ioSphere®

ioSphere Management Solution SDK

JANUARY 29, 2014

FUSION-IO

Copyright Notice

Fusion-io, Inc.
2855 E. Cottonwood Parkway
Suite 100
Salt Lake City, UT 84121
USA

(801) 424.5500

Part Number: D0005267-005_4

Published: January 29, 2014

Table of Contents

About the SDK

The ioSphere Management Solution SDK (Software Development Kit) is a C library that contains a set of low-level functions used to manage ioMemory devices. It provides functions to enumerate, query status, monitor, and perform operations on ioMemory devices. It is the basis on which the entire suite of Fusion-io management applications was built.

On Linux/*nix, the **`libvsl-dev`** package contains the SDK header files and libraries to link against. The **`fio_abstraction`** library contained in this package will work regardless of what driver is installed on the system. This means you can build and link your application once, and it will continue to work even if an older version of the driver is un-installed and a newer one is installed in its place. Note that newer versions of the SDK may be released with newer versions of the ioMemory VSL driver, but you are not required to upgrade to the new SDK and rebuild applications unless you want to gain access to the newer APIs.

Use cases for the SDK

Before you begin working with the SDK, you should think carefully about whether using the SDK is necessary for what you are trying to accomplish. Fusion-io already provides several management applications, such as ioManager, SNMP, SMI-S, and WMI, that you can use to manage products in your framework. In most cases, these management apps provide enough functionality that you won't need the SDK for basic tasks.

However, there may be cases where the SDK may be the right choice. For example, a customer has an appliance containing Fusion-io devices and wants to add a button to their appliance management application to detach a device.

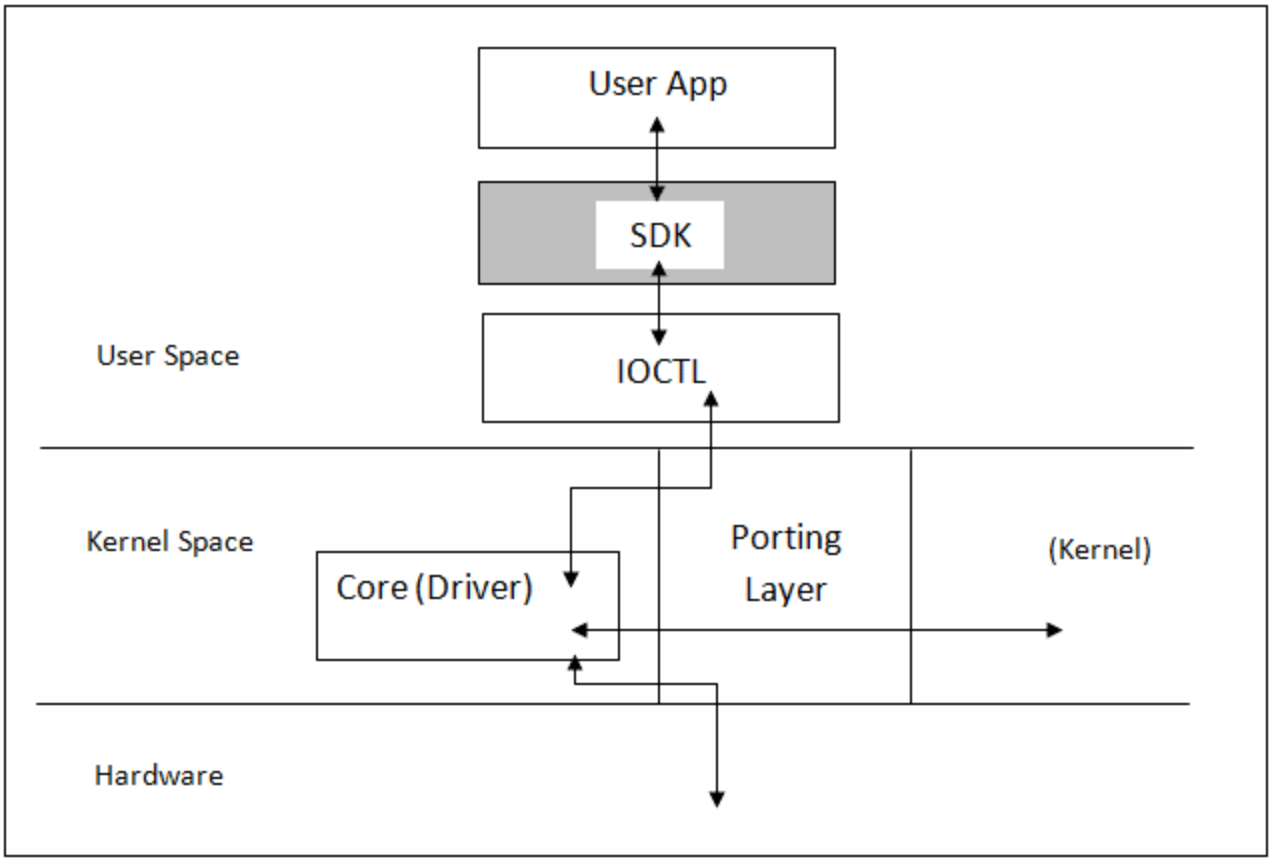
The header files, with structures and functions referred to in this document, are contained in the SDK as sample code for your convenience.

The source codes and supported handles are noted in blue.

SDK architecture

The diagram below shows how the SDK fits into the overall data flow between the user application and the driver. The basic flow works as follows:

1. The user application sends a data request through the SDK.
2. The SDK calls an internal library of IOCTLs to prepare the request to be transmitted to the hardware.
3. The IOCTL command is transferred through the Porting Layer to ensure that it works properly with the existing OS platform.
4. The Fusion-io driver communicates with the OS kernel (through the Porting Layer) and the hardware to satisfy the request.
5. Information is passed back along the path from the driver to the user application, via the SDK.


User App
SDK
IOCTL
User Space
Kernel Space
Core (Driver)
Porting
Layer
(Kernel)
Hardware

Global functions

The functions in this section are global to the entire SDK. They are valid for all or no handle types.

The source code for these functions is the **vsl_global.h** file.

fio_error_code_t on page 4 for a detailed list of error code enumerations,

vsl_init

Description

SDK initialization function
Function must be called prior to calling library APIs.

```
DllExport
fio_error_code_t vsl_init(void* initdata, fio_error_t* error);
```

initdata – initialization data, pass NULL for the initdata

Return

FIO_SUCCESS=0 is success.

vsl_fini

Description

SDK termination function
Once called, the library APIs can no longer be used until **vsl_init** is called again.

```
DllExport
void vsl_fini(void);
```

Return

Nothing is returned with this function.

vsl_destroy

Description

Destroys/deallocates any previously created handle

```
DllExport
fio_error_code_t vsl_destroy(void** handle, fio_error_t* error);
```

Return

FIO_SUCCESS=0 is success.

vsl_is_driver_loaded

Description

Checks if the VSL is loaded

```
DllExport
uint8_t vsl_is_driver_loaded(void);
```

Return

Returns non-zero if the VSL driver is loaded

fio_error_code_t

fio_error_code_t is described in the **vsl_global_types.h** file.

fio_error_code_t is capable of returning values for multiple SDKs. For this version of the Management SDK, the error codes will be one of the **vsl_error_code_t** values enumerated in the **vsl_error.h** file.

vsl_error_code_t Enumerations

FIO_SUCCESS = 0	Success
FIO_ERR_INVALID_HANDLE = 1	Invalid handle
FIO_ERR_DEVICE_NOT_FOUND = 2	Device not found
FIO_ERR_PERM = 3	Permission denied
FIO_ERR_INVALID_ATTR_ID = 4	Invalid attribute **<value>** specified for object **<value>**
FIO_ERR_CANNOT_SET_ATTR = 5	Attribute **<value>** not settable for object <value>
FIO_ERR_OUT_OF_MEMORY = 6	Out of memory
FIO_ERR_GENERIC_FAILURE = 7	Generic failure
FIO_ERR_INVALID_STATE = 8	Device not in a valid state to handle request
FIO_ERR_OPEN_CHANNEL = 9	Failed to communicate with device
FIO_ERR_LOCK_CHANNEL = 10	Failed to lock channel *Make sure the device is detached (by running* **fio-detach**) *and not running another configuration operation.*
FIO_ERR_INTERNAL_FAILURE = 11	Internal failure **<value>**

FIO_ERR_MISSING_REQUIRED_DATA = 12	Missing **<value>** required to perform function **<value>**
FIO_ERR_INITIALIZATION = 13	Initialization failed
FIO_ERR_SYNCH = 14	Synchronization failed
FIO_ERR_OS_NOT_SUPPORTED = 15	Not supported on current OS
FIO_ERR_DISK_WRITE_FAILED = 16	Failed to write file to disk
FIO_ERR_ATTR_NOT_AVAILABLE = 17	**<value>** is not available
FIO_ERR_NOT_IMPLEMENTED = 18	Capability not currently implemented
FIO_ERR_FEATURE_NOT_SUPPORTED = 19	**<value>** feature not supported by the hardware
FIO_ERR_INTERFACE_NOT_SUPPORTED = 20	Object not supported in the interface method
FIO_ERR_LIBRARY = 21	Shared library unavailable or could not be loaded
FIO_ERR_INVALID_LEB = 22	Selected LEB invalid / out of range
FIO_ERR_CANCEL = 23	Canceled
FIO_ERR_STATE_NOT_DETACHED = 24	Device not detached
FIO_ERR_OPERATION_NOT_SUPPORTED = 25	Requested operation not supported on the device *No changes were made to the device.*
FIO_ERR_ALREADY_IN_STATE = 26	Device already in the requested state
FIO_ERR_DRIVER_NOT_LOADED = 27	Driver not loaded
FIO_ERR_DRIVER_LOADED = 28	Driver loaded
FIO_ERR_INVALID_VERSION = 29	API version mismatch between device(s) and Fusion-io driver
FIO_ERR_LOCK_IOCTL = 30	Failed to acquire IOCTL lock
FIO_ERR_NO_EBMAP = 31	Device does not have a persistent LEB map
FIO_ERR_BUSY = 32	Device busy
FIO_ERR_INVALID_SECTOR_SIZE = 33	Specified sector size invalid
FIO_ERR_SDK_INIT = 34	SDK not initialized
FIO_ERR_IO = 35	Failed to read from or write to the device
FIO_ERR_INVALID_INPUT = 36	Invalid input
FIO_ERR_MISSING_NV_DATA = 37	NV-data scan skipped during VSL driver load

FIO_ERR_FORMAT_NONZERO_FORMAT_SIZE = 38	Proposed format size of zero not allowed *Specify a valid format size.*
FIO_ERR_FORMAT_MAX_SIZE_EXCEEDED = 39	Proposed format size **<value> <value>** larger than the default size **<value> <value>** *Specify a valid format size.*
FIO_ERR_FORMAT_MIN_SIZE_VIOLATED = 40	Minimum format size **<value> <value>** *Specify a larger format size.*
FIO_ERR_FORMAT_ONLY_DEFAULT = 41	Device only supports default formatting mode
FIO_ERR_FORMAT_OVERFORMAT_NOT_SUPPORTED = 42	Device does not support overformatting
FIO_ERR_FORMAT_SECTOR_SIZE_MULTIPLIER = 43	Sector size **<value>** bytes not matching required multiplier **<value>**
FIO_ERR_FORMAT_SECTOR_SIZE_POWER_OF_TWO = 44	Sector size **<value>** bytes not a power of two
FIO_ERR_FORMAT_SECTOR_SIZE_OUT_OF_RANGE = 45	Sector size **<value>** bytes out of range *Sector size must be between* **<value>** *and* **<value>** *bytes*.
FIO_ERR_FORMAT_NO_RESERVE = 46	Not enough reserve space available to support requested format size and sector size *Specify a smaller format size and/or larger sector size.*
FIO_ERR_ATTACH_IMPROPER_VERSION = 47	Unsupported data format version *Extract old data using your previous driver, re-format and then attach and restore your data.*
FIO_ERR_ATTACH_IS_DIRTY = 48	Failed because a clean only attach was specified *Full scan required.*
FIO_ERR_UPDATE_VERSION_UNAVAILABLE = 49	Unable to retrieve current firmware version from the device
FIO_ERR_UPDATE_FFF_FILE_NOT_FOUND = 50	Invalid **fff** path *File not found*
FIO_ERR_UPDATE_INVALID_FFF = 51	Invalid **fff** file
FIO_ERR_UPDATE_PART_NOT_FOUND = 52	No firmware for part in the **fff** file
FIO_ERR_UPDATE_ALREADY_UP_TO_DATE = 53	Firmware already up to date
FIO_ERR_UPDATE_BARRIER = 54	Firmware upgrade from current firmware not allowed
FIO_ERR_UPDATE_ECC_MISMATCH = 55	ECC for the current firmware does not match with the upgrade

FIO_ERR_UPDATE_SMP_NOT_FOUND = 56	Unable to communicate with the SMP
FIO_ERR_UPDATE_PROGRAMMING_FAILURE = 57	Firmware programming failure
FIO_ERR_UPDATE_INVALID_FIRMWARE_TYPE = 58	Firmware not a supported type
FIO_ERR_UPDATE_SMP_FAILURE = 59	SMP programming failure
FIO_ERR_HOST_TRIM_CANT_MODIFY_SETTINGS = 60	Unable to modify registry settings for the trim service
FIO_ERR_HOST_TRIM_CANT_MODIFY_SERVICE = 61	Unable to modify the status of the trim service
FIO_ERR_MEL_FILENAME_REQUIRED = 62	File name was not specified
FIO_ERR_MEL_INCOMPATIBLE = 63	MEL input incompatible with current driver version
FIO_ERR_MEL_TRANSFER = 64	Failed to transfer EB map to driver memory
FIO_ERR_MEL_PERSIST = 65	Failed to persist EB map on device
FIO_ERR_SURE_ERASE_FAIL_FLASH = 66	Could not clear flash data *Make sure the device is detached first by running* **fio-detach.**
FIO_ERR_SURE_ERASE_FAIL_MIDPROM = 67	Could not clear identification firmware
FIO_ERR_SURE_ERASE_FAIL_FIRMWARE = 68	Could not clear firmware
FIO_ERR_SURE_ERASE_FAIL_REPORT = 69	Could not open report file
FIO_ERR_SURE_ERASE_FAIL_BACKUP_LEB = 70	Could not backup LEB map
FIO_ERR_SURE_ERASE_FAIL_CLEAR_LEB = 71	Could not clear LEB map
FIO_ERR_STATE_NOT_ATTACHED = 72	Device not attached
FIO_ERR_BUFFER_TOO_SMALL = 73	Caller did not allocate a large enough buffer for the return
FIO_ERR_PPS_MISSING = 74	Device **<value>** not properly configured No PPS found on the device
FIO_ERR_PPS_INVALID = 75	Configuration area found but appears corrupted
FIO_ERR_PPS_FAIL = 76	PPS operation to the device **<value>** failed
FIO_ERR_PPS_VER = 77	Unexpected version of the PPS found on the device **<value>** *Upgrade to the latest driver and software versions and try again.*

