10.1: Flash Address

| 63 48 | 47 | | 00 |
|---|---|---|---|
| QoS Domain ID | Reserved | Super block ID | ADU Offset |

QoS domain IDs are 16 bits. The lower 48-bit field of LBA field consists of super block ID in the upper and ADU Offset in the lower, and the remaining part is reserved. The exact size of each field depends on the device type. The functions `SEFParseFlashAddress()` and `SEFCreateFlashAddress()` are used to pull apart and build flash memory addresses.

In Perfect and Packed modes, the ADU offset is contiguous from 0 up to the size of the super block. In Fragmented mode, the ADU Offset is non-contiguous and the defective planes are skipped. The ADU Offset is constructed with ADU number, Plane number, Die number and Page number in low-to-high order. Note that each element is not always a power of two.

Figure 10.2: Elements constructing ADU Offset in Fragmented mode

| | | | 00 |
|---|---|---|---|
| Page number | Die Number | Plane number | ADU number |

The function `SEFParseFlashAddress()` and `SEFCreateFlashAddress()` hide the details of deconstructing and constructing a flash memory address.

# 11 | API Management Commands

## 11.1 SEFLibraryInit

```
struct SEFStatus SEFLibraryInit(void)
```

Initializes the SEF Library, enumerates the SEF Units present, and returns the number of units found. Every successful call to SEFLibraryInit() must be balanced with a call to SEFLibraryCleanup().
*See Also: SEFLibraryCleanup*

Table 11.1: Return value of SEFLibraryInit

| Type | Description |
|---|---|
| struct SEFStatus | Status and info summarizing result. |

Table 11.2: Return values of SEFLibraryInit

| Error Value | Description |
|---|---|
| 0 | The info member returns the number of units |

## 11.2 SEFGetHandle

```
SEFHandle SEFGetHandle(uint16_t index)
```

Returns a handle to the SEF Unit at the specified index (zero based)

Table 11.3: Parameters of SEFGetHandle

| Type | Name | Direction | Description |
|---|---|---|---|
| uint16_t | index | In | Index of the SEF Unit |

Table 11.4: Return value of SEFGetHandle

| Type | Description |
| --- | --- |
| SEFHandle | Handle to the SEF Unit |

## 11.3 SEFLibraryCleanup

```
struct SEFStatus SEFLibraryCleanup(void)
```

Performs cleanup of the SEF Library and releases resources.
Every successful call to SEFLibraryInit() must be balanced with a call to SEFLibraryCleanup().
Note: When the returned status error and info fields are zero, all open handles are closed, invalidated and are unusable.
*See Also:* *SEFLibraryInit*

Table 11.5: Return value of SEFLibraryCleanup

| Type | Description |
| --- | --- |
| struct SEFStatus | Status and info summarizing result. |

Table 11.6: Return values of SEFLibraryCleanup

| Error Value | Description |
| --- | --- |
| 0 | The info field is the library's reference count. |
| -ENODEV | The SEF Library was not initialized |
| -EWOULDBLOCK | This function cannot be called on a callback thread |

## 11.4 SEFGetInformation

```
const struct SEFInfo* SEFGetInformation(SEFHandle sefHandle) NONNULL(1)
```

Gets device information.
Returns ADU size(s), number of channels, number of dies, and other associated information. Dynamic values are refreshed just before the structure is returned.

Table 11.7: Parameters of SEFGetInformation

| Type | Name | Direction | Description |
| --- | --- | --- | --- |
| SEFHandle | sefHandle | In | Handle to the SEF Unit |

Table 11.8: Return value of SEFGetInformation

| Type | Description |
|---|---|
| const struct SEFInfo * | SEFInfo struct or NULL if sefHandle is NULL. |

## 11.5 SEFListVirtualDevices

```
struct SEFStatus SEFListVirtualDevices(SEFHandle sefHandle, struct SEFVirtualDev
```

Returns a list of the defined Virtual Devices.
When list is NULL or insufficiently sized or bufferSize is 0, status.info returns the minimum buffer size for the complete list. The data that fits in an insufficiently sized buffer is valid but incomplete. The buffer must be at least the size of the list structure.

Table 11.9: Parameters of SEFListVirtualDevices

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFHandle | sefHandle | In | Handle to the SEF Unit |
| struct SEFVirtualDeviceList* | list | Out | Buffer for storing list of Virtual Devices |
| size_t | bufferSize | In | Buffer size |

Table 11.10: Return value of SEFListVirtualDevices

| Type | Description |
|---|---|
| struct SEFStatus | Status and info summarizing result. |

Table 11.11: Return values of SEFListVirtualDevices

| Error Value | Description |
|---|---|
| -ENODEV | The SEF Handle is not valid |
| -EINVAL | The function parameter is not valid; info returns the parameter index that is not valid |
| 0 | info field returns the minimum buffer size if the buffer is insufficient or NULL; otherwise, 0 |

## 11.6 SEFListQoSDomains

```
struct SEFStatus SEFListQoSDomains(SEFHandle sefHandle, struct SEFQoSDomainList
```

Returns a list of the defined QoS Domains.

When list is NULL or insufficiently sized or bufferSize is 0, status.info returns the minimum buffer size for the complete list. The data that fits in an insufficiently sized buffer is valid but incomplete. The buffer must be at least the size of the list structure.

Table 11.12: Parameters of SEFListQoSDomains

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFHandle | sefHandle | In | Handle to the SEF Unit |
| struct SEFQoSDomainList* | list | Out | Buffer for storing list of QoS Domains |
| size_t | bufferSize | In | Buffer size |

Table 11.13: Return value of SEFListQoSDomains

| Type | Description |
|---|---|
| struct SEFStatus | Status and info summarizing result. |

Table 11.14: Return values of SEFListQoSDomains

| Error Value | Description |
|---|---|
| -ENODEV | The SEF Handle is not valid |
| -EINVAL | The function parameter is not valid; info returns the parameter index that is not valid |
| 0 | info field returns the minimum buffer size if the buffer is insufficient or NULL; otherwise, 0 |

## 11.7 SEFGetUserAddressMeta

```
static uint32_t SEFGetUserAddressMeta(struct SEFUserAddress userAddress)
```

Table 11.15: Parameters of SEFGetUserAddressMeta

| Type | Name | Direction | Description |
|---|---|---|---|
| struct SEFUserAddress | userAddress | In | User address to be parsed |

Table 11.16: Return value of SEFGetUserAddressMeta

| Type | Description |
|---|---|
| uint32_t | Returns meta value from a user address |

## 11.8 SEFGetUserAddressLba

```
static uint64_t SEFGetUserAddressLba(struct SEFUserAddress userAddress)
```

Table 11.17: Parameters of SEFGetUserAddressLba

| Type | Name | Direction | Description |
|---|---|---|---|
| struct SEFUserAddress | userAddress | In | User address to be parsed |

Table 11.18: Return value of SEFGetUserAddressLba

| Type | Description |
|---|---|
| uint64_t | Returns LBA value from a user address |

## 11.9 SEFParseUserAddress

```
static void SEFParseUserAddress(struct SEFUserAddress userAddress, uint64_t *lba
```

Return LBA and meta values from a user address.

Table 11.19: Parameters of SEFParseUserAddress

| Type | Name | Direction | Description |
|---|---|---|---|
| struct SEFUserAddress | userAddress | In | User address to be parsed |
| uint64_t * | lba | Out | Lba parsed from the user address |
| uint32_t * | meta | Out | Meta parsed from the user address |

## 11.10 SEFCreateUserAddress

```
static struct SEFUserAddress SEFCreateUserAddress(uint64_t lba, uint32_t meta)
```

Creates a user address from lba and meta values.

Table 11.20: Parameters of SEFCreateUserAddress

| Type | Name | Direction | Description |
|---|---|---|---|
| uint64_t | lba | In | lba to be used to generate user address (40 bits) |
| uint32_t | meta | In | meta to be used to generate user address (24 bits) |

Table 11.21: Return value of SEFCreateUserAddress

| Type | Description |
|---|---|
| struct SEFUserAddress | Returns the user address created from lba and meta values |

## 11.11 SEFCreateVirtualDevices

```
struct SEFStatus SEFCreateVirtualDevices(SEFHandle sefHandle, uint16_t numVirtua
```

Creates the Virtual Devices and allocates physical resources.
Configuring the virtual devices for a SEF Unit is only done during pre-production. Once the flash of a SEF Unit has been written to, it is not possible to change the Virtual Device configuration.
Configuration is accomplished by supplying a array of pointers to virtualDeviceConfigs. Each Virtual Device being configured will have a single array entry. Each of those entries contains a list of die IDs that will define a specific Virtual Device. The superBlockDies in the config must be 0 or evenly divide into the number of dies specified by the die list.
Valid die IDs start at 0 and are less than the total number of dies in a SEF Unit. The total number of dies is equal to SEFInfo::numBanks * SEFInfo::numChannels. The die ID of a die at channel CH, bank BNK, is equal to CH + BNK*SEFInfo::numChannels. A die ID can only be used in at most one Virtual Device configuration. If a die is not included in any Virtual Device configuration, it will be lost capacity that cannot be used.
*See Also: SEFGetInformation*

Table 11.22: Parameters of SEFCreateVirtualDevices

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFHandle | sefHandle | In | Handle to the SEF Unit |
| uint16_t | numVirtualDevices | In | Number of entries in virtualDeviceConfigs |
| const struct SEFVirtualDeviceConfig[]* | virtualDeviceConfigs | In | Pointers to configurations describing how to create the virtual devices |

Table 11.23: Return value of SEFCreateVirtualDevices

| Type | Description |
|---|---|

| | |
|---|---|
| struct SEFStatus | Status and info summarizing result. Returns 0 on success and negative value on error. |

Table 11.24: Return values of SEFCreateVirtualDevices

| Error Value | Description |
|---|---|
| -ENODEV | The SEF Handle is not valid |
| -EINVAL | The function parameter is not valid; info returns the parameter index that is not valid |
| -EACCES | You don't have the needed permission to perform this operation |

## 11.12 SEFSetNumberOfPSLCSuperBlocks

```
struct SEFStatus SEFSetNumberOfPSLCSuperBlocks(SEFVDHandle vdHandle, uint32_t nu
```

Sets the number of pSLC super blocks.
This defines the number of regular super blocks which are transformed to use as pSLC super blocks. Because it applies to all the dies in the Virtual Device, the value must be a multiple of the ratio of the number of dies in the Virtual Device to the number of configured dies per super block.
Once super blocks have been allocated from the Virtual Device, it may not be possible to modify the number of pSLC super blocks and the call will fail with -ENOSPC.

Table 11.25: Parameters of SEFSetNumberOfPSLCSuperBlocks

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFVDHandle | vdHandle | In | Handle to the SEF Unit |
| uint32_t | numPSLCSuperBlocks | In | The number of pSLC super blocks to set |

Table 11.26: Return values of SEFSetNumberOfPSLCSuperBlocks

| Error Value | Description |
|---|---|
| 0 | The number of pSLC super blocks has been set |
| -ENODEV | The SEF Handle is not valid |
| -ENOSPC | No space is available for pSLC super blocks |
| -EINVAL | The function parameter is not valid; info returns the parameter index that is not valid. |

## 11.13 SEFGetVirtualDeviceUsage

```
struct SEFStatus SEFGetVirtualDeviceUsage(SEFVDHandle vdHandle, struct SEFVirtu
```

Returns Virtual Device usage.

Table 11.27: Parameters of SEFGetVirtualDeviceUsage

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFVDHandle | vdHandle | In | Handle to the Virtual Device |
| struct SEFVirtualDeviceUsage* | usage | Out | Buffer for storing VD usage |

Table 11.28: Return value of SEFGetVirtualDeviceUsage

| Type | Description |
|---|---|
| struct SEFStatus | Status and info summarizing result. |

Table 11.29: Return values of SEFGetVirtualDeviceUsage

| Error Value | Description |
|---|---|
| -ENODEV | The Virtual Device Handle is not valid |
| -EPERM | The Virtual Device Handle is not open |

## 11.14 SEFGetDieList

```
struct SEFStatus SEFGetDieList(SEFHandle sefHandle, struct SEFVirtualDeviceID vi
```

Returns Virtual Device die list.

When list is NULL or insufficiently sized or bufferSize is 0, status.info returns the minimum buffer size for the complete list. The data that fits in an insufficiently sized buffer is valid but incomplete. The buffer must be at least the size of the list structure.

Table 11.30: Parameters of SEFGetDieList

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFHandle | sefHandle | In | Handle to the SEF Unit |
| struct SEFVirtualDeviceID | virtualDeviceID | In | Virtual Device ID |
| struct SEFDieList* | list | Out | Buffer for storing VD information |
| size_t | bufferSize | In | Buffer size |

Table 11.31: Return value of SEFGetDieList

| Type | Description |
|---|---|

| | |
|---|---|
| struct SEFStatus | Status and info summarizing result. |

Table 11.32: Return values of SEFGetDieList

| Error Value | Description |
|---|---|
| -ENODEV | The SEF Handle is not valid |
| -EINVAL | The function parameter is not valid; info returns the parameter index that is not valid |
| 0 | info field returns the minimum buffer size if the buffer is insufficient or NULL; otherwise, 0 |

## 11.15 SEFGetVirtualDeviceInformation

```
struct SEFStatus SEFGetVirtualDeviceInformation(SEFHandle sefHandle, struct SEFV
```

Returns Virtual Device information.

