# DOCA Driver APIs

## Reference Manual

# Table of Contents

# Chapter 1. Change Log

This chapter list changes in API that were introduced to the library.

## 1.3.0

- Field Groups, GPU Groups, and field watches created with a handle returned from dcgmConnect() are now cleaned up upon disconnect. dcgmConnect_v2() can be used to get the old behavior of objects persisting after disconnect.
- dcgmConnect_v2() was added as a method for specifying additional connection options when connecting to the host engine.
- dcgmUnwatchFields() was added as a method of unwatching fields that were previously watched with dcgmWatchFields()
- dcgmActionValidate_v2() was added to be able to pass more parameters to the DCGM GPU Diagnostic.
- dcgmDiagResponse_t was increased from v2 to v3. See dcgmDiagResponse_v3 for details

## 1.2.3

- No API changes in this version.

## 1.1.1

- dcgmGetAllSupportedDevices() was added as a method to get DCGM-supported GPU Ids. dcgmGetAllDevices() can still be used to get all GPU Ids in the system.

## 1.0.0

- Initial Release.

# Chapter 2. Modules

Here is a list of all modules:



## 2.1. Host

Flex IO SDK host API for DPA programs. Mostly used for DPA resource management and invocation of DPA programs.

## struct flexio_affinity

## struct flexio_app_attr

## struct flexio_cmdq_attr

## struct flexio_cq_attr

## struct flexio_event_handler_attr

## struct flexio_log_dev_attr

## struct flexio_mkey_attr

## struct flexio_process_attr

## struct flexio_qmem

## struct flexio_qp_attr

## struct flexio_qp_attr_opt_param_mask

## struct flexio_wq_attr

## struct flexio_wq_sq_attr

## enum flexio_affinity_type

Flex IO thread affinity types.

### Values

**FLEXIO_AFFINITY_NONE**
**FLEXIO_AFFINITY_STRICT**
**FLEXIO_AFFINITY_GROUP**

## enum flexio_cmdq_state

Flex IO command queue states.

### Values

**FLEXIO_CMDQ_STATE_PENDING = 0**
**FLEXIO_CMDQ_STATE_RUNNING = 1**

## enum flexio_cq_period_mode_t

Flex IO CQ CQE compression period modes.

### Values

**FLEXIO_CQ_PERIOD_MODE_EVENT = 0x0**
**FLEXIO_CQ_PERIOD_MODE_CQE = 0x1**

## enum flexio_cqe_comp_type

Flex IO CQ CQE compression modes.

### Values

**FLEXIO_CQE_COMP_NONE = 0x0**
**FLEXIO_CQE_COMP_BASIC = 0x1**
**FLEXIO_CQE_COMP_ENH = 0x2**

## enum flexio_log_dev_sync_mode

Flex IO device logging synchronization modes.

### Values

**FLEXIO_LOG_DEV_SYNC_MODE_SYNC**
**FLEXIO_LOG_DEV_SYNC_MODE_ASYNC**

## enum flexio_log_lvl_t

Flex IO SDK host logging levels

### Values

**FLEXIO_LOG_LVL_ERR = 0**
**FLEXIO_LOG_LVL_WARN = 1**
**FLEXIO_LOG_LVL_INFO = 2**
**FLEXIO_LOG_LVL_DBG = 3**

## enum flexio_memtype

Flex IO memory types.

### Values

**FLEXIO_MEMTYPE_DPA = 0**
**FLEXIO_MEMTYPE_HOST = 1**

## enum flexio_qp_op_types

Flex IO QP operation types.

### Values

**FLEXIO_QP_WR_RDMA_WRITE = 0x4**
**FLEXIO_QP_WR_RDMA_READ = 0x8**
**FLEXIO_QP_WR_ATOMIC_CMP_AND_SWAP = 0x10**

## enum flexio_qp_qpc_mtu

Flex IO QP possible MTU values.

### Values

**FLEXIO_QP_QPC_MTU_BYTES_256 = 0x1**
**FLEXIO_QP_QPC_MTU_BYTES_512 = 0x2**
**FLEXIO_QP_QPC_MTU_BYTES_1K = 0x3**
**FLEXIO_QP_QPC_MTU_BYTES_2K = 0x4**
**FLEXIO_QP_QPC_MTU_BYTES_4K = 0x5**

## enum flexio_qp_state

Flex IO QP states.

### Values

**FLEXIO_QP_STATE_RST**
**FLEXIO_QP_STATE_INIT**
**FLEXIO_QP_STATE_RTR**
**FLEXIO_QP_STATE_RTS**

## enum flexio_qp_transport_type

Flex IO QP states.

### Values

**FLEXIO_QPC_ST_RC = 0x0**
**FLEXIO_QPC_ST_UC = 0x1**
**FLEXIO_QPC_ST_UD = 0x2**
**FLEXIO_QPC_ST_XRC = 0x3**
**FLEXIO_QPC_ST_IBL2 = 0x4**
**FLEXIO_QPC_ST_DCI = 0x5**
**FLEXIO_QPC_ST_QP0 = 0x7**
**FLEXIO_QPC_ST_QP1 = 0x8**
**FLEXIO_QPC_ST_RAW_DATAGRAM = 0x9**
**FLEXIO_QPC_ST_REG_UMR = 0xc**
**FLEXIO_QPC_ST_DC_CNAK = 0x10**

## enum flexio_status

Flex IO API function return codes.

### Values

**FLEXIO_STATUS_SUCCESS = 0**
**FLEXIO_STATUS_FAILED = 1**
**FLEXIO_STATUS_TIMEOUT = 2**
**FLEXIO_STATUS_FATAL_ERR = 3**

## typedef void (flexio_func_arg_pack_fn_t)

Callback function to pack the arguments for a function.

This function is called internally from the FlexIO runtime upon user making a call (e.g., flexio_process_call). It packs the arguments for a user function into the argument buffer provided in `argbuf`. The argument list can be arbitrarily long and is represented by `ap`. The correct usage of this function requires the caller to initialize the list using `va_start`.

## typedef void (flexio_func_t)

Flex IO application function prototype.

## typedef uint64_t flexio_uintptr_t

Flex IO address type.

# flexio_status flexio_app_create (flexio_app_attr *fattr, flexio_app **app)

Create a container for a FlexIO App.

## Parameters

**fattr**

- A pointer to the application attributes struct.

**app**

## Returns

flexio status value.

## Description

This function creates a named app with a given ELF buffer. It is called from within the constructor generated by the compiler.

# flexio_status flexio_app_destroy (flexio_app *app)

Destroy a flexio app.

## Parameters

**app**

- App that was created before.

## Returns

flexio status value.

## Description

This function destroys the state associated with the app and all registered functions. This function will free the internal elf buffer. It is called from within the destructor generated by the compiler.

## flexio_status flexio_app_get_elf (flexio_app *app, uint64_t *bin_buff, size_t bin_size)

Retrieve ELF binary associated with application.

### Parameters

**app**
- App that created before.

**bin_buff**
- Pointer to buffer to copy ELF binary.

**bin_size**
- Size of buffer pointed by bin_buff. If parameter is smaller than ELF binary size function will fail.

### Returns

flexio status value.

### Description

This function registers the function name, stub address with the runtime. Compiler calls this from within the constructor.

## size_t flexio_app_get_elf_size (flexio_app *app)

Gets a Flex IO application size.

### Parameters

**app**
- A pointer to a Flex IO application.

### Returns

the application's size (bytes) or NULL on error.

## flexio_status flexio_app_get_list (flexio_appapp_list, uint32_t *num_apps)

Get a list of FlexIO Apps that are available.

### Parameters

**app_list**
- A list of apps that are available.

**num_apps**

### Returns

flexio status value.

### Description

This function returns a list of Flex IO apps that are loaded.

## const char *flexio_app_get_name (flexio_app *app)

Gets a Flex IO application name.

### Parameters

**app**
- A pointer to a Flex IO application.

### Returns

the application's name or NULL on error.

## flexio_status flexio_app_list_free (flexio_app **apps_list)

Free the list of flexio apps.

### Returns

flexio status value.

### Description

This function frees the list of apps obtained from `flexio_app_get_list`.

## flexio_status flexio_buf_dev_alloc (flexio_process *process, size_t buff_bsize, flexio_uintptr_t *dest_daddr_p)

Allocates a buffer on Flex IO heap memory.

### Parameters

**process**
- A pointer to the Flex IO process context.

**buff_bsize**
- The size of the buffer to allocate.

**dest_daddr_p**
- A pointer to the Flex IO address, where the buffer was allocated.

### Returns

flexio status value.

### Description

This function allocates a buffer with the requested size on the Flex IO heap memory. On success - sets dest_daddr_p to the start address of the allocated buffer. On Failure - sets dest_daddr_p to 0x0.