FIO_ERR_PPS_FULL = 78	Area for storing configuration full *Cannot store new parameters*
FIO_ERR_PPS_ENTRY_MISSING = 79	Requested entry not found in the configuration area
FIO_ERR_PPS_BAD_ENTRY = 80	Requested PPS entry appears corrupted
FIO_ERR_PPS_BAD_KEY = 81	Malformed key used to attempt a request for data in the PPS
FIO_ERR_VSU_CREATE_ERR = 82	VSU factory could not be instantiated
FIO_ERR_VSU_DELETE_FAIL = 83	VSU could not be deleted
FIO_ERR_FORMAT_FIXED_PCT_VIOLATED = 84	ioMemory device has a fixed format percentage and cannot be formatted to the specified size
FIO_ERR_OBJECT_NOT_FOUND = 85	Object **<value>** not found on object <value>
FIO_ERR_UNSUPPORTED_SIZE = 86	Specified size not supported
FIO_ERR_CSR_LOCKED = 87	The PID: **<value>** currently holds a lock on the device in slot **<value>**
FIO_ERR_UPDATE_PIPE_NOT_SUPPORTED = 88	Device does not support firmware update
FIO_ERR_UPDATE_OPTROM_NOT_FOUND = 89	Firmware does not support option ROM
FIO_ERR_UPDATE_OPTROM_WRITE_FAILURE = 90	Option ROM programming failure
FIO_ERR_FORMAT_FAST_RESCAN_REQUIRED = 91	Fast rescan support required to support requested features
FIO_ERR_UPDATE_MULTIPIPE_PART_NOT_FOUND = 92	Device does not support virtual controllers with the firmware *No changes were made to the device.*
FIO_ERR_UPDATE_MULTIPIPE_UNSUITABLE_ CARD = 93	Card too worn or otherwise unsuitable for a virtual controller change
FIO_ERR_UPDATE_MULTIPIPE_NOT_MERGEABLE = 94	Unable to find all of the virtual controllers for a merge
FIO_ERR_FORMAT_MULTIPIPE_UPGRADE_ FAILURE = 95	Completion of a virtual controller change failed
FIO_ERR_MULTIPIPE_INVALID_STATE = 96	Related device **<value>** not in a valid state to handle request
FIO_ERR_UPDATE_MULTIPIPE_BOTH_DUOS_ REQUIRED = 97	Both adapters of a duo must be converted

Clear cache function

The clear cache function is used to clear the cache from any of the following handles. When you use a function to get the value of a field, the value is returned from the handle's cache (if available). Otherwise, the value is retrieved and stored in the cache. It will then return the same value in all subsequent calls to this function until the clear function is called or the handle is destroyed.

This function works with the following handle functions:

- **iom_handle_t** – ioMemory handle functions on page 10.
- **adapter_handle_t** – Adapter handle functions on page 13
- **vsu_handle_t** – VSU handle functions on page 13.
- **cluster_handle_t** – Cluster handle functions on page 117.
- **host_handle_t** – Host handle functions on page 120.

fio_error_code_t on page 4 for a detailed list of error code enumerations.

vsl_clear_cache

Description

Clears any cached data from the handle

```
DllExport
fio_error_code_t vsl_clear_cache(void* handle, fio_error_t* error);
```

Return

FIO_SUCCESS=0 is success.

Device handle functions

To manage the ioMemory device, you must create a handle that is related to the device.

This section describes the functions used to create handles for:

- **ioMemory devices:** Your ioMemory device consists of one or more ioMemory devices. Host handle functions on page 120.
- **Adapters:** Some ioMemory devices have adapters, as described above. Adapter handle functions on page 13.
- **VSUs:** VSUs represent the block device (**`/dev/fioa`**) that is created when an ioMemory device is attached to an operating system. VSU handle functions on page 13.

> ⚠ The ioDrive Duo device consists of two ioMemory devices, and one adapter for interfacing with both devices.

ioMemory handle functions

These functions deal with the **`iom_handle_t`** handle. This handle type is used in many functions and is used to create other handles.

The source code for these functions is the **`iom.h`** file.

fio_error_code_t on page 4 for a detailed list of error code enumerations.

vsl_iom_create_by_name

Description

Create a handle to the ioMemory device from a device path
Ex: **`"/dev/fct0"`**

```
DllExport
fio_error_code_t vsl_iom_create_by_name(const char* iom_path, iom_handle_t*
handle, fio_error_t* error);
```

Return

`FIO_SUCCESS=0` is success.

vsl_iom_create_by_copy

Description

Create a deep copy of an **`iom`** handle

```
DllExport
fio_error_code_t vsl_iom_create_by_copy(iom_handle_t input_handle, iom_handle_t*
handle, fio_error_t* error);
```

Return

FIO_SUCCESS=0 is success.

vsl_iom_create_by_index

Description

Create a handle to an ioMemory device from an index
Ex: 0 == **"/dev/fct0"**

```
DllExport
fio_error_code_t vsl_iom_create_by_index(uint32_t index, iom_handle_t* handle,
fio_error_t* error);
```

Return

FIO_SUCCESS=0 is success.

vsl_iom_create_by_vsu

Description

Create a handle to an ioMemory device from a VSU handle

```
DllExport
fio_error_code_t vsl_iom_create_by_vsu(vsu_handle_t vsu_handle, iom_handle_t*
handle, fio_error_t* error);
```

Return

FIO_SUCCESS=0 is success.

vsl_compare_iom

Description

Compares two ioMemory device handles

```
DllExport
int32_t vsl_compare_iom(void* lhs, void* rhs);
```

* **lhs** – will be compared to **rhs**

* **rhs** – will be compared to **lhs**

Return

Negative value if **lhs** is greater than **rhs**, 0 if equal, positive if **rhs** is greater than **lhs**

vsl_iom_iterate

Description

Iterate over all ioMemory device in system

* **handle** must be set to NULL to signify start of iteration.

```
DllExport
uint8_t vsl_iom_iterate(iom_handle_t* handle, uint8_t free, fio_error_t* error);
```

- If *free* is non-zero, the handle stored in handle will be deleted before being replaced.
- If *free* is zero, handle will not be deleted, and caller is responsible for calling the **vsl_destroy** function when done using handle.
- If iteration is stopped before the end, caller is responsible for calling the **vsl_destroy** function on last returned handle.

vsl_iom_get_raw_handle

Description

Get the (low level library) raw handle, if handle is currently open

```
DllExport
fio_error_code_t vsl_iom_get_raw_handle(iom_handle_t handle, int* raw_handle,
fio_error_t* error);
```

Return

FIO_SUCCESS=0 is success.

vsl_iom_set_gather_os_storage_fields

Description

Configures if the iom object will access OS storage objects to initialize OS storage related fields
This function can be used to prevent extra overhead. The default is **enabled (1).**

```
DllExport
fio_error_code_t vsl_iom_set_gather_os_storage_fields(uint8_t enabled, fio_
error_t* error);
```

Return

FIO_SUCCESS=0 is success.

Adapter handle functions

These functions deal with the **iom_handle_t** handle.

The source code for these functions is the **adapter.h** file.

fio_error_code_t on page 4 for a detailed list of error code enumerations.

vsl_adapter_create

Description

Creates a handle representing an adapter that one or more ioMemory devices are plugged into

```
DllExport
fio_error_code_t vsl_adapter_create(iom_handle_t iom, adapter_handle_t* handle,
fio_error_t* error);
```

Return

FIO_SUCCESS=0 is success.

vsl_compare_adapter

Description

Compares two adapter handles

```
DllExport
int32_t vsl_compare_adapter(void* lhs, void* rhs);
```

* **lhs** - will be compared to **rhs**

* **rhs** - will be compared to **lhs**

Return

Negative value if **lhs** is greater than **rhs**, 0 if equal, positive if **rhs** is greater than **lhs**

VSU handle functions

These functions deal with the **vsu_handle_t** handle.

The source code for these functions is the **vsu.h** file.

ioMemory handle functions on page 10 for more information.

vsl_vsu_iterate

Description

Iterate over all VSU in the ioMemory device

```
DllExport
uint8_t vsl_vsu_iterate(iom_handle_t parent_handle, vsu_handle_t* handle, uint8_
t free, fio_error_t* error);
```

* **handle** – must be set to NULL to signify start of iteration.

* **parent_handle** – should point to a valid iom handle.

vsl_vsu_create_by_copy

Description

Copies an existing handle and saves it as a new handle

```
DllExport
fio_error_code_t vsl_vsu_create_by_copy(vsu_handle_t input_handle, vsu_handle_t*
handle, fio_error_t* error);
```

Return

FIO_SUCCESS=0 is success.

vsl_compare_vsu

Description

Compares two VSU handles

```
DllExport
int32_t vsl_compare_vsu(void* lhs, void* rhs);
```

* **lhs** – will be compared to **rhs**

* **rhs** – will be compared to **lhs**

Return

Negative value if **lhs** is greater than **rhs**, 0 if equal, positive if **rhs** is greater than **lhs**

Openable functions

The following functions work with **iom_handle_t** and **adapter_handle_t** handles.

The source code for these functions is the **vsl_abstract.h** file.

ioMemory handle functions on page 10 for more information.

vsl_open

Description

Opens previously created handle

```
DllExport
fio_error_code_t vsl_open(void* handle, fio_error_t* error);
```

Return

FIO_SUCCESS=0 is success. ioMemory handle functions on page 10 for an enumerated list.

vsl_close

Description

Closes previously opened handle

```
DllExport
fio_error_code_t vsl_close(void* handle, fio_error_t* error);
```

Return

FIO_SUCCESS=0 is success.

vsl_is_handle_open

Description

Reports if the handle is currently open

```
DllExport
uint8_t vsl_is_handle_open(void* handle, fio_error_t* error);
```

Return

Returns true if handle is currently open, false otherwise

Persistent path functions

The following function(s) work with **iom_handle_t** and **vsu_handle_t** handles.

The source code for these function(s) is the **vsl_abstract.h** file.

ioMemory handle functions on page 10 for more information.

vsl_persistent_paths

Description

Get persistent paths for the device
Paths is an array of pointers to strings and will be populated with **[count]** entries. The paths returned is memory that is owned by the parent handle and should not be held long term. Rather, a copy should immediately be made.

```
DllExport
fio_error_code_t vsl_persistent_paths(void* handle, const char*** paths, uint16_
t* count, fio_error_t* error);
```

Return

FIO_SUCCESS=0 is success.

Device status fields

This section describes the fields available for displaying the status of ioMemory devices. These fields may be compatible with one or more of the Device Handles:

```
iom_handle_t
adapter_handle_t
vsu_handle_t
```

Device handle functions on page 10 for more information.

Status information fields

The source file for each field is noted in the description for that field, and the handle (or handles) that works with the field functions are listed below the functions.

The following fields may have a **HAS**, **GETTER**, and/or **SETTER** function. **HAS** and **GETTER** functions exist if the field is readable. **SETTER** exists if the field is writable.

The following describes the parameters and return values for each function type:

HAS Function

Description

Checks that the value is available in the object

* **handle** – The handle of the object
* **error** – An error return structure

Return

The error code if the field is not available. **FIO_SUCCESS=0** is success.

GETTER Function

Description

Returns the value for the field

When this function is called, the value is returned from the handle's cache if available, otherwise the value is retrieved and stored in the cache. It will then return the same value in all subsequent calls to this function, until the clear function is called or the handle is destroyed.

* **handle** – The handle of the object
* **error** – An error return structure

Return

The value of the field
If there was an error, the default value for this type will be returned. Users of this function should either call the HAS function first, or check the error return structure before using this value.

SETTER Function

Description

Sets the value for the field

When this function is called, the value is set and stored in the handle's cache.

* **`handle`** – The handle of the object
* **`value`** – The value set
* **`error`** – An error return structure

Return

FIO_SUCCESS=0 is success
The error code if the field was not able to be set.

For additional information:

- ioMemory handle functions on page 10
- Adapter handle functions on page 13
- VSU handle functions on page 13

active_media_pct

Description

Active media percentage

The source code for this field is the **`iom.h`** file.

HAS Function

```
DllExport
uint8_t vsl_has_active_media_pct(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
float vsl_active_media_pct(void* handle, fio_error_t* error);
```

Supported Handle(s)

`iom_handle_t`

client_progress_pct

Description

If the driver's active client is performing an operation, this field will show the progress.Otherwise, the field will be unavailable.

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_client_progress_pct(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint16_t vsl_client_progress_pct(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

current_errors

Description

Currently active errors on the the ioMemory device (bitset of **iom_errors_t**)

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_current_errors(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint64_t vsl_current_errors(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

iom_errors_t Enumeration

IOM_ERR_NONE = 0x00000000	None

IOM_ERR_FAILED_STATE = 0x00000001	The ioMemory device encountered an internal error and has been temporarily disabled *All reads and writes will fail.*
IOM_ERR_SLOT_BANDWIDTH = 0x00000002	The bandwidth of the PCI slot is incompatible with the ioMemory device
IOM_ERR_WRITE_THROTTLING_PARTIAL = 0x00000004	The ioMemory device has reduced its write performance
IOM_ERR_WRITE_THROTTLING_COMPLETE = 0x00000008	The ioMemory device is not allowing write operations
IOM_ERR_TEMPERATURE = 0x00000010	The temperature of the device is at a critical threshold. *All reads and writes will fail.*
IOM_ERR_OVERTEMPERATURE = 0x00000020	The temperature of this device has surpassed a critical threshold. All reads and writes will fail.
IOM_ERR_VCCINT = 0x00000040	VCCint voltage is out of range *All reads and writes will fail.*
IOM_ERR_VCCAUX = 0x00000080	VCCaux voltage is out of range *All reads and writes will fail.* *Flashback error to support management software written against VSL API abstracted against* **libfio** *(2.x drivers). 3.x drivers will never return this error.*
IOM_ERR_FLASHBACK = 0x00000100	The ioMemory device has exhausted its flashback protection *Promptly replace the device after backing up its data.*
IOM_ERR_PCIE_FATAL_ERRORS = 0x00000200	PCIe non-correctable errors encountered.

current_warnings

Description

Currently active warnings on the the ioMemory device (bitset of **iom_warnings_t**)

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_current_warnings(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint64_t vsl_current_warnings(void* handle, fio_error_t* error);
```

Supported Handle(s)

`iom_handle_t`

iom_warnings_t Enumeration

`IOM_WARN_NONE = 0x00000000`	None
`IOM_WARN_TEMPERATURE = 0x00000001`	The temperature of the device is approaching a critical threshold
`IOM_WARN_RESERVES = 0x00000002`	The ioMemory device is close to wearing out - reduced-write mode is triggered when reserve is depleted *Formatting to a smaller size will free up reserve.*
`IOM_WARN_SLOT_BANDWIDTH = 0x00000004`	The bandwidth of the PCI slot is not optimal for the ioMemory device
`IOM_WARN_PCIE_ERRORS = 0x00000008`	PCIe correctable errors were encountered
`IOM_WARN_PCI_POWERLOSS_PROT = 0x00000010`	Power loss protection has been disabled on the device, introducing the risk of data corruption in the event of a power failure
`IOM_WARN_WRITE_REG_POWER = 0x00000020`	Power write governing activated, performance may be limited. If this condition persists, switch to a higher powered PCIe slot or attach external power cable
`IOM_WARN_WRITE_REG_THERMAL = 0x00000040`	Thermal write governing activated, performance may be limited *If this condition persists, increase air flow, lower room temperature or reduce write load.*
`IOM_WARN_WRITE_REG_LIFESPAN = 0x00000080`	Write lifespan governing activated, performance may be limited I*f this condition persists, reduce write load or consider an ioMemory device with a higher write volume rating.*
`IOM_WARN_MINIMAL_STATE = 0x00000100`	The ioMemory device is currently running in a minimal state
`IOM_WARN_OVERPOWER = 0x00000200`	Over PCIe power budget alarm triggered
`IOM_WARN_MISSING_LEB_MAP = 0x00000400`	The ioMemory device is missing a LEB map
`IOM_WARN_UPGRADE_IN_PROGRESS = 0x00000800`	A media upgrade is in progress T*he ioMemory device will not be usable until it is low-*

	level formatted.
`IOM_WARN_RESERVES_DEPLETED = 0x00001000`	The ioMemory device reserve is depleted – reduced-write or read-only mode will be triggered when device is attached *Formatting to a smaller size will free up reserve.*

has_pcie_correctable_errors

Description

Reports if there have been correctable PCI errors on the device

The source code for this field is the **`vsl_abstract.h`** file.

HAS Function