When info is NULL or insufficiently sized or bufferSize is 0, status.info returns the minimum buffer size for the complete set of information. The data that fits in an insufficiently sized buffer is valid but incomplete. The buffer must be at least the size of the info structure.

Table 11.33: Parameters of SEFGetVirtualDeviceInformation

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFHandle | sefHandle | In | Handle to the SEF Unit |
| struct SEFVirtualDeviceID | virtualDeviceID | In | Virtual Device ID |
| struct SEFVirtualDeviceInfo* | info | Out | Buffer for storing VD information |
| size_t | bufferSize | In | Buffer size |

Table 11.34: Return value of SEFGetVirtualDeviceInformation

| Type | Description |
|---|---|
| struct SEFStatus | Status and info summarizing result. |

Table 11.35: Return values of SEFGetVirtualDeviceInformation

| Error Value | Description |
|---|---|
| -ENODEV | The SEF Handle is not valid |
| -EINVAL | The function parameter is not valid; info returns the parameter index that is not valid |

| | |
|---|---|
| 0 | info field returns the minimum buffer size if the buffer is insufficient or NULL; otherwise, 0 |

## 11.16 SEFSetVirtualDeviceSuspendConfig

```
struct SEFStatus SEFSetVirtualDeviceSuspendConfig(SEFVDHandle vdHandle, const st
```

Sets the suspend configuration for a Virtual Device.

Table 11.36: Parameters of SEFSetVirtualDeviceSuspendConfig

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFVDHandle | vdHandle | In | Handle to the SEF Unit |
| const struct SEFVirtualDeviceSuspendConfig* | config | In | Suspend configuration to set |

Table 11.37: Return values of SEFSetVirtualDeviceSuspendConfig

| Error Value | Description |
|---|---|
| 0 | The suspend configuration has been set |
| -ENODEV | The SEF Handle is not valid |
| -EINVAL | The function parameter is not valid; info returns the parameter index that is not valid. |

## 11.17 SEFCreateQoSDomain

```
struct SEFStatus SEFCreateQoSDomain(SEFVDHandle vdHandle, struct SEFQoSDomainID
```

Attempts to create a QoS Domain in the specified Virtual Device.
Returns an error when the target Virtual Device doesn't have enough flash memory space. The actual flash capacity reserved in the Virtual Device is typically larger than what was requested by flashCapacity.
*See Also: SEFGetInformation*

Table 11.38: Parameters of SEFCreateQoSDomain

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFVDHandle | vdHandle | In | Handle to the Virtual Device |

| | | | |
|---|---|---|---|
| struct SEFQoSDomainID* | QoSDomainID | Out | Assigned QoS Domain ID. |
| struct SEFQoSDomainCapacity* | flashCapacity | In | Number of required/reserved/maximum ADUs regular flash |
| struct SEFQoSDomainCapacity* | pSLCFlashCapacity | In | Number of required/reserved/maximum ADUs pSLC flash |
| int | ADUindex | In | Index into the ADUSize[] array in SEFInfo returned by SEFGetInformation() to select the data and metadata sizes of an ADU. |
| enum SEFAPIIdentifier | api | In | Specifies the API Identifier for this QoS Domain |
| enum SEFDefectManagementMethod | defectStrategy | In | Specifies the defect management strategy for the QoS Domain |
| enum SEFErrorRecoveryMode | recovery | In | Specifies the recovery mode for this QoS Domain |
| const char * | encryptionKey | In | NULL for disabled. |
| uint16_t | numPlacementIDs | In | The maximum number of Placement IDs that can be placed on the QoS Domain. |

| | | | |
|---|---|---|---|
| uint16_t | maxOpenSuperBlocks | In | The maximum number super blocks that can be open in a QoS Domain. If less than numPlacementIDs it will be set to numPlacementIDs+2. This affects resource/memory usage in the device. |
| uint8_t | defaultReadQueue | In | The default read queue assignment, 0 through numReadQueues-1 defined for the Virtual Device. |
| struct SEFWeights | weights | In | Weight values for each type of write I/O operations. |

Table 11.39: Return value of SEFCreateQoSDomain

| Type | Description |
|---|---|
| struct SEFStatus | Status and info summarizing result. |

Table 11.40: Return values of SEFCreateQoSDomain

| Error Value | Description |
|---|---|
| -ENODEV | The Virtual Device Handle is not valid |
| -EPERM | The Virtual Device Handle is not open |
| -EINVAL | The function parameter is not valid; info returns the parameter index that is not valid |
| -ENOMEM | The library was unable to allocate needed structures. status.info is set to the type of capacity that caused the failure (0 for kForWrite, 1 for kForPSLCWrite, 2 for QoSD max) |

## 11.18 SEFSetQoSDomainCapacity

```
struct SEFStatus SEFSetQoSDomainCapacity(SEFVDHandle vdHandle, struct SEFQoSDoma
```

Resets the capacity of a QoS Domain.
Sets a new capacity and quota for the QoS Domain. When the flashQuota is less than the flashCapacity or the used flashedCapacity, it will be set to the larger of the two.

Table 11.41: Parameters of SEFSetQoSDomainCapacity

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFVDHandle | vdHandle | In | Handle to the Virtual Device |
| struct SEFQoSDomainID | QoSDomainID | In | QoS Domain ID |
| enum SEFSuperBlockType | type | In | Type of super block |
| struct SEFQoSDomainCapacity* | capacity | In | Number of required/reserved/maximum ADUs the flash type |

Table 11.42: Return value of SEFSetQoSDomainCapacity

| Type | Description |
|---|---|
| struct SEFStatus | Status and info summarizing result. |

Table 11.43: Return values of SEFSetQoSDomainCapacity

| Error Value | Description |
|---|---|
| -ENODEV | The Virtual Device Handle is not valid |
| -EPERM | The Virtual Device Handle is not open |
| -EINVAL | The function parameter is not valid; info returns the parameter index that is not valid |
| -ENOSPC | The Virtual Device does not have enough space |

## 11.19 SEFSetRootPointer

```
struct SEFStatus SEFSetRootPointer(SEFQoSHandle qosHandle, int index, struct SEF
```

Sets the value of a QoSDomain root pointer.
A root pointer may be set to any value. Root pointer values are read back using SEFGetQoSDomainInformation(). When a root pointer is set to a flash address that is valid for the QoS Domain it's stored in, the ADU it points to can be read by SEFReadWithPhysicalAddress() using a flash address of just the root pointer index as the ADU offset with zeros for the QoS DomainId and super block index.
*See Also: SEFReadWithPhysicalAddress*

Table 11.44: Parameters of SEFSetRootPointer

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFQoSHandle | qosHandle | In | Handle to the QoS Domain |
| int | index | In | The index of the root pointer |
| struct SEFFlashAddress | value | In | Value of the pointer |

Table 11.45: Return value of SEFSetRootPointer

| Type | Description |
|---|---|
| struct SEFStatus | Status and info summarizing result. |

Table 11.46: Return values of SEFSetRootPointer

| Error Value | Description |
|---|---|
| -ENODEV | The QoS Domain handle is not valid |
| -EPERM | The QoS Domain Handle is not open |
| -EINVAL | The function parameter is not valid; info returns the parameter index that is not valid |

## 11.20 SEFSetReadDeadline

```
struct SEFStatus SEFSetReadDeadline(SEFQoSHandle qosHandle, enum SEFDeadlineType
```

Sets target QoS Domain's read deadline policy.

*See Also: SEFVirtualDeviceInfo*

Table 11.47: Parameters of SEFSetReadDeadline

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFQoSHandle | qosHandle | In | Handle to the QoS Domain |
| enum SEFDeadlineType | deadline | In | Deadline type for this QoS Domain |

Table 11.48: Return value of SEFSetReadDeadline

| Type | Description |
|---|---|
| struct SEFStatus | Status and info summarizing result. |

Table 11.49: Return values of SEFSetReadDeadline

| Error Value | Description |
|---|---|
| -ENODEV | The QoS Domain handle is not valid |
| -EPERM | The QoS Domain Handle is not open |

## 11.21 SEFSetWeights

```
struct SEFStatus SEFSetWeights(SEFQoSHandle qosHandle, struct SEFWeights weights
```

Sets target QoS Domain's default program and erase weights.

*See Also: SEFQoSDomainInfo*

Table 11.50: Parameters of SEFSetWeights

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFQoSHandle | qosHandle | In | Handle to the QoS Domain |
| struct SEFWeights | weights | In | Default program and erase weights for this QoS Domain |

Table 11.51: Return value of SEFSetWeights

| Type | Description |
|---|---|
| struct SEFStatus | Status and info summarizing result. |

Table 11.52: Return values of SEFSetWeights

| Error Value | Description |
|---|---|
| -ENODEV | The QoS Domain handle is not valid |
| -EPERM | The QoS Domain Handle is not open |

## 11.22 SEFGetSuperBlockList

```
struct SEFStatus SEFGetSuperBlockList(SEFQoSHandle qosHandle, struct SEFSuperBlo
```

Returns a list of super blocks assigned to the QoS Domain.

When list is NULL or insufficiently sized or bufferSize is 0, status.info returns the minimum buffer size for the complete list. The data that fits in an insufficiently sized buffer is valid but incomplete. The buffer must be at least the size of the list structure.

Table 11.53: Parameters of SEFGetSuperBlockList

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFQoSHandle | qosHandle | In | Handle to the QoS Domain |
| struct SEFSuperBlockList* | list | Out | List of super block records |
| size_t | bufferSize | In | Buffer size |

Table 11.54: Return value of SEFGetSuperBlockList

| Type | Description |
|---|---|
| struct SEFStatus | Status and info summarizing result. |

Table 11.55: Return values of SEFGetSuperBlockList

| Error Value | Description |
|---|---|
| -ENODEV | The QoS Domain handle is not valid |
| -EPERM | The QoS Domain Handle is not open |
| -EINVAL | The function parameter is not valid; info returns the parameter index that is not valid |
| 0 | info field returns the minimum buffer size if the buffer is insufficient or NULL; otherwise, 0 |

## 11.23 SEFGetQoSDomainInformation

```
struct SEFStatus SEFGetQoSDomainInformation(SEFHandle sefHandle, struct SEFQoSDo
```

Returns QoS Domain information, including the list of super blocks assigned to the QoS Domain.

Table 11.56: Parameters of SEFGetQoSDomainInformation

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFHandle | sefHandle | In | Handle to the SEF Unit |
| struct SEFQoSDomainID | QoSDomainID | In | QoS Domain ID |
| struct SEFQoSDomainInfo* | info | Out | Buffer for storing QoS Domain information |

Table 11.57: Return value of SEFGetQoSDomainInformation

| Type | Description |
|---|---|
| struct SEFStatus | Status and info summarizing result. |

Table 11.58: Return values of SEFGetQoSDomainInformation

| Error Value | Description |
|---|---|
| -ENODEV | The SEF handle is not valid |
| -EINVAL | The function parameter is not valid; info returns the parameter index that is not valid |
| 0 | SEFQoSDomainInfo was successfully returned. |

## 11.24 SEFGetReuseList

```
struct SEFStatus SEFGetReuseList(SEFQoSHandle qosHandle, struct SEFWearInfo *inf
```

Returns list of super blocks to process for wear-leveling.
Used in support of the implementation of a host-specified wear leveling policy. The SEF Unit has a built in wear-leveling mechanism. It returns closed blocks in the order they should be released if subject to the host-specified wear leveling policy.
When info is NULL or insufficiently sized or bufferSize is 0, status.info returns the minimum buffer size for the complete set of information. The data that fits in an insufficiently sized buffer is valid but incomplete. The buffer must be at least the size of the info structure.

Table 11.59: Parameters of SEFGetReuseList

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFQoSHandle | qosHandle | In | Handle to the QoS Domain |
| struct SEFWearInfo* | info | Out | Buffer for storing information of blocks to process |
| size_t | bufferSize | In | Buffer size |

Table 11.60: Return value of SEFGetReuseList

| Type | Description |
|---|---|
| struct SEFStatus | Status and info summarizing result. |

Table 11.61: Return values of SEFGetReuseList

| Error Value | Description |
|---|---|
| -ENODEV | The QoS Domain handle is not valid |
| -EPERM | The QoS Domain Handle is not open |
| -EINVAL | The function parameter is not valid; info returns the parameter index that is not valid |
| 0 | info field returns the minimum buffer size if the buffer is insufficient or NULL; otherwise, 0 |

## 11.25 SEFGetRefreshList

```
struct SEFStatus SEFGetRefreshList(SEFQoSHandle qosHandle, struct SEFRefreshInfo
```

Returns a list of blocks that have encountered ECC corrections.
These blocks subsequently need to be re-written, or else data loss may occur. This call should be part of a periodic background check to guard against data loss.
When info is NULL or insufficiently sized or bufferSize is 0, status.info returns the minimum buffer size for the complete set of information. The data that fits in an insufficiently sized buffer is valid but incomplete. The buffer must be at least the size of the info structure.