## flexio_status flexio_buf_dev_free (flexio_process *process, flexio_uintptr_t daddr)

Deallocates Flex IO heap memory buffer.

### Parameters

**process**
- A pointer to the Flex IO process context.

**daddr**
- A pointer to an address of allocated memory on the Flex IO heap. Zero value is valid argument.

### Returns

flexio status value.

### Description

This function frees Flex IO heap memory buffer by address.

## flexio_status flexio_buf_dev_memset (flexio_process *process, int value, size_t buff_bsize, flexio_uintptr_t dest_daddr)

Sets DPA heap memory buffer to a given value.

### Parameters

**process**
- A pointer to the Flex IO process context.

**value**
- A value to set the DPA heap memory buffer to.

**buff_bsize**
- The size of the Flex IO heap memory buffer.

**dest_daddr**
- Flex IO heap memory buffer address to set.

### Returns

flexio status value.

## flexio_status flexio_cmdq_create (flexio_process *process, flexio_cmdq_attr *fattr, flexio_cmdq **cmdq)

Create asynchronous rpc command queue.

### Parameters

**process**
- A pointer to the process context.

**fattr**
- A pointer to the command queue attributes struct.

**cmdq**
- A pointer to the created command queue context pointer.

### Returns

flexio status value.

### Description

This function creates the asynchronous rpc command queue infrastructure allowing background tasks execution.

## flexio_status flexio_cmdq_destroy (flexio_cmdq *cmdq)

Destroy the command queue infrastructure.

### Parameters

**cmdq**
- A pointer to the command queue context.

### Returns

flexio status value.

### Description

This function destroy the command queue infrastructure and release all its resources.

## flexio_cmdq_is_empty (flexio_cmdq *cmdq)

Check if command queue is empty.

### Parameters

**cmdq**

- A pointer to the command queue context.

### Returns

boolean.

### Description

This function checks if the command queue is empty and all jobs up to this point where performed.

## flexio_status flexio_cmdq_state_running (flexio_cmdq *cmdq)

Move command queue to running state.

### Parameters

**cmdq**

- A pointer to the command queue context.

### Returns

flexio status value.

### Description

This function moves the command queue to running state in the case the queue was create in pending state. Otherwise has no affect.

## flexio_status flexio_cmdq_task_add (flexio_cmdq *cmdq, flexio_func_t *host_func, uint64_t arg)

Add a task to the asynchronous rpc command queue.

### Parameters

**cmdq**
- A pointer to the command queue context.

**host_func**
- host stub function for DPA function to execute.

**arg**
- user argument to function.

### Returns

flexio status value.

### Description

This function adds a task to the asynchronous rpc command queue to be executed by DPA in background. allowing background jobs execution.

## flexio_status flexio_copy_from_host (flexio_process *process, void *src_haddr, size_t buff_bsize, flexio_uintptr_t *dest_daddr_p)

Copy from host memory to Flex IO heap memory buffer.

### Parameters

**process**
- A pointer to the Flex IO process context.

**src_haddr**
- An address of the buffer on the host memory.

**buff_bsize**
- The size of the buffer to copy.

**dest_daddr_p**
- A pointer to the Flex IO address, where the buffer was copied to.

### Returns

flexio status value.

### Description

This function copies data from a buffer on the host memory to the Flex IO memory. The function allocates memory on the device heap which dest_address points to. It is the caller responsibility to deallocate this memory when it is no longer used.

## flexio_status flexio_coredump_create (flexio_process *process, const char *corefile)

Create a DPA core dump of the process.

### Parameters

**process**

- A pointer to a flexio_process

**corefile**

- pathname to write ELF formatted core dump data too. The file/path is passed directly to fopen().

### Returns

flexio status value.

### Description

This function creates a core dump image of a process and all it's threads, and is intended to be used after a fatal error or abnormal termination to allow the user to debug DPA application code.

There must be sufficient free memory to allocate 2-3 times the maximum core file size for intermediate processing before the elf file is written.

Memory windows that may be referenced by DPA code are *not* dumped by this code and must be handled separately if the data is desired.

## flexio_status flexio_cq_create (flexio_process *process, ibv_context *ibv_ctx, const flexio_cq_attr *fattr, flexio_cq **cq)

Creates a Flex IO CQ.

### Parameters

**process**

- A pointer to the Flex IO process.

**ibv_ctx**
- A pointer to an IBV device context (might be different than process'). If NULL - process' will be used.

**fattr**
- A pointer to the CQ attributes struct.

**cq**
- A pointer to the created CQ context pointer.

### Returns

flexio status value.

### Description

This function creates a Flex IO CQ.

## flexio_status flexio_cq_destroy (flexio_cq *cq)

Destroys a Flex IO CQ.

### Parameters

**cq**
- A pointer to a CQ context.

### Returns

flexio status value.

### Description

This function destroys a Flex IO CQ.

## uint32_t flexio_cq_get_cq_num (flexio_cq *cq)

Gets the Flex IO CQ number.

### Parameters

**cq**
- A pointer to a Flex IO CQ.

### Returns

the CQ number or UINT32_MAX on error.

## flexio_status flexio_cq_modify_moderation (flexio_cq *cq, uint16_t max_count, uint16_t period, uint16_t mode)

Modifies a Flex IO CQ moderation configuration.

### Parameters

**cq**
- A pointer to a CQ context.

**max_count**
- CQ moderation max count value.

**period**
- CQ moderation period value.

**mode**
- CQ moderation mode value.

### Returns

flexio status value.

## flexio_status flexio_cq_query_moderation (flexio_cq *cq, uint16_t *max_count, uint16_t *period, uint16_t *mode)

Queries a Flex IO CQ moderation configuration.

### Parameters

**cq**
- A pointer to a CQ context.

**max_count**
- A pointer to the CQ moderation max count value.

**period**
- A pointer to the CQ moderation period value.

**mode**
- A pointer to the CQ moderation mode value.

### Returns

flexio status value.

## flexio_status flexio_device_mkey_create (flexio_process *process, flexio_mkey_attr *fattr, flexio_mkey **mkey)

Creates an Mkey to the process device UMEM.

### Parameters

**process**
- A pointer to the Flex IO process context.

**fattr**
- A pointer to a Flex IO MKey attribute struct.

**mkey**
- A pointer to a pointer to the created MKey struct.

### Returns

flexio status value.

### Description

This function creates an MKey over the provided PD for the provided process device UMEM. The mkey_id will point to the field in the containing flexio_mkey object.

## flexio_status flexio_device_mkey_destroy (flexio_mkey *mkey)

destroys an MKey object containing the given ID

### Parameters

**mkey**
- A pointer to the Flex IO MKey to destroy. NULL is a valid value.

### Returns

flexio status value.

### Description

This function destroys an Mkey object containing the given ID.

# flexio_err_handler_fd (flexio_process *process)

Get file descriptor for error handler.

## Parameters

**process**

- A pointer to the Flex IO process.

## Returns

- file descriptor.

## Description

User should get fd in order to monitor for nonrecoverable errors

User can poll all created processes, using select/poll/epoll functions family.

# flexio_err_status (flexio_process *process)

Check if unrecoverable error occurred.

## Parameters

**process**

- A pointer to the Flex IO process.

## Returns

- nonzero value if error happen.

## Description

User can call this function many times. Error status will be reset only after calling flexio_err_display();

It is suggested to check error status after every negotiation with DPA and periodically later. Alternative way - to poll file descriptor got from flexio_err_handler_fd() in order to get events about error

## flexio_status flexio_event_handler_create (flexio_process *process, flexio_event_handler_attr *fattr, const flexio_outbox *outbox, flexio_event_handler **event_handler_ptr)

Creates a Flex IO event handler.

### Parameters

**process**
- A pointer to the Flex IO process.

**fattr**
- A pointer to the event handler attributes struct.

**outbox**
- Default outbox (or NULL for no default outbox).

**event_handler_ptr**
- A pointer to the created event handler context pointer.

### Returns

flexio status value.

### Description

This function creates a Flex IO event handler for an existing Flex IO process.

## flexio_status flexio_event_handler_destroy (flexio_event_handler *event_handler)

Destroys a Flex IO event handler.

### Parameters

**event_handler**
- A pointer to an event handler context.

### Returns

flexio status value.

### Description

This function destroys a Flex IO event handler.

## uint32_t flexio_event_handler_get_id (flexio_event_handler *event_handler)

Gets the ID from a Flex IO event handler's thread metadata.

### Parameters

**event_handler**
- A pointer to a Flex IO event handler.

### Returns

the event handler's thread ID or UINT32_MAX on error.

