Software-Enabled Flash™
API Specification

Memory Storage Strategy Division

API Version: 1.10

Contents

1 Revision History

Version	Date	Description of change(s)
1.10	08/17/2020	Initial version of the document

2 | Introduction

This specification describes the core components of the Software-Enabled Flash™ (SEF) application programming interface.

The SEF API provides a simplified interface which abstracts away details of low-level flash memory device mechanics in such a way that allows hosts to interact with flash memory 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 hardware Units. SEF Units are PCIe® based NVMe™ devices, with certain SEF specific extensions to the NVMe command set. 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
- Providing mechanism to allow control over housekeeping functions to support predictable latency
- Offloading computational burden from hosts via powerful API primitives
- Extension of flash memory lifetime due to intelligent automatic resource allocation

Figure 2.1 illustrates where the SEF library is located in the context of a traditional SSD-like application stack. The SEF Library accepts I/O requests from a host-defined FTL, and issues a set of commands. The SEF Unit translates this down to an appropriate set of flash memory-level operations.

Figure 2.1: SEF Library location in the system layer

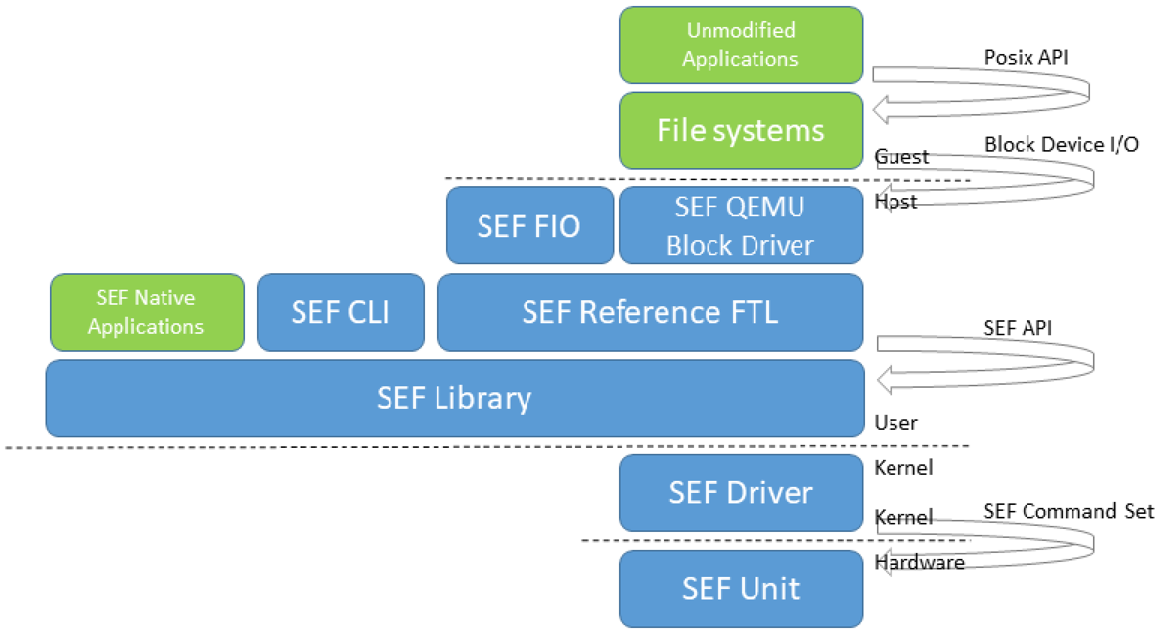
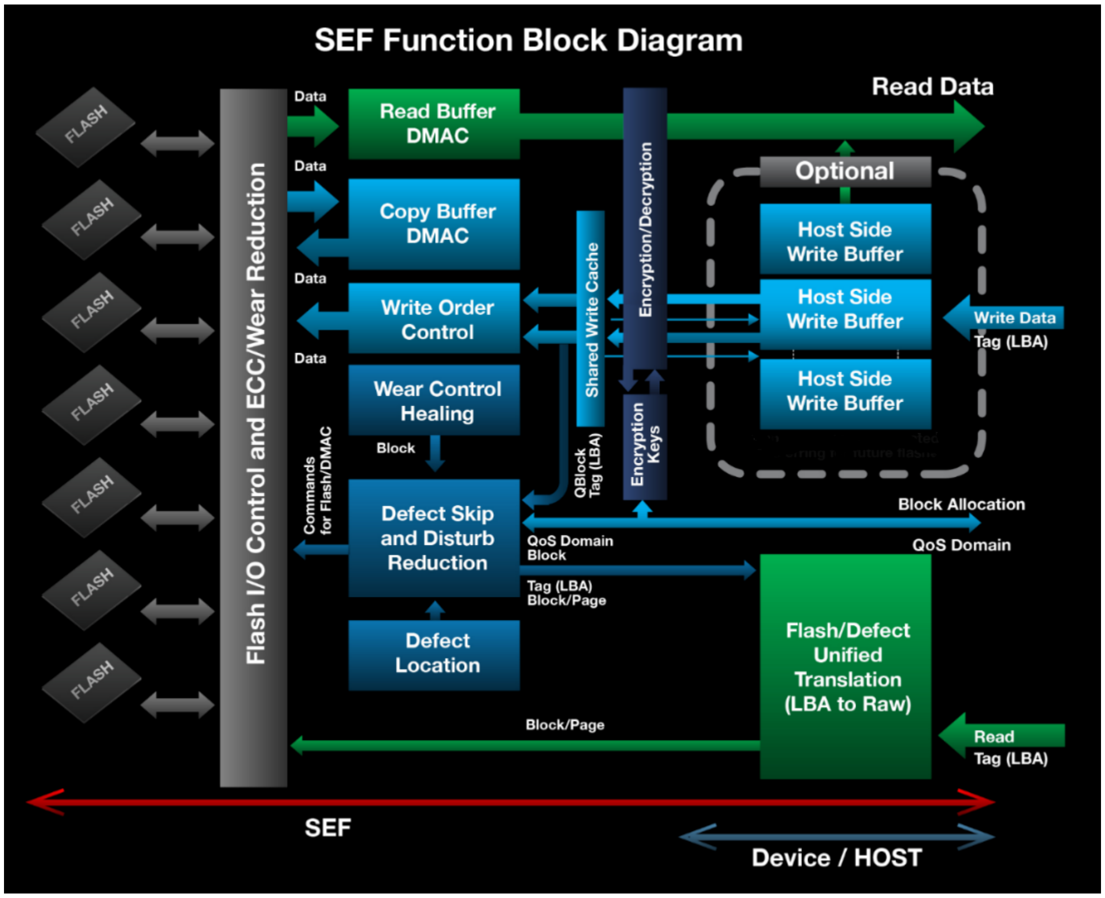


Figure 2.2 provides a detailed view of the system interface provided by the SEF API. SEF handles functionality including super block allocation, identifying and working around defective blocks, low-level flash memory I/O, scheduling, prioritization and other device-level concerns. The host layer in turn 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.

Figure 2.2: SEF Block Diagram



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.
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 some extensions.
Flash Translation Layer (FTL)	A mapping of Logical Block Addresses (LBA) to flash memory addresses providing a hard disk-like API on top of a flash memory API.
Virtual Device (VD)	A set of flash memory dies. Occupies one or more flash memory dies, 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. *Refer to Chapter 7 for more information*
QoS Domain (QD)	A logical construct exposed to the host and enumerated as a SEF device 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. *Refer to Chapter 8 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. *Refer to Chapter 10 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 quanta (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.
FMQ	Flash memory media queues allow for the control of I/O scheduling in a virtual device. *Refer to Chapter 7 for more information*
Root Pointer	Provides a bootstrapping mechanism to retrieve metadata from a QoS domain.

4 | Design Environment

The SEF library runs on a Linux® host. It supports user mode or kernel mode. The library and driver do not yet support forked processes. The SEF library API is defined by SEFAPI.h and implemented in libsef.a. It is platform-agnostic and is usable by any code that can use a C interface. The library I/O path functions come in both synchronous and asynchronous versions, which typically have identical functionality and semantics. When this is not true, the API will call out how the synchronous and asynchronous versions differ. Note that callbacks from the library are made from a static internal thread pool and so should not block for long periods of time. It is not allowed to call synchronous functions from a callback thread.

5 | Design Strategy

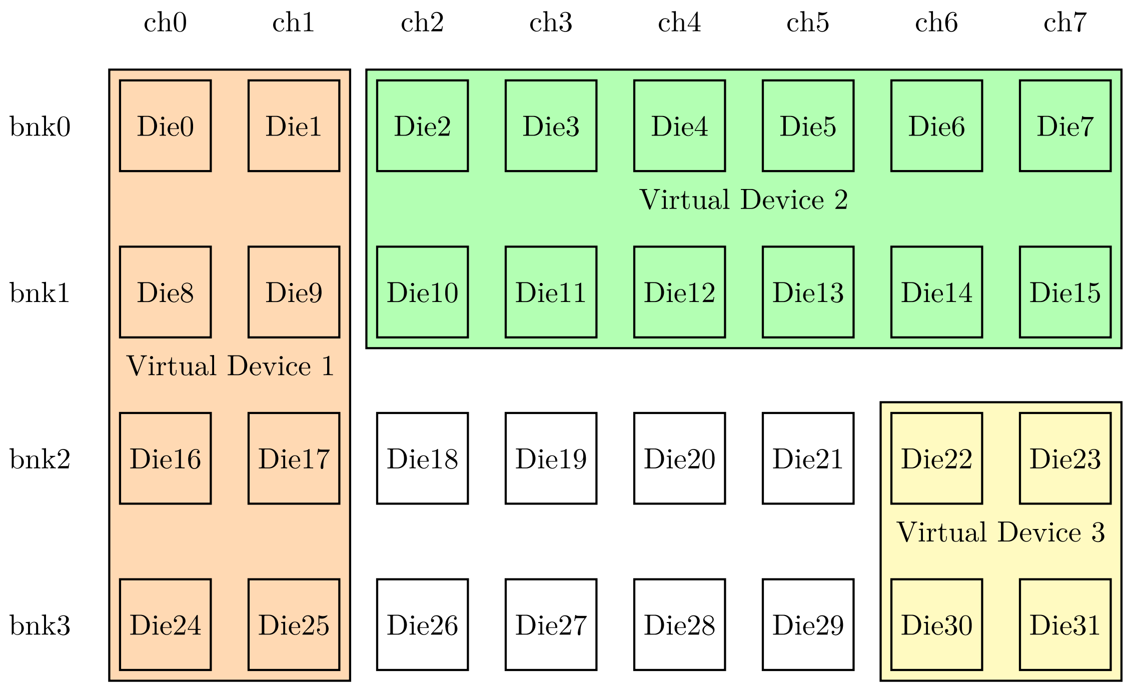
The SEF library is stateless. Nearly all requests result in one or more requests to the SEF driver. Those requests are submitted using the caller's thread. The completion of SEF driver requests is handled by an internal, statically sized thread pool based on the number of CPUs. Therefore, completion routines should not block on resources that require another completion routine to execute as that would risk deadlock. An example of a bad design is an I/O completion blocking waiting for a different I/O request to complete first. The SEF library specifically will fail synchronous calls into the SEF library from a completion routine for this reason. Of course, waiting for a resource owned by another completion thread won't cause deadlock, but it does reduce the number of threads available to process completions.