Table 11.62: Parameters of SEFGetRefreshList

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFQoSHandle | qosHandle | In | Handle to the QoS Domain |
| struct SEFRefreshInfo* | info | Out | Buffer for storing information of blocks to process |
| size_t | bufferSize | In | Buffer size |

Table 11.63: Return value of SEFGetRefreshList

| Type | Description |
|---|---|
| struct SEFStatus | Status and info summarizing result. |

Table 11.64: Return values of SEFGetRefreshList

| Error Value | Description |
|---|---|
| -ENODEV | The QoS Domain handle is not valid |
| -EPERM | The QoS Domain Handle is not open |
| -EINVAL | The function parameter is not valid; info returns the parameter index that is not valid |
| 0 | info field returns the minimum buffer size if the buffer is insufficient or NULL; otherwise, 0 |

## 11.26 SEFGetCheckList

```
struct SEFStatus SEFGetCheckList(SEFQoSHandle qosHandle, struct SEFCheckInfo *in
```

Returns a list of blocks that have encountered conditions that need to be checked.
In the event that this command indicates that blocks need to be checked, a subsequent patrol command (SEFCheckSuperBlock) should be issued in response. Detailed error statistics will be returned as part of the patrol, and appropriate corrective actions can be based on the returned information.

When info is NULL or insufficiently sized or bufferSize is 0, status.info returns the minimum buffer size for the complete set of information. The data that fits in an insufficiently sized buffer is valid but incomplete. The buffer must be at least the size of the info structure.
*See Also: SEFCheckSuperBlock*

Table 11.65: Parameters of SEFGetCheckList

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFQoSHandle | qosHandle | In | Handle to the QoS Domain |
| struct SEFCheckInfo* | info | Out | Buffer for storing information of blocks to process |
| size_t | bufferSize | In | Buffer size |

Table 11.66: Return value of SEFGetCheckList

| Type | Description |
|---|---|
| struct SEFStatus | Status and info summarizing result. |

Table 11.67: Return values of SEFGetCheckList

| Error Value | Description |
|---|---|
| -ENODEV | The QoS Domain handle is not valid |
| -EPERM | The QoS Domain Handle is not open |
| -EINVAL | The function parameter is not valid; info returns the parameter index that is not valid |
| 0 | info field returns the minimum buffer size if the buffer is insufficient or NULL; otherwise, 0 |

## 11.27 SEFGetUserAddressList

```
struct SEFStatus SEFGetUserAddressList(SEFQoSHandle qosHandle, struct SEFFlashAc
```

Returns the user address list in terms of its underlying super blocks.
Used as part of an FTL reconstruction activity. This can happen in the event of, for example, ungraceful shutdown. This mechanism can also be used to build custom diagnostic tools. This command is not needed during normal operation.
ADUs that have not been written return a user address equal to SEFUserAddressIgnore.
When list is NULL or insufficiently sized or bufferSize is 0, status.info returns the minimum buffer size for the complete list. The data that fits in an insufficiently sized buffer is valid but incomplete. The buffer must be at least the size of the list structure.

Table 11.68: Parameters of SEFGetUserAddressList

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFQoSHandle | qosHandle | In | Handle to the QoS Domain |
| struct SEFFlashAddress | flashAddress | In | Flash address of the super block |
| struct SEFUserAddressList* | list | Out | Buffer for storing list of user addresses |
| size_t | bufferSize | In | Buffer size |

Table 11.69: Return value of SEFGetUserAddressList

| Type | Description |
|---|---|
| struct SEFStatus | Status and info summarizing result. |

Table 11.70: Return values of SEFGetUserAddressList

| Error Value | Description |
|---|---|
| -ENODEV | The QoS Domain handle is not valid |
| -EPERM | The QoS Domain Handle is not open |
| -EINVAL | The function parameter is not valid; info returns the parameter index that is not valid |
| 0 | info field returns the minimum buffer size if the buffer is insufficient or NULL; otherwise, 0 |

## 11.28 SEFGetSuperBlockInfo

```
struct SEFStatus SEFGetSuperBlockInfo(SEFQoSHandle qosHandle, struct SEFFlashAdd
```

Returns information corresponding to the super block.

Table 11.71: Parameters of SEFGetSuperBlockInfo

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFQoSHandle | qosHandle | In | Handle to the QoS Domain |
| struct SEFFlashAddress | flashAddress | In | Flash address of the super block |
| int | getDefectMap | In | When non-zero populates the defectBitmap member of SEFSuperBlockInfo. See SEFSuperBlockInfo for information on the size of defectBitmap |
| struct SEFSuperBlockInfo* | info | Out | Buffer for storing super block information |

Table 11.72: Return value of SEFGetSuperBlockInfo

| Type | Description |
|---|---|
| struct SEFStatus | Status and info summarizing result. |

Table 11.73: Return values of SEFGetSuperBlockInfo

| Error Value | Description |
|---|---|
| -ENODEV | The QoS Domain handle is not valid |
| -EPERM | The QoS Domain Handle is not open |
| -EINVAL | The function parameter is not valid; info returns the parameter index that is not valid |

## 11.29 SEFCheckSuperBlock

```
struct SEFStatus SEFCheckSuperBlock(SEFQoSHandle qosHandle, struct SEFFlashAddre
```

This is a read patrol operation which is used in conjunction with SEFGetCheckList and the kRequirePatrol QoS Notification.
Patrol reads don't use the scheduling queues and are issued as soon as possible. Any actions required by the result of the patrol will generate the appropriate QoS Notification.
*See Also: SEFGetCheckList*

Table 11.74: Parameters of SEFCheckSuperBlock

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFQoSHandle | qosHandle | In | Handle to the QoS Domain |
| struct SEFFlashAddress | flashAddress | In | Flash address of the super block to be checked |

Table 11.75: Return value of SEFCheckSuperBlock

| Type | Description |
|---|---|
| struct SEFStatus | Status and info summarizing result. |

Table 11.76: Return values of SEFCheckSuperBlock

| Error Value | Description |
|---|---|
| 0 | The super block is checked |
| -ENODEV | The QoS Domain handle is not valid |
| -EPERM | The QoS Domain Handle is not open |

| | |
|---|---|
| -EINVAL | The function parameter is not valid; info returns the parameter index that is not valid |

## 11.30 SEFDeleteVirtualDevices

```
struct SEFStatus SEFDeleteVirtualDevices(SEFHandle sefHandle) NONNULL(1)
```

Deletes the Virtual Devices and allocated physical resources.
Deleting virtual devices for a SEF Unit can only be done during pre-production. Once the flash of a SEF Unit has been written to, it is not possible to delete the Virtual Device configuration.

Table 11.77: Parameters of SEFDeleteVirtualDevices

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFHandle | sefHandle | In | Handle to the SEF Unit |

Table 11.78: Return value of SEFDeleteVirtualDevices

| Type | Description |
|---|---|
| struct SEFStatus | Status and info summarizing result. Returns 0 on success and negative value on error. |

Table 11.79: Return values of SEFDeleteVirtualDevices

| Error Value | Description |
|---|---|
| -ENODEV | The SEF Handle is not valid |
| -EINVAL | The function parameter is not valid; info returns the parameter index that is not valid |
| -EACCES | You don't have the needed permission to perform this operation |
| -ENOTEMPTY | At least one QoS Domain exists |
| -EBUSY | The Virtual Device is in use and not all the handles are closed |

## 11.31 SEFDeleteQoSDomain

```
struct SEFStatus SEFDeleteQoSDomain(SEFHandle sefHandle, struct SEFQoSDomainID Q
```

Deletes the target QoS Domain.
The QoS Domain must be in the closed state before issuing this command. After closing the target QoS Domain, its assigned super blocks are returned to the Virtual Device's free pool.

Table 11.80: Parameters of SEFDeleteQoSDomain

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFHandle | sefHandle | In | Handle to the SEF Unit |
| struct SEFQoSDomainID | QoSDomainID | In | QoS Domain ID |

Table 11.81: Return value of SEFDeleteQoSDomain

| Type | Description |
|---|---|
| struct SEFStatus | Status and info summarizing result. |

Table 11.82: Return values of SEFDeleteQoSDomain

| Error Value | Description |
|---|---|
| -ENODEV | The SEF handle is not valid |
| -EINVAL | The QoS Domain ID is not valid |
| -EACCES | You don't have the needed permission to perform this operation |
| -EBUSY | The QoS Domain is in use and not all the handles are closed |

## 11.32 SEFResetEncryptionKey

```
struct SEFStatus SEFResetEncryptionKey(SEFVDHandle vdHandle, struct SEFQoSDomai
```

Resets the encryption key for a QoS Domain.

Table 11.83: Parameters of SEFResetEncryptionKey

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFVDHandle | vdHandle | In | Handle to the Virtual Drive |
| struct SEFQoSDomainID | QoSDomainID | In | QoS Domain ID |

Table 11.84: Return value of SEFResetEncryptionKey

| Type | Description |
|---|---|
| struct SEFStatus | Status and info summarizing result. |

Table 11.85: Return values of SEFResetEncryptionKey

| Error Value | Description |
|---|---|
| -ENODEV | The Virtual Device handle is not valid |
| -EPERM | The Virtual Device handle is not open |
| -EINVAL | The QoS Domain Id is not valid |

## 11.33 SEFOpenVirtualDevice

```
struct SEFStatus SEFOpenVirtualDevice(SEFHandle sefHandle, struct SEFVirtualDevi
```

Opens the target Virtual Device.

Since Virtual Devices are persistent, this provides the mechanism for opening a preexisting Virtual Device to resume I/O after reboot. This function needs to be called in order to receive notifications about the Virtual Device, such as in the event that a reduced capacity notification is issued.

Table 11.86: Parameters of SEFOpenVirtualDevice

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFHandle | sefHandle | In | Handle to the SEF Unit |
| struct SEFVirtualDeviceID | virtualDeviceID | In | Virtual Device ID |
| void(*)(void *, struct SEFVDNotification) | notifyFunc | In | Callback to be executed upon event generation |
| void * | context | In | A void* pointer passed to the async event notification function (used to pass user context information) |
| SEFVDHandle * | vdHandle | In | Handle to the Virtual Drive |

Table 11.87: Return value of SEFOpenVirtualDevice

| Type | Description |
|---|---|
| struct SEFStatus | Status and info summarizing result. |

Table 11.88: Return values of SEFOpenVirtualDevice

| Error Value | Description |
|---|---|

| | |
|---|---|
| -ENODEV | The SEF handle is not valid |
| -EINVAL | The function parameter is not valid; info returns the parameter index that is not valid |
| -EACCES | You don't have the needed permission to perform this operation |
| -ENOMEM | The library was unable to allocate needed structures |
| -EALREADY | The Virtual Device is already open |

## 11.34 SEFCloseVirtualDevice

```
struct SEFStatus SEFCloseVirtualDevice(SEFVDHandle vdHandle) NONNULL(1)
```

Closes an open Virtual Device and shuts down associated event notification.

Table 11.89: Parameters of SEFCloseVirtualDevice

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFVDHandle | vdHandle | In | Handle to the Virtual Device |

Table 11.90: Return value of SEFCloseVirtualDevice

| Type | Description |
|---|---|
| struct SEFStatus | Status and info summarizing result. |

Table 11.91: Return values of SEFCloseVirtualDevice

| Error Value | Description |
|---|---|
| -ENODEV | The Virtual Device handle is not valid |
| -EPERM | The Virtual Device Handle is not open |
| -EWOULDBLOCK | This function cannot be called on a callback thread |

## 11.35 SEFOpenQoSDomain

```
struct SEFStatus SEFOpenQoSDomain(SEFHandle sefHandle, struct SEFQoSDomainID QoS
```

Opens a previously created QoS Domain.
Since QoS Domains are persistent, this provides the mechanism for opening a preexisting QoS Domain to resume I/O after reboot. This function also provides a channel to receive notifications regarding this QoS Domain.

Table 11.92: Parameters of SEFOpenQoSDomain

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFHandle | sefHandle | In | Handle to the SEF Unit |
| struct SEFQoSDomainID | QoSDomainID | In | QoS Domain ID |
| void(*)(void *, struct SEFQoSNotification) | notifyFunc | In | Callback to be executed during event generation |
| void * | context | In | A void* pointer passed to the async event notification function (used to pass user context information) |
| const void * | encryptionKey | In | In a multi-tenant environment, different tenants will write to separate QoS domains. Provides for individualized encryption keys on a per-domain basis |
| SEFQoSHandle * | qosHandle | Out | Handle to the QoS Domain |

Table 11.93: Return value of SEFOpenQoSDomain

| Type | Description |
|---|---|
| struct SEFStatus | Status and info summarizing result. |

Table 11.94: Return values of SEFOpenQoSDomain

| Error Value | Description |
|---|---|
| -ENODEV | The SEF handle is not valid |
| -EINVAL | The function parameter is not valid; info returns the parameter index that is not valid |
| -EACCES | You don't have the needed permission to perform this operation |
| -ENOMEM | The library was unable to allocate needed structures |
| -EALREADY | The QoS Domain is already open |

## 11.36 SEFCloseQoSDomain

```
struct SEFStatus struct SEFStatus SEFCloseQoSDomain(SEFQoSHandle qosHandle) NONN
```

Closes an open QoS Domain.