## flexio_thread *flexio_event_handler_get_thread (flexio_event_handler *event_handler)

Gets a Flex IO thread object from a Flex IO event handler.

### Parameters

**event_handler**
- A pointer to a Flex IO event handler.

### Returns

the event handler's thread or NULL on error.

## flexio_status flexio_event_handler_run (flexio_event_handler *event_handler, uint64_t user_arg)

Run a Flex IO event handler.

### Parameters

**event_handler**
- A pointer to an event handler context.

**user_arg**
- A 64 bit argument for the event handler's thread.

### Returns

flexio status value.

### Description

This function makes a Flex IO event handler start running.

# flexio_status flexio_func_get_register_info (flexio_app *app, flexio_func_t *host_stub_func_addr, uint32_t *pup, char *dev_func_name, char *dev_unpack_func_name, size_t func_name_size, size_t *argbuf_size, flexio_func_arg_pack_fn_t **host_pack_func, flexio_uintptr_t *dev_func_addr, flexio_uintptr_t *dev_unpack_func_addr)

Obtain info for previously registered function.

## Parameters

**app**
- FlexIO app.

**host_stub_func_addr**
- Known host stub func addr.

**pup**
- Whether function has been registered with pack/unpack support (0: No, 1:Yes).

**dev_func_name**
- Name of device function.

**dev_unpack_func_name**
- Name of unpack routine on device, NA if pup == 0.

**func_name_size**
- Size of function name len allocated.

**argbuf_size**
- Size of argument buffer, NA if pup == 0.

**host_pack_func**
- Function pointer to host packing routine, NA if pup == 0.

**dev_func_addr**
- address of device function.

**dev_unpack_func_addr**
- address of device unpack function.

## Returns

flexio status value.

### Description

This function is used to obtain info about a previously registered function. It is used to compose higher-level libraries on top of DPACC / FlexIO interface. It is not intended to be used directly by the user.

The caller must ensure that the string pointers have been allocated and are at least `FLEXIO_MAX_NAME_LEN + 1` long to ensure that the call doesn't fail to copy full function name.

## flexio_status flexio_func_pup_register (flexio_app *app, const char *dev_func_name, const char *dev_unpack_func_name, flexio_func_t *host_stub_func_addr, size_t argbuf_size, flexio_func_arg_pack_fn_t *host_pack_func)

Register a function name at application start.

### Parameters

**app**
- App that created before.

**dev_func_name**
- The device function name (entry point). Length of name should be up to FLEXIO_MAX_NAME_LEN bytes.

**dev_unpack_func_name**
- The device wrapper function that unpacks the argument buffer. Length of name should be up to FLEXIO_MAX_NAME_LEN bytes.

**host_stub_func_addr**
- The host stub function that is used by the application to reference the device function.

**argbuf_size**
- Size of the argument buffer required by this function.

**host_pack_func**
- Host callback function that packs the arguments.

### Returns

flexio status value.

### Description

This function registers the function name, stub address with the runtime. It is called from within the constructor generated by the compiler.

## flexio_status flexio_func_register (flexio_app *app, const char *dev_func_name, flexio_func_t **out_func)

Register a function to be used later.

### Parameters

**app**
- previously created flexio app.

**dev_func_name**
- name of flexio function on device that will be called. Length of name should be up to FLEXIO_MAX_NAME_LEN bytes.

**out_func**
- opaque handle to use with flexio_process_call(), flexio_event_handler_create(), ...

### Returns

flexio status value.

### Description

This function is intended to be called directly by user in the situation where they don't desire pack/unpack support that is typically done by the compiler interface.

It is the user's responsibility to ensure that a function was annotated for event handler with __dpa_global__. The runtime will not provide any type checking. A mismatched call will result in undefined behavior.

## flexio_status flexio_host2dev_memcpy (flexio_process *process, void *src_haddr, size_t buff_bsize, flexio_uintptr_t dest_daddr)

Copy from host memory to a pre-allocted Flex IO heap memory buffer.

### Parameters

**process**
- A pointer to the Flex IO process context.

**src_haddr**
- An address of the buffer on the host memory.

**buff_bsize**
- The size of the buffer to copy.

**dest_daddr**
- Flex IO heap memory buffer address to copy to.

### Returns

flexio status value.

### Description

This function copies data from a buffer on the host memory to a buffer on the Flex IO heap memory.

## flexio_status flexio_log_dev_destroy (flexio_process *process)

Destroys a flexio print environment.

### Parameters

**process**
- A pointer to the Flex IO process.

### Returns

flexio status value.

### Description

This function destroys and releases all resources, allocated for process printing needs by flexio_log_dev_init().

## flexio_status flexio_log_dev_flush (flexio_process *process)

Flush print buffer in case of asynchronous print.

### Parameters

**process**
- A pointer to the Flex IO process.

### Returns

flexio status value.

### Description

All data from print buffer will be flushed to file, defined in flexio_log_dev_init().

In case of synchronous print this functions does nothing. This function allocates resources to support printing from Flex IO to HOST.

## flexio_status flexio_log_dev_init (flexio_process *process, flexio_log_dev_attr *log_dev_fattr, FILE *out, pthread_t *ppthread)

Create environment to support print from DPA.

### Parameters

**process**
- A pointer to the Flex IO process.

**log_dev_fattr**
- A pointer to the print attributes struct.

**out**
- file to save data from Flex IO. Use stdout if you want receive data on HOST's console

**ppthread**
- A pointer to receive phread ID of created thread. May be NULL if user doesn't need it.

### Returns

flexio status value.

### Description

This function allocates resources to support printing from Flex IO to HOST.

Print works in the following modes: synchronous or asynchronous. Under synchronous mode, a dedicated thread starts to receive data and prints it immediately.
When asynchronous mode is in operation, all print buffers will be flushed by flexio_log_dev_flush(). Buffer can be overrun.

This function doesn't have a "destroy" procedure. All printing infrastructure will be closed and the resources will be released using the flexio_process_destroy() function.

# flexio_log_lvl_t flexio_log_lvl_set (flexio_log_lvl_t lvl)

Sets host SDK logging level.

## Parameters

**lvl**

- logging level to set. All entries with this or higher priority level will be printed.

## Returns

flexio_log_lvl_t.

## Description

This function sets the host logging level. Changing the logging level may change the visibility of some logging entries in the SDK code.

# uint32_t flexio_mkey_get_id (flexio_mkey *mkey)

Gets the Flex IO MKey ID.

## Parameters

**mkey**

- A pointer to a Flex IO MKey.

## Returns

the Flex IO mkey ID or UINT32_MAX on error.

# flexio_status flexio_outbox_create (flexio_process *process, ibv_context *other_ctx, flexio_uar *uar, flexio_outbox **outbox)

Creates a Flex IO outbox.

## Parameters

**process**

- A pointer to the Flex IO process.

**other_ctx**

- An IBV context for creating the PRM object (Different than process's on multi GVMI case, NULL or same as process's otherwise).

**uar**
- A Flex IO UAR struct created for the outbox's IBV device.

**outbox**
- A pointer to the created outbox context pointer.

### Returns

flexio status value.

### Description

This function Creates a Flex IO outbox for the given process.

## flexio_status flexio_outbox_destroy (flexio_outbox *outbox)

Destroys a Flex IO outbox.

### Parameters

**outbox**
- A pointer to a outbox context.

### Returns

flexio status value.

### Description

This function destroys a Flex IO outbox.

## uint32_t flexio_outbox_get_id (flexio_outbox *outbox)

Gets the Flex IO outbox ID.

### Parameters

**outbox**
- A pointer to a Flex IO outbox.

### Returns

the Flex IO outbox ID or UINT32_MAX on error.

## flexio_uar *flexio_outbox_get_uar (flexio_outbox *outbox)

Gets a Flex IO UAR object from a Flex IO outbox.

### Parameters

**outbox**
- A pointer to a Flex IO outbox.

### Returns

the Flex IO outbox UAR object or NULL on error.

## flexio_status flexio_process_call (flexio_process *process, flexio_func_t *host_func, uint64_t *func_ret, ...)

Calls a Flex IO process.

### Parameters

**process**
- A pointer to the Flex IO process to run.

**host_func**
- The host stub function that is used by the application to reference the device function.

**func_ret**
- A pointer to the ELF function return value.

### Returns

flexio status value.

## flexio_status flexio_process_create (ibv_context *ibv_ctx, flexio_app *app, const flexio_process_attr *process_attr, flexio_process **process_ptr)

Create a new Flex IO process.

### Parameters

**ibv_ctx**
- A pointer to a device context.

**app**
- Device side application handle.

**process_attr**
- Optional, process attributes for create. Can be NULL.

**process_ptr**
- A pointer to the created process pointer.

### Returns

flexio status value.

### Description

This function creates a new Flex IO process with requested image.