```
DllExport
uint8_t vsl_has_has_pcie_correctable_errors(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_has_pcie_correctable_errors(void* handle, fio_error_t* error);
```

Supported Handle(s)

`iom_handle_t`
`adapter_handle_t`

has_pcie_errors

Description

Reports if there have been any type of PCI errors on the device

The source code for this field is the **`vsl_abstract.h`** file.

HAS Function

```
DllExport
uint8_t vsl_has_has_pcie_errors(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_has_pcie_errors(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
adapter_handle_t

has_pcie_fatal_errors

Description

Reports if there have been fatal PCI errors on the device

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_has_pcie_fatal_errors(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_has_pcie_fatal_errors(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
adapter_handle_t

has_pcie_nonfatal_errors

Description

Reports if there have been non-fatal PCI errors on the device

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_has_pcie_nonfatal_errors(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_has_pcie_nonfatal_errors(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
adapter_handle_t

has_pcie_unrecognized_reqs

Description

Reports if there have been unrecognizable PCI requests on the device

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_has_pcie_unrecognized_reqs(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_has_pcie_unrecognized_reqs(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
adapter_handle_t

iom_state

Description

State of the ioMemory VSL driver for the ioMemory device

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_iom_state(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
iom_state_t vsl_iom_state(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

iom_state_t Enumeration

IOM_STATE_DETACHED = 0	Device has not scanned internal metadata, VSU unavailable

IOM_STATE_ATTACHING = 1	Device is scanning internal metadata
IOM_STATE_ATTACHED = 2	Device has scanned internal metadata, VSU available
IOM_STATE_DETACHING = 3	Device is tearing down internal metadata
IOM_STATE_BUSY = 4	Device is busy performing an operation
IOM_STATE_MINIMAL = 5	Device is running in a minimal state minimal_mode_reason on page 25
IOM_STATE_FAILED = 6	Device has had an internal failure and is not operational
IOM_STATE_SHUTDOWN = 7	Device is shut down

minimal_mode_reason

Description

Repots if the state is minimal mode, contains the reason (bitset of **iom_minimal_mode_reason_t**)

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_minimal_mode_reason(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint64_t vsl_minimal_mode_reason(void* handle, fio_error_t* error)
```

This is a bit field composed of values from **iom_minimal_mode_reason_t.**

Supported Handle(s)

iom_handle_t

iom_minimal_mode_reason_t Enumeration

IOM_MMR_UNKNOWN = 0x00000000	Reason unknown
IOM_MMR_FW_OUT_OF_DATE = 0x00000001	Firmware is out of date
IOM_MMR_INSUFFICIENT_POWER = 0x00000002	Adapter power needs to be plugged in
IOM_MMR_DUAL_PLANE_FAIL = 0x00000004	Could not enter dual-plane mode, although user-specified

IOM_MMR_FORCED = 0x00000008	Forced into minimal mode by the user (module parameter)
IOM_MMR_INTERNAL = 0x00000010	Internal error forced the device into minimal mode
IOM_MMR_CARD_LIMIT_EXCEEDED = 0x00000020	Product-specific device limit has been exceeded in the machine
IOM_MMR_UNSUPPORTED_OS = 0x00000040	The device will not function on the operating system
IOM_MMR_INSUFFICIENT_MEMORY = 0x00000080	There is not enough memory to load the driver
IOM_MMR_SMP_BOOTLOADER_MODE = 0x00000100	SMP is in bootloader mode
IOM_MMR_MISSING_MIDPROM = 0x00000200	Missing mid
IOM_MMR_UNSUPPORTED_NAND = 0x00000400	Unsupported NAND
IOM_MMR_DRIVER_OUT_OF_DATE = 0x00000800	Driver is out of date
IOM_MMR_HARDWARE_FAILURE = 0x00001000	Hardware failure
IOM_MMR_GENERAL_CHANNEL_INIT_FAIL = 0x00002000	General Channel Initialization failure

pcie_bandwidth_compatibility

Description

PCIe link bw compatibility of parent adapter in the motherboard slot

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_pcie_bandwidth_compatibility(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
pci_slot_compatibility_t vsl_pcie_bandwidth_compatibility(void* handle, fio_
error_t* error);
```

Supported Handle(s)

iom_handle_t
adapter_handle_t

pci_slot_compatibility_t Enumeration

SC_INCOMPATIBLE = 0	Insufficient to run
SC_SUBOPTIMAL = 1	Sufficient to run, but with possibly degraded performance
SC_OPTIMAL = 2	The slot is providing sufficient for maximum performance

pcie_bandwidth_mbps

Description

Total bandwidth of the PCIe link (lanes * speed factoring in transfer overhead), in MBytes/sec

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_pcie_bandwidth_mbps(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
float vsl_pcie_bandwidth_mbps(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
adapter_handle_t

pcie_link_lanes

Description

Current number of PCIe lanes negotiated

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_pcie_link_lanes(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExportuint8_t vsl_pcie_link_lanes(void* handle, fio_error_t* error);
```

Supported Handle(s)

`iom_handle_t`
`adapter_handle_t`

pcie_link_speed_gtps

Description

Current negotiated linkspeed value in GT/sec

The source code for this field is the `vsl_abstract.h` file.

> ⚠ This link speed is measured in transfers per second, not bits.

HAS Function

```
DllExport
uint8_t vsl_has_pcie_link_speed_gtps(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
pcie_link_speed_t vsl_pcie_link_speed_gtps(void* handle, fio_error_t* error);
```

Supported Handle(s)

`iom_handle_t`
`adapter_handle_t`

pcie_link_speed_t Enumeration

`PCIE_LS_2_5 = 0`	PCIe (1.x) link speed is 2.5 GT/sec per lane
`PCIE_LS_5_0 = 1`	PCIe (2.0) link speed is 5.0 GT/sec per lane
`PCIE_LS_8_0 = 2`	PCIe (3.0) link speed is 8.0 GT/sec per lane

vsu_state

Description

State of the VSU (block device)

The source code for this field is the `vsu.h` file.

HAS Function

```
DllExport
uint8_t vsl_has_vsu_state(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
vsu_state_t vsl_vsu_state(void* handle, fio_error_t* error);
```

SETTER Function

```
DllExport
fio_error_code_t vsl_set_vsu_state(void* handle, vsu_state_t value, fio_error_t*
error);
```

Supported Handle(s)

`iom_handle_t`

vsu_state_t Enumeration

`VSU_STATE_OFFLINE = 0`	VSU is offline
`VSU_STATE_ONLINE = 1`	VSU is online

Identification and configuration information fields

The source file for each field is noted in the description for that field, and the handle (or handles) that works with the field functions are listed below the functions.

The following fields may have a **`HAS`**, **`GETTER`**, and/or **`SETTER`** function. **`HAS`** and **`GETTER`** functions exist if the field is readable. **`SETTER`** exists if the field is writable.

The following describes the parameters and return values for each function type:

HAS Function

Description

Checks that the value is available in the object

* **`handle`** – The handle of the object
* **`error`** – An error return structure

Return

The error code if the field is not available. **`FIO_SUCCESS=0`** is success.

GETTER Function

Description

Returns the value for the field

When this function is called, the value is returned from the handle's cache if available, otherwise the value is retrieved and stored in the cache. It will then return the same value in all subsequent calls to this function, until the clear function is called or the handle is destroyed.

* **`handle`** – The handle of the object
* **`error`** – An error return structure

Return

The value of the field
If there was an error, the default value for this type will be returned. Users of this function should either call the HAS function first, or check the error return structure before using this value.

SETTER Function

Description

Sets the value for the field

When this function is called, the value is set and stored in the handle's cache.

* **`handle`** – The handle of the object
* **`value`** – The value set
* **`error`** – An error return structure

Return

FIO_SUCCESS=0 is success
The error code if the field was not able to be set.

For additional information:

- ioMemory handle functions on page 10
- Adapter handle functions on page 13
- VSU handle functions on page 13

atomic_writes_available

Description

Reports if atomic writes feature is available on the device

The source code for this field is the **`iom.h`** file.

HAS Function

```
DllExport
uint8_t vsl_has_atomic_writes_available(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_atomic_writes_available(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

beacon_state

Description

State of the beacon
If non-zero, all LED's on the card will be on, otherwise LED's function normally. This field is settable and may be used to turn beacon on/off.

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_beacon_state(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_beacon_state(void* handle, fio_error_t* error);
```

SETTER Function

```
DllExport
fio_error_code_t vsl_set_beacon_state(void* handle, uint8_t value, fio_error_t*
error);
```

Set to 1 to turn the beacon on.

Supported Handle(s)

iom_handle_t

board_kind

Description

Kind of board

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_board_kind(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
board_kind_t vsl_board_kind(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
adapter_handle_t

board_kind_t Enumeration

BOARD_KIND_SINGLE_ADAPTER = 0	
BOARD_KIND_DUAL_ADAPTER = 1	
BOARD_KIND_IOSAN_ADAPTER = 2	
BOARD_KIND_OCTAL_BASE_ADAPTER = 3	
BOARD_KIND_OCTAL_MEZZ_ADAPTER = 4	
BOARD_KIND_IODIMM = 5	
BOARD_KIND_IOMONO = 6	
BOARD_KIND_IOXTREME = 7	
BOARD_KIND_HP_MEZZ = 8	
BOARD_KIND_CONTROLLER = 9	
BOARD_KIND_HP_A3_ADAPTER = 10	
BOARD_KIND_2_5_INCH_FUSED = 11	
BOARD_KIND_NAND_MODULE = 12	
BOARD_KIND_DUAL_CONTROLLER_ADAPTER = 13	
BOARD_KIND_SINGLE_CONTROLLER_ADAPTER = 14	Single ctrl adapter differs from ioMono in flash pack capability vs built in flash
BOARD_KIND_CISCO_MEZZ = 15	

board_name

Description

Human readable name of board (8-bit ASCII)

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_board_name(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
const char* vsl_board_name(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
adapter_handle_t

ctrl_dev_path

Description

Path to the control device

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_ctrl_dev_path(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
const char* vsl_ctrl_dev_path(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

device_label

Description

The device label, printed in all log messages

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_device_label(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
const char* vsl_device_label(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

device_name

Description

Device name
Ex: will be set to **"fct0"** for **/dev/fct0**

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_device_name(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
const char* vsl_device_name(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
vsu_handle_t

device_path

Description

Device path
Ex: **/dev/fct1, /dev/sda, /dev/sda1**

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_device_path(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
const char* vsl_device_path(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
vsu_handle_t

external_power_requirement

Description

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_external_power_requirement(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
ext_power_reqt_t vsl_external_power_requirement(void* handle, fio_error_t*
error);
```

Supported Handle(s)

iom_handle_t
adapter_handle_t

ext_power_reqt_t Enumeration

EXT_PWR_NONE = 0	
EXT_PWR_OPTIONAL = 1	
EXT_PWR_REQUIRED = 2	

fio_serial_number

Description

Serial number or id of entire product assembly

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_fio_serial_number(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
const char* vsl_fio_serial_number(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
adapter_handle_t

hardware_generation

Description

The hardware generation of the ioMemory device

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_hardware_generation(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint16_t vsl_hardware_generation(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

has_adapter

Description

Reports if the board has an adapter

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_has_adapter(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_has_adapter(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
adapter_handle_t

iom_capabilities

Description

Capabilities of the ioMemory device

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_iom_capabilities(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint64_t vsl_iom_capabilities(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

iom_capabilities_t Enumeration

IOM_CAPABILITY_NONE = 0x00000000	None Reports if the ioMemory device can update its firmware.
IOM_CAPABILITY_UPDATE_FW = 0x00000001	Update firmware

location_in_adapter

Description

Human readable string for location within adapter

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_location_in_adapter(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
const char* vsl_location_in_adapter(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

nand_module_count

Description

Derived count of nand modules based on value in **nand_module_mask**

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_nand_module_count(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint16_t vsl_nand_module_count(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

nand_module_mask

Description

Bitmask of shipped nand modules (bitset of **nand_module_presence_t**)

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_nand_module_mask(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint64_t vsl_nand_module_mask(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

nand_module_presence_t Enumeration

IOM_NAND_MODULE_NONE = 0x00	None
IOM_NAND_MODULE_0 = 0x01	NM 0
IOM_NAND_MODULE_1 = 0x02	NM 1
IOM_NAND_MODULE_2 = 0x04	NM 2

num_expected_iom

Description

The number of expected devices in the adapter

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_num_expected_iom(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint16_t vsl_num_expected_iom(void* handle, fio_error_t* error);
```

Supported Handle(s)

vsu_handle_t

numeric_id

Description

Numeric ID of the VSU
This ID is not guaranteed to remain constant across detach/attach cycle.

The source code for this field is the **vsu.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_numeric_id(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint64_t vsl_numeric_id(void* handle, fio_error_t* error);
```

Supported Handle(s)

vsu_handle_t

oem

Description

The OEM

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_oem(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
fio_oem_t vsl_oem(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
adapter_handle_t

fio_oem_t Enumeration

OEM_FIO = 0	
OEM_HP = 1	
OEM_IBM = 2	
OEM_SUN = 3	
OEM_ORACLE = 4	
OEM_NEC = 5	

OEM_SUPERMICRO = 6	
OEM_DELL = 7	
OEM_CISCO = 8	

oem_part_number_replacement

Description

OEM replacement part number

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_oem_part_number_replacement(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
const char* vsl_oem_part_number_replacement(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
adapter_handle_t

oem_serial_number

Description

OEM specified serial number (unused for Fusion-io products)

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_oem_serial_number(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
const char* vsl_oem_serial_number(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
adapter_handle_t

oem_sku

Description

OEM SKU (unused for Fusion-io products)

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_oem_sku(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
const char* vsl_oem_sku(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
adapter_handle_t

part_number_legacy

Description

Fusion-io legacy part number (8-bit ASCII)

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_part_number_legacy(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
const char* vsl_part_number_legacy(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
adapter_handle_t

part_number_pa

Description

Fusion-io PA part number of programmed assembly

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_part_number_pa(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
const char* vsl_part_number_pa(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
adapter_handle_t

part_number_sku

Description

Fusion-io SKU or part number of product

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_part_number_sku(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
const char* vsl_part_number_sku(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
adapter_handle_t

pci_addr

Description

PCI address (bus:device.function) (all hex)

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_pci_addr(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
const char* vsl_pci_addr(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

pci_bus

Description

Bus component of PCI address

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_pci_bus(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_pci_bus(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

pci_device

Description

Device component of PCI address

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_pci_device(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_pci_device(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

pci_device_id

Description

PCI Device ID

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_pci_device_id(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint16_t vsl_pci_device_id(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

pci_function

Description

Function component of PCI address

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_pci_function(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_pci_function(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

pci_slot_num

Description

PCI slot number

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_pci_slot_num(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_pci_slot_num(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

pci_subsys_device_id

Description

PCI Subsystem Device ID

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_pci_subsys_device_id(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint16_t vsl_pci_subsys_device_id(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

pci_subsys_vendor_id

Description

PCI Subsystem Vendor ID

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_pci_subsys_vendor_id(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint16_t vsl_pci_subsys_vendor_id(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

pci_vendor_id

Description

PCI Subsystem Vendor ID

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_pci_vendor_id(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint16_t vsl_pci_vendor_id(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

persistent_uid

Description

A unique identifier that remains persistent across reboots, attaches, detaches, etc
The format of this identifier may vary across different OSs.