Writes to a SEF device complete before the final flash memory address has been assigned. A notification is sent if the final flash memory address is different than the preliminary address. However, no notification occurs when the preliminary flash memory address is the final flash memory address. It can be inferred by the super block closed notification. As a result, metadata that is tracking the allocated flash memory addresses is not final until the super block close notification is processed. In the case of a power failure, up-to-date metadata structures can be rebuilt from the user address data supplied when the data was written.

6 | Software-Enabled Flash™ (SEF) Unit

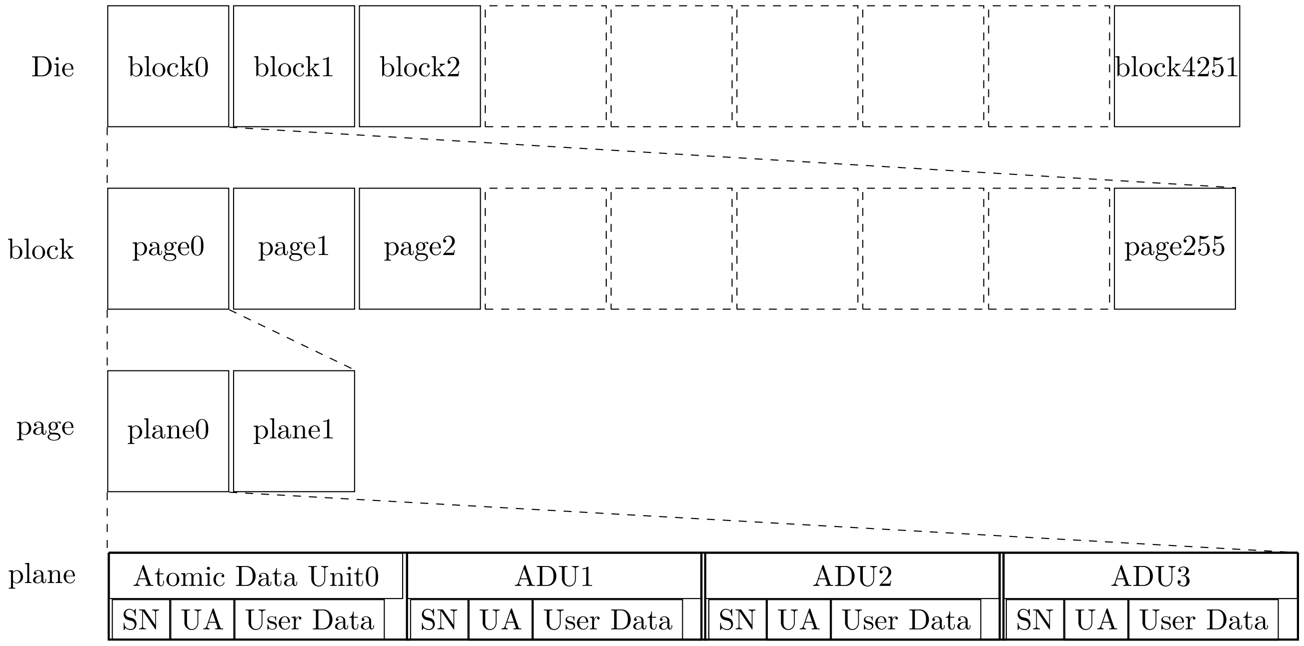
A SEF Unit is a set of dies and the associated control logic and (optional) DRAM. The dies are grouped by user-defined rectangular regions to form a many-to-one mapping of dies to a virtual device. A virtual device represents physical isolation with the number of virtual devices limited by the number of dies in the SEF Unit. Figure 6.1 shows an example of three virtual devices overlaid on an 8 x 4 SEF Unit with eight dies left unallocated.

Figure 6.1: SEF Unit Geometry



As shown in the figure 6.2, 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 programing 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 SEF-unit-created monotonically increasing serial number (SN).

Figure 6.2: Die Geometry



7 | 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 a virtual device is created, several parameters are specified to define the characteristics of the virtual device. Virtual devices are created by using the SEFCreateVirtualDevice() function. The size of the virtual device is user-configurable and dependent on the resources available. When a new virtual device is created, it 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. Deleting and creating new virtual devices is supported but may mix dies with different amounts of wear. In this case, wear leveling becomes the application's responsibility.

7.1 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 device. The maximum allowed ID is one less than the number of dies in the SEF Unit.

dieMap: Requests a rectangular region of dies that will be owned by the created virtual device.

defectStrategy: Specifies how defective ADUs are handled by the virtual device. 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 11 for more details.

numFMQueues: Specifies the number of Flash Media Queues per die. The maximum value is firmware-specific and can be retrieved with SEFGetInformation().

weights: Specifies the default weights for each Flash Media Queue. There is a default weight for each type of operation: read, erase for write, write, read for copy, erase for copy and write for copy. The weights affect which queue will supply the next die operation. If all the weights are the same, the queues are round-robined. If all the weights are 0, the queue number is used as a priority with queue 0 being the highest priority. Otherwise, weighted fair queuing is used and the queue with the lowest current die time is selected. When an I/O completes, the weighting for the I/O is added to the die time for the queue. The minimum die time from all the queues is then subtracted from all the queues. Additionally, when a read operation arrives at the head of a queue with a die time less than a currently executing write or erase, that operation is automatically suspended to execute the read. When choosing different command weights for weighted fair queuing, it is necessary to know the actual die time for each command and the percent of die time that is desired for each command. Command die time can be found in the SEFInfo structure returned by SEFGetInformation(). In general, if there are n commands whose die times are c_i with a desired percent p_i of die time, the weights w_i are then:

$$C = \sum_{i=0}^{n-1} c_i$$

$$w_i = \frac{c_i}{C \cdot p_i}$$

This will certainly yield fractional weights. To make them integers, they can be normalized by multiplying all the weights by a scaling factor (e.g., 100 and/or the inverse of the smallest weight). For example, if you wish reads to use 75% of the bandwidth and writes to use 25% with hypothetically reads taking 100us and writes taking 2000us.

$$C = 2100$$

$$Read = \frac{100}{2100 \times 0.75} = 0.0635$$

$$Write = \frac{2000}{2100 \times 0.25} = 3.81$$

Normalizing by dividing by 0.0635 yields a read weight of 1 and a write weight of 60.

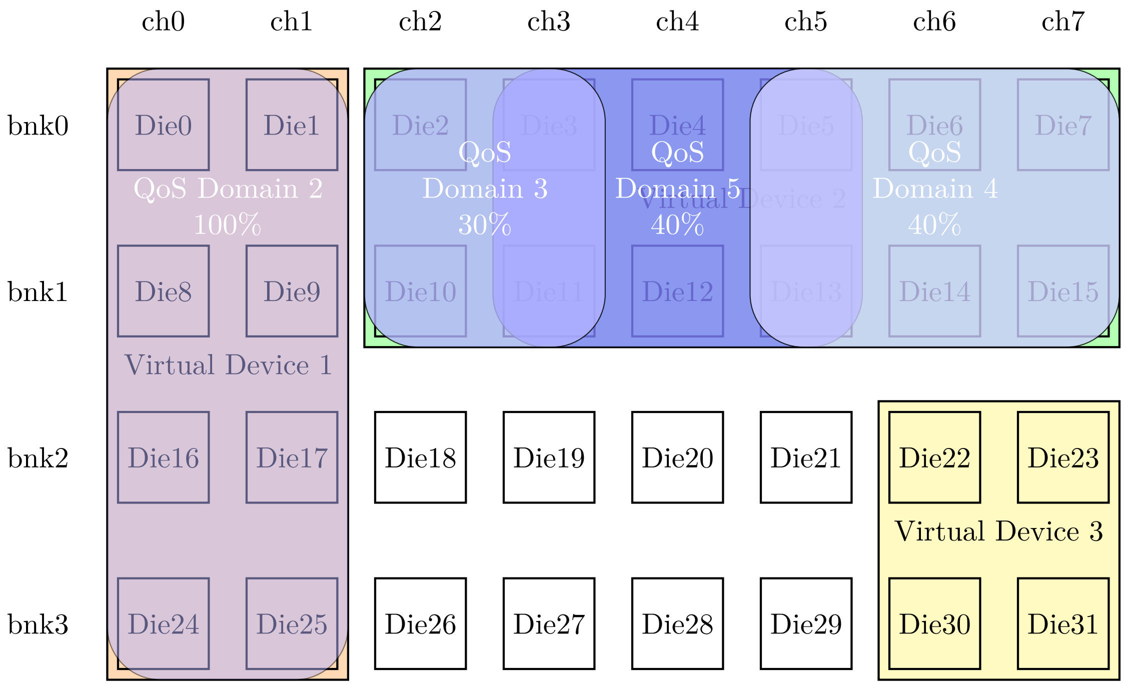
8 | QoS Domains

A QoS domain is the mechanism used to access data within a SEF device. 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. At boot time the SEF device 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.

Figure 8.1 shows an example of how the virtual devices of a SEF Unit could be divided into QoS domains. 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 specific hardware limitations.

Figure 8.1: QoS Domain Example



Allocated super blocks are owned by only one QoS domain at a time and 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.

8.1 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 device. IDs 0 and 1 are reserved.

flashCapacity: the number of ADUs reserved for the QoS domain. It is subtracted from the available ADUs from the virtual device so must be less than the currently available ADUs.

flashQuota: the quota for the amount of space this QoS domain may consume. The value is specified as the total number of ADUs. If less than flashCapacity, it will be set to flashCapacity. Since a super block is never shared between QoS domains, the actual capacity allocated for the QoS domain may be greater than requested to fill out an entire super block.