## flexio_status flexio_process_destroy (flexio_process *process)

Destroys a new Flex IO process.

### Parameters

**process**
- A pointer to a process.

### Returns

flexio status value.

### Description

This function destroys a new Flex IO process.

## flexio_status flexio_process_error_handler_set (flexio_process *process, flexio_func_t *error_handler)

Set the Flexio process error handler.

### Parameters

**process**
- A pointer to a process

**error_handler**
- The host stub function that is used as a reference to the error handler function.

### Returns

flexio status value.

### Description

This function sets the Flex IO process error handler. The error handler must be set after the process is created, and before the first thread is created. The function registered for error handler should be annotated with __dpa_global__.

## ibv_pd *flexio_process_get_pd (flexio_process *process)

Gets a Flex IO IBV PD object from a Flex IO process.

### Parameters

**process**
- A pointer to a Flex IO process.

### Returns

the process's PD object or NULL on error.

## flexio_status flexio_qp_create (flexio_process *process, ibv_context *ibv_ctx, flexio_qp_attr *qp_fattr, flexio_qp **qp_ptr)

Creates a Flex IO QP.

### Parameters

**process**
- A pointer to the Flex IO process.

**ibv_ctx**
- A pointer to an IBV device context (might be different than process'). If NULL - process' will be used.

**qp_fattr**
- A pointer to the QP attributes struct.

**qp_ptr**
- A pointer to the created QP context pointer.

### Returns

flexio status value.

### Description

This function creates a Flex IO QP.

## flexio_status flexio_qp_destroy (flexio_qp *qp)

Destroys a Flex IO QP.

### Parameters

**qp**
- A pointer to the QP context.

### Returns

flexio status value.

### Description

This function destroys a Flex IO QP.

## uint32_t flexio_qp_get_qp_num (flexio_qp *qp)

Gets the Flex IO QP number.

### Parameters

**qp**
- A pointer to a Flex IO QP.

### Returns

the QP number or UINT32_MAX on error.

## flexio_status flexio_qp_modify (flexio_qp *qp, flexio_qp_attr *fattr, flexio_qp_attr_opt_param_mask *mask)

Modify Flex IO QP.

### Parameters

**qp**
- A pointer to the QP context.

**fattr**
- A pointer to the QP attributes struct that will also define the QP connection.

**mask**
- A pointer to the optional QP attributes mask.

### Returns

flexio status value.

### Description

This function modifies Flex IO QP and transition it between states. At the end of the procedure Flex IO QP would have moved from it's current state to to next state, given in the fattr, if the move is a legal transition in the QP's state machine.

## flexio_status flexio_rq_create (flexio_process *process, ibv_context *ibv_ctx, uint32_t cq_num, const flexio_wq_attr *fattr, flexio_rq **flexio_rq_ptr)

Creates a Flex IO RQ.

### Parameters

**process**
- A pointer to the Flex IO process.

**ibv_ctx**
- A pointer to an IBV device context (might be different than process'). If NULL - process' will be used.

**cq_num**
- A CQ number.

**fattr**
- A pointer to the RQ attributes struct.

**flexio_rq_ptr**

### Returns

flexio status value.

### Description

This function creates a Flex IO RQ.

## flexio_status flexio_rq_destroy (flexio_rq *flexio_rq)

Destroys a Flex IO RQ.

### Returns

flexio status value.

### Description

This function destroys a Flex IO RQ.

## mlx5dv_devx_obj *flexio_rq_get_tir (flexio_rq *rq)

Gets the Flex IO RQ TIR object.

### Parameters

**rq**
- A pointer to a Flex IO RQ.

### Returns

the RQ TIR object or NULL on error.

## uint32_t flexio_rq_get_wq_num (flexio_rq *rq)

Gets the Flex IO RQ number.

### Parameters

**rq**
- A pointer to a Flex IO RQ.

### Returns

the RQ number or UINT32_MAX on error.

## flexio_status flexio_rq_set_err_state (flexio_rq *rq)

Sets a Flex IO RQ to error state.

### Parameters

**rq**
- A pointer to the RQ context to move to error state.

### Returns

flexio status value.

### Description

This function sets a Flex IO RQ to error state.

# flexio_status flexio_sq_create (flexio_process *process, ibv_context *ibv_ctx, uint32_t cq_num, const flexio_wq_attr *fattr, flexio_sq **flexio_sq_ptr)

Creates a Flex IO SQ.

## Parameters

**process**
- A pointer to the Flex IO process.

**ibv_ctx**
- A pointer to an IBV device context (might be different than process'). If NULL - process' will be used.

**cq_num**
- A CQ number (can be Flex IO or host CQ).

**fattr**
- A pointer to the SQ attributes struct.

**flexio_sq_ptr**

## Returns

flexio status value.

## Description

This function creates a Flex IO SQ.

# flexio_status flexio_sq_destroy (flexio_sq *flexio_sq)

Destroys a Flex IO SQ.

## Returns

flexio status value.

## Description

This function destroys a Flex IO SQ.

## uint32_t flexio_sq_get_wq_num (flexio_sq *sq)

Gets the Flex IO SQ number.

### Parameters

**sq**
- A pointer to a Flex IO SQ.

### Returns

the SQ number or UINT32_MAX on error.

## flexio_status flexio_uar_create (flexio_process *process, mlx5dv_devx_uar *devx_uar, flexio_uar **flexio_uar)

Creates a Flex IO UAR object.

### Parameters

**process**
- A pointer to the Flex IO process context.

**devx_uar**
- A pointer to a DevX UAR struct.

**flexio_uar**
- A pointer to a pointer to the created Flex IO UAR struct.

### Returns

flexio status value.

### Description

This function creates a Flex IO UAR object from a DevX UAR object.

## flexio_status flexio_uar_destroy (flexio_uar *uar)

destroys a Flex IO UAR object

### Parameters

**uar**
- A pointer to the Flex IO UAR to destroy.

### Returns

flexio status value.

### Description

This function destroys a Flex IO UAR object.

## mlx5dv_devx_uar *flexio_uar_get_devx_uar (flexio_uar *flexio_uar)

Gets a DevX UAR object from a Flex IO UAR.

### Parameters

**flexio_uar**
- A pointer to a Flex IO UAR.

### Returns

the Flex IO UAR internal DevX object or NULL on error.

## flexio_status flexio_window_create (flexio_process *process, ibv_pd *pd, flexio_window **window)

Creates a Flex IO window.

### Parameters

**process**
- A pointer to the Flex IO process.

**pd**
- A pointer to a protection domain struct to the memory the window should access.

**window**
- A pointer to the created window context pointer.

### Returns

flexio status value.

### Description

This function Creates a Flex IO window for the given process.

## flexio_status flexio_window_destroy (flexio_window *window)

Destroys a Flex IO window.

### Parameters

**window**
- A pointer to a window context.

### Returns

flexio status value.

### Description

This function destroys a Flex IO window.

## uint32_t flexio_window_get_id (flexio_window *window)

Gets the Flex IO window ID.

### Parameters

**window**
- A pointer to a Flex IO window.

### Returns

the Flex IO window ID or UINT32_MAX on error.

## #define FLEXIO_MAX_NAME_LEN (256)

Maximum length of application and device function names

# 2.2. Dev

Flex IO SDK device API for DPA programs. Includes services for DPA programs.

## struct spinlock_s

## enum cq_ce_mode

Flex IO dev CQ CQE creation modes.

### Values

**MLX5_CTRL_SEG_CE_CQE_ON_CQE_ERROR = 0x0**
**MLX5_CTRL_SEG_CE_CQE_ON_FIRST_CQE_ERROR = 0x1**
**MLX5_CTRL_SEG_CE_CQE_ALWAYS = 0x2**
**MLX5_CTRL_SEG_CE_CQE_AND_EQE = 0x3**

## enum flexio_dev_status_t

Return status of Flex IO dev API functions.

### Values

**FLEXIO_DEV_STATUS_SUCCESS = 0**
**FLEXIO_DEV_STATUS_FAILED = 1**

## typedef void (flexio_dev_async_rpc_handler_t)

Asynchronous RPC handler callback function type.

Defines an RPC handler callback function.

arg - argument of the RPC function.

return void.

## typedef void (flexio_dev_event_handler_t)

Event handler callback function type.

Defines an event handler callback function. On handler function end, need to call flexio_dev_process_finish() instead of a regular return statement, in order to properly release resources back to the OS.

thread_arg - an argument for the executing thread.

return void.

## flexio_dev_get_thread_ctx (flexio_dev_thread_ctx **dtctx)

Request thread context.

### Parameters

**dtctx**

- A pointer to a pointer of flexio_dev_thread_ctx structure.

### Returns

0 on success negative value on failure.