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_persistent_uid(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
const char* vsl_persistent_uid(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
vsu_handle_t

port_in_adapter

Description

Indicates the ioMemory device's port within the adapter
Reports if the device is the only ioMemory device in the adapter, the value will be set to 0.

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_port_in_adapter(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint16_t vsl_port_in_adapter(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

product_name

Description

Human-readable product name (8-bit ASCII)

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_product_name(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
const char* vsl_product_name(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
adapter_handle_t

product_name_short

Description

Short (< 20 character) version of the product name (8-bit ASCII)

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_product_name_short(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
const char* vsl_product_name_short(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
adapter_handle_t

ptrim_available

Description

Reports if persistent trim feature is available on the device

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_ptrim_available(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_ptrim_available(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

required_pcie_bandwidth_mbps

Description

Required PCIe bandwidth to achieve full performance, in MB/sec

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_required_pcie_bandwidth_mbps(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_required_pcie_bandwidth_mbps(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
adapter_handle_t

serial_number

Description

Serial number or ID

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_serial_number(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
const char* vsl_serial_number(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
adapter_handle_t

virtual_controller_active_count

Description

The number of active virtual controllers on the parent physical ioMemory device

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_virtual_controller_active_count(void* handle, fio_error_t*
error);
```

GETTER Function

```
DllExport
uint8_t vsl_virtual_controller_active_count(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

virtual_controller_configured_count

Description

Reports if multiple virtual controllers are configured and if so how many

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_virtual_controller_configured_count(void* handle, fio_error_t*
error);
```

GETTER Function

```
DllExport
uint8_t vsl_virtual_controller_configured_count(void* handle, fio_error_t*
error);
```

Supported Handle(s)

iom_handle_t

virtual_controller_number

Description

The virtual controller number

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_virtual_controller_number(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint16_t vsl_virtual_controller_number(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

vsl_vsu_set_gather_os_storage_fields

Description

Configures if the VSU object will access OS storage objects to initialize OS storage related fields
Can be used to prevent extra overhead. The default is enabled (1).

The source code for this field is the **vsu.h** file.

SETTER Function

```
DllExport
uint64_t vsl_numeric_id(void* handle, fio_error_t* error);
```

Supported Handle(s)

vsu_handle_t

vsu_type

Description

The type of VSU (block, cache, etc.)

The source code for this field is the **vsu.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_vsu_type(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
vsu_type_t vsl_vsu_type(void* handle, fio_error_t* error);
```

Supported Handle(s)

`vsu_handle_t`

vsu_type_t Enumeration

VSU_TYPE_BLOCK = 0	Format should produce a standard block device
VSU_TYPE_CACHE = 1	Format for use as a cache device

Format information fields

The source file for each field is noted in the description for that field, and the handle (or handles) that works with the field functions are listed below the functions.

The following fields may have a **`HAS`**, **`GETTER`**, and/or **`SETTER`** function. **`HAS`** and **`GETTER`** functions exist if the field is readable. **`SETTER`** exists if the field is writable.

The following describes the parameters and return values for each function type:

HAS Function

Description

Checks that the value is available in the object

* **`handle`** – The handle of the object
* **`error`** – An error return structure

Return

The error code if the field is not available. **`FIO_SUCCESS=0`** is success.

GETTER Function

Description

Returns the value for the field

When this function is called, the value is returned from the handle's cache if available, otherwise the value is retrieved and stored in the cache. It will then return the same value in all subsequent calls to this function, until the clear function is called or the handle is destroyed.

* **handle** – The handle of the object
* **error** – An error return structure

Return

The value of the field
If there was an error, the default value for this type will be returned. Users of this function should either call the HAS function first, or check the error return structure before using this value.

SETTER Function

Description

Sets the value for the field

When this function is called, the value is set and stored in the handle's cache.

* **handle** – The handle of the object
* **value** – The value set
* **error** – An error return structure

Return

FIO_SUCCESS=0 is success
The error code if the field was not able to be set.

For additional information:

- ioMemory handle functions on page 10
- Adapter handle functions on page 13
- VSU handle functions on page 13

eb_index_type

Description

The type of packet indexing drive was formatted with

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_eb_index_type(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
iom_format_eb_index_type_t vsl_eb_index_type(void* handle, fio_error_t* error);
```

Supported Handle(s)

`iom_handle_t`

iom_format_eb_index_type_t Enumeration

IOM_FORMAT_EBIDX_TYPE_NONE = 0	No packet indexing
IOM_FORMAT_EBIDX_TYPE_FULL = 1	Full packet indexing for all packet types
IOM_FORMAT_EBIDX_TYPE_BRIEF = 2	Brief packet indexing for certain system packet types only

factory_size_bytes

Description

The factory capacity on the device when shipped

The source code for this field is the `iom.h` file.

HAS Function

```
DllExport
uint8_t vsl_has_factory_size_bytes(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint64_t vsl_factory_size_bytes(void* handle, fio_error_t* error);
```

Supported Handle(s)

`iom_handle_t`

format_uuid

Description

The UUID given to the volume during low-level fio-format

The source code for this field is the `iom.h` file.

HAS Function

```
DllExport
uint8_t vsl_has_format_uuid(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
const char* vsl_format_uuid(void* handle, fio_error_t* error);
```

Supported Handle(s)

`iom_handle_t`

format_version

Description

The source code for this field is the `iom.h` file.

HAS Function

```
DllExport
uint8_t vsl_has_format_version(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint16_t vsl_format_version(void* handle, fio_error_t* error);
```

Supported Handle(s)

`iom_handle_t`

in_use_physical_sectors

Description

Number of physical sectors that contain valid data in the VSU

The source code for this field is the `vsu.h` file.

HAS Function

```
DllExport
uint8_t vsl_has_in_use_physical_sectors(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint64_t vsl_in_use_physical_sectors(void* handle, fio_error_t* error);
```

Supported Handle(s)

vsu_handle_t

leb_map_present

Description

Reports if the driver has a LEB mapping

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_leb_map_present(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_leb_map_present(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

max_overformat_size_bytes

Description

Maximum formatted block device capacity, used as hard limit when overformatting

The source code for this field is the **adapter.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_max_overformat_size_bytes(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint64_t vsl_max_overformat_size_bytes(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

max_physical_allowed_sectors

Description

Maximum number of physical sectors that the VSU will be allowed to use
Default: max physical size

The source code for this field is the **vsu.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_max_physical_allowed_sectors(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint64_t vsl_max_physical_allowed_sectors(void* handle, fio_error_t* error);
```

Supported Handle(s)

vsu_handle_t

min_physical_reserved_sectors

Description

Minimum number of physical sectors reserved for the VSU
Default: 0 sectors

The source code for this field is the **vsu.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_min_physical_reserved_sectors(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint64_t vsl_min_physical_reserved_sectors(void* handle, fio_error_t* error);
```

Supported Handle(s)

vsu_handle_t

sector_count

Description

Number of sectors available

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_sector_count(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint64_t vsl_sector_count(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
vsu_handle_t

sector_size_bytes

Description

The sector size in bytes

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_sector_size_bytes(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_sector_size_bytes(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
vsu_handle_t

shipped_capacity_bytes

Description

Capacity of block device at time of shipment in bytes

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_shipped_capacity_bytes(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint64_t vsl_shipped_capacity_bytes(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

shipped_sector_size_bytes

Description

Block device sector size at time of shipment in bytes

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_shipped_sector_size_bytes(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_shipped_sector_size_bytes(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

size_bytes

Description

The size of the device in bytes

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_size_bytes(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint64_t vsl_size_bytes(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
vsu_handle_t

RAM usage information fields

The source file for each field is noted in the description for that field, and the handle (or handles) that works with the field functions are listed below the functions.

The following fields may have a **HAS**, **GETTER**, and/or **SETTER** function. **HAS** and **GETTER** functions exist if the field is readable. **SETTER** exists if the field is writable.

The following describes the parameters and return values for each function type:

HAS Function

Description

Checks that the value is available in the object

* **handle** – The handle of the object
* **error** – An error return structure

Return

The error code if the field is not available. **FIO_SUCCESS=0** is success.

GETTER Function

Description

Returns the value for the field

When this function is called, the value is returned from the handle's cache if available, otherwise the value is retrieved and stored in the cache. It will then return the same value in all subsequent calls to this function, until the clear function is called or the handle is destroyed.

* **handle** – The handle of the object
* **error** – An error return structure

Return

The value of the field
If there was an error, the default value for this type will be returned. Users of this function should either call the HAS function first, or check the error return structure before using this value.

SETTER Function

Description

Sets the value for the field

When this function is called, the value is set and stored in the handle's cache.

* **`handle`** – The handle of the object
* **`value`** – The value set
* **`error`** – An error return structure

Return

FIO_SUCCESS=0 is success
The error code if the field was not able to be set.

For additional information:

- ioMemory handle functions on page 10
- Adapter handle functions on page 13
- VSU handle functions on page 13

prealloc_enabled

Description

Reports if full memory preallocation is enabled on the ioMemory device

The source code for this field is the **`iom.h`** file.

HAS Function

```
DllExport
uint8_t vsl_has_prealloc_enabled(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_prealloc_enabled(void* handle, fio_error_t* error);
```

SETTER Function

```
DllExport
fio_error_code_t vsl_set_prealloc_enabled(void* handle, uint8_t value, fio_
error_t* error);
```

If the value is set to 1, preallocation is enabled.

Supported Handle(s)

`iom_handle_t`

prealloc_ram_required_mb

Description

Amount of RAM (MiB) the ioMemory device would require if preallocation was turned on

The source code for this field is the `iom.h` file.

HAS Function

```
DllExport
uint8_t vsl_has_prealloc_ram_required_mb(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint64_t vsl_prealloc_ram_required_mb(void* handle, fio_error_t* error);
```

Supported Handle(s)

`iom_handle_t`

ram_usage_current_bytes

Description

The current amount of memory in bytes allocated for the device

The source code for this field is the `iom.h` file.

HAS Function

```
DllExport
uint8_t vsl_has_ram_usage_current_bytes(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint64_t vsl_ram_usage_current_bytes(void* handle, fio_error_t* error);
```

Supported Handle(s)

`iom_handle_t`

ram_usage_peak_bytes

Description

The highest amount of memory in bytes allocated for the device since driver load

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_ram_usage_peak_bytes(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint64_t vsl_ram_usage_peak_bytes(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

Health and warranty information

The source file for each field is noted in the description for that field, and the handle (or handles) that works with the field functions are listed below the functions.

The following fields may have a **HAS**, **GETTER**, and/or **SETTER** function. **HAS** and **GETTER** functions exist if the field is readable. **SETTER** exists if the field is writable.

The following describes the parameters and return values for each function type:

HAS Function

Description

Checks that the value is available in the object

* **handle** – The handle of the object
* **error** – An error return structure

Return

The error code if the field is not available. **FIO_SUCCESS=0** is success.

GETTER Function

Description

Returns the value for the field

When this function is called, the value is returned from the handle's cache if available, otherwise the value is retrieved and stored in the cache. It will then return the same value in all subsequent calls to this function, until the clear function is called or the handle is destroyed.

* **`handle`** – The handle of the object
* **`error`** – An error return structure

Return

The value of the field
If there was an error, the default value for this type will be returned. Users of this function should either call the HAS function first, or check the error return structure before using this value.

SETTER Function

Description

Sets the value for the field

When this function is called, the value is set and stored in the handle's cache.

* **`handle`** – The handle of the object
* **`value`** – The value set
* **`error`** – An error return structure

Return

FIO_SUCCESS=0 is success
The error code if the field was not able to be set.

For additional information:

- ioMemory handle functions on page 10
- Adapter handle functions on page 13
- VSU handle functions on page 13

first_use_ts

Description

Timestamp of when the device was first used by the customer (posix epoch format)

The source code for this field is the **`iom.h`** file.

HAS Function

```
DllExport
uint8_t vsl_has_first_use_ts(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
int64_t vsl_first_use_ts(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

origin_ts

Description

Timestamp of when the device first started to operate (posix epoch format)

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_origin_ts(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
int64_t vsl_origin_ts(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

rated_endurance_tbytes

Description

Rated write endurance of the device, in terabytes written

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_rated_endurance_tbytes(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_rated_endurance_tbytes(void* handle, fio_error_t* error);
```

Supported Handle(s)

`iom_handle_t`

reserve_space_pct

Description

Percent of reserve space left on the ioMemory device
This is based on format size and may change when the device is formatted. Formatting to a smaller size will increase reserve space.

The source code for this field is the `iom.h` file.

HAS Function

```
DllExport
uint8_t vsl_has_reserve_space_pct(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
float vsl_reserve_space_pct(void* handle, fio_error_t* error);
```

Supported Handle(s)

`iom_handle_t`

reserve_status

Description

Status of the ioMemory device reserve pool

The source code for this field is the `iom.h` file.

HAS Function

```
DllExport
uint8_t vsl_has_reserve_status(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
iom_reserve_status_t vsl_reserve_status(void* handle, fio_error_t* error);
```

Supported Handle(s)

`iom_handle_t`

iom_reserve_status_t Enumeration

IOM_RESERVE_HEALTHY = 0	Device is healthy
IOM_RESERVE_WARNING = 1	Device is getting close to entering reduced write mode
IOM_RESERVE_DEPLETED = 2	Device has entered reduced write or read only mode to preserve the flash from further wearout

reserve_warning_threshold_percent

Description

The percentage of reserve space under which a warning is issued

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_reserve_warning_threshold_percent(void* handle, fio_error_t*
error);
```

GETTER Function

```
DllExport
float vsl_reserve_warning_threshold_percent(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

total_physical_bytes_read

Description

Bytes actually read from the NAND array, including ioMemory VSL overhead

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_total_physical_bytes_read(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint64_t vsl_total_physical_bytes_read(void* handle, fio_error_t* error);
```

Supported Handle(s)

`iom_handle_t`

total_physical_bytes_written

Description

Bytes actually written to the NAND array, including driver overhead

The source code for this field is the `iom.h` file.

HAS Function

```
DllExport
uint8_t vsl_has_total_physical_bytes_written(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint64_t vsl_total_physical_bytes_written(void* handle, fio_error_t* error);
```

Supported Handle(s)

`iom_handle_t`

Firmware and driver information fields

The source file for each field is noted in the description for that field, and the handle (or handles) that works with the field functions are listed below the functions.