ADUsize: this is the ADU size requested for this QoS domain. Different QoS domains, even within the same virtual device, may have different ADU sizes. A list of supported ADU sizes for a SEF device may be queried from the SEF device.

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

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

encryption: specifies that the QoS domain is to be encrypted.

numRootPointers: specifies the number of user defined metadata pointers to be created for this QoS domain. A typical use for this would be to preserve metadata for the QoS domain within the QoS domain itself. For example, an L2P lookup table for a block FTL could be written/persisted within the QoS domain, keeping track of the physical addresses of the table. ADUs could then be written containing a tree of pointers to all of the ADUs making up the table, and finally the root pointer of the tree of pointers could be saved. At initialization time the root pointer could be read to restore the entire table. Up to 16 root pointers may be saved per QoS domain.

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.

FMQDefaults: specifies the default FMQ to use for each type of I/O operation. This can be optionally overridden when submitting I/O to a QoS domain.

9 | 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 virtual device. 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.

Unlike a regular page, the size of a super page is not static but is defined by the geometry of the virtual device. 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.

10 | Super Blocks

Super blocks are the main units of allocation used within the SEF API. Like super pages, super blocks span all the dies within a single virtual device. The number of super pages in a super block is fixed and is the same as the number of pages in a die. The size of a super block in ADUs, however, is dependent on the configuration of the virtual device that it resides in. A super block is only 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.

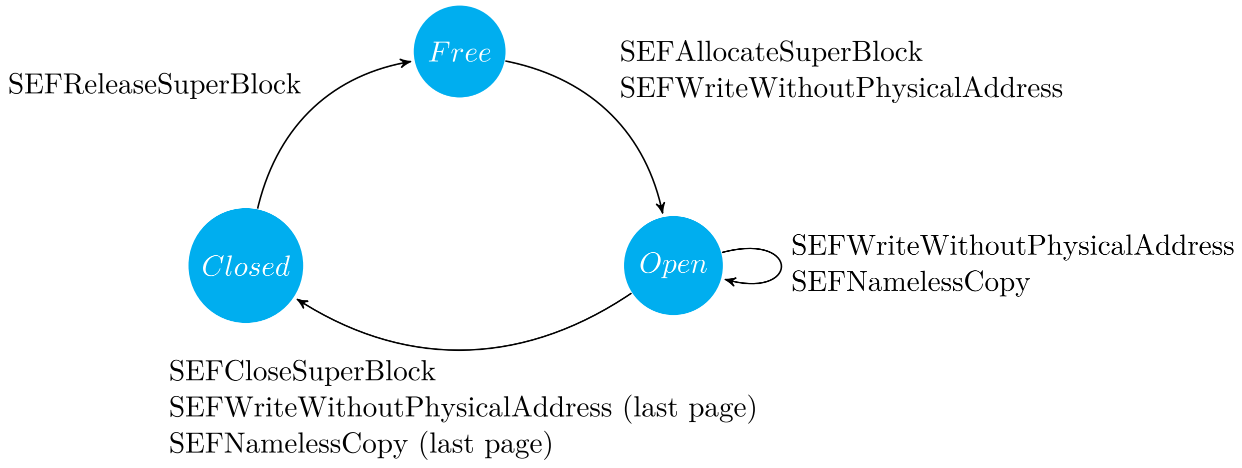
10.1 Super Block Management Commands

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 10.1 shows the state transitions regarding super blocks.

Super blocks are allocated either explicitly by the “Allocate” command, or implicitly by the “Write” command. When a reserved flash memory address (0xFFFFFFFFFFFFFFFF) 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 is allocated (assuming the capacity limit is not exceeded) once the current super block is filled.

Host does not need to erase super block. When the defect strategy is packed or fragmented, the apparent size of the super block may shrink after it is erased. This affects SEFWriteWithoutPhysicalAddress1(), SEFGetSuperBlockInfo() and SEFAllocateSuperBlock(). The number of available ADUs may also shrink as the super block is programmed.

Figure 10.1: Super Block State Transitions


Free
Open
Closed
SEFAllocateSuperBlock
SEFWriteWithoutPhysicalAddress
SEFReleaseSuperBlock
SEFWriteWithoutPhysicalAddress
SEFNamelessCopy
SEFCloseSuperBlock
SEFWriteWithoutPhysicalAddress (last page)
SEFNamelessCopy (last page)

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 block transits to ‘Free’ upon the Release command.

Open State

This is the state of super blocks in the middle of being programmed. ‘Free’ super block transits to ‘Open’ by Allocate command and Write Without Physical Address command.

Internally, there are three ‘Open’ states:

> *Open for Write Without Physical Address* : A super block dedicated to Write Without Physical Address. The super block transits to this state by 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 for Nameless Copy*: A super block dedicated to Nameless Copy. The super block transits to this state by a Nameless Copy command without explicit super block ID. Only one super block can be open for nameless copy at a time.
>
> *Open by Erase*: A super block opened by the super block management command “Allocate”. This super block can be used for Nameless Write by specifying explicit super block ID. For special case only.

Closed

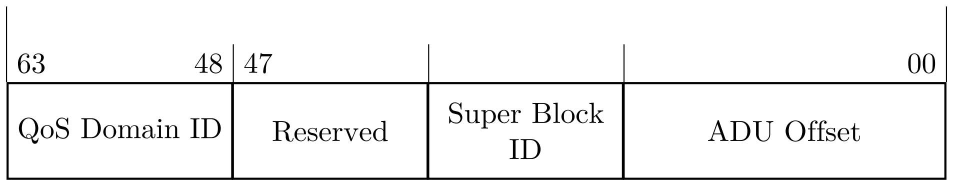
This is the state of super blocks which retain effective data after all Super Pages have been programmed. ‘Open’ super block transits to ‘Closed’ by either a Nameless Write command, a

Nameless Copy command, an explicit Close command or a device-initiated automatic close.

11 | 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 device, 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 11.1.

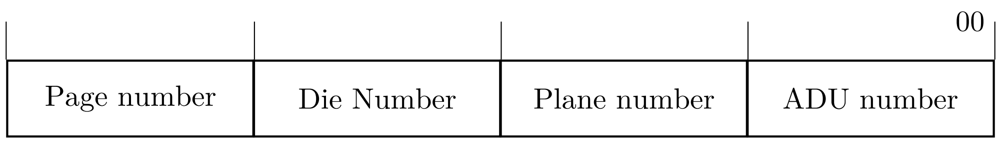
Figure 11.1: Flash Address



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. 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 11.2: Elements constructing ADU Offset in Fragmented mode



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

12 | API Management Commands

12.1 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: SEFStatus, SEFLibraryCleanup

Table 12.1: Return value of SEFLibraryInit

Type	Description
struct SEFStatus	Status and info summarizing result. The 'info' member contains number of units.

12.2 SEFGetHandle

```
SEFHandle SEFGetHandle(int index)
```

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

Table 12.2: Parameters of SEFGetHandle

Name	Type	Direction	Description
index	int	In	Index of the SEF Unit

Table 12.3: Return value of SEFGetHandle

Type	Description
SEFHandle	Handle to the SEF Unit

12.3 SEFLibraryCleanup

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

Performs cleanup of the SEF Library and releases resources.
See Also: SEFStatus, SEFLibraryInit

Table 12.4: Return value of SEFLibraryCleanup

Type	Description
struct SEFStatus	Status and info summarizing result.

12.4 SEFGetInformation

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

Gets device information.
Returns ADU size(s), number of channels, number of dies, and other associated information.
See Also: SEFStatus

Table 12.5: Parameters of SEFGetInformation

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

Table 12.6: Return value of SEFGetInformation

Type	Description
const struct SEFInfo *	Status and info summarizing result.

12.5 SEFListVirtualDevices

```
struct SEFStatus SEFListVirtualDevices(SEFHandle sefHandle, struct
   SEFVirtualDeviceList *list, int bufferSize)
```

Returns a list of the defined Virtual Devices.
See Also: SEFStatus

Table 12.7: Parameters of SEFListVirtualDevices

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

Table 12.8: Return value of SEFListVirtualDevices

Type	Description
struct SEFStatus	Status and info summarizing result.

12.6 SEFListQoSDomains

```
struct SEFStatus SEFListQoSDomains(SEFHandle sefHandle, struct
    SEFQoSDomainList *list, int bufferSize)
```

Returns a list of the defined QoS Domains.
See Also: SEFStatus

Table 12.9: Parameters of SEFListQoSDomains

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

Table 12.10: Return value of SEFListQoSDomains

Type	Description
struct SEFStatus	Status and info summarizing result.

12.7 SEFCreateVirtualDevice

```
struct SEFStatus SEFCreateVirtualDevice(SEFHandle sefHandle, struct
    SEFVirtualDeviceID virtualDeviceID, struct SEFDieMap dieMap, enum
    SEFDefectManagementMethod defectStrategy, uint8_t numFMQueues, const
     struct SEFWeights weights[])
```

Creates a Virtual Device and allocates physical resources.

See Also: SEFStatus, SEFGetInformation

Table 12.11: Parameters of SEFCreateVirtualDevice

Name	Type	Direction	Description
sefHandle	SEFHandle	In	Handle to the SEF Unit
virtualDeviceID	struct SEFVirtualDeviceID	In	Virtual Device ID
dieMap	struct SEFDieMap	In	Dies requested for virtual device
defectStrategy	enum SEFDefectManagement-Method	In	Defect management strategy for the Virtual Device
numFMQueues	uint8_t	In	Number of Flash Media Queues per die in the Virtual Device
weights	const struct SEFWeights	In	Weight values for each Flash Media Queue

Table 12.12: Return value of SEFCreateVirtualDevice

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

12.8 SEFGetVirtualDeviceInformation

```
struct SEFStatus SEFGetVirtualDeviceInformation(SEFHandle sefHandle,
   struct SEFVirtualDeviceID virtualDeviceID, struct
   SEFVirtualDeviceInfo *info, int bufferSize)
```

Returns Virtual Device information.

See Also: SEFStatus

Table 12.13: Parameters of SEFGetVirtualDeviceInformation

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

Table 12.14: Return value of SEFGetVirtualDeviceInformation

Type	Description
struct SEFStatus	Status and info summarizing result.