### Description

This function requests the thread context. Should be called for every start of thread.

## uint32_t flexio_dev_get_thread_id (flexio_dev_thread_ctx *dtctx)

Get thread ID from thread context.

### Parameters

**dtctx**

- A pointer to a flexio_dev_thread_ctx structure.

### Returns

thread ID value.

### Description

This function queries a thread context for its thread ID (from thread metadata).

## flexio_uintptr_t flexio_dev_get_thread_local_storage (flexio_dev_thread_ctx *dtctx)

Get thread local storage address from thread context.

### Parameters

**dtctx**

- A pointer to a flexio_dev_thread_ctx structure.

### Returns

thread local storage value.

### Description

This function queries a thread context for its thread local storage (from thread metadata).

## flexio_dev_log (flexio_log_dev_level level, const char *format, ...)

Creates logging entries and outputs from the device to the host side.

### Returns

- same as from regular printf.

### Description

[in] level - logging level. [in] format, ... - same as for regular printf.

## flexio_dev_status_t flexio_dev_outbox_config (flexio_dev_thread_ctx *dtctx, uint16_t outbox_config_id)

Config thread outbox object.

### Parameters

**dtctx**
- A pointer to flexio_dev_thread_ctx structure.

**outbox_config_id**
- The outbox object config id.

### Returns

flexio_dev_status_t.

### Description

This function updates the thread outbox object of the given thread context.

# int int flexio_dev_print (const char *format, ...)

Same as a regular printf but with protection from simultaneous print from different threads.

## Returns

- same as from regular printf.

## Description

[in] - same as for regular printf.

# flexio_dev_process_finish (void)

Exit flexio process (no errors).

## Description

This function releases resources back to OS and returns '0x40' in dpa_process_status. All threads for the current process will stop executing and no new threads will be able to trigger for this process. Threads state will NOT be changes to 'finished' (will remain as is).

# flexio_dev_puts (flexio_dev_thread_ctx *dtctx, char *str)

Put a string to printing queue.

## Parameters

**dtctx**
- A pointer to a pointer of flexio_dev_thread_ctx structure.

**str**
- A pointer to string.

## Returns

length of printed string.

## Description

This function puts a string to host printing queue. This queue has been serviced by host application. Would have no effect, if the host application didn't configure printing environment. In order to initalize/configure printing envoirment - On HOST side - after flexio_process_create, flexio_log_dev_init should be called. On DEV side - before using flexio_dev_puts, the thread context is needed, therefore flexio_dev_get_thread_ctx should be called before.

# flexio_dev_thread_finish (void)

Exit from a thread, mark it as finished.

## Description

This function releases resources back to OS. The thread will be marked as finished so next DUAR will not trigger it.

# flexio_dev_thread_reschedule (void)

Exit from a thread, leave process active.

## Description

This function releases resources back to OS. For the next DUAR the thread will restart from the beginning.

# flexio_dev_status_t flexio_dev_window_config (flexio_dev_thread_ctx *dtctx, uint16_t window_config_id, uint32_t mkey)

Config thread window object.

## Parameters

**dtctx**
- A pointer to a flexio_dev_thread_ctx structure.

**window_config_id**
- The window object id.

**mkey**
- mkey object.

## Returns

flexio_dev_status_t.

## Description

This function updates the thread window object of the given thread context.

# flexio_dev_status_t flexio_dev_window_copy_from_host (flexio_dev_thread_ctx *dtctx, void *daddr, uint64_t haddr, uint32_t size)

Copy a buffer from host memory to device memory.

## Parameters

**dtctx**
- A pointer to a flexio_dev_thread_ctx structure.

**daddr**
- A pointer to the device memory buffer.

**haddr**
- A pointer to the host memory allocated buffer.

**size**
- Number of bytes to copy.

## Returns

flexio_dev_status_t.

## Description

This function copies specified number of bytes from host memory to device memory. UNSUPPORTED at this time.

# flexio_dev_status_t flexio_dev_window_copy_to_host (flexio_dev_thread_ctx *dtctx, uint64_t haddr, const void *daddr, uint32_t size)

Copy a buffer from device memory to host memory.

## Parameters

**dtctx**
- A pointer to a flexio_dev_thread_ctx structure.

**haddr**
- A pointer to the host memory allocated buffer.

**daddr**
- A pointer to the device memory buffer.

**size**
- Number of bytes to copy.

### Returns

flexio_dev_status_t.

### Description

This function copies specified number of bytes from device memory to host memory.

## flexio_dev_status_t flexio_dev_window_mkey_config (flexio_dev_thread_ctx *dtctx, uint32_t mkey)

Config thread window mkey object.

### Parameters

**dtctx**
- A pointer to a flexio_dev_thread_ctx structure.

**mkey**
- mkey object.

### Returns

flexio_dev_status_t.

### Description

This function updates the thread window mkey object of the given thread context.

## flexio_dev_status_t flexio_dev_window_ptr_acquire (flexio_dev_thread_ctx *dtctx, uint64_t haddr, flexio_uintptr_t *daddr)

Generate device address from host allocated memory.

### Parameters

**dtctx**
- A pointer to a flexio_dev_thread_ctx structure.

**haddr**
- Host allocated address.

**daddr**
- A pointer to write the device generated matching address.

### Returns

flexio_dev_status_t.

### Description

This function generates a memory address to be used by device to access host side memory, according to already create window object. from a host allocated address.

## int int void flexio_dev_yield (flexio_dev_thread_ctx *dtctx)

exit point for continuable event handler routine

### Parameters

**dtctx**
- A pointer to a flexio_dev_thread_ctx structure.

### Returns

Function does not return.

### Description

This function is used to mark the exit point on continuable event handler where user wishes to continue execution on next event. In order to use this API the event handler must be created with continuable flag enabled, otherwise call will have no effect.

## #define spin_init __atomic_store_n(&((lock)->locked), 0, __ATOMIC_SEQ_CST)

Initialize a spinlock mechanism.

Initialize a spinlock mechanism, must be called before use.

### #define spin_lock do { \ while (__atomic_exchange_n(&((lock)->locked), 1, __ATOMIC_SEQ_CST)); \ } while (0)

Lock a spinlock mechanism.

Lock a spinlock mechanism.

### #define spin_trylock __atomic_exchange_n(&((lock)->locked), 1, __ATOMIC_SEQ_CST)

Atomic try to catch lock.

makes attempt to take lock. Returns immediately.

### #define spin_unlock __atomic_store_n(&((lock)->locked), 0, __ATOMIC_SEQ_CST)

Unlock a spinlock mechanism.

Unlock a spinlock mechanism.

## 2.3. DevErr

Flex IO SDK device API for DPA programs error handling.

### enum flexio_dev_error_t

Flex IO dev errors.

#### Values

**FLEXIO_DEV_ERROR_ILLEGAL_ERR = 0x42**

## __attribute__ ((__noreturn__))

Exit the process and return a user (fatal) error code.

### Description

Error codes returned to the host in the dpa_process_status field of the DPA_PROCESS object are defined as follows: 0: OK 1-63: RTOS or Firmware errors 64-127: Flexio-SDK errors 129-255: User defined

## uint64_t flexio_dev_get_and_rst_errno (flexio_dev_thread_ctx *dtctx)

Get and Reset thread error flag (errno) of recoverable (non fatal) error.

### Parameters

**dtctx**

- A pointer to a flexio_dev_thread_ctx structure.

### Returns

- void.

## uint64_t flexio_dev_get_errno (flexio_dev_thread_ctx *dtctx)

Get thread error flag (errno) of recoverable (non fatal) error.

### Parameters

**dtctx**

- A pointer to a flexio_dev_thread_ctx structure.

### Returns

thread error code.

### Description

This function queries an errno field from thread context.

## flexio_dev_rst_errno (flexio_dev_thread_ctx *dtctx)

Reset thread error flag (errno) of recoverable (non fatal) error.

### Parameters

**dtctx**
- A pointer to a flexio_dev_thread_ctx structure.

# 2.4. DevQueueAccess

Flex IO SDK device API for DPA programs queue access. Provides an API for handling networking queues (WQs/CQs).

## enum flexio_ctrl_seg_t

Flex IO dev WQE control segment types.

### Values

**FLEXIO_CTRL_SEG_SEND_EN = 0**
**FLEXIO_CTRL_SEG_SEND_RC = 1**
**FLEXIO_CTRL_SEG_LDMA = 2**
**FLEXIO_CTRL_SEG_RDMA_WRITE = 3**
**FLEXIO_CTRL_SEG_RDMA_READ = 4**
**FLEXIO_CTRL_SEG_ATOMIC_COMPARE_AND_SWAP = 5**
**FLEXIO_CTRL_SEG_LSO = 6**
**FLEXIO_CTRL_SEG_NOP = 7**
**FLEXIO_CTRL_SEG_RDMA_WRITE_IMM = 8**

## flexio_dev_status_t flexio_dev_cq_arm (flexio_dev_thread_ctx *dtctx, uint32_t ci, uint32_t qnum)

Arm CQ function.