This will close any open super blocks associated with this domain. All outstanding kSuperBlockChangeState events will be delivered before this function returns. A QoS Domain must be in the closed state to be deleted.

Table 11.95: Parameters of SEFCloseQoSDomain

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFQoSHandle | qosHandle | In | Handle to the QoS Domain |

Table 11.96: Return value of SEFCloseQoSDomain

| Type | Description |
|---|---|
| struct SEFStatus struct SEFStatus | Status and info summarizing result. |

Table 11.97: Return values of SEFCloseQoSDomain

| Error Value | Description |
|---|---|
| -ENODEV | The QoS Domain handle is not valid |
| -EPERM | The QoS Domain Handle is not open |
| -EWOULDBLOCK | This function cannot be called on a callback thread |

## 11.37 SEFGetQoSHandleProperty

```
struct SEFProperty SEFGetQoSHandleProperty(SEFQoSHandle qos, enum SEFPropertyID
```

This function gets a property given a SEFQoSHandle.

Table 11.98: Parameters of SEFGetQoSHandleProperty

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFQoSHandle | qosHandle | In | Handle to the QoS Domain |
| enum SEFPropertyID | propID | In | The Property ID requested |

Table 11.99: Return value of SEFGetQoSHandleProperty

| Type | Description |
|---|---|

| | |
|---|---|
| struct SEFProperty | Returns the property stored given the property ID; If an unknown property ID is passed in, the returned type of the property will be kSefPropertyTypeNull. If kSefPropertyPrivateData is not set, the returned type of the property will be kSefPropertyTypeNull. |

## 11.38 SEFSetQoSHandleProperty

```
struct SEFStatus SEFSetQoSHandleProperty(SEFQoSHandle qos, enum SEFPropertyID pr
```

This function sets a property given a SEFQoSHandle.
The only settable property is kSefPropertyPrivateData.

Table 11.100: Parameters of SEFSetQoSHandleProperty

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFQoSHandle | qosHandle | In | Handle to the QoS Domain |
| enum SEFPropertyID | propID | In | The Property ID being stored |
| struct SEFProperty | value | In | The value of the property being stored |

Table 11.101: Return value of SEFSetQoSHandleProperty

| Type | Description |
|---|---|
| struct SEFStatus | Status and info summarizing result. |

Table 11.102: Return values of SEFSetQoSHandleProperty

| Error Value | Description |
|---|---|
| -ENODEV | The QoS Domain handle is not valid |
| -EPERM | The QoS Domain Handle is not open |
| -EINVAL | The function parameter is not valid; info returns the parameter index that is not valid |

## 11.39 SEFParseFlashAddress

```
struct SEFStatus SEFParseFlashAddress(SEFQoSHandle qosHandle, struct SEFFlashAdd
```

This function is used to extract info needed by FTL from an opaque flash address.
The QoS Domain ID of the passed in qosHandle does not have to match the QoS Domain ID of the passed in flash address. No validation is performed and the address is parsed as if it came from the

QoS Domain of the passed in qosHandle. When they differ, it's up to the client to ensure the two different QoS Domain IDs are compatible. That is, the virtual devices they live in have the same value for superBlockDies.

Table 11.103: Parameters of SEFParseFlashAddress

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFQoSHandle | qosHandle | In | Handle to a QoS Domain to interpret/parse the flash address. May be NULL if only the QoSDomainID is being returned. |
| struct SEFFlashAddress | flashAddress | In | The opaque address to be parsed |
| struct SEFQoSDomainID* | QoSDomainID | Out | A pointer to where to return the QoS Domain ID. A null pointer indicates that the QoS Domain ID is not to be returned |
| uint32_t * | blockNumber | Out | A pointer to where to return the block number. A null pointer indicates that the block number is not to be returned |
| uint32_t * | ADUOffset | Out | A pointer to where to return the ADU Offset. A null pointer indicates that the ADU Offset is not to be returned |

Table 11.104: Return value of SEFParseFlashAddress

| Type | Description |
|---|---|
| struct SEFStatus | Status and info summarizing result. |

## 11.40 SEFCreateFlashAddress

```
struct SEFFlashAddress SEFCreateFlashAddress(SEFQoSHandle qosHandle, struct SEF
```

This function is used to create an opaque flash address.

A generated flash address may be rejected by the device if it specifies an illegal ADUOffset, a super block number not owned by the QoSDomainID, or a QoSDomainID that has not been opened by the caller.

Table 11.105: Parameters of SEFCreateFlashAddress

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFQoSHandle | qosHandle | In | Handle of a QoS Domain to create a flash address for. |

| | | | |
|---|---|---|---|
| struct SEFQoSDomainID | QoSDomainID | In | The desired QoS Domain ID. It is not validated against the QoS Domain ID of the qosHandle. |
| uint32_t | blockNumber | In | The desired super block number. |
| uint32_t | ADUOffset | In | The desired ADU Offset. |

Table 11.106: Return value of SEFCreateFlashAddress

| Type | Description |
|---|---|
| struct SEFFlashAddress | The generated flash address or the NULL flashAddress if the qosHandle is invalid. |

## 11.41 SEFReleaseSuperBlock

```
struct SEFStatus SEFReleaseSuperBlock(SEFQoSHandle qosHandle, struct SEFFlashAdd
```

Releases the specific super block to the free pool owned by the Virtual Device to which the specified QoS Domain belongs.
The target super block must have been assigned by a previous call to SEFAllocateSuperBlock() or as part of SEFWriteWithoutPhysicalAddress(). The super block may be in an open or closed state.

Table 11.107: Parameters of SEFReleaseSuperBlock

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFQoSHandle | qosHandle | In | Handle to the QoS Domain of the super block |
| struct SEFFlashAddress | flashAddress | In | Flash address of the super block to release |

Table 11.108: Return value of SEFReleaseSuperBlock

| Type | Description |
|---|---|
| struct SEFStatus | Status and info summarizing result. |

Table 11.109: Return values of SEFReleaseSuperBlock

| Error Value | Description |
|---|---|
| -ENODEV | The QoS Domain handle is not valid |
| -EPERM | The QoS Domain Handle is not open |
| -EFAULT | The Flash Address is not valid |

## 11.42 SEFAllocateSuperBlock

```
struct SEFStatus SEFAllocateSuperBlock(SEFQoSHandle qosHandle, struct SEFFlashAc
```

Allocates a super block that will be assigned to the specified QoS Domain and returns the flash address of this super block.

The returned super block will be in the open state. These super blocks in turn can be used as part of the parameter set for the SEFNamelessCopy and SEFWriteWithoutPhysicalAddress functions. When allocating a super block, The SEF Unit intelligently selects a location in a manner designed to optimize the lifetime of flash memory and will return the flash address that was selected. Note that each open super block will allocate a write buffer and therefore consume memory, so there is a tradeoff in the number of open super blocks and the amount of memory consumed.

It's required that the total ADUs in the QoS Domain be less than its flash quota and its Virtual Device have an available super block. The ADUs in use by a QoS Domain can be known by summing the writableADUs of each super block in the QoS Domain.

*See Also: SEFGetQoSDomainInformation*

Table 11.110: Parameters of SEFAllocateSuperBlock

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFQoSHandle | qosHandle | In | Handle to the QoS Domain |
| struct SEFFlashAddress* | flashAddress | Out | The flash address of the allocated block |
| enum SEFSuperBlockType | type | In | kForWrite or kForPSLCWrite |
| uint8_t * | defectMap | Out | Optional buffer to receive the block's defect map. Used for kFragmented QoS Domains. When supplied, the buffer must be at least as large as SEFQoSDomainInfo::defectMapSize. |
| const struct SEFAllocateOverrides* | overrides | In | Overrides to scheduler parameters; pointer can be null for none required. |

Table 11.111: Return value of SEFAllocateSuperBlock

| Type | Description |
|---|---|
| struct SEFStatus | Status and info summarizing result. |

Table 11.112: Return values of SEFAllocateSuperBlock

| Error Value | Description |
|---|---|
| 0 | The info member contains number of ADUs in allocated super block |
| -ENODEV | The QoS Domain handle is not valid |
| -EPERM | The QoS Domain Handle is not open |
| -EINVAL | The function parameter is not valid; info returns the parameter index that is not valid |
| -ENOSPC | The QoS Domain is out of space |

## 11.43 SEFFlushSuperBlock

```
struct SEFStatus struct SEFStatus SEFFlushSuperBlock(SEFQoSHandle qosHandle, str
```

Flushes the target super block.

This command causes all written data for the super block that is still in the write buffer and not persisted to flash memory to be persisted to flash memory. The device will automatically append data if necessary to finish programming of all pending user data writes. This command will not return until all the data is persistent and all kAddressUpdate change notifications generated by the flush have been processed.

When a flush causes a super block to have no more writable ADUs, the super block will be closed and a QoS Domain notification of the close will be sent.

Table 11.113: Parameters of SEFFlushSuperBlock

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFQoSHandle | qosHandle | In | Handle to the QoS Domain of the super block |
| struct SEFFlashAddress | flashAddress | In | Flash address of the super block to be flushed. |
| uint32_t * | distanceToEndOfSuperBlock | Out | Indicates remaining size in ADU after this flush operation. May be NULL. |

Table 11.114: Return value of SEFFlushSuperBlock

| Type | Description |
|---|---|
| struct SEFStatus struct SEFStatus | Status and info summarizing result. |

Table 11.115: Return values of SEFFlushSuperBlock

| Error Value | Description |
|---|---|
| -ENODEV | The QoS Domain handle is not valid |
| -EPERM | The QoS Domain Handle is not open |
| -EINVAL | The function parameter is not valid; info returns the parameter index that is not valid |

## 11.44 SEFCloseSuperBlock

```
struct SEFStatus SEFCloseSuperBlock(SEFQoSHandle qosHandle, struct SEFFlashAddre
```

Closes the target super block.
If there is remaining unwritten space in the super block, that space will be padded with dummy data. This can be used by the FTL as a means of closing a super block without invoking a Write command. This command will not return until all the data is persistent and all kAddressUpdate change notifications generated by the close have been processed ensuring that all addresses have either transitioned from tentative to permanent or have been updated.

Table 11.116: Parameters of SEFCloseSuperBlock

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFQoSHandle | qosHandle | In | Handle to the QoS Domain of the super block |
| struct SEFFlashAddress | flashAddress | In | Flash address of the super block to move to Closed state by filling data |

Table 11.117: Return value of SEFCloseSuperBlock

| Type | Description |
|---|---|
| struct SEFStatus | Status and info summarizing result. When .error is zero, .info is the size of the super block. |

Table 11.118: Return values of SEFCloseSuperBlock

| Error Value | Description |
|---|---|
| 0 | The super block is was closed or was already closed |
| -ENODEV | The QoS Domain handle is not valid |
| -EPERM | The QoS Domain Handle is not open |
| -EFAULT | The Flash Address is not valid |

## 11.45 SEFReleaseSuperBlockAsync

```
void SEFReleaseSuperBlockAsync(SEFQoSHandle qosHandle, struct SEFReleaseSuperBlo
```

This function is the asynchronous version of SEFReleaseSuperBlock().
A caller allocated IOCB is supplied to initiate the release. The caller can free the IOCB once the kSefIoFlagDone bit is set in common.flags or the common.callback_func has been called. When the IOCB is malformed, the callback_func may be called on the submitters thread.
*See Also: SEFReleaseSuperBlock*

Table 11.119: Parameters of SEFReleaseSuperBlockAsync

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFQoSHandle | qosHandle | In | Handle to the QoS Domain |
| struct SEFReleaseSuperBlockIOCB* | iocb | In/Out | For asynchronous response from SEF Library Unused fields should be set to 0. |

## 11.46 SEFAllocateSuperBlockAsync

```
void SEFAllocateSuperBlockAsync(SEFQoSHandle qosHandle, struct SEFAllocateSuperB
```

This function is the asynchronous version of SEFAllocateSuperBlock().
A caller allocated IOCB is supplied to initiate the allocation. The caller can free the IOCB once the kSefIoFlagDone bit is set in common.flags or the common.callback_func has been called. When the IOCB is malformed, the callback_func may be called on the submitters thread.
*See Also: SEFAllocateSuperBlock*

Table 11.120: Parameters of SEFAllocateSuperBlockAsync

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFQoSHandle | qosHandle | In | Handle to the QoS Domain |
| struct SEFAllocateSuperBlockIOCB* | iocb | In/Out | For asynchronous response from SEF Library Unused fields should be set to 0. |

## 11.47 SEFCloseSuperBlockAsync

```
void SEFCloseSuperBlockAsync(SEFQoSHandle qosHandle, struct SEFCloseSuperBlockIO
```

This function is the asynchronous version of SEFCloseSuperBlock().