12.9 SEFCreateQoSDomain

```
struct SEFStatus SEFCreateQoSDomain(SEFVDHandle vdHandle, struct
    SEFQoSDomainID QoSDomainID, uint64_t flashCapacity, uint64_t
    flashQuota, uint32_t ADUsize, enum SEFAPIIdentifier api, enum
    SEFErrorRecoveryMode recovery, int encryption, uint16_t
    numRootPointers, uint16_t numPlacementIDs, struct SEFFMQAssignments
    FMQDefaults)
```

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. When the flashQuota is less than the flashCapacity, it will be set to the flashCapacity.
See Also: SEFGetInformation

Table 12.15: Parameters of SEFCreateQoSDomain

Name	Type	Direction	Description
vdHandle	SEFVDHandle	In	Handle to the Virtual Device
QoSDomainID	struct SEFQoSDomainID	In	QoS Domain ID. Unique across all QoS Domains
flashCapacity	uint64_t	In	Number of required/reserved ADUs
flashQuota	uint64_t	In	Number of ADUs that can be allocated
ADUsize	uint32_t	In	Size of ADU in this QoS domain in bytes. Must be one of the values in ADUSize[] in SEFInfo returned by SEFGetInformation(). SEF Unit should support 4kiB.
api	enum SEFAPIIdentifier	In	Specifies the API Identifier for this QoS domain
recovery	enum SEFErrorRecoveryMode	In	Specifies the recovery mode for this QoS domain

encryption	int	In	0 for disabled, non-zero for enabled
numRootPointers	uint16_t	In	Specifies the number of root pointers corresponding to this QoS domain
numPlacementIDs	uint16_t	In	The maximum number of Placement IDs that can be placed on the QoS domain. (The number of Placement IDs would affect memory usage)
FMQDefaults	struct SEFFMQAssignments	In	The default die FMQ assignments for I/O commands

Table 12.16: Return value of SEFCreateQoSDomain

Type	Description
struct SEFStatus	Status and info summarizing result.

12.10 SEFSetQoSDomainCapacity

```
struct SEFStatus SEFSetQoSDomainCapacity(SEFQoSHandle qosHandle,
    uint64_t flashCapacity, uint64_t flashQutoa)
```

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, it is set to the flashCapacity. Returns an error when the total capacity of assigned super blocks exceeds the new capacity.

Table 12.17: Parameters of SEFSetQoSDomainCapacity

Name	Type	Direction	Description
qosHandle	SEFQoSHandle	In	Handle to the QoS Domain
flashCapacity	uint64_t	In	Number of required/reserved ADUs
flashQutoa	uint64_t	In	Number of ADUs that can be allocated

Table 12.18: Return value of SEFSetQoSDomainCapacity

Type	Description
struct SEFStatus	Status and info summarizing result.

12.11 SEFSetRootPointer

```
struct SEFStatus SEFSetRootPointer(SEFQoSHandle qosHandle, int index,
   struct SEFFlashAddress value)
```

Sets the physical address of the QoSDomain root ADU 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 SEFReadWithPhysicalAddress1() using a flash address of just the root pointer index as the ADU.
See Also: SEFStatus, SEFReadWithPhysicalAddress1

Table 12.19: Parameters of SEFSetRootPointer

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

Table 12.20: Return value of SEFSetRootPointer

Type	Description
struct SEFStatus	Status and info summarizing result.

12.12 SEFSetReadDeadline

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

Sets target QoS Domain's read deadline policy.
See Also: SEFStatus, SEFVirtualDeviceInfo

Table 12.21: Parameters of SEFSetReadDeadline

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

Table 12.22: Return value of SEFSetReadDeadline

Type	Description
struct SEFStatus	Status and info summarizing result.

12.13 SEFGetQoSDomainInformation

```
struct SEFStatus SEFGetQoSDomainInformation(SEFHandle sefHandle, struct
    SEFQoSDomainID QoSDomainID, struct SEFQoSDomainInfo *info, int
   bufferSize)
```

Returns QoS Domain information, including the list of super blocks assigned to the QoS Domain.
See Also: SEFStatus

Table 12.23: Parameters of SEFGetQoSDomainInformation

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

Table 12.24: Return value of SEFGetQoSDomainInformation

Type	Description
struct SEFStatus	Status and info summarizing result.

12.14 SEFGetReuseList

```
struct SEFStatus SEFGetReuseList(SEFQoSHandle qosHandle, struct
   SEFWearInfo *info, int bufferSize)
```

Returns list of SuperBlocks to process for wear-leveling.
Used in support of the implementation of a host-specified wear leveling policy. SEF has a built in wear-leveling mechanism.
See Also: SEFStatus

Table 12.25: Parameters of SEFGetReuseList

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

Table 12.26: Return value of SEFGetReuseList

Type	Description
struct SEFStatus	Status and info summarizing result.

12.15 SEFGetRefreshList

```
struct SEFStatus SEFGetRefreshList(SEFQoSHandle qosHandle, struct
   SEFRefreshInfo *info, int bufferSize)
```

Returns a list of blocks that have encountered ECC errors.
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.
See Also: SEFStatus

Table 12.27: Parameters of SEFGetRefreshList

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

Table 12.28: Return value of SEFGetRefreshList

Type	Description
struct SEFStatus	Status and info summarizing result.

12.16 SEFGetCheckList

```
struct SEFStatus SEFGetCheckList(SEFQoSHandle qosHandle, struct
   SEFCheckInfo *info, int bufferSize)
```

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 (SEFCheckPage) 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.
See Also: SEFStatus, SEFCheckPage

Table 12.29: Parameters of SEFGetCheckList

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

Table 12.30: Return value of SEFGetCheckList

Type	Description
struct SEFStatus	Status and info summarizing result.

12.17 SEFGetUserAddressList

```
struct SEFStatus SEFGetUserAddressList(SEFQoSHandle qosHandle, struct
    SEFFlashAddress flashAddress, struct SEFUserAddressRecord *list, int
     bufferSize)
```

Returns the user address list in terms of its underlying superblocks.
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.
See Also: SEFStatus

Table 12.31: Parameters of SEFGetUserAddressList

Name	Type	Direction	Description
qosHandle	SEFQoSHandle	In	Handle to the QoS Domain
flashAddress	struct SEFFlashAddress	In	Physical address of the superblock
list	struct SEFUserAddressRecord *	Out	Buffer for storing list of user addresses
bufferSize	int	In	Buffer size

Table 12.32: Return value of SEFGetUserAddressList

Type	Description
struct SEFStatus	Status and info summarizing result.

12.18 SEFGetSuperBlockInfo

```
struct SEFStatus SEFGetSuperBlockInfo(SEFQoSHandle qosHandle, struct
    SEFFlashAddress flashAddress, struct SEFSuperBlockRecord *info)
```

Returns information corresponding to the superblock.
See Also: SEFStatus

Table 12.33: Parameters of SEFGetSuperBlockInfo

Name	Type	Direction	Description
qosHandle	SEFQoSHandle	In	Handle to the QoS Domain
flashAddress	struct SEFFlashAddress	In	Physical address of the superblock
info	struct SEFSuperBlockRecord *	Out	Buffer for storing superblock information

Table 12.34: Return value of SEFGetSuperBlockInfo

Type	Description
struct SEFStatus	Status and info summarizing result.

12.19 SEFCheckPage

```
struct SEFStatus SEFCheckPage(SEFQoSHandle qosHandle, struct
    SEFFlashAddress flashAddress)
```

This is a read patrol operation which can be used in conjunction with SEFGetCheckList.
Returns detailed information concerning checked pages to allow host software to take appropriate corrective actions.
See Also: SEFStatus, SEFGetCheckList

Table 12.35: Parameters of SEFCheckPage

Name	Type	Direction	Description
qosHandle	SEFQoSHandle	In	Handle to the QoS Domain
flashAddress	struct SEFFlashAddress	In	Physical address to be checked

Table 12.36: Return value of SEFCheckPage

Type	Description
struct SEFStatus	Status and info summarizing result.

12.20 SEFDeleteVirtualDevice

```
struct SEFStatus SEFDeleteVirtualDevice(SEFHandle sefHandle, struct
    SEFVirtualDeviceID virtualDeviceID)
```

Deletes the target virtual device.
The Virtual Device must be in the closed state before issuing this command. Moreover, this command will fail if the Virtual Device contains any QoS Domains.
See Also: SEFStatus

Table 12.37: Parameters of SEFDeleteVirtualDevice

Name	Type	Direction	Description
sefHandle	SEFHandle	In	Handle to the SEF Unit
virtualDeviceID	struct SEFVirtualDeviceID	In	Virtual Device ID

Table 12.38: Return value of SEFDeleteVirtualDevice

Type	Description
struct SEFStatus	Status and info summarizing result.

12.21 SEFDeleteQoSDomain

```
struct SEFStatus SEFDeleteQoSDomain(SEFHandle sefHandle, struct
    SEFQoSDomainID QoSDomainID)
```

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 superblocks are returned to the virtual device's free pool.
See Also: SEFStatus

Table 12.39: Parameters of SEFDeleteQoSDomain

Name	Type	Direction	Description

sefHandle	SEFHandle	In	Handle to the SEF Unit
QoSDomainID	struct SEFQoSDomainID	In	QoS Domain ID

Table 12.40: Return value of SEFDeleteQoSDomain

Type	Description
struct SEFStatus	Status and info summarizing result.

12.22 SEFResetEncryptionKey

```
struct SEFStatus SEFResetEncryptionKey(SEFHandle sefHandle, struct
   SEFQoSDomainID QoSDomainID)
```

Resets the encryption key for a QoS Domain.
See Also: SEFStatus

Table 12.41: Parameters of SEFResetEncryptionKey

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

Table 12.42: Return value of SEFResetEncryptionKey

Type	Description
struct SEFStatus	Status and info summarizing result.

12.23 SEFOpenVirtualDevice

```
struct SEFStatus SEFOpenVirtualDevice(SEFHandle sefHandle, struct
   SEFVirtualDeviceID virtualDeviceID, void(*notifyFunc)(void *, struct
    SEFVDNotification), void *context, SEFVDHandle *vdHandle)
```

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.
See Also: SEFStatus

Table 12.43: Parameters of SEFOpenVirtualDevice

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

Table 12.44: Return value of SEFOpenVirtualDevice

Type	Description
struct SEFStatus	Status and info summarizing result.

12.24 SEFCloseVirtualDevice

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

Closes an open Virtual Device and shuts down associated event notification.
See Also: SEFStatus

Table 12.45: Parameters of SEFCloseVirtualDevice

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

Table 12.46: Return value of SEFCloseVirtualDevice

Type	Description
struct SEFStatus	Status and info summarizing result.