### Parameters

**dtctx**
- A pointer to a pointer of flexio_dev_thread_ctx structure.

**ci**
- Current CQ consumer index.

**qnum**
- Number of the CQ to arm.

### Returns

flexio_dev_status_t.

### Description

Moves a CQ to 'armed' state. This means that next CQE created for this CQ will result in an EQE on the relevant EQ.

## uint32_t flexio_dev_cqe_get_byte_cnt (flexio_dev_cqe64 *cqe)

Get byte count field from CQE function.

### Parameters

**cqe**
- CQE to parse.

### Returns

uint32_t - Byte count field value of the CQE.

### Description

Parse a CQE for its byte count field.

## uint8_t flexio_dev_cqe_get_csum_ok (flexio_dev_cqe64 *cqe)

Get csum OK field from CQE function.

### Parameters

**cqe**
- CQE to parse.

### Returns

uint8_t - csum_ok field value of the CQE.

### Description

Parse a CQE for its csum OK field.

## uint8_t flexio_dev_cqe_get_opcode (flexio_dev_cqe64 *cqe)

Get the opcode field from CQE function.

### Parameters

**cqe**
- CQE to parse.

### Returns

uint8_t - Opcode field value of the CQE.

## uint8_t flexio_dev_cqe_get_owner (flexio_dev_cqe64 *cqe)

Get owner field from CQE function.

### Parameters

**cqe**
- CQE to parse.

### Returns

uint8_t - Owner field value of the CQE.

### Description

Parse a CQE for its owner field.

## uint32_t flexio_dev_cqe_get_qpn (flexio_dev_cqe64 *cqe)

Get QP number field from CQE function.

### Parameters

**cqe**
- CQE to parse.

### Returns

uint32_t - QP number field value of the CQE.

### Description

Parse a CQE for its QP number field.

## uint32_t flexio_dev_cqe_get_user_index (flexio_dev_cqe64 *cqe)

Get the user index field from CQE function.

### Parameters

**cqe**
- CQE to parse.

### Returns

uint32_t - User index field value of the CQE.

## uint16_t flexio_dev_cqe_get_wqe_counter (flexio_dev_cqe64 *cqe)

Get WQE counter filed from CQE function.

### Parameters

**cqe**
- CQE to parse.

### Returns

uint16_t - WQE counter field value of the CQE.

### Description

Parse a CQE for its WQE counter field.

## flexio_dev_status_t flexio_dev_db_ctx_arm (flexio_dev_thread_ctx *dtctx, uint32_t cqn, uint32_t emu_ctx_id)

arm the emulation context

### Parameters

**dtctx**
- A pointer to a pointer of flexio_dev_thread_ctx structure.

**cqn**
- CQ number provided by host.

**emu_ctx_id**
- Emulation context ID, provided by a call on the host to flexio_emu_db_to_cq_ctx_get_id.

## flexio_dev_status_t flexio_dev_db_ctx_force_trigger (flexio_dev_thread_ctx *dtctx, uint32_t cqn, uint32_t emu_ctx_id)

force trigger of emulation context

### Parameters

**dtctx**

**cqn**
- CQ number provided by host.

**emu_ctx_id**
- Emulation context ID, provided by a call on the host to flexio_emu_db_to_cq_ctx_get_id.

## flexio_dev_status_t flexio_dev_dbr_cq_set_ci (uint32_t *cq_dbr, uint32_t ci)

Set consumer index value for a CQ function.

### Parameters

**cq_dbr**
- A pointer to the CQ's doorbell record address.

**ci**
- The consumer index value to update.

### Returns

flexio_dev_status_t.

### Description

Writes an updated consumer index number to a CQ's doorbell record

# flexio_dev_status_t flexio_dev_dbr_rq_inc_pi (uint32_t *rq_dbr)

Increment producer index of an RQ by 1 function.

## Parameters

**rq_dbr**
- A pointer to the CQ's doorbell record address.

## Returns

flexio_dev_status_t.

## Description

Mark a WQE for reuse by incrementing the relevant RQ producer index by 1

# flexio_dev_status_t flexio_dev_dbr_sq_set_pi (uint32_t *sq_dbr, uint32_t pi)

Set producer index value for an SQ function.

## Parameters

**sq_dbr**
- A pointer to the SQ's doorbell record address.

**pi**
- The producer index value to update.

## Returns

flexio_dev_status_t.

## Description

Writes an updated producer index number to an SQ's doorbell record

# flexio_dev_status_t flexio_dev_eq_update_ci (flexio_dev_thread_ctx *dtctx, uint32_t ci, uint32_t qnum)

Update an EQ consumer index function.

## Parameters

**dtctx**
- A pointer to a pointer of flexio_dev_thread_ctx structure.

**ci**
- Current EQ consumer index.

**qnum**
- Number of the EQ to update.

## Returns

flexio_dev_status_t.

## Description

Updates the consumer index of an EQ after handling an EQE.

# uint32_t flexio_dev_eqe_get_cqn (flexio_dev_eqe *eqe)

Get CQ number field from EQE function.

## Parameters

**eqe**
- EQE to parse.

## Returns

uint32_t - CQ number field value of the EQE.

## Description

Parse an EQE for its CQ number field.

## uint8_t flexio_dev_eqe_get_owner (flexio_dev_eqe *eqe)

Get owner field from EQE function.

### Parameters

**eqe**
- EQE to parse.

### Returns

uint32_t - owner field value of the EQE.

### Description

Parse an EQE for its owner field.

## flexio_dev_status_t flexio_dev_msix_send (flexio_dev_thread_ctx *dtctx, uint32_t cqn)

Send msix on the cq linked to the msix eq.

### Parameters

**dtctx**
- A pointer to a flexio_dev_thread_ctx structure.

**cqn**
- CQ number to trigger db on. Trigger is done via currently configured outbox, this can be changed with outbox config API according to CQ.

### Returns

flexio_dev_status_t.

### Description

This function trigger msix on the given cq.

# flexio_dev_status_t flexio_dev_qp_sq_ring_db (flexio_dev_thread_ctx *dtctx, uint16_t pi, uint32_t qnum)

QP/SQ ring doorbell function.

## Parameters

**dtctx**
- A pointer to a pointer of flexio_dev_thread_ctx structure.

**pi**
- Current queue producer index.

**qnum**
- Number of the queue to update.

## Returns

flexio_dev_status_t.

## Description

Rings the doorbell of a QP or SQ in order to alert the HW of pending work.

# void *flexio_dev_rwqe_get_addr (flexio_dev_wqe_rcv_data_seg *rwqe)

Get address field from receive WQE function.

## Parameters

**rwqe**
- WQE to parse.

## Returns

void* - Address field value of the receive WQE.

## Description

Parse a receive WQE for its address field.

# flexio_dev_status_t flexio_dev_swqe_seg_atomic_set (flexio_dev_sqe_seg *swqe, uint64_t swap_or_add_data, uint64_t compare_data)

Fill out an Atomic send queue wqe segment function.

## Parameters

**swqe**
- Send WQE segment to fill.

**swap_or_add_data**
- The data that will be swapped in or the data that will be added.

**compare_data**
- The data that will be compared with. Unused in fetch & add operation.

## Returns

flexio_dev_status_t.

## Description

Fill the fields of a send WQE segment (2 DWORDs) with Atomic segment information. This segment can service a compare & swap or fetch & add operation.

# flexio_dev_status_t flexio_dev_swqe_seg_ctrl_set (flexio_dev_sqe_seg *swqe, uint32_t sq_pi, uint32_t sq_number, uint32_t ce, flexio_ctrl_seg_t ctrl_seg_type)

Fill out a control send queue wqe segment function.

## Parameters

**swqe**
- Send WQE segment to fill.

**sq_pi**
- Producer index of the send WQE.

**sq_number**
- SQ number that holds the WQE.

**ce**
- wanted CQ policy for CQEs. Value is taken from cq_ce_mode enum.

**ctrl_seg_type**
- Type of control segment.

### Returns

flexio_dev_status_t.

### Description

Fill the fields of a send WQE segment (4 DWORDs) with control segment information. This should always be the 1st segment of the WQE.

## flexio_dev_status_t flexio_dev_swqe_seg_eth_set (flexio_dev_sqe_seg *swqe, uint16_t cs_swp_flags, uint16_t mss, uint16_t inline_hdr_bsz, uint8_t inline_hdrs)

Fill out an ethernet send queue wqe segment function.