The following fields may have a **`HAS`**, **`GETTER`**, and/or **`SETTER`** function. **`HAS`** and **`GETTER`** functions exist if the field is readable. **`SETTER`** exists if the field is writable.

The following describes the parameters and return values for each function type:

HAS Function

Description

Checks that the value is available in the object

* **`handle`** – The handle of the object
* **`error`** – An error return structure

Return

The error code if the field is not available. **`FIO_SUCCESS=0`** is success.

GETTER Function

Description

Returns the value for the field

When this function is called, the value is returned from the handle's cache if available, otherwise the value is retrieved and stored in the cache. It will then return the same value in all subsequent calls to this function, until the clear function is called or the handle is destroyed.

* **`handle`** – The handle of the object
* **`error`** – An error return structure

Return

The value of the field
If there was an error, the default value for this type will be returned. Users of this function should either call the HAS function first, or check the error return structure before using this value.

SETTER Function

Description

Sets the value for the field

When this function is called, the value is set and stored in the handle's cache.

* **`handle`** – The handle of the object
* **`value`** – The value set
* **`error`** – An error return structure

Return

FIO_SUCCESS=0 is success
The error code if the field was not able to be set.

For additional information:

- ioMemory handle functions on page 10
- Adapter handle functions on page 13
- VSU handle functions on page 13

driver_barrier_build_ver

Description

Barrier driver build number

The source code for this field is the **`vsl_abstract.h`** file.

HAS Function

```
DllExport
uint8_t vsl_has_driver_barrier_build_ver(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_driver_barrier_build_ver(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

driver_barrier_major_ver

Description

Barrier driver major version

The source code for this field is the *vsl_abstract.h* file.

HAS Function

```
DllExport
uint8_t vsl_has_driver_barrier_major_ver(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_driver_barrier_major_ver(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

driver_barrier_micro_ver

Description

Barrier driver micro version

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_driver_barrier_micro_ver(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_driver_barrier_micro_ver(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

driver_barrier_minor_ver

Description

Barrier driver minor version

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_driver_barrier_minor_ver(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_driver_barrier_minor_ver(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

driver_barrier_version

Description

Barrier driver version in the format of major.minor.micro.build

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_driver_barrier_version(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
const char* vsl_driver_barrier_version(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

driver_build_ver

Description

Driver build number

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_driver_build_ver(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_driver_build_ver(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

driver_major_ver

Description

Driver major version

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_driver_major_ver(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_driver_major_ver(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

driver_micro_ver

Description

Driver micro version

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_driver_micro_ver(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_driver_micro_ver(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

driver_version

Description

Driver version in the format of major.minor.micro

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_driver_version(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
const char* vsl_driver_version(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

driver_version_detail

Description

Driver version in the format of major.minor.micro.build codename@src_control_uuid

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_driver_version_detail(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
const char* vsl_driver_version_detail(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

fw_barrier_major_ver

Description

Barrier major version of the firmware

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_fw_barrier_major_ver(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_fw_barrier_major_ver(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

fw_barrier_micro_ver

Description

Barrier micro version of the firmware

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_fw_barrier_micro_ver(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_fw_barrier_micro_ver(void* handle, fio_error_t* error);
```

Supported Handle(s)

`iom_handle_t`

fw_barrier_minor_ver

Description

Barrier minor version of the firmware

The source code for this field is the **`vsl_abstract.h`** file.

HAS Function

```
DllExport
uint8_t vsl_has_fw_barrier_minor_ver(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_fw_barrier_minor_ver(void* handle, fio_error_t* error);
```

Supported Handle(s)

`iom_handle_t`

fw_barrier_revision

Description

Barrier revision (legacy) of the firmware

The source code for this field is the **`vsl_abstract.h`** file.

HAS Function

```
DllExport
uint8_t vsl_has_fw_barrier_revision(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_fw_barrier_revision(void* handle, fio_error_t* error);
```

Supported Handle(s)

`iom_handle_t`

fw_current_major_ver

Description

Current major version of the firmware

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_fw_current_major_ver(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_fw_current_major_ver(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

fw_current_micro_ver

Description

Current micro version of the firmware

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_fw_current_micro_ver(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_fw_current_micro_ver(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

fw_current_minor_ver

Description

Current minor version of the firmware

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_fw_current_minor_ver(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_fw_current_minor_ver(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

fw_current_release_status

Description

Release status of currently installed firmware

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_fw_current_release_status(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
fw_release_status_t vsl_fw_current_release_status(void* handle, fio_error_t*
error);
```

Supported Handle(s)

iom_handle_t

fw_release_status_t Enumeration

FW_PUBLIC = 0	Firmware is public
FW_INTERNAL = 1	Firmware is internal

fw_current_revision

Description

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_fw_current_revision(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_fw_current_revision(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

fw_current_version

Description

Current version of the firmware in format major.minor.micro

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_fw_current_version(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
const char* vsl_fw_current_version(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

fw_max_major_ver

Description

Max major version of the firmware

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_fw_max_major_ver(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_fw_max_major_ver(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

fw_max_micro_ver

Description

Max micro version of the firmware

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_fw_max_micro_ver(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_fw_max_micro_ver(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

fw_max_minor_ver

Description

Max minor version of the firmware

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_fw_max_minor_ver(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_fw_max_minor_ver(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

fw_min_major_ver

Description

Min major version of the firmware

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_fw_min_major_ver(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_fw_min_major_ver(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

fw_min_micro_ver

Description

Min micro version of the firmware

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_fw_min_micro_ver(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_fw_min_micro_ver(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

fw_min_revision

Description

Min revision (legacy) of the firmware

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_fw_min_revision(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_fw_min_revision(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

min_driver_version

Description

Minimum driver version in the format of major.minor.micro

The source code for this field is the **adapter.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_min_driver_version(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
const char* vsl_min_driver_version(void* handle, fio_error_t* error);
```

Supported Handle(s)

adapter_handle_t

optrom_current_major_ver

Description

Current major version of the option ROM

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_optrom_current_major_ver(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_optrom_current_major_ver(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

optrom_current_micro_ver

Description

Current micro version of the option ROM

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_optrom_current_micro_ver(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_optrom_current_micro_ver(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

optrom_current_minor_ver

Description

Current minor version of the option ROM

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_optrom_current_minor_ver(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_optrom_current_minor_ver(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

optrom_current_revision

Description

Current revision of the option ROM

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_optrom_current_revision(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_optrom_current_revision(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

optrom_current_version

Description

Current version of the option ROM in format major.minor.micro

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_optrom_current_version(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_optrom_current_version(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

optrom_enabled

Description

Current enabled/disabled state of the option ROM

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_optrom_enabled(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_optrom_enabled(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

smp_app_major_ver

Description

SMP App (AVR firmware) Major Version

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_smp_app_major_ver(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_smp_app_major_ver(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
adapter_handle_t

smp_app_micro_ver

Description

SMP App Micro Version

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_smp_app_micro_ver(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_smp_app_micro_ver(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
adapter_handle_t

smp_app_minor_ver

Description

SMP App Minor Version

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_smp_app_minor_ver(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_smp_app_minor_ver(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
adapter_handle_t

smp_app_nano_ver

Description

SMP App Nano Version

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_smp_app_nano_ver(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_smp_app_nano_ver(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
adapter_handle_t

smp_app_version

Description

SMP App Version String

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_smp_app_version(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
const char* vsl_smp_app_version(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
adapter_handle_t

smp_boot_major_ver

Description

SMP Boot Major Version

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_smp_boot_major_ver(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_smp_boot_major_ver(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
adapter_handle_t

smp_boot_micro_ver

Description

SMP Boot Micro Version

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_smp_boot_micro_ver(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_smp_boot_micro_ver(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
adapter_handle_t

smp_boot_minor_ver

Description

SMP Boot Minor Version

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_smp_boot_minor_ver(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_smp_boot_minor_ver(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
adapter_handle_t

smp_boot_nano_ver

Description

SMP Boot Nano Version

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_smp_boot_nano_ver(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_smp_boot_nano_ver(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
adapter_handle_t

smp_boot_version

Description

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_smp_boot_version(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
const char* vsl_smp_boot_version(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
adapter_handle_t

Power information fields

The source file for each field is noted in the description for that field, and the handle (or handles) that works with the field functions are listed below the functions.

The following fields may have a **HAS**, **GETTER**, and/or **SETTER** function. **HAS** and **GETTER** functions exist if the field is readable. **SETTER** exists if the field is writable.

The following describes the parameters and return values for each function type:

HAS Function

Description

Checks that the value is available in the object

* **handle** – The handle of the object
* **error** – An error return structure

Return

The error code if the field is not available. **FIO_SUCCESS=0** is success.

GETTER Function

Description

Returns the value for the field

When this function is called, the value is returned from the handle's cache if available, otherwise the value is retrieved and stored in the cache. It will then return the same value in all subsequent calls to this function, until the clear function is called or the handle is destroyed.

* **handle** – The handle of the object
* **error** – An error return structure

Return

The value of the field
If there was an error, the default value for this type will be returned. Users of this function should either call the HAS function first, or check the error return structure before using this value.

SETTER Function

Description

Sets the value for the field

When this function is called, the value is set and stored in the handle's cache.

* **handle** – The handle of the object
* **value** – The value set
* **error** – An error return structure

Return

FIO_SUCCESS=0 is success
The error code if the field was not able to be set.

For additional information:

- ioMemory handle functions on page 10
- Adapter handle functions on page 13
- VSU handle functions on page 13

bus_amps

Description

The current value of amperage flowing in the PCIE 12V bus

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
uint8_t vsl_has_bus_amps(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
float vsl_bus_amps(void* handle, fio_error_t* error);
```

Supported Handle(s)

adapter_handle_t

.

bus_min_volts

Description

The min voltage seen on the PCIE 12V bus since last cleared

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_bus_min_volts(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
float vsl_bus_min_volts(void* handle, fio_error_t* error);
```

Supported Handle(s)

adapter_handle_t

bus_peak_amps

Description

The peak amperage seen on the PCIE 12V bus since last cleared.

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_bus_peak_amps(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
float vsl_bus_peak_amps(void* handle, fio_error_t* error);
```

Supported Handle(s)

adapter_handle_t

bus_peak_volts

Description

The max voltage seen on the PCIE 12V bus since last cleared

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_bus_peak_volts(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
float vsl_bus_peak_volts(void* handle, fio_error_t* error);
```

Supported Handle(s)

adapter_handle_t

bus_peak_watts

Description

The peak wattage seen on the PCIE 12V bus since last cleared

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_bus_peak_watts(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
float vsl_bus_peak_watts(void* handle, fio_error_t* error);
```

Supported Handle(s)

adapter_handle_t

bus_volts

Description

The voltage on the PCIE 12V bus

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_bus_volts(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
float vsl_bus_volts(void* handle, fio_error_t* error);
```

Supported Handle(s)

adapter_handle_t

bus_watts

Description

The wattage being drawn on the PCIE 12V bus

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_bus_watts(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
float vsl_bus_watts(void* handle, fio_error_t* error);
```

Supported Handle(s)

adapter_handle_t

external_power_connected

Description

Status of external power connection

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_external_power_connected(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_external_power_connected(void* handle, fio_error_t* error);
```

Supported Handle(s)

adapter_handle_t

external_power_override

Description

Reports if external power override has been set

The source code for this field is the **adapter.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_external_power_override(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_external_power_override(void* handle, fio_error_t* error);
```

Supported Handle(s)

adapter_handle_t

pcie_slot_power_watts

Description

Power available from the slot into which the device is inserted

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_pcie_slot_power_watts(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
float vsl_pcie_slot_power_watts(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
adapter_handle_t

power_limit_watts

Description

The current setting for power-throttling limit

The source code for this field is the **adapter.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_power_limit_watts(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
float vsl_power_limit_watts(void* handle, fio_error_t* error);
```

Supported Handle(s)

adapter_handle_t

power_monitor_enabled

Description

The source code for this field is the **adapter.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_power_monitor_enabled(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_power_monitor_enabled(void* handle, fio_error_t* error);
```

Supported Handle(s)

adapter_handle_t

power_setpoint_high_watts

Description

High setpoint for watts

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_power_setpoint_high_watts(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
float vsl_power_setpoint_high_watts(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

powercut_holdup_type

Description

Type of power cut holdup

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_powercut_holdup_type(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
powercut_holdup_t vsl_powercut_holdup_type(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t
adapter_handle_t

powercut_holdup_t Enumeration

POWERCUT_NONE = 0	
POWERCUT_SELF = 1	
POWERCUT_EXTERNAL = 2	

powerloss_protect_avail

Description

Reports if the card and firmware support power loss protection (powercut)

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_powerloss_protect_avail(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_powerloss_protect_avail(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

powerloss_protect_enabled

Description

Displays if power loss protection is enabled

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_powerloss_protect_enabled(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_powerloss_protect_enabled(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

vccaux_avg_volts

Description

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_vccaux_avg_volts(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
float vsl_vccaux_avg_volts(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

vccaux_peak_volts

Description

FPGA voltage supply (max)

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_vccaux_peak_volts(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
float vsl_vccaux_peak_volts(void* handle, fio_error_t* error);
```

Supported Handle(s)

`iom_handle_t`

vccaux_setpoint_high_volts

Description

High setpoint for VCCaux

The source code for this field is the `iom.h` file.

HAS Function

```
DllExport
uint8_t vsl_has_vccaux_setpoint_high_volts(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
float vsl_vccaux_setpoint_high_volts(void* handle, fio_error_t* error);
```

Supported Handle(s)

`iom_handle_t`

vccaux_setpoint_low_volts

Description

Low setpoint for VCCaux

The source code for this field is the `iom.h` file.

HAS Function

```
DllExport
uint8_t vsl_has_vccaux_setpoint_low_volts(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
float vsl_vccaux_setpoint_low_volts(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

vccint_avg_volts

Description

Core supply voltage (avg)

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_vccint_avg_volts(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
float vsl_vccint_avg_volts(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

vccint_peak_volts

Description

Core supply voltage (max)

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_vccint_peak_volts(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
float vsl_vccint_peak_volts(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

vccint_setpoint_high_volts

Description

High setpoint for VCCint

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_vccint_setpoint_high_volts(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
float vsl_vccint_setpoint_high_volts(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

vccint_setpoint_low_volts

Description

Low setpoint for VCCint

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_vccint_setpoint_low_volts(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
float vsl_vccint_setpoint_low_volts(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

Temperature information fields

The source file for each field is noted in the description for that field, and the handle (or handles) that works with the field functions are listed below the functions.

The following fields may have a **HAS**, **GETTER**, and/or **SETTER** function. **HAS** and **GETTER** functions exist if the field is readable. **SETTER** exists if the field is writable.

The following describes the parameters and return values for each function type:

HAS Function

Description

Checks that the value is available in the object

* **handle** – The handle of the object
* **error** – An error return structure

Return

The error code if the field is not available. **FIO_SUCCESS=0** is success.

GETTER Function

Description

Returns the value for the field

When this function is called, the value is returned from the handle's cache if available, otherwise the value is retrieved and stored in the cache. It will then return the same value in all subsequent calls to this function, until the clear function is called or the handle is destroyed.