12.25 SEFOpenQoSDomain

```
struct SEFStatus SEFOpenQoSDomain(SEFHandle sefHandle, struct
    SEFQoSDomainID QoSDomainID, void(*notifyFunc)(void *, struct
    SEFQoSNotification), void *context, const void *encryptionKey,
    SEFQoSHandle *qosHandle)
```

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.
See Also: SEFStatus

Table 12.47: Parameters of SEFOpenQoSDomain

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

Table 12.48: Return value of SEFOpenQoSDomain

Type	Description
struct SEFStatus	Status and info summarizing result.

12.26 SEFCloseQoSDomain

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

Closes an open QoS Domain.
This in turn will close any open superblocks 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.
See Also: SEFStatus

Table 12.49: Parameters of SEFCloseQoSDomain

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

Table 12.50: Return value of SEFCloseQoSDomain

Type	Description
struct SEFStatus	Status and info summarizing result.

12.27 SEFParseFlashAddress

```
struct SEFStatus SEFParseFlashAddress(SEFHandle sefHandle, struct
    SEFFlashAddress flashAddress, struct SEFQoSDomainID *QoSDomainID,
    uint16_t *blockNumber, uint32_t *ADUOffset)
```

This function is used to extract info needed by FTL from an opaque flash address.
See Also: SEFStatus

Table 12.51: Parameters of SEFParseFlashAddress

Name	Type	Direction	Description
sefHandle	SEFHandle	In	Handle to the SEF Unit
flashAddress	struct SEFFlashAddress	In	The opaque address to be parsed
QoSDomainID	struct SEFQoSDomainID *	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
blockNumber	uint16_t *	Out	A pointer to where to return the block number. A null pointer indicates that the block number is not to be returned
ADUOffset	uint32_t *	Out	A pointer to where to return the ADU Offset. A null pointer indicates that the ADU Offset is not to be returned

Table 12.52: Return value of SEFParseFlashAddress

Type	Description
struct SEFStatus	Status and info summarizing result.

12.28 SEFCreateFlashAddress

```
struct SEFFlashAddress SEFCreateFlashAddress(SEFHandle sefHandle,
    struct SEFQoSDomainID QoSDomainID, uint16_t blockNumber, uint32_t
    ADUOffset)
```

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 block number not owned by the QoSDomainID, or a QoSDomainID that has not been opened by the caller.

Table 12.53: Parameters of SEFCreateFlashAddress

Name	Type	Direction	Description
sefHandle	SEFHandle	In	Handle to the SEF Unit
QoSDomainID	struct SEFQoSDomainID	In	The desired QoS Domain ID.
blockNumber	uint16_t	In	The desired block number.
ADUOffset	uint32_t	In	The desired ADU Offset.

Table 12.54: Return value of SEFCreateFlashAddress

Type	Description
struct SEFFlashAddress	The generated flash address.

12.29 SEFReleaseSuperBlock

```
struct SEFStatus SEFReleaseSuperBlock(SEFQoSHandle qosHandle, struct
    SEFFlashAddress flashAddress)
```

Releases the specific Super Block to the free pool owned by the Virtual Device to which the specified QoS Domain belongs.
The target superblock must have been assigned by a previous call to SEFAllocateSuperBlock() or as part of SEFWriteWithoutPhysicalAddress1(). The superblock must be closed, otherwise the call will fail.
See Also: SEFStatus

Table 12.55: Parameters of SEFReleaseSuperBlock

Name	Type	Direction	Description
qosHandle	SEFQoSHandle	In	Handle to the QoS Domain of the Super Block

flashAddress	struct SEFFlashAddress	In	Physical address of the superblock to release

Table 12.56: Return value of SEFReleaseSuperBlock

Type	Description
struct SEFStatus	Status and info summarizing result.

12.30 SEFAllocateSuperBlock

```
struct SEFStatus SEFAllocateSuperBlock(SEFQoSHandle qosHandle, struct
    SEFFlashAddress *flashAddress, uint32_t retention, enum
    SEFSuperBlockType type, const struct SEFAllocateOverrides *overrides
    )
```

Allocates a superblock that will be assigned to the specified QoS Domain and returns the physical address of this superblock.
Any number of superblocks can be kept open for write for each QoS domain. These superblocks in turn can be used as part of the parameter set for SEFWriteWithoutPhysicalAddress(). When allocating a superblock, SEF intelligently selects a location in a manner designed to optimize the lifetime of flash memory and will return the physical address that was selected. Note that each open superblock will allocate a write buffer and therefore consume memory, so there is a tradeoff in the number of open superblocks and the amount of memory consumed.
Required that the total ADUs in the domain be less than its flash quota. This can be known by summing the writableADUs of each superblock in the domain.
See Also: SEFStatus, SEFGetQoSDomainInformation

Table 12.57: Parameters of SEFAllocateSuperBlock

Name	Type	Direction	Description
qosHandle	SEFQoSHandle	In	Handle to the QoS Domain
flashAddress	struct SEFFlashAddress *	Out	The flash address of the allocated block
retention	uint32_t	In	Retention period in hours
type	enum SEFSuperBlockType	In	kForWrite, kForCopy or kForDeviceMetadata
overrides	const struct SEFAllocateOverrides *	In	Overrides to scheduler parameters; pointer can be null for none required.

Table 12.58: Return value of SEFAllocateSuperBlock

Type	Description
struct SEFStatus	Status and info summarizing result. .info contains number of ADUs in allocated superblock

12.31 SEFFlushSuperBlock

```
struct SEFStatus SEFFlushSuperBlock(SEFQoSHandle qosHandle, struct
    SEFFlashAddress flashAddress, uint32_t *distanceToEndOfSuperBlock)
```

Flushes the target superblock.
This command causes all written data for the superblock 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 any address change notifications for the superblock being flushed have been processed, ensuring that all previously tentative addresses are now permanent.
See Also: SEFStatus

Table 12.59: Parameters of SEFFlushSuperBlock

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

Table 12.60: Return value of SEFFlushSuperBlock

Type	Description
struct SEFStatus	Status and info summarizing result.

12.32 SEFCloseSuperBlock

struct SEFStatus SEFCloseSuperBlock(SEFQoSHandle qosHandle, **struct** SEFFlashAddress flashAddress)

Closes the target superblock.
If there is remaining unwritten space in the superblock, that space will be padded with dummy data. This can be used by the FTL as a means of closing a superblock without invoking a Write command.
This command will not return until all address change and superblock state change notifications for the superblock being closed have been processed, ensuring that all previously tentative addresses are now permanent.
See Also: SEFStatus

Table 12.61: Parameters of SEFCloseSuperBlock

Name	Type	Direction	Description
qosHandle	SEFQoSHandle	In	Handle to the QoS Domain of the Super Block
flashAddress	struct SEFFlashAddress	In	Physical address of the Super Block to move to Closed state by filling data

Table 12.62: Return value of SEFCloseSuperBlock

Type	Description
struct SEFStatus	Status and info summarizing result.

12.33 SEFPrepareBufferForNamelessCopy

void* SEFPrepareBufferForNamelessCopy(**struct** SEFCopySource *copySource, **const struct** SEFUserAddressFilter *filter, uint32_t numAddressChangeRecords, **struct** SEFAddressChangeRequest ** addressChangeInfo)

This function allocates a data buffer SEFNamelessCopy().
It initializes filter parameters and returns pointers into members of copySource and addressChangeInfo.

Table 12.63: Parameters of SEFPrepareBufferForNamelessCopy

Name	Type	Direction	Description

copySource	struct SEFCopySource *	In	Description of copy source; format and arraysize MUST be initialized. The validBitmap pointer or flashAddressList pointer will be set by this function.
filter	const struct SEFUserAddressFilter *	In	Pointer to filter parameters, may be null for no filtering. This function will set the filter fields in the buffer.
numAddressChangeRecords	uint32_t	In	Size of addressChangeRequest userAddress array
addressChangeInfo	struct SEFAddressChangeRequest **	Out	A pointer to pointer to the address change info within the buffer (set by this function)

Table 12.64: Return value of SEFPrepareBufferForNamelessCopy

Type	Description
void *	Pointer to allocated buffer or NULL if error

12.34 SEFFreeBufferForNamelessCopy

```
void SEFFreeBufferForNamelessCopy(void *copyContext)
```

Frees the buffer allocated with SEFPrepareBufferForNamelessCopy().

Table 12.65: Parameters of SEFFreeBufferForNamelessCopy

Name	Type	Direction	Description
copyContext	void *	In	A pointer to the memory to free

12.35 SEFReleaseSuperBlockAsync

```
void SEFReleaseSuperBlockAsync(SEFQoSHandle qosHandle, struct
    SEFReleaseSuperBlockIOCB *iocb)
```

This function is the asynchronous version of SEFReleaseSuperBlock().
See Also: SEFReleaseSuperBlock

Table 12.66: Parameters of SEFReleaseSuperBlockAsync

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

12.36 SEFAllocateSuperBlockAsync

```
void SEFAllocateSuperBlockAsync(SEFQoSHandle qosHandle, struct
    SEFAllocateSuperBlockIOCB *iocb)
```

This function is the asynchronous version of SEFAllocateSuperBlock().
See Also: SEFAllocateSuperBlock

Table 12.67: Parameters of SEFAllocateSuperBlockAsync

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

12.37 SEFCloseSuperBlockAsync

```
void SEFCloseSuperBlockAsync(SEFQoSHandle qosHandle, struct
    SEFCloseSuperBlockIOCB *iocb)
```

This function is the asynchronous version of SEFCloseSuperBlock().
kSuperblockStateChanged will have been sent before the completion routine is called and the iocb is marked as done.
See Also: SEFCloseSuperBlock

Table 12.68: Parameters of SEFCloseSuperBlockAsync

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

13 | Data Access Commands

13.1 SEFWriteWithoutPhysicalAddress1

```
struct SEFStatus SEFWriteWithoutPhysicalAddress1(SEFQoSHandle qosHandle
    , struct SEFFlashAddress flashAddress, struct SEFPlacementID
    placementID, struct SEFUserAddress userAddress, uint32_t numADU,
    const struct iovec *iov, uint16_t iovcnt, struct SEFFlashAddress *
    permanentAddresses, uint32_t *distanceToEndOfSuperBlock, const
    struct SEFWriteOverrides *overrides)
```

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

If auto-allocate was enabled on the superblock, when the assigned superblock is filled and closed, SEF assigns a new super-block for following writes. If auto-allocate is not enabled, host software will know about the superblock size as part of the allocation, and can use this information to construct appropriately-sized write commands. Manually allocated superblocks for writes MUST be of type kForWrite. 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 SEFReadWithPhysicalAddress1(). In kSuperblock mode, the LBA portion of the user address is incremented for each ADU when writing multiple ADUs. The user address value 0xFFFFFFFFFFFFFFFF is reserved and is invalid.