A caller allocated IOCB is supplied to initiate the close. The caller can free the IOCB once the kSefIoFlagDone bit is set in common.flags or the common.callback_func has been called. When the IOCB is malformed, the callback_func may be called on the submitters thread.

Note: kSuperBlockStateChanged will have been sent and processed before the completion routine is called and the iocb is marked as done.

*See Also: SEFCloseSuperBlock*

Table 11.121: Parameters of SEFCloseSuperBlockAsync

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFQoSHandle | qosHandle | In | Handle to the QoS Domain |
| struct SEFCloseSuperBlockIOCB* | iocb | In/Out | For asynchronous response from SEF Library |

# 12 | Data Access Commands

## 12.1 SEFWriteWithoutPhysicalAddress

```
struct SEFStatus SEFWriteWithoutPhysicalAddress(SEFQoSHandle qosHandle, struct S
```

Writes data to the specified user address to an underlying physical flash page that is assigned for the QoS Domain.

If auto-allocate is enabled for the super block, when the assigned super block is filled and closed, the SEF Unit assigns a new super block for the remaining writes. If auto-allocate is not enabled, host software will know about the super block size as part of the allocation, and can use this information to construct appropriately-sized write commands. This call will not return until the data has been persisted, and will automatically pad the user data with dummy data if required to complete flash memory programming. The userAddress supplied here will be checked when reading the data back with SEFReadWithPhysicalAddress(). If storing a user address is not required, a userAddress of SEFUserAddressIgnore may be used. The check can optionally be disabled when reading and must be disabled to read data written with a user address of SEFUserAddressIgnore. In kSuperBlock mode and writing multiple ADUs, the LBA portion of the user address is incremented for each ADU. The write will fail if the userAddress is incremented to a value equal to SEFUserAddressIgnore. The userAddresses in a super block can be read using SEFGetUserAddressList.

Note: The synchronous and asynchronous versions differ in how data is committed to flash. As described above, the synchronous version flushes data to flash returning permanent flash addresses.

In contrast, the asynchronous version lazily flushes data to flash. The flash addresses returned are tentative instead. Once the SEF Unit eventually flushes a tentative address to flash, the original address may be discovered to be bad. When this happens, a kAddressUpdate QoS Domain notification is sent indicating the data has moved to a new permanent flash address. When the IOCB flag kSefIoFlagNotifyBufferRelease is set, the domain notification kBufferRelease will be sent for each piece of the IOCB iov as it becomes committed to flash. It is then the responsibility of the caller to maintain the lifetime of the iov buffers until the release notifications are sent. When not set, the commit state can be inferred instead by the kSuperBlockStateChanged QoS notification for the owning super block. Buffer lifetime is managed by library in this case by copying write data into library managed buffers.

Table 12.1: Parameters of SEFWriteWithoutPhysicalAddress

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFQoSHandle | qosHandle | In | Handle to the QoS Domain |
| struct SEFFlashAddress | flashAddress | In | Flash address of the super block. SEFAutoAllocate(PSLC) if auto allocate. |
| struct SEFPlacementID | placementID | In | Only valid if the flashAddress is auto allocated. The type of FLASH is set when the write allocates a new super block. A value from 0 to numPlacementIds–1 indicating what logical data group to place this data in. |
| struct SEFUserAddress | userAddress | In | FTL can store meta-data related to this operation by this field. For example, storing LBA address to bind to this write operation such as data tags. |
| uint32_t | numADU | In | Total amount of write data size calculated in QoS Domain ADUs. |
| const struct iovec* | iov | In | A pointer to the scatter gather list |
| uint16_t | iovcnt | In | The number of elements in the scatter gather list |

| | | | |
|---|---|---|---|
| const void * | metadata | In | Pointer to metadata to write with the data; The number of bytes per ADU required is SEFQoSDomainInfo::ADUsize.meta. May be NULL. |
| struct SEFFlashAddress* | permanentAddresses | Out | Must allocate space for returned permanent physical addresses equal to 8*length (i.e. 8*number of ADUs) |
| uint32_t * | distanceToEndOfSuperBlock | Out | Indicates remaining size in ADU after this write operation. May be NULL. This is not a guarantee as the block may be forced closed if too many super blocks are open. When this returns 0, the block was closed. |
| const struct SEFWriteOverrides* | overrides | In | Overrides to scheduler parameters; pointer can be null for none required. |

Table 12.2: Return value of SEFWriteWithoutPhysicalAddress

| Type | Description |
|---|---|
| struct SEFStatus | Status and info summarizing result. When .error is non-zero, .info is the number of ADUs written. |

Table 12.3: Return values of SEFWriteWithoutPhysicalAddress

| Error Value | Description |
|---|---|
| -ENODEV | The QoS Domain handle is not valid |
| -EPERM | The QoS Domain Handle is not open |

| | |
|---|---|
| -EINVAL | The function parameter is not valid; info returns the parameter index that is not valid |
| -ENOSPC | The QoS Domain is out of space |

## 12.2 SEFReadWithPhysicalAddress

```
struct SEFStatus SEFReadWithPhysicalAddress(SEFQoSHandle qosHandle, struct SEFF
```

Reads data from a specified physical address.

While writes are expressed in terms of a placement ID or super block flash addresses, reads are expressed in terms of physical flash addresses. Read commands may interrupt other types of commands. When there is an in-flight flash memory command to the same flash die other than a read command, the in-flight command may be suspended in order to maintain deterministic read latency. If the target physical address is currently in the process of being programmed, data will instead be returned from the write buffer.

The userAddress must either match what was stored when the data was written or be SEFUserAddressIgnore to disable checking. In kSuperBlock mode, the LBA portion of the user address is incremented for each ADU in a multi-ADU write.

Note: When reading data that was just written, a read error will be returned when the data's original flash address has been updated but the notification has yet to be processed by the client. In this case, the caller must retry the read after the flash address change notification has been processed.

*See Also: SEFSetRootPointer*

Table 12.4: Parameters of SEFReadWithPhysicalAddress

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFQoSHandle | qosHandle | In | Handle to the QoS Domain |
| struct SEFFlashAddress | flashAddress | In | Physical address for the read command; When the QoS Domain ID and block number are 0, the ADU offset is the root pointer index for the flash address to read. |
| uint32_t | numADU | In | Length of data to read (in ADUs). Maximum allowed is superBlockCapacity. |
| const struct iovec* | iov | In | A pointer to the scatter gather list |
| uint16_t | iovcnt | In | The number of elements in the scatter gather list |
| size_t | iovOffset | In | Starting byte offset into iov array |

| | | | |
|---|---|---|---|
| struct SEFUserAddress | userAddress | In | Stored data by the FTL. It will be validated with what was stored when the data was written except when SEFUserAddressIgnore is supplied |
| void * | metadata | In | Buffer to receive metadata stored with the data; The number of bytes per ADU required is SEFQoSDomainInfo::ADUsize.meta. May be NULL |
| const struct SEFReadOverrides* | overrides | In | Overrides to scheduler parameters; pointer can be null for none required. |

Table 12.5: Return value of SEFReadWithPhysicalAddress

| Type | Description |
|---|---|
| struct SEFStatus | Status and info summarizing result. |

Table 12.6: Return values of SEFReadWithPhysicalAddress

| Error Value | Description |
|---|---|
| -ENODEV | The QoS Domain handle is not valid |
| -EPERM | The QoS Domain Handle is not open |
| -EINVAL | The function parameter is not valid; info returns the parameter index that is not valid |

## 12.3 SEFNamelessCopy

```
struct SEFStatus SEFNamelessCopy(SEFQoSHandle srcQosHandle, struct SEFCopySource
```

Performs Nameless Copy with map or list; optional user address filtering.
Copies ADUs as described by copySource to the copyDestination. Source addresses can only reference closed super blocks.

Table 12.7: Parameters of SEFNamelessCopy

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFQoSHandle | srcQosHandle | In | Handle to the source QoS Domain |

| | | | |
|---|---|---|---|
| struct SEFCopySource | copySource | In | Physical addresses to copy |
| SEFQoSHandle | dstQosHandle | In | Handle to the destination QoS Domain |
| struct SEFFlashAddress | copyDestination | In | Flash address of destination super block |
| const struct SEFUserAddressFilter* | filter | In | Pointer to user address filter parameters, null indicates no filtering |
| const struct SEFCopyOverrides* | overrides | In | Pointer to overrides to scheduler parameters; pointer can be null for none required. |
| uint32_t | numAddressChangeRecords | In | Maximum number of ADUs to copy (size of SEFAddressChangeRequest userAddress array) |
| struct SEFAddressChangeRequest* | addressChangeInfo | In | Filled with changed addresses |

Table 12.8: Return value of SEFNamelessCopy

| Type | Description |
|---|---|
| struct SEFStatus | Status and info summarizing result |

Table 12.9: Return values of SEFNamelessCopy

| Error Value | Description |
|---|---|

| | |
|---|---|
| 0 | the info member contains: Source list indication referenced non-closed super blocks (kCopyNonClosedSuperBlock), Destination super block has defective planes (kCopyDestinationDefectivePlanes), Read error was detected on source (kCopyReadErrorOnSource), Data that is out of User Address range is detected (kCopyFilteredUserAddresses), Filled addressChangeInfo array and stopped the copy (kCopyFilledAddressChangeInfo), Destination super block was filled/closed (kCopyClosedDestination), Consumed entire source bitmap or list (kCopyConsumedSource) |
| -ENODEV | The QoS Domain handle is not valid |
| -EPERM | The QoS Domain Handle is not open |
| -EINVAL | The function parameter is not valid; info returns the parameter index that is not valid |

## 12.4 SEFWriteWithoutPhysicalAddressAsync

```
void SEFWriteWithoutPhysicalAddressAsync(SEFQoSHandle qosHandle, struct SEFWrite
```

This function is the asynchronous version of SEFWriteWithoutPhysicalAddress().
A caller allocated IOCB is supplied to initiate the write. The caller can free the IOCB once the kSefIoFlagDone bit is set in common.flags or the common.callback_func has been called. When the IOCB is malformed, the callback_func may be called on the submitters thread.
Note: When the kSefIoFlagCommit flag is set in the IOCB's flag field, the returned tentative addresses will be permanent, potentially adding padding.
Note: Any kAddressUpdate and kSuperBlockStateChange QoS notifications for the returned tentative addresses will occur after the iocb completion routine has returned. When no completion routine is set, the caller must handle the race condition of acting on done being set and the notifications being sent.
*See Also: SEFWriteWithoutPhysicalAddress*

Table 12.10: Parameters of SEFWriteWithoutPhysicalAddressAsync

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFQoSHandle | qosHandle | In | Handle to the QoS Domain |
| struct SEFWriteWithoutPhysicalAddressIOCB* | iocb | In/Out | For asynchronous response from SEF Library. Unused fields should be set to 0. |

## 12.5 SEFReadWithPhysicalAddressAsync

```
void SEFReadWithPhysicalAddressAsync(SEFQoSHandle qosHandle, struct SEFReadWithP
```

This function is the asynchronous version of SEFReadWithPhysicalAddress().
A caller allocated IOCB is supplied to initiate the read. The caller can free the IOCB once the kSefIoFlagDone bit is set in common.flags or the common.callback_func has been called. When the IOCB is malformed, the callback_func may be called on the submitters thread.
*See Also: SEFReadWithPhysicalAddress*

Table 12.11: Parameters of SEFReadWithPhysicalAddressAsync

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFQoSHandle | qosHandle | In | Handle to the QoS Domain |
| struct SEFReadWithPhysicalAddressIOCB* | iocb | In/Out | For asynchronous response from SEF Library Unused fields should be set to 0. |

## 12.6 SEFNamelessCopyAsync

```
void SEFNamelessCopyAsync(SEFQoSHandle qosHandle, struct SEFNamelessCopyIOCB *i
```

This function is the asynchronous version of SEFNamelessCopy().
A caller allocated IOCB is supplied to initiate the copy. The caller can free the IOCB once the kSefIoFlagDone bit is set in common.flags or the common.callback_func has been called. When the IOCB is malformed, the callback_func may be called on the submitters thread.
*See Also: SEFNamelessCopy*

Table 12.12: Parameters of SEFNamelessCopyAsync

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFQoSHandle | qosHandle | In | Handle to the source QoS Domain |
| struct SEFNamelessCopyIOCB* | iocb | In/Out | For asynchronous response from SEF Library Unused fields should be set to 0. |

# 13 Common Structures

## 13.1 SEFUserAddressLbaBits

Number of bits in a user address lba value.

Table 13.1: SEFUserAddressLbaBits

| Name | Definition |
| --- | --- |
| SEFUserAddressLbaBits | 40 |

## 13.2 SEFUserAddressMetaBits

Number of bits in a user address meta value.