* **handle** – The handle of the object
* **error** – An error return structure

Return

The value of the field
If there was an error, the default value for this type will be returned. Users of this function should either call the HAS function first, or check the error return structure before using this value.

SETTER Function

Description

Sets the value for the field

When this function is called, the value is set and stored in the handle's cache.

* **handle** – The handle of the object
* **value** – The value set
* **error** – An error return structure

Return

FIO_SUCCESS=0 is success
The error code if the field was not able to be set.

For additional information:

- ioMemory handle functions on page 10
- Adapter handle functions on page 13
- VSU handle functions on page 13

temp_internal_deg_c

Description

Internal (FPGA) temperature in degrees Celsius

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_temp_internal_deg_c(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
float vsl_temp_internal_deg_c(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

temp_internal_peak_deg_c

Description

Peak Internal (FPGA) temperature in degrees Celsius

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_temp_internal_peak_deg_c(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
float vsl_temp_internal_peak_deg_c(void* handle, fio_error_t* error);
```

Supported Handle(s)

`iom_handle_t`

temp_internal_warn_deg_c

Description

Temperature threshold where the systemwill begin warning user about temperature being too high, in degrees Celsius

The source code for this field is the `iom.h` file.

HAS Function

```
DllExport
uint8_t vsl_has_temp_internal_warn_deg_c(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
float vsl_temp_internal_warn_deg_c(void* handle, fio_error_t* error);
```

Supported Handle(s)

`iom_handle_t`

temp_setpoint_high_deg_c

Description

Temperature point at which the driver will shut itself down to avoid overheating FPGA

The source code for this field is the `iom.h` file.

HAS Function

```
DllExport
uint8_t vsl_has_temp_setpoint_high_deg_c(void* handle, fio_error_t* error);
```

GETTER Function

DllExport
float vsl_temp_setpoint_high_deg_c(void* handle, fio_error_t* error);

Supported Handle(s)

`iom_handle_t`

temp_setpoint_shutdown_deg_c

Description

Temperature point at which FPGA will shut down to avoid physical damage

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_temp_setpoint_shutdown_deg_c(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
float vsl_temp_setpoint_shutdown_deg_c(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

Throttling information fields

The source file for each field is noted in the description for that field, and the handle (or handles) that works with the field functions are listed below the functions.

The following fields may have a **HAS**, **GETTER**, and/or **SETTER** function. **HAS** and **GETTER** functions exist if the field is readable. **SETTER** exists if the field is writable.

The following describes the parameters and return values for each function type:

HAS Function

Description

Checks that the value is available in the object

* **handle** – The handle of the object
* **error** – An error return structure

Return

The error code if the field is not available. **FIO_SUCCESS=0** is success.

GETTER Function

Description

Returns the value for the field

When this function is called, the value is returned from the handle's cache if available, otherwise the value is retrieved and stored in the cache. It will then return the same value in all subsequent calls to this function, until the clear function is called or the handle is destroyed.

* **handle** – The handle of the object
* **error** – An error return structure

Return

The value of the field
If there was an error, the default value for this type will be returned. Users of this function should either call the HAS function first, or check the error return structure before using this value.

SETTER Function

Description

Sets the value for the field

When this function is called, the value is set and stored in the handle's cache.

* **handle** – The handle of the object
* **value** – The value set
* **error** – An error return structure

Return

FIO_SUCCESS=0 is success
The error code if the field was not able to be set.

For additional information:

- ioMemory handle functions on page 10
- Adapter handle functions on page 13
- VSU handle functions on page 13

thermal_threshold_deg_c

Description

Thermal threshold in degrees Celsius for throttling

The source code for this field is the **vsl_abstract.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_thermal_threshold_deg_c(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint16_t vsl_thermal_threshold_deg_c(void* handle, fio_error_t* error);
```

Supported Handle(s)

`iom_handle_t`

write_pwr_throttling_count

Description

A counter that is incremented every time power-throttling is invoked on a write

The source code for this field is the `iom.h` file.

HAS Function

```
DllExport
uint8_t vsl_has_write_pwr_throttling_count(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint64_t vsl_write_pwr_throttling_count(void* handle, fio_error_t* error);
```

Supported Handle(s)

`iom_handle_t`

write_pwr_throttling_sec_since_last

Description

Seconds elapsed since last incidence of power-throttling

The source code for this field is the `iom.h` file.

HAS Function

```
DllExport
uint8_t vsl_has_write_pwr_throttling_sec_since_last(void* handle, fio_error_t*
error);
```

GETTER Function

```
DllExport
uint32_t vsl_write_pwr_throttling_sec_since_last(void* handle, fio_error_t*
error);
```

Supported Handle(s)

`iom_handle_t`

write_reg_power_level

Description

Power write regulation level

The source code for this field is the `iom.h` file.

HAS Function

```
DllExport
uint8_t vsl_has_write_reg_power_level(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
iom_write_reg_level_t vsl_write_reg_power_level(void* handle, fio_error_t*
error);
```

Supported Handle(s)

`iom_handle_t`

iom_write_reg_level_t Enumeration

IOM_WR_REG_INACTIVE = 0	
IOM_WR_REG_ACTIVE = 1	

write_reg_thermal_level

Description

Thermal write regulation level

The source code for this field is the `iom.h` file.

HAS Function

```
DllExport
uint8_t vsl_has_write_reg_thermal_level(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
iom_write_reg_level_t vsl_write_reg_thermal_level(void* handle, fio_error_t*
error);
```

Supported Handle(s)

`iom_handle_t`

write_reg_total_level

Description

Total write regulation level

The source code for this field is the `iom.h` file.

HAS Function

```
DllExport
uint8_t vsl_has_write_reg_total_level(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
iom_write_reg_level_t vsl_write_reg_total_level(void* handle, fio_error_t*
error);
```

Supported Handle(s)

`iom_handle_t`

write_thermal_throttling_count

Description

A counter that is incremented every time thermal throttling is invoked on a write

The source code for this field is the `iom.h` file.

HAS Function

```
DllExport
uint8_t vsl_has_write_thermal_throttling_count(void* handle, fio_error_t*
error);
```

GETTER Function

```
DllExport
uint64_t vsl_write_thermal_throttling_count(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

write_thermal_throttling_sec_since_last

Description

Seconds elapsed since last incidence of thermal throttling

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_write_thermal_throttling_sec_since_last(void* handle, fio_error_
t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_write_thermal_throttling_sec_since_last(void* handle, fio_error_t*
error);
```

Supported Handle(s)

iom_handle_t

write_throttling_reason

Description

Reports the reason the VSL is currently write throttling the device

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_write_throttling_reason(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
iom_write_throttling_reason_t vsl_write_throttling_reason(void* handle, fio_
error_t* error);
```

Supported Handle(s)

`iom_handle_t`

iom_write_throttling_reason_t Enumeration

IOM_WTHROT_REASON_NONE = 0	No write throttling
IOM_WTHROT_REASON_USER = 1	User call caused the current write-throttling mode
IOM_WTHROT_REASON_NO_MD_BLOCKS = 2	Out of metadata blocks
IOM_WTHROT_REASON_NOMEM = 3	Out of memory (RAM)
IOM_WTHROT_REASON_DEAD_DIE = 4	Dead chip die or permanent parity substitution
IOM_WTHROT_REASON_WEAROUT = 5	Too many failed LEBs to continue unthrottled
IOM_WTHROT_REASON_ADAPTER_POWER = 6	Adapter power isn't plugged in. *Note:* This is not the behavior for all devices; some devices start up in minimal mode in this case.
IOM_WTHROT_REASON_INTERNAL = 7	Internal failure caused throttled mode
IOM_WTHROT_REASON_LIMIT_POWER = 8	Attempt to minimize power consumption
IOM_WTHROT_REASON_GROOM_FAILS = 9	Groomer could not free enough blocks to continue

write_throttling_state

Description

State of write throttling

The source code for this field is the `iom.h` file.

HAS Function

```
DllExport
uint8_t vsl_has_write_throttling_state(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
iom_write_throttling_state_t vsl_write_throttling_state(void* handle, fio_error_
t* error);
```

Supported Handle(s)

`iom_handle_t`

iom_write_throttling_state_t Enumeration

IOM_WTHROT_NONE = 0	Device is not currently throttling writes
IOM_WTHROT_PARTIAL = 1	Device is throttling writes to a very low speed
IOM_WTHROT_COMPLETE = 2	Device has stopped all writes

Other information fields

The source file for each field is noted in the description for that field, and the handle (or handles) that works with the field functions are listed below the functions.

The following fields may have a **`HAS`**, **`GETTER`**, and/or **`SETTER`** function. **`HAS`** and **`GETTER`** functions exist if the field is readable. **`SETTER`** exists if the field is writable.

The following describes the parameters and return values for each function type:

HAS Function

Description

Checks that the value is available in the object

* **`handle`** – The handle of the object
* **`error`** – An error return structure

Return

The error code if the field is not available. **`FIO_SUCCESS=0`** is success.

GETTER Function

Description

Returns the value for the field

When this function is called, the value is returned from the handle's cache if available, otherwise the value is retrieved and stored in the cache. It will then return the same value in all subsequent calls to this function, until the clear function is called or the handle is destroyed.

* **`handle`** – The handle of the object
* **`error`** – An error return structure

Return

The value of the field
If there was an error, the default value for this type will be returned. Users of this function should either call the HAS function first, or check the error return structure before using this value.

SETTER Function

Description

Sets the value for the field

When this function is called, the value is set and stored in the handle's cache.

* **`handle`** – The handle of the object
* **`value`** – The value set
* **`error`** – An error return structure

Return

FIO_SUCCESS=0 is success
The error code if the field was not able to be set.

For additional information:

- ioMemory handle functions on page 10
- Adapter handle functions on page 13
- VSU handle functions on page 13

ioctl_api_compatible

Description

Reports if the IOCTL API is compatible with the SDK for the device

The source code for this field is the **`iom.h`** file.

HAS Function

```
DllExport
uint8_t vsl_has_ioctl_api_compatible(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_ioctl_api_compatible(void* handle, fio_error_t* error);
```

Supported Handle(s)

`iom_handle_t`

ioctl_api_version_major

Description

API major version supported by the device

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_ioctl_api_version_major(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_ioctl_api_version_major(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

ioctl_api_version_minor

Description

API minor version supported by the device

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_ioctl_api_version_minor(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_ioctl_api_version_minor(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

session_copied_bytes

Description

Number of bytes copied internal to the driver (for example, grooming) since driver loaded

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_session_copied_bytes(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint64_t vsl_session_copied_bytes(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

session_erased_bytes

Description

Number of bytes erased since driver loaded, including overhead (for example, grooming)

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_session_erased_bytes(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint64_t vsl_session_erased_bytes(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

session_read_ops

Description

Number of read operations since last attach

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_session_read_ops(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint64_t vsl_session_read_ops(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

session_write_ops

Description

Number of write operations since last attach

The source code for this field is the **iom.h** file.

HAS Function

```
DllExport
uint8_t vsl_has_session_write_ops(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint64_t vsl_session_write_ops(void* handle, fio_error_t* error);
```

Supported Handle(s)

iom_handle_t

Cluster handle functions

The source code for these functions is the **cluster.h** file.

vsl_create_cluster_handle

Description

Regarding the IP address of the cluster (from **has_ip_address**):

The cluster IP address is implemented (by the underlying cluster software) as a special cluster resource that listens on an extra IP address designated for the cluster itself. This ensures external management software will always get a connection to at least one node in the cluster, that has the corresponding IP address bound. It is a convenience for external clients to connect to any node in the cluster using the cluster IP address.

DllExport

```
fio_error_code_t vsl_create_cluster_handle(cluster_handle_t* handle, fio_error_
t* error);
```

Return

FIO_SUCCESS=0 is success.

fio_error_code_t on page 4 for an enumerated list.

Cluster-specific fields

The source file for each field is noted in the description for that field, and the handle (or handles) that works with the field functions are listed below the functions.

The following fields works with cluster handles, and have HAS and GETTER functions. HAS and a GETTER functions exist if the field is readable.

The following documentation describes the parameters and return values for each function type:

HAS Function

Description

Checks that the value is available in the object

* **handle** – The handle of the object
* **error** – An error return structure

Return

The error code if the field is not available. **FIO_SUCCESS=0** is success.

GETTER Function

Description

Returns the value for the field

When this function is called, the value is returned from the handle's cache if available, otherwise the value is retrieved and stored in the cache. It will then return the same value in all subsequent calls to this function, until the clear function is called or the handle is destroyed.

* **`handle`** – The handle of the object

* **`error`** – An error return structure

Return

The value of the field
If there was an error, the default value for this type will be returned. Users of this function should either call the HAS function first, or check the error return structure before using this value.

For additional information:

- ioMemory handle functions on page 10
- Adapter handle functions on page 13
- VSU handle functions on page 13

cluster_name

Description

The name of the cluster
This is the administrative name of the entire cluster itself, if that is a configurable property of the underlying cluster software.

HAS Function

```
DllExport
uint8_t vsl_has_cluster_name(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
const char* vsl_cluster_name(void* handle, fio_error_t* error);
```

ip_address

Description

The source code for this function is found in the **`vsl_abstract.h`** file.

HAS Function

```
DllExport
uint8_t vsl_has_ip_address(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_has_ip_address(void* handle, fio_error_t* error);
```

Host handle functions

The source code for these functions is the **host.h** file.

vsl_host_create

Description

Creates a handle for the host.

DllExport

```
fio_error_code_t vsl_host_create(host_handle_t* handle, fio_error_t* error);
```

Return

FIO_SUCCESS=0 is success.

fio_error_code_t on page 4 for an enumerated list.

Host-specific fields

The source file for each field is noted in the description for that field, and the handle (or handles) that works with the field functions are listed below the functions.

The following fields may have a **HAS**, **GETTER**, and/or **SETTER** function. **HAS** and **GETTER** functions exist if the field is readable. **SETTER** exists if the field is writable.

The following describes the parameters and return values for each function type:

HAS Function

Description

Checks that the value is available in the object

* **handle** – The handle of the object
* **error** – An error return structure

Return

The error code if the field is not available. **FIO_SUCCESS=0** is success.

GETTER Function

Description

Returns the value for the field

When this function is called, the value is returned from the handle's cache if available, otherwise the value is retrieved and stored in the cache. It will then return the same value in all subsequent calls to this function, until the clear function is called or the handle is destroyed.

* **handle** – The handle of the object
* **error** – An error return structure

Return

The value of the field
If there was an error, the default value for this type will be returned. Users of this function should either call the HAS function first, or check the error return structure before using this value.

SETTER Function

Description

Sets the value for the field

When this function is called, the value is set and stored in the handle's cache.

* **handle** – The handle of the object
* **value** – The value set
* **error** – An error return structure

Return

FIO_SUCCESS=0 is success
The error code if the field was not able to be set.

For additional information:

- ioMemory handle functions on page 10
- Adapter handle functions on page 13
- VSU handle functions on page 13

driver_loaded

Description

Reports if the driver is loaded

HAS Function

```
DllExport
uint8_t vsl_has_driver_loaded(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_driver_loaded(void* handle, fio_error_t* error);
```

fusion_trim_running

Description

Reports if the Fusion-io trim service is running (Windows only)

HAS Function