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 device eventually flushes a tentative address to flash it may be discovered to be bad. When this happens, a kAddressUpdate QoS notification is sent indicating the data has moved to a new permanent flash address. There is no notification for addresses that have successfully flushed and are now permanent. It can be inferred instead by the kSuperblockStateChanged QoS notification for the owning superblock.

See Also: SEFStatus

Table 13.1: Parameters of SEFWriteWithoutPhysicalAddress1

Name	Type	Direction	Description
qosHandle	SEFQoSHandle	In	Handle to the QoS Domain
flashAddress	struct SEFFlashAddress	In	Physical address of the superblock. 0xFFFFFFFFFFFFFFFF if auto allocate.
placementID	struct SEFPlacementID	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.
userAddress	struct SEFUserAddress	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.
numADU	uint32_t	In	Total amount of write data size calculated in ADU. Maximum allowed is 64k ADUs.
iov	const struct iovec *	In	A pointer to the scatter gather list
iovcnt	uint16_t	In	The number of elements in the scatter gather list
permanentAddresses	struct SEFFlashAddress *	Out	Must allocate space for returned permanent addresses equal to 8*length (e.g. 8*number of ADUs)

distanceToEndOfSuperBlock	uint32_t *	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 superblocks are open.
overrides	const struct SEFWriteOverrides *	In	Overrides to scheduler parameters; pointer can be null for none required.

Table 13.2: Return value of SEFWriteWithoutPhysicalAddress1

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

13.2 SEFReadWithPhysicalAddress1

```
struct SEFStatus SEFReadWithPhysicalAddress1(SEFQoSHandle qosHandle,
   struct SEFFlashAddress flashAddress, uint32_t numADU, const struct
   iovec *iov, uint16_t iovcnt, uint32_t iovOffset, struct
   SEFUserAddress userAddress, const struct SEFReadOverrides *overrides
   )
```

Reads data from a specified physical address.

While writes are expressed in terms of logical addresses, reads are expressed in terms of physical 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 will 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 0 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 changed flash address notification has been processed.

See Also: SEFStatus, SEFSetRootPointer

Table 13.3: Parameters of SEFReadWithPhysicalAddress1

Name	Type	Direction	Description
qosHandle	SEFQoSHandle	In	Handle to the QoS Domain
flashAddress	struct SEFFlashAddress	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.
numADU	uint32_t	In	Length of data to read (in ADUs). Maximum allowed is superblockCapacity.
iov	const struct iovec *	In	A pointer to the scatter gather list
iovcnt	uint16_t	In	Number of elements in the scatter gather list
iovOffset	uint32_t	In	Starting byte offset into iov array
userAddress	struct SEFUserAddress	In	Stored data by the FTL. It will be validated with what was stored when the data was written except when SEFUserAddressIgnore is supplied
overrides	const struct SEFReadOverrides *	In	Overrides to scheduler parameters; pointer can be null for none required.

Table 13.4: Return value of SEFReadWithPhysicalAddress1

Type	Description
struct SEFStatus	Status and info summarizing result.

13.3 SEFNamelessCopy

```
struct SEFStatus SEFNamelessCopy(SEFQoSHandle srcQosHandle, struct
    SEFCopySource copySource, SEFQoSHandle dstQosHandle, struct
    SEFFlashAddress copyDestination, const struct SEFUserAddressFilter *
    filter, const struct SEFCopyOverrides *overrides, uint32_t
    numAddressChangeRecords, struct SEFAddressChangeRequest *
    addressChangeInfo, void *copyContext)
```

Performs Nameless Copy with map or list; optional user address filtering.
Copies ADUs as described by copySource to the copyDestination. If the destination superblock was allocated by SEFAllocateSuperBlock() the type must be kForCopy.
Note: Padding is added when the copy is not a multiple of the minimum writeable unit.
See Also: SEFStatus, SEFProcessAddressChangeRequests, SEFPrepareBufferForNamelessCopy

Table 13.5: Parameters of SEFNamelessCopy

Name	Type	Direction	Description
srcQosHandle	SEFQoSHandle	In	Handle to the source QoS Domain
copySource	struct SEFCopySource	In	Physical addresses to copy
dstQosHandle	SEFQoSHandle	In	Handle to the destination QoS Domain
copyDestination	struct SEFFlashAddress	In	Physical address of destination superblock
filter	const struct SEFUserAddressFilter *	In	Pointer to user address filter parameters, null indicates no filtering
overrides	const struct SEFCopyOverrides *	In	Pointer to overrides to scheduler parameters; pointer can be null for none required.
numAddressChangeRecords	uint32_t	In	Maximum number of ADUs to copy (size of addressChangeRequest userAddress array)
addressChangeInfo	struct SEFAddressChangeRequest *	Out	Information to record changed addresses
copyContext	void *	In	Pointer to working buffer returned by SEFPrepareBufferForNamelessCopy()

Table 13.6: Return value of SEFNamelessCopy

Type	Description

struct SEFStatus	Status and info summarizing result, .info contains:Destination super block has defective planes (1bit)Read error was detected on source (1bit)Data that is out of User Address range is detected (1bit)Destination superblock was filled/closed (1bit)Consumed entire source bitmap or list (1bit)

13.4 SEFProcessAddressChangeRequests

```
struct SEFStatus SEFProcessAddressChangeRequests(SEFQoSHandle
    srcQosHandle, struct SEFCopySource copySource, SEFQoSHandle
    dstQosHandle, uint32_t copyInfo, const struct
    SEFAddressChangeRequest *addressChangeInfo)
```

Performs post processing of address change records for Nameless Copy.
See Also: SEFStatus, SEFNamelessCopy

Table 13.7: Parameters of SEFProcessAddressChangeRequests

Name	Type	Direction	Description
srcQosHandle	SEFQoSHandle	In	Handle to the source QoS Domain
copySource	struct SEFCopySource	In	Physical addresses to copy
dstQosHandle	SEFQoSHandle	In	Handle to the destination QoS Domain
copyInfo	uint32_t	In	Information returned from namelessCopy in status.info field. copyInfo contains:Destination super block has defective planes (1bit)Read error was detected on source (1bit)Data that is out of User Address range is detected (1bit)Destination superblock was filled/closed (1bit)Consumed entire source bitmap or list (1bit)
addressChangeInfo	const struct SEFAddressChangeRequest *	In	Information to record changed addresses

Table 13.8: Return value of SEFProcessAddressChangeRequests

Type	Description
struct SEFStatus	Status and info summarizing result.

13.5 SEFWriteWithoutPhysicalAddress1Async

```
void SEFWriteWithoutPhysicalAddress1Async(SEFQoSHandle qosHandle,
    struct SEFWriteWithoutPhysicalAddressIOCB *iocb)
```

This function is the asynchronous version of SEFWriteWithoutPhysicalAddress1().
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: SEFWriteWithoutPhysicalAddress1

Table 13.9: Parameters of SEFWriteWithoutPhysicalAddress1Async

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

13.6 SEFReadWithPhysicalAddress1Async

```
void SEFReadWithPhysicalAddress1Async(SEFQoSHandle qosHandle, struct
    SEFReadWithPhysicalAddressIOCB *iocb)
```

This function is the asynchronous version of SEFReadWithPhysicalAddress1().
See Also: SEFReadWithPhysicalAddress1

Table 13.10: Parameters of SEFReadWithPhysicalAddress1Async

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

13.7 SEFNamelessCopyAsync

```
void SEFNamelessCopyAsync(SEFQoSHandle qosHandle, struct
   SEFNamelessCopyIOCB *iocb)
```

This function is the asynchronous version of SEFNamelessCopy().
See Also: SEFNamelessCopy

Table 13.11: Parameters of SEFNamelessCopyAsync

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

14 Common Structures

14.1 SEFStatus

Table 14.1: Members of SEFStatus

Name	Type	Description
error	int64_t	Status information
info	int64_t	Additional context-based descriptive information

14.2 SEFVirtualDeviceID

Table 14.2: Members of SEFVirtualDeviceID

Name	Type
id	uint16_t

14.3 SEFQoSDomainID

Table 14.3: Members of SEFQoSDomainID

Name	Type
id	uint16_t

14.4 SEFPlacementID

Table 14.4: Members of SEFPlacementID

Name	Type
id	uint16_t

14.5 SEFInfo

Table 14.5: Members of SEFInfo

Name	Type	Description
vendor	char[8]	Vendor field
serialNumber	char[20]	Device serial number
FWVersion	char[8]	Device firmware version
HWVersion	char[8]	Device hardware version
maxQoSDomains	uint16_t	Hardware version specific, may be less than 65535 defined by architecture
maxRootPointers	uint16_t	Firmware version specific, may be less than 16 defined by architecture
supportedOptions	uint64_t	Bitmap of supported features
maxPlacementIDs	uint16_t	Firmware version specific, max number of open superblocks per QoS domain
numFlashMediaQueues	uint16_t	Firmware version specific, max number of scheduling queues per die
numVirtualDevices	uint16_t	Number of currently defined virtual devices
numQoSDomains	uint16_t	Number of currently defined QoS Domains
APIVersion	uint16_t	API Version
numDies	uint16_t	Number of dies per channel
numChannels	uint16_t	Number of channels per SEF Unit
numPlanes	uint16_t	Number of planes per die
numADUSizes	uint16_t	Size of ADUsize array that follows at end of structure
reserved_0	uint16_t	
numBlocks	uint32_t	Number of blocks per die
numPages	uint32_t	Number of pages per block
pageSize	uint32_t	Physical page size
metaSize	uint32_t	Meta size per ADU

totalBandWidth	uint32_t	Total bandwidth corresponding to the underlying NAND component on this device
readLatency	uint32_t	Read latency corresponding to the underlying NAND components on this device
writeLatency	uint32_t	Write latency corresponding to the underlying NAND components on this device
eraseLatency	uint32_t	Erase latency corresponding to the underlying NAND components on this device
openExpirationPeriod	uint32_t	Granularity in seconds for entire block
ADUsize	uint32_t[0]	Array of supported ADU sizes (in bytes)

14.6 SEFVirtualDeviceList

Table 14.6: Members of SEFVirtualDeviceList

Name	Type	Description
numVirtualDevices	uint16_t	Number of virtual devices
virtualDeviceID	struct SEFVirtualDeviceID[0]	An Array of all Virtual device IDs

14.7 SEFQoSDomainList

Table 14.7: Members of SEFQoSDomainList

Name	Type	Description
numQoSDomains	uint16_t	Number of QoS domains
QoSDomainID	struct SEFQoSDomainID[0]	An Array of all QoS Domain IDs

14.8 SEFUserAddress

Structure of SEFUserAddress may be redefined by user.
The limitations for redefining the structure are:size must be metaSize from SEFInfo struct, 8 bytes for foreseeable futureValue of 0xFFFFFFFFFFFFFFFF is reservedmulti-adu writes will auto

increment the LBA valueFor kSuperblock, the LBA is limited to 40 bits and the meta to 24. The bits member is in little endian format.