Table 13.2: SEFUserAddressMetaBits

| Name | Definition |
| --- | --- |
| SEFUserAddressMetaBits | (64-SEFUserAddressLbaBits) |

## 13.3 SEFAutoAllocate

Flash address value to indicate device should allocate the super block from standard FLASH while doing a write.
*See Also: SEFWriteWithoutPhysicalAddress*

Table 13.3: SEFAutoAllocate

| Name | Definition |
| --- | --- |
| SEFAutoAllocate | ((struct SEFFlashAddress) {UINT64_C(0xffffffffffffffff)}) |

## 13.4 SEFAutoAllocatePSLC

Flash address value to indicate device should allocate the super block from pSLC while doing a write.
*See Also: SEFWriteWithoutPhysicalAddress*

Table 13.4: SEFAutoAllocatePSLC

| Name | Definition |
|---|---|
| SEFAutoAllocatePSLC | ((struct SEFFlashAddress) { htole64(UINT64_C(0xfffffffffffffffe))}) |

## 13.5 SEFUserAddressIgnore

User address value to indicate it should not be validated by the SEF device.
*See Also: SEFReadWithPhysicalAddress*

Table 13.5: SEFUserAddressIgnore

| Name | Definition |
|---|---|
| SEFUserAddressIgnore | ((struct SEFUserAddress) {UINT64_C(0xffffffffffffffff)}) |

## 13.6 SEFNullFlashAddress

Flash address value indicating empty.

Table 13.6: SEFNullFlashAddress

| Name | Definition |
|---|---|
| SEFNullFlashAddress | ((struct SEFFlashAddress){(int64_t)0x0}) |

## 13.7 SEFIsNullFlashAddress

```
static int SEFIsNullFlashAddress(struct SEFFlashAddress flashAddress)
```

Checks whether the flash address is null.

Table 13.7: Parameters of SEFIsNullFlashAddress

| Type | Name | Direction | Description |
|---|---|---|---|

| | | | |
|---|---|---|---|
| struct SEFFlashAddress | flashAddress | In | The opaque address to be checked |

Table 13.8: Return value of SEFIsNullFlashAddress

| Type | Description |
|---|---|
| int | Returns 1 if the flashAddress is null |

## 13.8 SEFIsEqualFlashAddress

```
static int SEFIsEqualFlashAddress(struct SEFFlashAddress flashAddress1, struct S
```

Checks whether two flash addresses are equal.

Table 13.9: Parameters of SEFIsEqualFlashAddress

| Type | Name | Direction | Description |
|---|---|---|---|
| struct SEFFlashAddress | flashAddress1 | In | The opaque address to be compared |
| struct SEFFlashAddress | flashAddress2 | In | The opaque address to be compared |

Table 13.10: Return value of SEFIsEqualFlashAddress

| Type | Description |
|---|---|
| int | Returns 1 if the flashAddress1 equals flashAddress2 |

## 13.9 SEFNextFlashAddress

```
struct SEFFlashAddress SEFNextFlashAddress(SEFQoSHandle qosHandle, struct SEFFla
```

Returns the next flash address by incrementing the ADU Offset.
Doesn't guarantee that the returned flash address is valid

Table 13.11: Parameters of SEFNextFlashAddress

| Type | Name | Direction | Description |
|---|---|---|---|
| SEFQoSHandle | qosHandle | In | Handle to a QoS Domain to interpret/parse the flash address. |
| struct SEFFlashAddress | flashAddress | In | The opaque address to be incremented |

Table 13.12: Return value of SEFNextFlashAddress

| Type | Description |
|---|---|
| struct SEFFlashAddress | Returns the next flash address if the qosHandle is valid, otherwise it returns SEFNullFlashAddress. |

## 13.10 SEFStatus

Table 13.13: Members of SEFStatus

| Type | Name | Description |
|---|---|---|
| int32_t | error | Status information |
| int32_t | info | Additional context-based descriptive information |

## 13.11 SEFVirtualDeviceID

Table 13.14: Members of SEFVirtualDeviceID

| Type | Name |
|---|---|
| uint16_t | id |

## 13.12 SEFQoSDomainID

Table 13.15: Members of SEFQoSDomainID

| Type | Name |
|---|---|
| uint16_t | id |

## 13.13 SEFPlacementID

Table 13.16: Members of SEFPlacementID

| Type | Name |
|---|---|
| uint16_t | id |

## 13.14 SEFADUsize

Table 13.17: Members of SEFADUsize

| Type | Name | Description |
|---|---|---|
| uint32_t | data | ADU data size in bytes |
| uint16_t | meta | ADU meta data size in bytes |
| uint16_t | reserved | Reserved/unused |

## 13.15 SEFInfo

Table 13.18: Members of SEFInfo

| Type | Name | Description |
|---|---|---|
| const char * | name | Device Name from O/S |
| char[8] | vendor | Vendor field |
| char[20] | serialNumber | Device serial number |
| char[8] | FWVersion | Device firmware version |
| char[8] | HWVersion | Device hardware version |
| uint16_t | unitNumber | Unit number of the SEFInfo struct |
| uint16_t | APIVersion | API Version |
| uint64_t | supportedOptions | Supported features - see kSupported defines |
| uint32_t | maxOpenSuperBlocks | Firmware version specific, max number of open super blocks for the device. When 0, the limit is per Virtual Device instead. SEFVirtualDeviceInfo |
| uint16_t | maxQoSDomains | Hardware version specific, may be less than 65535 defined by architecture |
| uint16_t | maxRootPointers | Firmware version specific, may be less than 8 defined by architecture |
| uint16_t | maxPlacementIDs | Firmware version specific, max number of auto opened super blocks per QoS Domain |
| uint16_t | reserved_0 | Reserved/unused |
| uint16_t | numReadQueues | Firmware version specific, max number of read queues total |
| uint16_t | numVirtualDevices | Number of currently defined virtual devices |
| uint16_t | numQoSDomains | Number of currently defined QoS Domains |
| uint16_t | numBanks | Number of banks per channel |
| uint16_t | numChannels | Number of channels per SEF Unit |
| uint16_t | numPlanes | Number of planes per die |
| uint32_t | pageSize | Physical page size |
| uint32_t | numPages | Number of pages per block |
| uint32_t | numBlocks | Number of blocks per die |

| | | |
|---|---|---|
| uint32_t | totalBandWidth | Total bandwidth in MiBs corresponding to the underlying flash component on this device |
| uint32_t | readTime | Read time in microseconds corresponding to the underlying flash components on this device |
| uint32_t | programTime | Program time in microseconds corresponding to the underlying flash components on this device |
| uint32_t | eraseTime | Erase time in microseconds corresponding to the underlying flash components on this device |
| uint16_t | minReadWeight | Advisory minimum read weight to allow timely house keeping internal I/O |
| uint16_t | minWriteWeight | Advisory minimum write weight to allow timely house keeping internal I/O |
| uint32_t | openExpirationPeriod | Granularity in seconds for entire block |
| uint16_t | reserved_1 | Reserved/unused |
| uint16_t | numADUSizes | Size of ADUsize array that follows at end of structure |
| struct SEFADUsize[] | ADUsize | Array of supported ADU sizes |

## 13.16 SEFVirtualDeviceList

Table 13.19: Members of SEFVirtualDeviceList

| Type | Name | Description |
|---|---|---|
| uint16_t | numVirtualDevices | Number of virtual devices |
| struct SEFVirtualDeviceID[] | virtualDeviceID | An Array of all Virtual device IDs |

## 13.17 SEFQoSDomainList

Table 13.20: Members of SEFQoSDomainList

| Type | Name | Description |
|---|---|---|
| uint16_t | numQoSDomains | Number of QoS domains |
| struct SEFQoSDomainID[] | QoSDomainID | An Array of all QoS Domain IDs |

## 13.18 SEFUserAddress

Structure of SEFUserAddress may be redefined by user.

The limitations for redefining the structure are:

- size must be 8 bytes
- multi-adu writes will auto increment the LBA value and must not equal SEFUserAddressIgnore. However SEFUserAddressIgnore is supported as a starting user address.

For kSuperBlock, the LBA is limited to 40 bits and the meta to 24. The unformatted member is in little endian format.

Table 13.21: Members of SEFUserAddress

| Type | Name |
|---|---|
| uint64_t | unformatted |

## 13.19 SEFFlashAddress

Opaque flash address value parsable by SEFParseFlashAddress()

Table 13.22: Members of SEFFlashAddress

| Type | Name |
|---|---|
| uint64_t | bits |

## 13.20 SEFDieList

Table 13.23: Members of SEFDieList

| Type | Name | Description |
|---|---|---|
| uint16_t | numDies | Number of dies in dieIDs |
| uint16_t[] | dieIDs | List of dies by ID |

## 13.21 SEFWeights

Relative die time weights for write type of I/O operations.

Table 13.24: Members of SEFWeights

| Type | Name | Description |
|---|---|---|
| uint16_t | programWeight | Default weight for a program operation by the Nameless Write and Nameless Copy commands |

| | | |
|---|---|---|
| uint16_t | eraseWeight | Default weight for an erase operation by SEFAllocateSuperBlock, SEFFlushSuperBlock and SEFCloseSuperBlock |

## 13.22 SEFVirtualDeviceConfig

Table 13.25: Members of SEFVirtualDeviceConfig

| Type | Name | Description |
|---|---|---|
| struct SEFVirtualDeviceID | virtualDeviceID | Virtual Device ID |
| uint8_t | numReadQueues | Number of read queues to define for this Virtual Device |
| uint8_t | reserved | Reserved, must be initialized to zero |
| uint16_t[SEFMaxReadQueues] | readWeights | Default weight for read operations for each possible read queue |
| uint16_t | superBlockDies | Number of dies in a super block, 0 uses dieList.numDies |
| struct SEFDieList | dieList | List of dies in ascending order for the Virtual Device |

## 13.23 SEFVirtualDeviceUsage

Table 13.26: Members of SEFVirtualDeviceUsage

| Type | Name | Description |
|---|---|---|
| uint32_t | eraseCount | Count of super blocks erased. Used to populate eraseOrder in SEFSuperBlockInfo |
| uint32_t | numUnallocatedSuperBlocks | Number of unallocated super blocks |
| uint32_t | numSuperBlocks | Number of allocated super blocks |
| uint32_t | numUnallocatedPSLCSuperBlocks | Number of unallocated pSLC super blocks |
| uint32_t | numPSLCSuperBlocks | Number of allocated pSLC super blocks |
| struct SEFVirtualDeviceID | vdID | Virtual device ID of the handle |
| uint8_t | averagePEcount | Average program/erase count |
| uint8_t | maxPEcount | Max program/erase count |

| | | |
|---|---|---|
| uint16_t | patrolCycleTime | Recommended Patrol Cycle in minutes |
| uint16_t | reserved | Reserved, must be initialized to zero |

## 13.24 SEFVirtualDeviceSuspendConfig

Configuration for Erase/Program suspend.
The weights supplied with i/o represents virtual time. These parameters control how often, and for how long an erase/program can be interrupted by reads.

Table 13.27: Members of SEFVirtualDeviceSuspendConfig

| Type | Name |
|---|---|
| uint32_t | maxTimePerSuspend |
| uint32_t | minTimeUntilSuspend |
| uint32_t | maxSuspendInterval |

## 13.25 SEFVirtualDeviceInfo

Table 13.28: Members of SEFVirtualDeviceInfo

| Type | Name | Description |
|---|---|---|
| uint64_t | flashCapacity | Flash capacity in 4k ADUs |
| uint64_t | flashAvailable | Available flash capacity in 4k ADUs |
| uint64_t | pSLCFlashCapacity | pSLC Flash capacity in 4k ADUs |
| uint64_t | pSLCFlashAvailable | pSLC Available flash capacity in 4k ADUs |
| uint32_t | superBlockCapacity | Super block capacity in 4k ADUs |
| uint32_t | pSLCSuperBlockCapacity | pSLC super block capacity in 4k ADUs |
| uint32_t | maxOpenSuperBlocks | Maximum number of open super blocks per Virtual Device. When 0, the limit is per device instead. See SEFInfo |

| | | |
|---|---|---|
| uint32_t | numPSLCSuperBLocks | Number of pSLC super blocks |
| struct SEFVirtualDeviceSuspendConfig | suspendConfig | |
| uint16_t | superBlockDies | Number of dies used for a super block |
| uint8_t | aduOffsetBitWidth | Number of bits that make up the adu offset in a flash address |
| uint8_t | superBlockIdBitWidth | Number of bits that make up the super block id in a flash address |
| uint16_t[SEFMaxReadQueues] | readWeights | Default weight for read operations for each possible read queue |
| uint8_t | numReadQueues | Number of read queues defined for the Virtual Device |
| uint8_t[5] | reserved | Reserved, must be initialized to zero |
| struct SEFQoSDomainList | QoSDomains | List of domains |