```
DllExport
uint8_t vsl_has_fusion_trim_running(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_fusion_trim_running(void* handle, fio_error_t* error);
```

is_cluster_master

Description

Reports if a host is the cluster master (one node in the current cluster membership considered the master/controller of the cluster)

For Linux clusters, this is an attribute of the node object. For Windows clusters, it can be the node with the highest IP address presently running in the current cluster membership.

This property can change as a result of change in cluster membership. Only one (running) node can be the master. Highly layer software can test this flag to coordinate running logic on only one node for the whole cluster.

HAS Function

```
DllExport
uint8_t vsl_has_is_cluster_master(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_is_cluster_master(void* handle, fio_error_t* error);
```

ram_available_kernel_bytes

Description

Reports the total available kernel memory (in bytes)

HAS Function

```
DllExport
uint8_t vsl_has_ram_available_kernel_bytes(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint64_t vsl_ram_available_kernel_bytes(void* handle, fio_error_t* error);
```

ram_available_physical_bytes

Description

Reports the available physical memory (in bytes)

HAS Function

```
DllExport
uint8_t vsl_has_ram_available_physical_bytes(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint64_t vsl_ram_available_physical_bytes(void* handle, fio_error_t* error);
```

ram_total_physical_bytes

Description

Reports the total physical memory on the system (in bytes)

HAS Function

```
DllExport
uint8_t vsl_has_ram_total_physical_bytes(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint64_t vsl_ram_total_physical_bytes(void* handle, fio_error_t* error);
```

ram_total_virtual_bytes

Description

Reports the total virtual memory on the system (in bytes)

HAS Function

```
DllExport
uint8_t vsl_has_ram_total_virtual_bytes(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint64_t vsl_ram_total_virtual_bytes(void* handle, fio_error_t* error);
```

trim_enabled

Description

Repots if trim is enabled (Windows only).

> This handle does not report if trim is currently active, only if it is supposed to be. When setting this, it is possible to receive **FIO_ERR_HOST_TRIM_CANT_MODIFY_SETTINGS** or **FIO_ERR_HOST_TRIM_CANT_MODIFY_SERVICE** errors.

HAS Function

```
DllExport
uint8_t vsl_has_trim_enabled(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_trim_enabled(void* handle, fio_error_t* error);
```

SETTER Function

```
DllExport
fio_error_code_t vsl_set_trim_enabled(void* handle, uint8_t value, fio_error_t*
error);
```

Management operation functions

These functions execute a management operation against an ioMemory device. Before the operation can be executed, a handle for the operation must be created for the device. These operation functions are described in the **vsl_abstract.h** file and work with the following handles:

- **iom_attach_handle_t** – Attach operation functions on page 127.
- **iom_detach_handle_t** – Detach operation functions on page 130.
- **iom_format_handle_t** – Format operation functions on page 132.
- **iom_sure_erase_handle_t** – Sure erase operation functions on page 143.
- **iom_update_handle_t** – Update operation functions on page 147.

vsl_op_get_progress_percent

Description

Reports the current operation progress in percent

DllExport

```
uint8_t vsl_op_get_progress_percent(void* handle);
```

Return

Returns the current operation progress in percent

vsl_op_is_running

Description

Reports if the operation is currently executing

DllExport

```
uint8_t vsl_op_is_running(void* handle);
```

Return

Returns if the operation is currently executing

vsl_op_start

Description

Validates and executes the operation
Returns immediately

DllExport

```
fio_error_code_t vsl_op_start(void* handle, fio_error_t* error);
```

Return

FIO_SUCCESS=0 is success. fio_error_code_t on page 4 for an enumerated list.

vsl_op_wait_for_result

Description

Blocks until the operation completes and returns success(0) or an error code

```
DllExport
fio_error_code_t vsl_op_wait_for_result(void* handle, fio_error_t* error);
```

Return

FIO_SUCCESS=0 is success. fio_error_code_t on page 4 for an enumerated list.

Attach operation functions

These functions and fields work with the Management Operation Functions to execute an operation on the ioMemory device.

Attach handle function

The source code for these functions is the **iom_attach.h** file.

vsl_iom_attach_create

Description

Creates a handle for attaching the ioMemory device.

```
DllExportfio_error_code_t vsl_iom_attach_create(iom_handle_t iom, iom_attach_
handle_t* handle, fio_error_t* error);
```

Return

FIO_SUCCESS=0 is success. fio_error_code_t on page 4 for an enumerated list.

Attach field

The source file for each field is noted in the description for that field, and the handle (or handles) that works with the field functions are listed below the functions.

The following fields may have a **HAS**, **GETTER**, and/or **SETTER** function. **HAS** and **GETTER** functions exist if the field is readable. **SETTER** exists if the field is writable.

The following describes the parameters and return values for each function type:

HAS Function

Description

Checks that the value is available in the object

* **handle** – The handle of the object
* **error** – An error return structure

Return

The error code if the field is not available. **FIO_SUCCESS=0** is success.

GETTER Function

Description

Returns the value for the field

When this function is called, the value is returned from the handle's cache if available, otherwise the value is retrieved and stored in the cache. It will then return the same value in all subsequent calls to this function, until the clear function is called or the handle is destroyed.

* **handle** – The handle of the object
* **error** – An error return structure

Return

The value of the field
If there was an error, the default value for this type will be returned. Users of this function should either call the HAS function first, or check the error return structure before using this value.

SETTER Function

Description

Sets the value for the field

When this function is called, the value is set and stored in the handle's cache.

* **handle** – The handle of the object
* **value** – The value set
* **error** – An error return structure

Return

FIO_SUCCESS=0 is success
The error code if the field was not able to be set.

For additional information:

- ioMemory handle functions on page 10
- Adapter handle functions on page 13
- VSU handle functions on page 13

scan_mode

Description

Reports options for the scanning meta-data

HAS Function

```
DllExport
uint8_t vsl_iom_attach_has_scan_mode(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
iom_attach_scan_mode_t vsl_iom_attach_scan_mode(void* handle, fio_error_t*
error);
```

SETTER Function

```
DllExport
fio_error_code_t vsl_iom_attach_set_scan_mode(void* handle, iom_attach_scan_
mode_t value, fio_error_t* error);
```

iom_attach_scan_mode_t Enumeration

IOM_ATTACH_SCAN_DEFAULT = 0	*Default:* Attempt attach, with full-scan if required
IOM_ATTACH_CLEAN_ONLY = 1	Attach only if clean shutdown and full scan is not required
IOM_ATTACH_FORCE_RESCAN = 2	Always force a full scan during attach

Detach operation functions

These functions and fields work with the Management Operation Functions to execute an operation on the ioMemory device.

Detach handle functions

The source code for these functions is the **iom_detach.h** file.

vsl_iom_detach_create

Description

Creates a handle for detaching the ioMemory device

```
DllExport
fio_error_code_t vsl_iom_detach_create(iom_handle_t iom, iom_detach_handle_t*
handle, fio_error_t* error);
```

Return

FIO_SUCCESS=0 is success. fio_error_code_t on page 4 for an enumerated list.

Detach fields

The source file for each field is noted in the description for that field, and the handle (or handles) that works with the field functions are listed below the functions.

The following fields may have a **SETTER** function, which exists if the field is writable.

The following describes the parameters and return values of the function type:

SETTER Function

Description

Sets the value for the field

When this function is called, the value is set and stored in the handle's cache.

* **handle** – The handle of the object
* **value** – The value set
* **error** – An error return structure

Return

FIO_SUCCESS=0 is success
The error code if the field was not able to be set.

For additional information:

- ioMemory handle functions on page 10
- Adapter handle functions on page 13
- VSU handle functions on page 13

force

Description

Forces the detach to happen even if it is in use (Windows only)

SETTER Function

```
DllExport
fio_error_code_t vsl_iom_detach_set_force(void* handle, uint8_t value, fio_
error_t* error);
```

Default: false (0)
Set to 1 to force detach.

Format operation functions

These functions and fields work with the Management Operation Functions to execute an operation on the ioMemory device.

Format handle functions

The source code for these functions is the **iom_format.h** file.

vsl_iom_format_create

Description

Creates a handle for formatting the ioMemory device

```
DllExport
fio_error_code_t vsl_iom_format_create(iom_handle_t iom, iom_format_handle_t*
handle, fio_error_t* error);
```

Return

FIO_SUCCESS=0 is success. fio_error_code_t on page 4 for an enumerated list.

vsl_iom_format_validate

Description

Performs all validation to determine if the current parameters are valid

```
DllExport
fio_error_code_t vsl_iom_format_validate(iom_format_handle_t handle, fio_error_
t* error);
```

Return

FIO_SUCCESS=0 is success. fio_error_code_t on page 4 for an enumerated list.

Format fields

The source file for each field is noted in the description for that field, and the handle (or handles) that works with the field functions are listed below the functions.

The following fields may have a **HAS**, **GETTER**, and/or **SETTER** function. **HAS** and **GETTER** functions exist if the field is readable. **SETTER** exists if the field is writable.

The following describes the parameters and return values for each function type:

HAS Function

Description

Checks that the value is available in the object

* **handle** – The handle of the object
* **error** – An error return structure

Return

The error code if the field is not available. **FIO_SUCCESS=0** is success.

GETTER Function

Description

Returns the value for the field

When this function is called, the value is returned from the handle's cache if available, otherwise the value is retrieved and stored in the cache. It will then return the same value in all subsequent calls to this function, until the clear function is called or the handle is destroyed.

* **handle** – The handle of the object
* **error** – An error return structure

Return

The value of the field
If there was an error, the default value for this type will be returned. Users of this function should either call the HAS function first, or check the error return structure before using this value.

SETTER Function

Description

Sets the value for the field

When this function is called, the value is set and stored in the handle's cache.

* **handle** – The handle of the object
* **value** – The value set
* **error** – An error return structure

Return

FIO_SUCCESS=0 is success
The error code if the field was not able to be set.

For additional information:

- ioMemory handle functions on page 10
- Adapter handle functions on page 13
- VSU handle functions on page 13

create_pstore

Description

Creates a PStore.

HAS Function

```
DllExport
uint8_t vsl_iom_format_has_create_pstore(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_iom_format_create_pstore(void* handle, fio_error_t* error);
```

SETTER Function

```
fio_error_code_t vsl_iom_format_set_create_pstore(void* handle, uint8_t value,
fio_error_t* error);
```

1= PStore will be created, 0= no PPS will be created (default)

disallow_overformat

Description

Reports if non-zero, then does not allow formatting larger than **`max_format_size_mb`**.

HAS Function

```
DllExport
uint8_t vsl_iom_format_has_disallow_overformat(void* handle, fio_error_t*
error);
```

GETTER Function

```
DllExport
uint8_t vsl_iom_format_disallow_overformat(void* handle, fio_error_t* error);
```

iom_format_atomic_writes_available

Description

If set, atomic writes will be enabled for the device.

HAS Function

```
DllExport
uint8_t vsl_iom_format_has_atomic_writes_available(void* handle, fio_error_t*
error);
```

GETTER Function

```
DllExport
uint8_t vsl_iom_format_atomic_writes_available(void* handle, fio_error_t*
error);
```

SETTER Function

```
DllExport
fio_error_code_t vsl_iom_format_set_atomic_writes_available(void* handle, uint8_
t value, fio_error_t* error);
```

1= atomic writes enabled, 0= not enabled (default)

iom_format_eb_index_type

Description

Sets the type of the packet indexing to be used

HAS Function

```
DllExport
uint8_t vsl_iom_format_has_eb_index_type(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
iom_format_eb_index_type_t vsl_iom_format_eb_index_type(void* handle, fio_error_
t* error);
```

SETTER Function

```
DllExport
fio_error_code_t vsl_iom_format_set_eb_index_type(void* handle, iom_format_eb_
index_type_t value, fio_error_t* error);
```

iom_format_eb_index_type_t Enumeration

IOM_FORMAT_EBIDX_TYPE_NONE = 0	No packet indexing

IOM_FORMAT_EBIDX_TYPE_FULL = 1	Full packet indexing for all packet types
IOM_FORMAT_EBIDX_TYPE_BRIEF = 2	Brief packet indexing for certain system packet types only

iom_format_force

Description

Allows the user to force a format to occur, even if the **nv_data** has not been loaded

> Note that this will void the warranty on the device. The warranty will be voided only if the **nv_data** has not been loaded. If the **nv_data** has been loaded, this field has no effect.

HAS Function

```
DllExport
uint8_t vsl_iom_format_has_force(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_iom_format_force(void* handle, fio_error_t* error);
```

SETTER Function

```
DllExport
fio_error_code_t vsl_iom_format_set_force(void* handle, uint8_t value, fio_
error_t* error);
```

iom_format_ptrim_available

Description

If set, persistent trim feature will be enabled for the device

HAS Function

```
DllExport
uint8_t vsl_iom_format_has_ptrim_available(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_iom_format_ptrim_available(void* handle, fio_error_t* error);
```

SETTER Function

```
DllExport
fio_error_code_t vsl_iom_format_set_ptrim_available(void* handle, uint8_t value,
fio_error_t* error);
```

1= persistent trim enabled, 0= not enabled (default)

iom_format_sector_size_bytes

Description

Setting will request a specific sector size
Getting this handle before setting it will retrieve the default value.

HAS Function

```
DllExport
uint8_t vsl_iom_format_has_sector_size_bytes(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_iom_format_sector_size_bytes(void* handle, fio_error_t* error);
```

SETTER Function

```
DllExport
fio_error_code_t vsl_iom_format_set_sector_size_bytes(void* handle, uint32_t
value, fio_error_t* error);c
```

iom_format_size_bytes

Description

Format to a specific size in bytes

> ⚠ An error may be returned during validate if the size does not fall within the boundaries defined by the size mode and sector size.

This field will also return the current size that will be formatted to based current parameters. Setting this value will also change the **`[size_pct]`** value.

HAS Function

```
DllExport
uint8_t vsl_iom_format_has_size_bytes(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint64_t vsl_iom_format_size_bytes(void* handle, fio_error_t* error);
```

SETTER Function

```
DllExport
fio_error_code_t vsl_iom_format_set_size_bytes(void* handle, uint64_t value,
fio_error_t* error);
```

max_sector_size_bytes

Description

Maximum supported sector size in bytes

HAS Function

```
DllExport
uint8_t vsl_iom_format_has_max_sector_size_bytes(void* handle, fio_error_t*
error);
```

GETTER Function

```
DllExport
uint32_t vsl_iom_format_max_sector_size_bytes(void* handle, fio_error_t* error);
```

max_size_bytes

Description

Formattable size with the given **`iom_format_sector_size_bytes`** and **`size_mode`**

HAS Function

```
DllExport
uint8_t vsl_iom_format_has_max_size_bytes(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint64_t vsl_iom_format_max_size_bytes(void* handle, fio_error_t* error);
```

min_sector_size_bytes

Description

Minimum supported sector size in bytes

HAS Function

```
DllExport
uint8_t vsl_iom_format_has_min_sector_size_bytes(void* handle, fio_error_t*
error);
```

GETTER Function

```
DllExport
uint32_t vsl_iom_format_min_sector_size_bytes(void* handle, fio_error_t* error);
```

power_of_two_sector_size

Description

Set to 1 if the sector size is required to be a power of 2

HAS Function

```
DllExport
uint8_t vsl_iom_format_has_power_of_two_sector_size(void* handle, fio_error_t*
error);
```

GETTER Function