Table 14.8: Members of SEFUserAddress

Name	Type
unformatted	uint64_t

14.9 SEFFlashAddress

Opaque flash address value parsable by SEFParseFlashAddress()

Table 14.9: Members of SEFFlashAddress

Name	Type
bits	uint64_t

14.10 SEFDieMap

Table 14.10: Members of SEFDieMap

Name	Type	Description
startChannel	uint8_t	starting channel number for rectangular region
startBank	uint8_t	starting bank number for rectangular region
numChannels	uint8_t	width for rectangular region
numBanks	uint8_t	height for rectangular region

14.11 SEFWeights

Relative die time weights for basic operations.

Table 14.11: Members of SEFWeights

Name	Type	Description
readWeight	uint16_t	Default Weight for a Read operation by Read commands

eraseWeight	uint16_t	Default Weight for an Erase operation by SEFAllocateSuperBlock for user Nameless writes
programWeight	uint16_t	Default Weight for a Program operation by Nameless Write commands
read4CopyWeight	uint16_t	Default Weight for a Read operation by Nameless Copy commands
erase4CopyWeight	uint16_t	Default Weight for an Erase operation by SEFAllocateSuperBlock for Nameless Copy
program4CopyWeight	uint16_t	Default Weight for a Program operation by Nameless Copy commands

14.12 SEFVirtualDeviceInfo

Table 14.12: Members of SEFVirtualDeviceInfo

Name	Type	Description
flashCapacity	uint64_t	Flash capacity in ADUs
flashAvailable	uint64_t	Available flash capacity in ADUs
superBlockCapacity	uint32_t	Total SuperBlock capacity in ADUs
eraseCount	uint32_t	Number of superblocks erased. Used to populate eraseOrder in SEFSuperBlockRecord
dieMap	struct SEFDieMap	Dies allocated to this virtual device
weights	struct SEFWeights[SEFMaxFMQueues]	Weights for each FMQ
numFMQueues	uint8_t	Number of flash media queues per die
defectStrategy	enum SEFDefectManagementMethod	Defect management strategy for the Virtual Device
averagePEcount	uint8_t	Average program/erase count
maxPEcount	uint8_t	Max program/erase count

numUnallocatedSuperBlocks	uint16_t	Number of unallocated super blocks
numSuperBlocks	uint16_t	Number of allocated super blocks
QoSDomains	struct SEFQoSDomainList	List of domains

14.13 SEFFMQAssignments

Table 14.13: Members of SEFFMQAssignments

Name	Type	Description
readFMQ	uint8_t	Default FMQ for user read commands
programFMQ	uint8_t	Default FMQ for user nameless write commands
read4CopyFMQ	uint8_t	Default FMQ for read by nameless copy commands
program4CopyFMQ	uint8_t	Default FMQ for write by nameless copy commands

14.14 SEFSuperBlockRecord

Table 14.14: Members of SEFSuperBlockRecord

Name	Type	Description
flashAddress	struct SEFFlashAddress	Flash address where this superblock resides
eraseOrder	uint32_t	Indication of when a superblock was erased. Can be used to determine the order blocks were allocated or to version a superblock. Values only increase over time and are unique at the virtual device level
writableADUs	uint32_t	If superblock is closed, writableADUs and writtenADUs are equal; if they are not equal, the superblock must still be open

writtenADUs	uint32_t	This field increments as ADUs in the superblock are written
placementID	struct SEFPlacementID	When auto-allocated, indicates the placement id supplied to SEFWriteWithoutPhysicalAddress1(). Otherwise it will be 0xffff
PEIndex	uint8_t	This is the block's erase count normalized to be between 0 and 255

14.15 SEFQoSDomainInfo

Table 14.15: Members of SEFQoSDomainInfo

Name	Type	Description
virtualDeviceID	struct SEFVirtualDeviceID	Virtual device ID
numPlacementIDs	uint16_t	Specifies the number of Placement IDs corresponding to this QoS domain
numRootPointers	uint16_t	Specifies the number of root pointers corresponding to this QoS domain
encryption	uint8_t	0 for disabled, non-zero for enabled
api	enum SEFAPIIdentifier	Specifies the API Identifier for this QoS domain
capacity	uint64_t	Reserved capacity of the QoS domain in ADUs
quota	uint64_t	Number of ADUs that can be allocated by the QoS domain
recoveryMode	enum SEFErrorRecoveryMode	Specifies the recovery mode for this QoS domain
deadline	enum SEFDeadlineType	Deadline type for the QoS domain
FMQDefaults	struct SEFFMQAssignments	The default die FMQ assignments for I/O commands
reserved_0	uint16_t	
rootPointers	struct SEFFlashAddress[SEFMaxRootPointer]	List of root pointers
ADUsize	uint32_t	Size of ADU in bytes
numSuperBlocks	uint32_t	Number of superblocks in use by the QoS Domain

superBlockRecords	struct SEFSuperBlockRecord[0]	List of superblock records

14.16 SEFWearInfo

Table 14.16: Members of SEFWearInfo

Name	Type	Description
numSuperBlocks	uint32_t	Number of superblocks
reserved_0	uint32_t	
superBlockRecords	struct SEFSuperBlockRecord[0]	List of superblock records

14.17 SEFRefreshInfo

Table 14.17: Members of SEFRefreshInfo

Name	Type	Description
numSuperBlocks	uint32_t	Number of superblocks
reserved_0	uint32_t	
superBlockRecords	struct SEFSuperBlockRecord[0]	List of superblock records

14.18 SEFCheckInfo

Super blocks returned by SEFGetCheckList()

Table 14.18: Members of SEFCheckInfo

Name	Type	Description
numSuperBlocks	uint32_t	Number of superblocks
reserved_0	uint32_t	
superBlockRecords	struct SEFSuperBlockRecord[0]	List of superblock records

14.19 SEFUserAddressRecovery

Table 14.19: Members of SEFUserAddressRecovery

Name	Type	Description

serial	uint64_t	Monotonically increasing generational counter that indicates the order in which blocks were written. For example, it can be used for replay for data recovery
userAddress	struct SEFUserAddress	Contains LBA information

14.20 SEFUserAddressRecord

Table 14.20: Members of SEFUserAddressRecord

Name	Type	Description
numADUs	uint32_t	Number of ADUs
reserved_0	uint32_t	
userAddressesRecovery	struct SEFUserAddressRecovery[0]	User address recovery scheme

14.21 SEFWriteOverrides

Supplied to override default write FMQ and weights.
May be used when calling SEFWriteWithoutPhysicalAddress1() or SEFWriteWithoutPhysicalAddress1Async(). Any of these fields can be set to -1 to use the default

Table 14.21: Members of SEFWriteOverrides

Name	Type	Description
eraseWeight	uint16_t	Weight to use for erase instead of virtual device default
programWeight	uint16_t	Weight to use for program instead of virtual device default
programFMQ	uint8_t	Flash Media Queue to use for erase and write instead of QoS Domain default

14.22 SEFReadOverrides

Supplied to override default read FMQ and weight.
May be used when calling SEFReadWithPhysicalAddress1() or SEFReadWithPhysicalAddress1Async(). Any of these fields can be set to -1 to use the default

Table 14.22: Members of SEFReadOverrides

Name	Type	Description
readWeight	uint16_t	Weight to use for read instead of virtual device default
readFMQ	uint8_t	Flash Media Queue to use for read instead of QoS Domain default

14.23 SEFAllocateOverrides

Supplied to override default superblock allocation FMQ and weight.
May be used when calling SEFAllocateSuperBlock() or SEFAllocateSuperBlockAsync(). Any of these fields can be set to -1 to use the default

Table 14.23: Members of SEFAllocateOverrides

Name	Type	Description
eraseWeight	uint16_t	Weight to use for erase instead of virtual device default
eraseFMQ	uint8_t	Flash Media Queue to use for erase instead of QoS Domain default

14.24 SEFCopySource

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

Table 14.24: Members of SEFCopySource

Name	Type	Description
format	enum SEFCopySourceType	Specifies the format to use
reserved_0	uint8_t[3]	
arraySize	uint32_t	Number of items in bitmap array or Flash Address List (QWORD count)

srcFlashAddress	struct SEFFlashAddress	flash address of source block. ADU and 0x3f indicates the ADU of bit 0 of validBitmap and ADU and 0x3f is the starting bit in validBiMap
validBitmap	uint64_t *	pointer to COPY of valid bitmap array (little endian), memory allocated by SEFPrepareBufferForNamelessCopy()
flashAddressList	struct SEFFlashAddress *	pointer to flash address list, memory allocated by SEFPrepareBufferForNamelessCopy()

14.25 SEFUserAddressFilter

Optional filtering on user address data during copy.

Table 14.25: Members of SEFUserAddressFilter

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

14.26 SEFAddressChangeRequest

Address change records.
This structure is used internally to implement SEFProcessChangeAddressRequest(). It may change in the future so it should be treated as opaque.

Table 14.26: Members of SEFAddressChangeRequest

Name	Type
numProcessedADUs	uint32_t
nextADUOffset	uint32_t
numReadErrorADUs	uint32_t
numDefectivePlanes	uint16_t

reserved	uint16_t
startingDstFlashAddress	struct SEFFlashAddress
userAddress	struct SEFUserAddress[0]

14.27 SEFCopyOverrides

Flash Meida Queue overrides for SEFNamelessCopy()
When any of these fields are set to 0, the default weight is used as defined by SEFCreateVirtualDevice() and default FMQ as defined by SEFCreateQoSDomain().