## 13.26 SEFQoSDomainCapacity

Table 13.29: Members of SEFQoSDomainCapacity

| Type | Name | Description |
|---|---|---|
| uint64_t | flashCapacity | Number of ADUs to assign to a QoS Domain |
| uint64_t | flashQuota | Maximum number of ADUs allowed for a QoS Domain |

## 13.27 SEFSuperBlockInfo

Table 13.30: Members of SEFSuperBlockInfo

| Type | Name | Description |
|---|---|---|
| struct SEFFlashAddress | flashAddress | Flash address where this super block resides |
| uint32_t | eraseOrder | Indication of when a super block was erased. Can be used to determine the order blocks were allocated or to version a super block. Values only increase over time and are unique at the Virtual Device level |

| | | |
|---|---|---|
| uint32_t | writableADUs | For a fresh, unwritten, open super block, this how much QoS Domain quota is being used by the super block. It will decrease if defects are encountered while writing |
| uint32_t | writtenADUs | This field increases as ADUs in the super block are written. For kPerfect and kPacked, it will equal writableADUs when the block is closed. For kFragmented, it will equal super block capacity because it includes defective portions of the flash |
| struct SEFPlacementID | placementID | When auto-allocated, indicates the placement id supplied to SEFWriteWithoutPhysicalAddress(). Otherwise it will be SEFPlacementIdUnused |
| uint16_t | numDefects | Number of defective planes per super page. This may increase as the super block is written |
| uint16_t | timeLeft | Time in minutes left to handle an integrity that is not kIntegrityGood before risking data loss |
| uint8_t | PEIndex | This is the block's erase count normalized to be between 0 and 255 |
| enum SEFSuperBlockType | type | This is the type of the super block, normal or pSLC |
| enum SEFSuperBlockState | state | This is the block's current state |
| enum SEFDataIntegrity | integrity | This is the integrity of the super block, If not kIntegrityGood, the super block requires a SEFCheckSuperBlock to patrol or refresh |
| uint8_t[] | defects | This is a bitmap indicating which planes are defective. SEFQoSDomainInfo::defectMapSize is the size of this field. |

## 13.28 SEFSuperBlockRecord

Entry in a SEFSuperBlockList.

Table 13.31: Members of SEFSuperBlockRecord

| Type | Name | Description |
|---|---|---|
| struct SEFFlashAddress | flashAddress | Flash address where this super block resides |
| uint8_t[6] | reserved | Reserved |
| uint8_t | PEIndex | This is the block's erase count normalized to be between 0 and 255 |
| enum SEFSuperBlockState | state | This is the block's current state |

## 13.29 SEFSuperBlockList

Table 13.32: Members of SEFSuperBlockList

| Type | Name | Description |
|---|---|---|
| uint32_t | numSuperBlocks | Number of super blocks in use by the QoS Domain |
| uint32_t | reserved | Reserved |
| struct SEFSuperBlockRecord[] | superBlockRecords | List of super block records |

## 13.30 SEFQoSDomainInfo

Table 13.33: Members of SEFQoSDomainInfo

| Type | Name | Description |
|---|---|---|
| struct SEFVirtualDeviceID | virtualDeviceID | Virtual device ID |
| uint16_t | numPlacementIDs | Specifies the number of Placement IDs corresponding to this QoS Domain |
| uint8_t | encryption | 0 for disabled, non-zero for enabled |
| enum SEFErrorRecoveryMode | recoveryMode | Specifies the recovery mode for this QoS Domain |
| enum SEFDefectManagementMethod | defectStrategy | Defect management strategy for the QoS Domain |
| enum SEFAPIIdentifier | api | Specifies the API Identifier for this QoS Domain |
| uint64_t | flashCapacity | Reserved capacity of the QoS Domain in QoS Domain ADUs |
| uint64_t | flashQuota | Number of QoS Domain ADUs that can be allocated by the QoS Domain |

| | | |
|---|---|---|
| uint64_t | flashUsage | Number of QoS Domain ADUs allocated by the QoS Domain |
| uint64_t | pSLCFlashCapacity | Reserved pSLC capacity of the QoS Domain in QoS Domain ADUs |
| uint64_t | pSLCFlashQuota | Number of pSLC QoS Domain ADUs that can be allocated by the QoS Domain |
| uint64_t | pSLCFlashUsage | Number of pSLC QoS Domain ADUs allocated by the QoS Domain |
| struct SEFFlashAddress[SEFMaxRootPointer] | rootPointers | List of root pointers |
| struct SEFADUsize | ADUsize | Size of QoS Domain ADUs, data and metadata in bytes |
| uint32_t | superBlockCapacity | Super block capacity in QoS Domain ADUs |
| uint32_t | pSLCSuperBlockCapacity | pSLC super block capacity in QoS Domain ADUs |
| uint16_t | maxOpenSuperBlocks | Maximum number of open super blocks for the QoS Domain |
| uint16_t | defectMapSize | Size of a super block defect map |
| struct SEFWeights | weights | Default i/o weights for erase and program |
| enum SEFDeadlineType | deadline | Deadline type for the QoS Domain |
| uint8_t | defaultReadQueue | Default read queue assignment |
| uint8_t | numReadQueues | Number of read queues as defined by the Virtual Device |
| uint8_t[5] | reserved | Reserved |

## 13.31 SEFWearInfo

Table 13.34: Members of SEFWearInfo

| Type | Name | Description |
|---|---|---|
| uint32_t | numSuperBlocks | Number of super blocks |
| uint32_t | reserved_0 | Reserved, must be initialized to zero |
| struct SEFSuperBlockRecord[] | superBlockRecords | List of super block records |

## 13.32 SEFRefreshInfo

Table 13.35: Members of SEFRefreshInfo

| Type | Name | Description |
|---|---|---|
| uint32_t | numSuperBlocks | Number of super blocks |
| uint32_t | reserved_0 | Reserved, must be initialized to zero |
| struct SEFSuperBlockRecord[] | superBlockRecords | List of super block records |

## 13.33 SEFCheckInfo

SuperBlocks returned by SEFGetCheckList()

Table 13.36: Members of SEFCheckInfo

| Type | Name | Description |
|---|---|---|
| uint32_t | numSuperBlocks | Number of super blocks |
| uint32_t | reserved_0 | Reserved, must be initialized to zero |
| struct SEFSuperBlockRecord[] | superBlockRecords | List of super block records |

## 13.34 SEFUserAddressList

Table 13.37: Members of SEFUserAddressList

| Type | Name | Description |
|---|---|---|
| uint32_t | numADUs | Number of ADUs |
| uint32_t | reserved_0 | Reserved, must be initialized to zero |
| struct SEFUserAddress[] | userAddressesRecovery | User addresses |

## 13.35 SEFProperty

Table 13.38: Members of SEFProperty

| Type | Name | Description |
|---|---|---|
| union | | 5 Members |
| ¬ int | intVal | Valid when type is kSefPropertyTypeInt |
| ¬ void * | ptr | Valid when type is kSefPropertyTypePtr |
| ¬ struct SEFQoSDomainID | qosID | Valid when type is kSefPropertyTypeQoSDomainID |
| ¬ struct SEFVirtualDeviceID | vdID | Valid when type is kSefPropertyTypeVirtualDeviceID |
| ¬ void(*)(void *, struct SEFQoSNotification) | qosNotify | Valid when type is kSefPropertyTypeQoSNotify |
| enum SEFPropertyType | type | Denotes the property type |

## 13.36 SEFWriteOverrides

Supplied to override default write weights.
May be used when calling SEFWriteWithoutPhysicalAddress() or SEFWriteWithoutPhysicalAddressAsync().

Table 13.39: Members of SEFWriteOverrides

| Type | Name | Description |
|---|---|---|
| uint16_t | programWeight | Weight to use for program instead of the QoS domain default. 0 will use the QoS Domain default. |
| uint16_t | eraseWeight | Weight to use for erase instead of the QoS domain default. 0 will use the QoS Domain default. |

## 13.37 SEFReadOverrides

Supplied to override default read weight.
May be used when calling SEFReadWithPhysicalAddress() or SEFReadWithPhysicalAddressAsync().

Table 13.40: Members of SEFReadOverrides

| Type | Name | Description |
|---|---|---|

| | | |
|---|---|---|
| uint16_t | readWeight | Weight to use for read instead of the read queue's default. 0 will use the read queue's default. |
| uint8_t | readQueue | Read queue to use for read instead of QoS Domain's default. A value greater than number of read queues defined for the QoS Domain will use the default read queue for the QoS Domain. |
| uint8_t | reserved | Reserved, must be initialized to zero. |

## 13.38 SEFAllocateOverrides

Supplied to override default super block allocation weight.
May be used when calling SEFAllocateSuperBlock() or SEFAllocateSuperBlockAsync().

Table 13.41: Members of SEFAllocateOverrides

| Type | Name | Description |
|---|---|---|
| uint16_t | eraseWeight | Weight to use for erase instead of the QoS Domain default. 0 will use the QoS Domain default. |

## 13.39 SEFCopySource

Source addresses for SEFNamelessCopy().
The Source addresses format controls if the validBitmap or list of flash addresses is used.SEFNamelessCopy()SEFUserAddressFilter

Table 13.42: Members of SEFCopySource

| Type | Name | Description |
|---|---|---|
| enum SEFCopySourceType | format | Specifies the format to use |
| uint8_t[3] | reserved_0 | Reserved, must be initialized to zero |
| uint32_t | arraySize | Number of items in bitmap array or Flash Address List (QWORD count) |
| union | | 2 Members |
| ¬ const struct SEFFlashAddress* | flashAddressList | pointer to flash address list |
| ¬ struct | | 2 Members |
| ¬ ¬ struct SEFFlashAddress | srcFlashAddress | flash address of source block. ADU & 0x3f indicates the ADU of bit 0 of validBitmap and ADU & 0x3f is the starting bit in validBitMap |

| | | |
|---|---|---|
| ¬ ¬ const uint64_t * | validBitmap | pointer to COPY of valid bitmap array (little endian) |

## 13.40 SEFUserAddressFilter

Optional filtering on user address data during copy.

Table 13.43: Members of SEFUserAddressFilter

| Type | Name | Description |
|---|---|---|
| struct SEFUserAddress | userAddressStart | Starting user address of filter |
| uint64_t | userAddressRangeLength | Length of filter range (0 indicates no filtering) |
| uint32_t | userAddressRangeType | Zero to copy data in range; non-zero to copy outside of range |

## 13.41 SEFAddressChangeRequest

Detailed information about results of the SEFNamelessCopy() request.

Table 13.44: Members of SEFAddressChangeRequest

| Type | Name | Description |
|---|---|---|
| uint32_t | numProcessedADUs | The number of processed ADUs including errors |
| uint32_t | nextADUOffset | Given a bitmap source, it indicates the next ADU offset of the source flash address; Given a list source, it indicates the next entry number in the source flash address list |
| uint32_t | numReadErrorADUs | The number of ADUs that couldn't be processed due to errors |
| uint32_t | numADUsLeft | The number of remaining ADUs in the destination super block |
| uint8_t | copyStatus | A bit array denoting the results of the request |
| uint8_t[7] | reserved | Reserved, must be initialized to zero |
| struct [] | addressUpdate | 3 Members; An array of information about copied ADUs |
| ¬ struct SEFUserAddress | userAddress | The user address |
| ¬ struct SEFFlashAddress | oldFlashAddress | The old flash address |

| | | |
|---|---|---|
| ¬ struct SEFFlashAddress | newFlashAddress | The new flash address |

## 13.42 SEFCopyOverrides

Scheduling Queue overrides for SEFNamelessCopy()

Table 13.45: Members of SEFCopyOverrides

| Type | Name | Description |
|---|---|---|
| uint16_t | programWeight | Weight to use for program instead of the QoS domain default. 0 will use the QoS Domain default |

# 14 | Callback Structures

## 14.1 SEFCommonIOCB

Table 14.1: Members of SEFCommonIOCB

| Type | Name | Direction | Description |
|---|---|---|---|
| struct SEFStatus | status | Out | Library sets error field to a non-zero value to indicate any error when a command completes |
| int16_t | opcode | In | Should never be accessed - for internal use by library |
| int16_t | flags | In/Out | SEFIOCBFlags |
| int32_t | reserved | In | Reserved, must be initialized to zero |
| void * | param1 | In | Ignored by the library; the caller can store context information that may be accessed from the completion function |
| void(*)(struct SEFCommonIOCB *) | complete_func | In | If non-zero, treated as the address of a function to be called when a command completes |