### Parameters

**swqe**
- Send WQE segment to fill.

**cs_swp_flags**
- Flags for checksum and swap, see PRM section 8.9.4.2, Send WQE Construction Summary.

**mss**
- Maximum Segment Size - For LSO WQEs - the number of bytes in the TCP payload to be transmitted in each packet. Must be 0 on non LSO WQEs.

**inline_hdr_bsz**
- Length of inlined packet headers in bytes. This includes the headers in the inline_data segment as well.

**inline_hdrs**
- First 2 bytes of the inlined packet headers.

### Returns

flexio_dev_status_t.

### Description

Fill the fields of a send WQE segment (4 DWORDs) with Ethernet segment information.

# flexio_dev_status_t flexio_dev_swqe_seg_inline_data_set (flexio_dev_sqe_seg *swqe, uint32_t data_sz, uint32_t *data)

Fill out an inline data send queue wqe segment function.

## Parameters

**swqe**
- Send WQE segment to fill.

**data_sz**
- Size of the data.

**data**
- Inline data array (3 DWORDs).

## Returns

flexio_dev_status_t.

## Description

Fill the fields of a send WQE segment (4 DWORDs) with inline data segment information.

# flexio_dev_status_t flexio_dev_swqe_seg_mem_ptr_data_set (flexio_dev_sqe_seg *swqe, uint32_t data_sz, uint32_t lkey, uint64_t data_addr)

Fill out a memory pointer data send queue wqe segment function.

## Parameters

**swqe**
- Send WQE segment to fill.

**data_sz**
- Size of the data.

**lkey**
- Local memory access key for the data operation.

**data_addr**
- Address of the data for the data operation.

### Returns

flexio_dev_status_t.

### Description

Fill the fields of a send WQE segment (4 DWORDs) with memory pointer data segment information.

## flexio_dev_status_t flexio_dev_swqe_seg_rdma_set (flexio_dev_sqe_seg *swqe, uint32_t rkey, uint64_t data_addr)

Fill out an RDMA send queue wqe segment function.

### Parameters

**swqe**
- Send WQE segment to fill.

**rkey**
- Remote memory access key for the RDMA operation.

**data_addr**
- Address of the data for the RDMA operation.

### Returns

flexio_dev_status_t.

### Description

Fill the fields of a send WQE segment (4 DWORDs) with RDMA segment information.

# 2.5. DevQueueTypes

Flex IO SDK device queue types for DPA programs. Defines basic networking elements structure.

## struct flexio_dev_cqe64

## struct flexio_dev_eqe

## union flexio_dev_sqe_seg

## struct flexio_dev_wqe_atomic_seg

## struct flexio_dev_wqe_ctrl_seg

## struct flexio_dev_wqe_eth_seg

## struct flexio_dev_wqe_inline_data_seg

## struct flexio_dev_wqe_inline_send_data_seg

## struct flexio_dev_wqe_mem_ptr_send_data_seg

## struct flexio_dev_wqe_rcv_data_seg

## struct flexio_dev_wqe_rdma_seg

## struct flexio_dev_wqe_transpose_seg

## enum flexio_dev_wqe_eth_seg_cs_swp_flags_t

Flex IO dev ethernet segment bitmask for CS / SWP flags

### Values

**FLEXIO_ETH_SEG_L4CS = 0x8000**
**FLEXIO_ETH_SEG_L3CS = 0x4000**
**FLEXIO_ETH_SEG_L4CS_INNER = 0x2000**
**FLEXIO_ETH_SEG_L3CS_INNER = 0x1000**
**FLEXIO_ETH_SEG_TRAILER_ALIGN = 0x0200**
**FLEXIO_ETH_SEG_SWP_OUTER_L4_TYPE = 0x0040**
**FLEXIO_ETH_SEG_SWP_OUTER_L3_TYPE = 0x0020**

**FLEXIO_ETH_SEG_SWP_INNER_L4_TYPE = 0x0002**
**FLEXIO_ETH_SEG_SWP_INNER_L3_TYPE = 0x0001**

## struct flexio_dev_eqe ::packed

Describes Flex IO dev EQE.

Describes Flex IO dev CQE.

Describes Flex IO dev WQE memory pointer send data segment.

Describes Flex IO dev WQE inline send data segment.

Describes Flex IO dev WQE receive data segment.

Describes Flex IO dev WQE control segment.

Describes Flex IO dev WQE ethernet segment.

Describes Flex IO dev WQE inline data segment.

Describes Flex IO dev WQE RDMA segment.

Describes Flex IO dev WQE ATOMIC segment.

Describes Flex IO dev WQE transpose segment.

## #define LOG_SQE_NUM_SEGS 2

SQ depth (log_sq_depth) is measured in WQEBBs, each one is 64B. We have to understand difference between wqe_idx and seg_idx. For example wqe with index 5 built from 4 segments with indexes 20, 21, 22 and 23.

# Chapter 3. Data Structures

Here are the data structures with brief descriptions:



## 3.1. flexio_affinity Struct Reference

Describes Flex IO thread affinity information.

### uint32_t flexio_affinity::id

ID of the chosen resource (HART / DPA HART group). Reserved if affinity type none is set.

### enumflexio_affinity_type flexio_affinity::type

Affinity type to use for a Flex IO thread (none, strict or group).

## 3.2. flexio_app_attr Struct Reference

Describes process attributes for creating a Flex IO application.

### size_t flexio_app_attr::app_bsize

DPA application size (bytes).

### const char *flexio_app_attr::app_name

DPA application name.

### void *flexio_app_attr::app_ptr

Pointer in the ELF file for the DPA application.

### char *flexio_app_attr::app_sig_sec_name

Application signature section name.

## 3.3. flexio_cmdq_attr Struct Reference

Describes process attributes for creating a Flex IO command queue (async RPC).

### int flexio_cmdq_attr::batch_size

Number of tasks to be executed to completion by invoked thread.

### enumflexio_cmdq_state flexio_cmdq_attr::state

Command queue initial state.

### int flexio_cmdq_attr::workers

Number of available workers, each worker can handle up to batch_size number of tasks in a single invocation.

## 3.4. flexio_cq_attr Struct Reference

Describes attributes for creating a Flex IO CQ.

### uint8_t flexio_cq_attr::always_armed

Indication to always arm for the created CQ

### bool flexio_cq_attr::cc

Indication to enable collapsed CQE for the created CQ.

### flexio_uintptr_t flexio_cq_attr::cq_dbr_daddr

DBR memory address for the created CQ.

### uint16_t flexio_cq_attr::cq_max_count

CQE compression max count (number of CQEs before creating an event).

### uint16_t flexio_cq_attr::cq_period

CQE compression period (number of usecs before creating an event).

### flexio_cq_period_mode_t flexio_cq_attr::cq_period_mode

CQE compression period mode (by CQE or by event).

### struct flexio_qmem flexio_cq_attr::cq_ring_qmem

Ring memory info for the created CQ.

### flexio_cqe_comp_type flexio_cq_attr::cqe_comp_type

CQE compression type to use for the CQ (none, basic or enhanced).

### uint8_t flexio_cq_attr::element_type

Type of the element attached to the created CQ (thread, EQ, none, emulated EQ).

### uint32_t flexio_cq_attr::emulated_eqn

Emulated EQ number to attach to the created CQ

### uint8_t flexio_cq_attr::log_cq_depth

Log number of entries for the created CQ.

### bool flexio_cq_attr::no_arm

Indication to not arm the CQ on creation.

### uint8_t flexio_cq_attr::overrun_ignore

Indication to ignore overrun for the created CQ.

### flexio_thread *flexio_cq_attr::thread

Thread object to attach to the created CQ (only valid for element type thread).

### void *flexio_cq_attr::uar_base_addr

CQ UAR base address.

### uint32_t flexio_cq_attr::uar_id

CQ UAR ID.

## 3.5. flexio_dev_cqe64 Struct Reference

Describes Flex IO dev CQE.

## __be32 flexio_dev_cqe64::byte_cnt

0Bh - Byte count.

## uint8_t flexio_dev_cqe64::csum_ok

07h 24..26 - checksum ok bits.

## uint8_t flexio_dev_cqe64::op_own

0Fh 0 - Ownership bit.

## __be32 flexio_dev_cqe64::qpn

0Eh - QPN.

## __be32 flexio_dev_cqe64::rsvd0

00h..06h - Reserved.

## uint8_t flexio_dev_cqe64::rsvd29

07h 0..23 - Reserved.

## __be32 flexio_dev_cqe64::rsvd36

09h..0Ah - Reserved.

## __be32 flexio_dev_cqe64::rsvd48

0Ch..0Dh - Reserved.