```
DllExport
uint8_t vsl_iom_format_power_of_two_sector_size(void* handle, fio_error_t*
error);
```

pstore_reserved_bytes

Description

Amount of ioMemory space to reserve for all the large values that could be stored in the PPS
Default: 0
UINT64: pstore_large_value_bytes

HAS Function

```
DllExport
uint8_t vsl_iom_format_has_pstore_reserved_bytes(void* handle, fio_error_t*
error);
```

GETTER Function

```
DllExport
uint64_t vsl_iom_format_pstore_reserved_bytes(void* handle, fio_error_t* error);
```

SETTER Function

```
DllExport
fio_error_code_t vsl_iom_format_set_pstore_reserved_bytes(void* handle, uint64_t
value, fio_error_t* error);
```

resulting_reserve_space

Description

Based on proposed sector size and format size, get the resulting reserve space remaining on the device

HAS Function

```
DllExport
uint8_t vsl_iom_format_has_resulting_reserve_space(void* handle, fio_error_t*
error);
```

GETTER Function

```
DllExport
float vsl_iom_format_resulting_reserve_space(void* handle, fio_error_t* error);
```

sector_size_multiplier

Description

If non-zero, the sector size must be a multiple of this number

HAS Function

```
DllExport
uint8_t vsl_iom_format_has_sector_size_multiplier(void* handle, fio_error_t*
error);
```

GETTER Function

```
DllExport
uint32_t vsl_iom_format_sector_size_multiplier(void* handle, fio_error_t*
error);
```

size_mode

HAS Function

```
DllExport
uint8_t vsl_iom_format_has_size_mode(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
iom_format_size_mode_t vsl_iom_format_size_mode(void* handle, fio_error_t*
error);
```

SETTER Function

```
DllExport
fio_error_code_t vsl_iom_format_set_size_mode(void* handle, iom_format_size_
mode_t value, fio_error_t* error);
```

iom_format_size_mode_t Enumeration

`FORMAT_SIZE_DEFAULT = 0xA`	*Default:* Format based on an advertised size of the device
`FORMAT_SIZE_OVERFORMAT = 0xB`	Format based on the maximum size of the device Use to overformat
`FORMAT_SIZE_SPARSE = 0xC`	Advanced use only, for specialized filesystems *This causes the size checks to be overridden and allow the device to be formatted to a larger size than it has available. Set to non-zero to force.*

target

Description

The type of the format: standard or special

HAS Function

```
DllExport
uint8_t vsl_iom_format_has_target(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
iom_format_target_t vsl_iom_format_target(void* handle, fio_error_t* error);
```

SETTER Function

```
DllExport
fio_error_code_t vsl_iom_format_set_target(void* handle, iom_format_target_t
value, fio_error_t* error);
```

iom_format_target_t Enumeration

FORMAT_TARGET_STANDARD = 0	*Default:* Format should produce a standard block device
FORMAT_TARGET_CACHE = 1	Format for use as a cache device
FORMAT_TARGET_UNKNOWN = 2	Target of this format is unknown

Sure erase operation functions

These functions and fields work with the Management Operation Functions to execute an operation on the ioMemory device.

Sure erase handle functions

The source code for these functions is the **iom_sure_erase.h** file.

vsl_iom_sure_erase_create

Description

Creates the ioMemory device handle for executing a sure erase on the device

```
DllExport
fio_error_code_t vsl_iom_sure_erase_create(iom_handle_t iom, iom_sure_erase_
handle_t* handle, fio_error_t* error);
```

Return

FIO_SUCCESS=0 is success. fio_error_code_t on page 4 for an enumerated list.

vsl_iom_sure_erase_read_report

Description

Gets the report log
vsl_iom_sure_erase_free_report must be called to deallocate memory when done using it.

```
DllExport
fio_error_code_t vsl_iom_sure_erase_read_report(iom_sure_erase_handle_t handle,
char** report, fio_error_t* error);
```

Return

FIO_SUCCESS=0 is success. fio_error_code_t on page 4 for an enumerated list.

vsl_iom_sure_erase_free_report

Description

Deallocates memory previously allocated in call to **vsl_iom_sure_erase_read_report**

```
DllExport
fio_error_code_t vsl_iom_sure_erase_free_report(iom_sure_erase_handle_t handle,
char** report, fio_error_t* error);
```

Return

FIO_SUCCESS=0 is success. fio_error_code_t on page 4 for an enumerated list.

Sure erase fields

The source file for each field is noted in the description for that field, and the handle (or handles) that works with the field functions are listed below the functions.

The following fields may have a **HAS**, **GETTER**, and/or **SETTER** function. **HAS** and **GETTER** functions exist if the field is readable. **SETTER** exists if the field is writable.

The following describes the parameters and return values for each function type:

HAS Function

Description

Checks that the value is available in the object

* **handle** – The handle of the object
* **error** – An error return structure

Return

The error code if the field is not available. **FIO_SUCCESS=0** is success.

GETTER Function

Description

Returns the value for the field

When this function is called, the value is returned from the handle's cache if available, otherwise the value is retrieved and stored in the cache. It will then return the same value in all subsequent calls to this function, until the clear function is called or the handle is destroyed.

* **handle** – The handle of the object
* **error** – An error return structure

Return

The value of the field
If there was an error, the default value for this type will be returned. Users of this function should either call the HAS function first, or check the error return structure before using this value.

SETTER Function

Description

Sets the value for the field

When this function is called, the value is set and stored in the handle's cache.

* **`handle`** – The handle of the object
* **`value`** – The value set
* **`error`** – An error return structure

Return

FIO_SUCCESS=0 is success
The error code if the field was not able to be set.

For additional information:

- ioMemory handle functions on page 10
- Adapter handle functions on page 13
- VSU handle functions on page 13

make_inoperable

Description

Erases not just data on the device, but all internal meta-data for the device as well, making the device inoperable
Default: false

> ⛔ This option makes your device permanently inoperable.

HAS Function

```
DllExport
uint8_t vsl_iom_sure_erase_has_make_inoperable(void* handle, fio_error_t*
error);
```

GETTER Function

```
DllExport
uint8_t vsl_iom_sure_erase_make_inoperable(void* handle, fio_error_t* error);
```

SETTER Function

```
DllExport
fio_error_code_t vsl_iom_sure_erase_set_make_inoperable(void* handle, uint8_t
value, fio_error_t* error);
```

purge

Description

Causes sure erase to perform a write followed by an erase

Default: false

HAS Function

```
DllExport
uint8_t vsl_iom_sure_erase_has_purge(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_iom_sure_erase_purge(void* handle, fio_error_t* error);
```

SETTER Function

```
DllExport
fio_error_code_t vsl_iom_sure_erase_set_purge(void* handle, uint8_t value, fio_
error_t* error);
```

Update operation functions

These functions and fields work with the Management Operation Functions to execute an operation on the ioMemory device.

Update handle functions

The source code for these functions is the **iom_update.h** file.

vsl_iom_update_create_from_fff

Description

Creates ioMemory update handle for the ioMemory device

```
DllExport
fio_error_code_t vsl_iom_update_create_from_fff(iom_handle_t iom, const char*
fff_path, iom_update_handle_t* handle, fio_error_t* error);
```

* **fff_path** – This is the path to the firmware archive file that will be used in the update.

Return

FIO_SUCCESS=0 is success. fio_error_code_t on page 4 for an enumerated list.

vsl_iom_update_validate

Description

Performs all validation to determine if the current parameters are valid.

DllExport

fio_error_code_t vsl_iom_update_validate(iom_update_handle_t handle, fio_error_t* error);

Return

FIO_SUCCESS=0 is success. See fio_error_code_t in the Global Functions section for an enumerated list.

Update fields

The source file for each field is noted in the description for that field, and the handle (or handles) that works with the field functions are listed below the functions.

The following fields may have a **HAS**, **GETTER**, and/or **SETTER** function. **HAS** and **GETTER** functions exist if the field is readable. **SETTER** exists if the field is writable.

The following describes the parameters and return values for each function type:

HAS Function

Description

Checks that the value is available in the object

* **`handle`** – The handle of the object
* **`error`** – An error return structure

Return

The error code if the field is not available. **`FIO_SUCCESS=0`** is success.

GETTER Function

Description

Returns the value for the field

When this function is called, the value is returned from the handle's cache if available, otherwise the value is retrieved and stored in the cache. It will then return the same value in all subsequent calls to this function, until the clear function is called or the handle is destroyed.

* **`handle`** – The handle of the object
* **`error`** – An error return structure

Return

The value of the field
If there was an error, the default value for this type will be returned. Users of this function should either call the HAS function first, or check the error return structure before using this value.

SETTER Function

Description

Sets the value for the field

When this function is called, the value is set and stored in the handle's cache.

* **`handle`** – The handle of the object
* **`value`** – The value set
* **`error`** – An error return structure

Return

FIO_SUCCESS=0 is success
The error code if the field was not able to be set.

For additional information:

- ioMemory handle functions on page 10
- Adapter handle functions on page 13
- VSU handle functions on page 13

adaptpdi_result

Description

The result from programming the adapter PDI

HAS Function

```
DllExport
uint8_t vsl_iom_update_has_adaptpdi_result(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
iom_update_programming_result_t vsl_iom_update_adaptpdi_result(void* handle,
fio_error_t* error);
```

iom_update_programming_result_t Enumeration

IOM_UPDATE_NOT_PROGRAMMED = 0x00000000	
IOM_UPDATE_PROGRAMMED = 0x00000001	
IOM_UPDATE_PROGRAMMING_FAILED = 0x00000002	

bypass_check

Description

Flags for bypassing/ignoring firmware checks

Default: none

SETTER Function

```
DllExport
fio_error_code_t vsl_iom_update_set_bypass_check(void* handle, iom_update_
bypass_check_t value, fio_error_t* error);
```

iom_update_bypass_check_t Enumeration

BYPASS_CHECK_NONE = 0x00000000	Dont bypass anything
BYPASS_CHECK_UP_TO_DATE = 0x00000001	Firmware is already up to date
BYPASS_CHECK_ECC = 0x00000002	Firmware ECC levels are not compatible
BYPASS_CHECK_BARRIER = 0x00000004	Firmware exceeds barrier version

ctrlpdi_result

Description

The result from programming the controller PDI

HAS Function

```
DllExport
uint8_t vsl_iom_update_has_ctrlpdi_result(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
iom_update_programming_result_t vsl_iom_update_ctrlpdi_result(void* handle, fio_
error_t* error);
```

forcing_downgrade

Description

Reports if forcing a downgrade of the firmware version

Default: false.

HAS Function

```
DllExport
uint8_t vsl_iom_update_has_forcing_downgrade(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint8_t vsl_iom_update_forcing_downgrade(void* handle, fio_error_t* error);
```

fpga_result

Description

The result from programming the FPGA

HAS Function

```
DllExport
uint8_t vsl_iom_update_has_fpga_result(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
iom_update_programming_result_t vsl_iom_update_fpga_result(void* handle, fio_
error_t* error);
```

fw_after_major_ver

HAS Function

```
DllExport
uint8_t vsl_iom_update_has_fw_after_major_ver(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_iom_update_fw_after_major_ver(void* handle, fio_error_t* error);
```

fw_after_micro_ver

HAS Function

```
DllExport
uint8_t vsl_iom_update_has_fw_after_micro_ver(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_iom_update_fw_after_micro_ver(void* handle, fio_error_t* error);
```

fw_after_minor_ver

HAS Function

```
DllExport
uint8_t vsl_iom_update_has_fw_after_minor_ver(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_iom_update_fw_after_minor_ver(void* handle, fio_error_t* error);
```

fw_after_revision

HAS Function

```
DllExport
uint8_t vsl_iom_update_has_fw_after_revision(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_iom_update_fw_after_revision(void* handle, fio_error_t* error);
```

fw_before_major_ver

HAS Function

```
DllExport
uint8_t vsl_iom_update_has_fw_before_major_ver(void* handle, fio_error_t*
error);
```

GETTER Function

```
DllExport
uint32_t vsl_iom_update_fw_before_major_ver(void* handle, fio_error_t* error);
```

fw_before_micro_ver

HAS Function

```
DllExport
uint8_t vsl_iom_update_has_fw_before_micro_ver(void* handle, fio_error_t*
error);
```

GETTER Function

```
DllExport
uint32_t vsl_iom_update_fw_before_micro_ver(void* handle, fio_error_t* error);
```

fw_before_minor_ver

HAS Function

```
DllExport
uint8_t vsl_iom_update_has_fw_before_minor_ver(void* handle, fio_error_t*
error);
```

GETTER Function

```
DllExport
uint32_t vsl_iom_update_fw_before_minor_ver(void* handle, fio_error_t* error);
```

fw_before_revision

HAS Function

```
DllExport
uint8_t vsl_iom_update_has_fw_before_revision(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
uint32_t vsl_iom_update_fw_before_revision(void* handle, fio_error_t* error);
```

optrom_arch

Description

Flags for selecting the option ROM architecture

Default: none

SETTER Function

```
DllExport
fio_error_code_t vsl_iom_update_set_optrom_arch(void* handle, iom_update_optrom_
arch_t value, fio_error_t* error);
```

iom_update_optrom_arch_t Enumeration

OPTROM_ARCH_DEFAULT = 0x00000000	The default optrom arch is the same as the running platform
OPTROM_ARCH_X86 = 0x00000001	Override the optrom arch to X86
OPTROM_ARCH_IA64 = 0x00000002	Override the optrom arch to ia64

optrom_result

Description

The result from programming the option ROM

HAS Function

```
DllExport
uint8_t vsl_iom_update_has_optrom_result(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
iom_update_programming_result_t vsl_iom_update_optrom_result(void* handle, fio_
error_t* error);
```

iom_update_programming_result_t Enumeration

IOM_UPDATE_NOT_PROGRAMMED = 0x00000000	
{{IOM_UPDATE_PROGRAMMED = 0x00000001	
{{IOM_UPDATE_PROGRAMMING_FAILED = 0x00000002	

part_number

Description

The part number used to find the appropriate firmware.

HAS Function

```
DllExport
uint8_t vsl_iom_update_has_part_number(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
const char* vsl_iom_update_part_number(void* handle, fio_error_t* error);
```

pretend

Description

Show what updates would be done (firmware will not be modified)

Default: false.
Set to 1 to enable.

SETTER Function

```
DllExport
fio_error_code_t vsl_iom_update_set_pretend(void* handle, uint8_t value, fio_
error_t* error);
```

smp_result

Description

The result from programming the SMP

HAS Function

```
DllExport
uint8_t vsl_iom_update_has_smp_result(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
iom_update_programming_result_t vsl_iom_update_smp_result(void* handle, fio_
error_t* error);
```

smpbase_result

Description

The result from programming the SMPbase

HAS Function

```
DllExport
uint8_t vsl_iom_update_has_smpbase_result(void* handle, fio_error_t* error);
```

GETTER Function

```
DllExport
iom_update_programming_result_t vsl_iom_update_smpbase_result(void* handle, fio_
error_t* error);
```

virtual_controller_convert

Description

Convert to this number of virtual controllers; no conversion if zero (default)

SETTER Function

```
DllExport
fio_error_code_t vsl_iom_update_set_virtual_controller_convert(void* handle,
uint8_t value, fio_error_t* error);
```