Table 14.27: Members of SEFCopyOverrides

Name	Type	Description
readWeight	uint16_t	Weight to use for read instead of virtual device default
eraseWeight	uint16_t	Weight to use for erase instead of virtual device default
programWeight	uint16_t	Weight to use for program instead of virtual device default
readFMQ	uint8_t	Flash Media Queue to use for read instead of QoS Domain default
programFMQ	uint8_t	Flash Media Queue to use for erase and write instead of QoS Domain default

15 | Callback Structures

15.1 SEFWriteWithoutPhysicalAddressIOCB

Table 15.1: Members of SEFWriteWithoutPhysicalAddressIOCB

Name	Type	Description
status	struct SEFStatus	Library sets error field to a non-zero value to indicate any error when a command completes
opcode	int16_t	Should never be accessed - for internal use by library
done	int16_t	Flag for polled I/O - library sets this field to a non-zero value once the command completes
param1	void *	Ignored by the library; the caller can store context information that may be accessed from the completion function
complete_func	void(*)(struct SEFWriteWithoutPhysicalAddressIOCB *)	If non-zero, treated as the address of a function to be called when a command completes
tentativeAddresses	struct SEFFlashAddress *	List of tentative addresses return
overrides	const struct SEFWriteOverrides *	Override parameters for scheduling purposes, may be NULL
flashAddress	struct SEFFlashAddress	Address of the superblock for this write; -1 for auto-allocate, or can use value from previous superblock allocation call
userAddress	struct SEFUserAddress	Contains LBA information

iov	const struct iovec *	A pointer to the scatter gather list
iovcnt	uint16_t	number of elements in the scatter gather list
placementID	struct SEFPlacementID	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
numADU	uint32_t	Length in ADUs, maximum is 64k ADUs
distanceToEndOfSuperBlock	uint32_t	Return value in units of ADUs

15.2 SEFReadWithPhysicalAddressIOCB

Table 15.2: Members of SEFReadWithPhysicalAddressIOCB

Name	Type	Description
status	struct SEFStatus	Library sets error field to a non-zero value to indicate any error when a command completes
opcode	int16_t	Should never be accessed - for internal use by library
done	int16_t	Flag for polled I/O - library sets this field to a non-zero value once the command completes
param1	void *	Ignored by the library; the caller can store context information that may be accessed from the completion function
complete_func	void(*)(struct SEFReadWithPhysicalAddressIOCB *)	If non-zero, treated as the address of a function to be called when a command completes
overrides	const struct SEFReadOverrides *	Override parameters for scheduling purposes, may be NULL

flashAddress	struct SEFFlashAddress	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.
userAddress	struct SEFUserAddress	Contains LBA information
iov	const struct iovec *	A pointer to the scatter gather list
iovOffset	uint32_t	Starting byte offset into iov array
numADU	uint32_t	Number of ADUs to be read, maximum is superblockCapacity
iovcnt	uint16_t	Number of elements in the scatter gather list

15.3 SEFReleaseSuperBlockIOCB

Table 15.3: Members of SEFReleaseSuperBlockIOCB

Name	Type	Description
status	struct SEFStatus	Library sets error field to a non-zero value to indicate any error when a command completes
opcode	int16_t	Should never be accessed - for internal use by library
done	int16_t	Flag for polled I/O - library sets this field to a non-zero value once the command completes
param1	void *	Ignored by the library; the caller can store context information that may be accessed from the completion function
complete_func	void(*)(struct SEFReleaseSuperBlockIOCB *)	If non-zero, treated as the address of a function to be called when a command completes
ALIGN_FOR_LONG		
flashAddress	struct SEFFlashAddress	Address of superblock

15.4 SEFAllocateSuperBlockIOCB

IOCB for SEFAllocateSuperBlockAsync()

Table 15.4: Members of SEFAllocateSuperBlockIOCB

Name	Type	Description
status	struct SEFStatus	Library sets error field to a non-zero value to indicate any error when a command completes
opcode	int16_t	Should never be accessed - for internal use by library
done	int16_t	Flag for polled I/O - library sets this field to a non-zero value once the command completes
param1	void *	Ignored by the library; the caller can store context information that may be accessed from the completion function
complete_func	void(*)(struct SEFAllocateSuperBlockIOCB *)	If non-zero, treated as the address of a function to be called when a command completes
overrides	const struct SEFAllocateOverrides *	Override parameters for scheduling purposes, may be NULL
flashAddress	struct SEFFlashAddress	Address of superblock
retention	uint32_t	Desired retention period in hours
type	enum SEFSuperBlockType	kForWrite, kForCopy or kForDeviceMetadata

15.5 SEFCloseSuperBlockIOCB

IOCB for SEFCloseSuperBlockAsync()

Table 15.5: Members of SEFCloseSuperBlockIOCB

Name	Type	Description
status	struct SEFStatus	Library sets error field to a non-zero value to indicate any error when a command completes

opcode	int16_t	Should never be accessed - for internal use by library
done	int16_t	Flag for polled I/O - library sets this field to a non-zero value once the command completes
param1	void *	Ignored by the library; the caller can store context information that may be accessed from the completion function
complete_func	void(*)(struct SEFCloseSuperBlockIOCB *)	If non-zero, treated as the address of a function to be called when a command completes
flashAddress	struct SEFFlashAddress	Address of the superblock

15.6 SEFNamelessCopyIOCB

Table 15.6: Members of SEFNamelessCopyIOCB

Name	Type	Description
status	struct SEFStatus	Library sets error field to a non-zero value to indicate any error when a command completes. See SEFNamelessCopy() for details of the info field.
opcode	int16_t	Should never be accessed - for internal use by library
done	int16_t	Flag for polled I/O - library sets this field to a non-zero value once the command completes
param1	void *	Ignored by the library; the caller can store context information that may be accessed from the completion function
complete_func	void(*)(struct SEFNamelessCopyIOCB *)	If non-zero, treated as the address of a function to be called when a command completes
dstQosHandle	SEFQoSHandle	Handle to the destination QoS Domain

copyDestination	struct SEFFlashAddress	Physical address of destination superblock
addressChangeInfo	struct SEFAddressChangeRequest *	Information to record changed addresses
numAddressChangeRecords	uint32_t	Maximum number of ADUs to copy (size of addressChangeRequest userAddress array)
reserved_0	uint32_t	
copySource	struct SEFCopySource	Physical addresses to copy
filter	const struct SEFUserAddressFilter *	Pointer to user address filter parameters, null for no filtering
overrides	const struct SEFCopyOverrides *	Override parameters for scheduling purposes, may be NULL
copyContext	void *	Working buffer returned by SEFPrepareBufferForNamelessCopy()

16 | Events

16.1 SEFQoSNotification

This event is issued at the QoS Domain level.

Table 16.1: Members of SEFQoSNotification

Name	Type	Description
type	enum SEFNotificationType	See union below...
reserved_0	uint8_t[5]	
QoSDomainID	struct SEFQoSDomainID	QoSDomainID for this notification
changedUserAddress	struct SEFUserAddress	User address that moved
oldFlashAddress	struct SEFFlashAddress	Old flash address
newFlashAddress	struct SEFFlashAddress	New flash address
maintenanceFlashAddress	struct SEFFlashAddress	kRequireMaintenance
patrolFlashAddress	struct SEFFlashAddress	kRequirePatrol
userData	char *	pointer to buffered data
unflushedUserAddress	struct SEFUserAddress	affected user address
unreadableFlashAddress	struct SEFFlashAddress	kUnreadable
changedFlashAddress	struct SEFFlashAddress	kSuperblockStateChanged open=>closed

16.2 SEFVDNotification

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

Table 16.2: Members of SEFVDNotification

Name	Type	Description
type	enum SEFNotificationType	Is kReducedCapacity or kOutOfCapacity
reserved_0	uint8_t	
virtualDeviceID	struct SEFVirtualDeviceID	virtual device for this notification
numADUs	uint32_t	kReducedCapacity - Amount of space that is no longer available

17 | Restrictions On Product Use

KIOXIA Corporation and its subsidiaries and affiliates are collectively referred to as “KIOXIA”. Hardware, software and systems described in this document are collectively referred to as “Product”.

- KIOXIA reserves the right to make changes to the information in this document and related Product without notice.
- This document and any information herein may not be reproduced without prior written permission from KIOXIA. Even with KIOXIA’s written permission, reproduction is permissible only if reproduction is without alteration/omission.
- Though KIOXIA works continually to improve Product’s quality and reliability, Product can malfunction or fail. Customers are responsible for complying with safety standards and for providing adequate designs and safeguards for their hardware, software and systems which minimize risk and avoid situations in which a malfunction or failure of Product could cause loss of human life, bodily injury or damage to property, including data loss or corruption. Before customers use the Product, create designs including the Product, or incorporate the Product into their own applications, customers must also refer to and comply with (a) the latest versions of all relevant KIOXIA information, including without limitation, this document, the specifications, the data sheets and application notes for Product and the precautions and conditions set forth in the "Reliability Information" in KIOXIA Corporation’s website and (b) the instructions for the application with which the Product will be used with or for. Customers are solely responsible for all aspects of their own product design or applications, including but not limited to (a) determining the appropriateness of the use of this Product in such design or applications; (b) evaluating and determining the applicability of any information contained in this document, or in charts, diagrams, programs, algorithms, sample application circuits, or any other referenced documents; and (c) validating all operating parameters for such designs and applications. **KIOXIA ASSUMES NO LIABILITY FOR CUSTOMERS’ PRODUCT DESIGN OR APPLICATIONS.**
- **PRODUCT IS NEITHER INTENDED NOR WARRANTED FOR USE IN EQUIPMENTS OR SYSTEMS THAT REQUIRE EXTRAORDINARILY HIGH LEVELS OF QUALITY AND/OR RELIABILITY, AND/OR A MALFUNCTION OR FAILURE OF WHICH MAY CAUSE LOSS OF HUMAN LIFE, BOD-**

- Please contact your KIOXIA sales representative for details as to environmental matters such as the RoHS compatibility of Product. Please use Product in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. **KIOXIA ASSUMES NO LIABILITY FOR DAMAGES OR LOSSES OCCURRING AS A RESULT OF NONCOMPLIANCE WITH APPLICABLE LAWS AND REGULATIONS**.

18 | Trademarks

- PCIe is a registered trademark of PCI-SIG.
- NVMe is a trademark of NVM Express, Inc.
- Linux is a registered trademark of Linus Torvalds in the U.S. and other countries.
- Other company names, product names, and service names may be trademarks of their respective companies.