## 14.2 SEFWriteWithoutPhysicalAddressIOCB

Table 14.2: Members of SEFWriteWithoutPhysicalAddressIOCB

| Type | Name | Direction | Description |
|---|---|---|---|
| struct SEFCommonIOCB | common | In/Out | Common fields for all IOCBs |

| | | | |
|---|---|---|---|
| struct SEFFlashAddress | flashAddress | In | Address of the super block for this write; -1 for auto-allocate, or can use value from previous super block allocation call |
| struct SEFUserAddress | userAddress | In | Contains LBA information |
| struct SEFFlashAddress* | tentativeAddresses | Out | List of tentative physical addresses return |
| const void * | metadata | In | Metadata to write with data; The number of bytes per ADU required is SEFQoSDomainInfo::ADUsize.meta. May be NULL |
| const struct iovec* | iov | In | A pointer to the scatter gather list |
| uint16_t | iovcnt | In | The number of elements in the scatter gather list |
| struct SEFPlacementID | placementID | In | Only valid if the flashAddress is auto allocated. A value from 0 to numPlacementIds – 1 indicating what logical data group to place this data in |
| uint32_t | numADU | In | Length in QoS Domain ADUs |
| uint32_t | distanceToEndOfSuperBlock | Out | Return value in units of ADUs |
| struct SEFWriteOverrides | overrides | In | Override parameters for scheduling purposes. Must set kSefIoFlagOverride in flags to apply |

## 14.3 SEFReadWithPhysicalAddressIOCB

Table 14.3: Members of SEFReadWithPhysicalAddressIOCB

| Type | Name | Direction | Description |
|---|---|---|---|
| struct SEFCommonIOCB | common | In/Out | Common fields for all IOCBs |

| | | | |
|---|---|---|---|
| struct SEFFlashAddress | flashAddress | In | Physical address for the read command; When the QoS Domain ID and block number are 0, the ADU offset is the root pointer index for the flash address to read. |
| struct SEFUserAddress | userAddress | In | Contains LBA information |
| const struct iovec* | iov | In | A pointer to the scatter gather list |
| void * | metadata | In | Receives ADU metadata; The number of bytes per ADU required is SEFQoSDomainInfo::ADUsize.meta. May be NULL |
| size_t | iovOffset | In | Starting byte offset into iov array |
| uint32_t | numADU | In | Number of ADUs to be read, maximum is superBlockCapacity |
| uint16_t | iovcnt | In | The number of elements in the scatter gather list |
| struct SEFReadOverrides | overrides | In | Override parameters for scheduling purposes. Must set kSefIoFlagOverride in flags to apply |
| uint16_t[3] | reserved | In | reserved, must be initialized to 0 |

## 14.4 SEFReleaseSuperBlockIOCB

Table 14.4: Members of SEFReleaseSuperBlockIOCB

| Type | Name | Direction | Description |
|---|---|---|---|
| struct SEFCommonIOCB | common | In/Out | Common fields for all IOCBs |
| struct SEFFlashAddress | flashAddress | In | Address of super block |

## 14.5 SEFAllocateSuperBlockIOCB

IOCB for SEFAllocateSuperBlockAsync()

Table 14.5: Members of SEFAllocateSuperBlockIOCB

| Type | Name | Direction | Description |
|---|---|---|---|
| struct SEFCommonIOCB | common | In/Out | Common fields for all IOCBs |
| struct SEFFlashAddress | flashAddress | Out | Address of super block |

| | | | |
|---|---|---|---|
| uint8_t * | defectMap | In | Optional buffer to receive the block's defect map. Used for kFragmented QoS Domains. When supplied, the buffer must be at least as large as SEFQoSDomainInfo::defectMapSize. |
| struct SEFAllocateOverrides | overrides | In | Override parameters for scheduling purposes. Must set kSefIoFlagOverride in flags to apply |
| enum SEFSuperBlockType | type | In | kForWrite or kForPSLCWrite |

## 14.6 SEFCloseSuperBlockIOCB

IOCB for SEFCloseSuperBlockAsync()

Table 14.6: Members of SEFCloseSuperBlockIOCB

| Type | Name | Direction | Description |
|---|---|---|---|
| struct SEFCommonIOCB | common | In/Out | Common fields for all IOCBs |
| struct SEFFlashAddress | flashAddress | In | Address of the super block |

## 14.7 SEFNamelessCopyIOCB

Table 14.7: Members of SEFNamelessCopyIOCB

| Type | Name | Direction | Description |
|---|---|---|---|
| struct SEFCommonIOCB | common | In/Out | Common fields for all IOCBs |
| SEFQoSHandle | dstQosHandle | In | Handle to the destination QoS Domain |
| struct SEFFlashAddress | copyDestination | In | Flash address of destination super block |
| uint32_t | reserved_0 | In | Reserved, must be initialized to zero |

| | | | |
|---|---|---|---|
| uint32_t | numAddressChangeRecords | In | Maximum number of ADUs to copy (size of addressChangeRequest userAddress array) |
| struct SEFAddressChangeRequest* | addressChangeInfo | Out | Output of changed addresses |
| struct SEFCopySource | copySource | In | Physical addresses to copy |
| const struct SEFUserAddressFilter* | filter | In | Pointer to user address filter parameters, null for no filtering |
| struct SEFCopyOverrides | overrides | In | Override parameters for scheduling purposes. Must set kSefIoFlagOverride in flags to apply |

# 15 | Events

## 15.1 SEFQoSNotification

This event is issued at the QoS Domain level.

Table 15.1: Members of SEFQoSNotification

| Type | Name | Description |
|---|---|---|
| enum SEFNotificationType | type | See union below... |
| uint8_t[5] | reserved_0 | Reserved, must be initialized to zero |
| struct SEFQoSDomainID | QoSDomainID | QoSDomainID for this notification |
| union | | 7 Members |
| ¬ struct SEFFlashAddress | maintenanceFlashAddress | Valid when type is kRequireMaintenance |
| ¬ struct | | 3 Members |
| ¬ ¬ struct SEFUserAddress | changedUserAddress | User address that moved |
| ¬ ¬ struct SEFFlashAddress | oldFlashAddress | Old flash address |
| ¬ ¬ struct SEFFlashAddress | newFlashAddress | New flash address |
| ¬ struct SEFFlashAddress | patrolFlashAddress | Valid when type is kRequirePatrol |
| ¬ struct | | 2 Members |
| ¬ ¬ struct SEFUserAddress | unflushedUserAddress | Affected user address |
| ¬ ¬ char * | userData | Pointer to buffered data |
| ¬ struct SEFFlashAddress | unreadableFlashAddress | Valid when type is kUnreadable |
| ¬ struct SEFFlashAddress | changedFlashAddress | Valid when type is kSuperBlockStateChanged (open=>closed) |
| ¬ uint32_t | writtenADUs | Number of ADUs written by user i/o commands to the super block |
| ¬ uint32_t | numADUs | Capacity of the super block in ADUs |
| ¬ struct | | 2 Members |
| ¬ ¬ const struct iovec* | iov | A pointer to the scatter gather list |

| | | |
|---|---|---|
| ¬ ¬ int16_t | iovcnt | The number of elements in the scatter gather list |

## 15.2 SEFVDNotification

This event indicates to the host that it should respond in some appropriate manner to the reduced capacity condition.

This event is issued at the Virtual Device level. Due to failure of blocks, actual available capacity may fall below the allocated capacity of the attached QoS Domains. The host should take action to release super blocks back to the Virtual Device's free pool before it is entirely consumed.

Table 15.2: Members of SEFVDNotification

| Type | Name | Description |
|---|---|---|
| enum SEFNotificationType | type | Is kReducedCapacity, kOutOfCapacity or kOutOfPSLCCapacity |
| uint8_t | reserved_0 | Reserved, must be initialized to zero |
| struct SEFVirtualDeviceID | virtualDeviceID | Virtual Device for this notification |
| uint32_t | numADUs | kReducedCapacity - Amount of space that is no longer available |

# 16 | Enumerated Types

## 16.1 SEFDefectManagementMethod

Table 16.1: Members of SEFDefectManagementMethod

| Member | Description |
|---|---|
| kPacked | Offset address in a super block is consecutive. Size of super block is reduced with defected block(s). This results in slower reads due to the extra level of indirection incurred. |
| kFragmented | Defective blocks are left in place, and are simply marked as non-addressable. Over time, this can result in a device with a gradually decreasing usable size. This scheme has the fastest read performance, but comes at the cost of additional management complexity that the host will be responsible for. |
| kPerfect | Offset address is consecutive. Size of super block is fixed. Number of super blocks is reduced with defected block(s). This has the slowest read performance because this remapping has the potential to cross block boundaries |

## 16.2 SEFAPIIdentifier

Table 16.2: Members of SEFAPIIdentifier

| Member | Description |
|---|---|
| kSuperBlock | Currently the only mode supported by the API |
| kInDriveGC | Reserved for future use |
| kVirtualSSD | Reserved for future use |

## 16.3 SEFErrorRecoveryMode

Table 16.3: Members of SEFErrorRecoveryMode

| Member | Description |
|---|---|

| | |
|---|---|
| kAutomatic | Automatic recovery mode |
| kHostControlled | Host is responsible for recovery |

## 16.4 SEFDeadlineType

Table 16.4: Members of SEFDeadlineType

| Member | Description |
|---|---|
| kFastest | Does not attempt a corrective action, but instead sends a notification to allow higher layer to read from a separate redundant store. |
| kTypical | Attempts to perform basic error recovery in the event of a read error condition |
| kLong | Attempts to perform more advanced error recovery in the event of a read error condition |
| kHeroic | Attempts to perform full recovery in the event of a read error condition |

## 16.5 SEFNotificationType

Asynchronous notifications from SEF.

Table 16.5: Members of SEFNotificationType

| Member | Description |
|---|---|
| kAddressUpdate | The flash address has changed |
| kUnflushedData | The super block data was flushed to the Flash Memory |
| kRequirePatrol | The super block requires to be patrolled; A list of super blocks requiring patrol can be retrieved using SEFGetCheckList <linebreak /> |
| kRequireMaintenance | The super block requires maintenance; In other words, the data should be copied off and the super block should be freed |
| kReducedCapacity | The Virtual Device's capacity has been reduced |
| kUnreadableData | The data stored at the flash address cannot be read |
| kSuperBlockStateChanged | The super block's state has changed |
| kOutOfCapacity | The Virtual Device is full |
| kOutOfPSLCCapacity | The Virtual Device is out of pSLC |
| kBufferRelease | The buffer pointed to by iov can be freed |

## 16.6 SEFSuperBlockType

Table 16.6: Members of SEFSuperBlockType

| Member | Description |
|---|---|
| kForWrite | Super block is for writes |
| kForPSLCWrite | Super block is for pSLC writes |

## 16.7 SEFSuperBlockState

Table 16.7: Members of SEFSuperBlockState

| Member | Description |
|---|---|
| kSuperBlockClosed | This is the state of super blocks which retain effective data after all super pages have been programmed |
| kSuperBlockOpenedByErase | This is the state of super blocks in the middle of being programmed and were allocated by SEFAllocateSuperBlock() |
| kSuperBlockOpenedByPlacementId | This is the state of super blocks in the middle of being programmed and were allocated automatically by placement id |

## 16.8 SEFDataIntegrity

Integrity of a super block.

Table 16.8: Members of SEFDataIntegrity

| Member | Description |
|---|---|
| kSefIntegretyUnknown | The block needs to be patrolled |
| kSefIntegretyGood | Reading the block requires little to no error correction |
| kSefIntegretyAllowable | Reading the block requires an acceptable amount of error correction |
| kSefIntegretyMarginal | The data in the block needs to be relocated |

## 16.9 SEFPropertyID

Table 16.9: Members of SEFPropertyID

| Member | Description |
|---|---|
| kSefPropertyQoSDomainID | Get QoS Domain ID in qosID |
| kSefPropertyVirtualDeviceID | Get Virtual Device ID as vdID |
| kSefPropertyUnitNumber | Get Unit number as intVal |
| kSefPropertyQoSNotify | Get QoS notification fnc as qosNotify |

| | |
|---|---|
| kSefPropertyPrivateData | Get/Set private data |
| kSefPropertyNumActiveRequests | Get Number of active requests as intVal |

## 16.10 SEFPropertyType

Table 16.10: Members of SEFPropertyType

| Member | Description |
|---|---|
| kSefPropertyTypeInvalid | The SEFPropertyID is not supported |
| kSefPropertyTypeNull | The Property has no value (not set) |
| kSefPropertyTypeInt | The intVal member is valid |
| kSefPropertyTypePtr | The ptr member is valid |
| kSefPropertyTypeQoSDomainID | The qosID member is valid |
| kSefPropertyTypeVirtualDeviceID | The vdID member is valid |
| kSefPropertyTypeQoSNotify | The qosNotify member is valid |

## 16.11 SEFCopySourceType

The source format to be used when copying a super block.

Table 16.11: Members of SEFCopySourceType

| Member | Description |
|---|---|
| kBitmap | Use validBitmap as the copy source |
| kList | Use flashAddressList as the copy source |

## 16.12 SEFIOCBFlags

Table 16.12: Members of SEFIOCBFlags

| Member | Description |
|---|---|
| kSefIoFlagDone | Flag for polled I/O - library sets this bit to a 1 value once the command completes |
| kSefIoFlagNotifyBufferRelease | Flag set to indicate caller is managing buffer lifetime. See Also: SEFWriteWithoutPhysicalAddress() |
| kSefIoFlagCommit | Flag set to force data to flash before completing, potentially adding padding |
| kSefIoFlagOverride | Flag set to apply weight override to i/o |