## uint8_t flexio_dev_cqe64::signature

0Fh 8..15 - Signature.

## __be32 flexio_dev_cqe64::srqn_uidx

08h - SRQ number or user index.

## __be16 flexio_dev_cqe64::wqe_counter

0Fh 16..31 - WQE counter.

# 3.6. flexio_dev_eqe Struct Reference

Describes Flex IO dev EQE.

## __be32 flexio_dev_eqe::cqn

18h 24 lsb - CQN.

## flexio_dev_eqe::@8 flexio_dev_eqe::event_data

20h - Event data.

## uint8_t flexio_dev_eqe::owner

3Fh - Owner.

## __be32 flexio_dev_eqe::rsvd00

00h..17h - Reserved.

## uint8_t flexio_dev_eqe::rsvd00

00h - Reserved.

## uint8_t flexio_dev_eqe::rsvd02

02h - Reserved.

## __be16 flexio_dev_eqe::rsvd3c

3Ch - Reserved.

## uint8_t flexio_dev_eqe::rsvd4

04h..1fh - Reserved.

## uint8_t flexio_dev_eqe::signature

3Eh - Signature.

### uint8_t flexio_dev_eqe::sub_type

03h - Sub type.

### uint8_t flexio_dev_eqe::type

01h - EQE type.

## 3.7. flexio_dev_sqe_seg Union Reference

Describes Flex IO dev send WQE segments. Only one segment can be set at a given time.

### struct flexio_dev_wqe_atomic_seg flexio_dev_sqe_seg::atomic

Atomic segment.

### struct flexio_dev_wqe_ctrl_seg flexio_dev_sqe_seg::ctrl

Control segment.

### struct flexio_dev_wqe_eth_seg flexio_dev_sqe_seg::eth

Ethernet segment.

### struct flexio_dev_wqe_inline_data_seg flexio_dev_sqe_seg::inline_data

Inline data segment.

### struct flexio_dev_wqe_inline_send_data_seg flexio_dev_sqe_seg::inline_send_data

Inline send data segment.

### struct flexio_dev_wqe_mem_ptr_send_data_seg flexio_dev_sqe_seg::mem_ptr_send_data

Memory pointer send data segment.

### struct flexio_dev_wqe_rdma_seg flexio_dev_sqe_seg::rdma

RDMA segment.

### struct flexio_dev_wqe_transpose_seg flexio_dev_sqe_seg::transpose

Transpose segment.

## 3.8. flexio_dev_wqe_atomic_seg Struct Reference

Describes Flex IO dev WQE ATOMIC segment.

### __be64 flexio_dev_wqe_atomic_seg::compare_data

02h..03h - Compare operation data.

### __be64 flexio_dev_wqe_atomic_seg::swap_or_add_data

00h..01h - Swap or Add operation data.

## 3.9. flexio_dev_wqe_ctrl_seg Struct Reference

Describes Flex IO dev WQE control segment.

### __be32 flexio_dev_wqe_ctrl_seg::general_id

03h - Control general ID.

### __be32 flexio_dev_wqe_ctrl_seg::idx_opcode

00h - WQE index and opcode.

### __be32 flexio_dev_wqe_ctrl_seg::qpn_ds

01h - QPN and number of data segments.

### __be32 flexio_dev_wqe_ctrl_seg::signature_fm_ce_se

02h - Signature, fence mode, completion mode and solicited event.

## 3.10. flexio_dev_wqe_eth_seg Struct Reference

Describes Flex IO dev WQE ethernet segment.

### __be16 flexio_dev_wqe_eth_seg::cs_swp_flags

01h 16..31 - CS and SWP flags.

### __be16 flexio_dev_wqe_eth_seg::inline_hdr_bsz

03h 16..31 - Inline headers size (bytes).

### uint8_t flexio_dev_wqe_eth_seg::inline_hdrs

03h 0..15 - Inline headers (first two bytes).

### __be16 flexio_dev_wqe_eth_seg::mss

01h 0..15 - Max segment size.

### __be32 flexio_dev_wqe_eth_seg::rsvd0

00h - Reserved.

### __be32 flexio_dev_wqe_eth_seg::rsvd2

02h - Reserved.

## 3.11. flexio_dev_wqe_inline_data_seg Struct Reference

Describes Flex IO dev WQE inline data segment.

### uint8_t flexio_dev_wqe_inline_data_seg::inline_data

00h..03h - Inline data.

## 3.12. flexio_dev_wqe_inline_send_data_seg Struct Reference

Describes Flex IO dev WQE inline send data segment.

### __be32 flexio_dev_wqe_inline_send_data_seg::byte_count

00h - Byte count.

### __be32 flexio_dev_wqe_inline_send_data_seg::data_and_padding

01h..03h - Data and padding array.

## 3.13. flexio_dev_wqe_mem_ptr_send_data_seg Struct Reference

Describes Flex IO dev WQE memory pointer send data segment.

### __be64 flexio_dev_wqe_mem_ptr_send_data_seg::addr

02h..03h - Address.

### __be32 flexio_dev_wqe_mem_ptr_send_data_seg::byte_count

00h - Byte count.

### __be32 flexio_dev_wqe_mem_ptr_send_data_seg::lkey

01h - Local key.

## 3.14. flexio_dev_wqe_rcv_data_seg Struct Reference

Describes Flex IO dev WQE receive data segment.

### __be64 flexio_dev_wqe_rcv_data_seg::addr

02h..03h - Address.

### __be32 flexio_dev_wqe_rcv_data_seg::byte_count

00h - Byte count.

### __be32 flexio_dev_wqe_rcv_data_seg::lkey

01h - Local key.

## 3.15. flexio_dev_wqe_rdma_seg Struct Reference

Describes Flex IO dev WQE RDMA segment.

### __be64 flexio_dev_wqe_rdma_seg::raddr

00h..01h - Remote address.

### __be32 flexio_dev_wqe_rdma_seg::rkey

02h - Remote key.

### __be32 flexio_dev_wqe_rdma_seg::rsvd0

03h - Reserved.

## 3.16. flexio_dev_wqe_transpose_seg Struct Reference

Describes Flex IO dev WQE transpose segment.

### uint8_t flexio_dev_wqe_transpose_seg::element_size

00h 0..7 - Matrix element size.

### uint8_t flexio_dev_wqe_transpose_seg::num_of_cols

01h 16..22 - Number of columns in matrix (7b).

### uint8_t flexio_dev_wqe_transpose_seg::num_of_rows

01h 0..6 - Number of rows in matrix (7b).

### uint8_t flexio_dev_wqe_transpose_seg::rsvd0

00h 8..31 - Reserved.

### uint8_t flexio_dev_wqe_transpose_seg::rsvd1

01h - Reserved.

### uint8_t flexio_dev_wqe_transpose_seg::rsvd2

01h - Reserved.

### uint8_t flexio_dev_wqe_transpose_seg::rsvd4

02h..03h - Reserved.

## 3.17. flexio_event_handler_attr Struct Reference

Describes attributes for creating a Flex IO event handler.

### struct flexio_affinity flexio_event_handler_attr::affinity

Thread's affinity information.

### uint64_t flexio_event_handler_attr::arg

Thread argument.

### int flexio_event_handler_attr::continuable

Thread continuable flag.

### flexio_func_t *flexio_event_handler_attr::host_stub_func

Stub for the entry function of the thread.

### flexio_uintptr_t flexio_event_handler_attr::thread_local_storage_daddr

Address of the local storage buffer of the thread.

## 3.18. flexio_log_dev_attr Struct Reference

Describes DPA log thread attributes for logging from the Device to the Host side.

### size_t flexio_log_dev_attr::data_bsize

Size of buffer, used for data transfer from Flex IO to HOST MUST be power of two and be at least 2Kb.

### enumflexio_log_dev_sync_mode flexio_log_dev_attr::sync_mode

Select between sync/async modes.

### flexio_uar *flexio_log_dev_attr::uar

A pointer to a Flex IO UAR object created by the caller for the device side.

## 3.19. flexio_mkey_attr Struct Reference

Describes process attributes for creating a Flex IO MKey.

### int flexio_mkey_attr::access

access contains the access mask for the MKey (Expected values: IBV_ACCESS_REMOTE_WRITE, IBV_ACCESS_LOCAL_WRITE).

### flexio_uintptr_t flexio_mkey_attr::daddr

DPA address the MKey is created for.

### size_t flexio_mkey_attr::len

Length of the address space the MKey is created for.

### ibv_pd *flexio_mkey_attr::pd

IBV protection domain information for the created MKey.

## 3.20. flexio_process_attr Struct Reference

Describes attributes for creating a Flex IO process.

### ibv_pd *flexio_process_attr::pd

IBV protection domain information for the created process. Passing NULL will result in an internal PD being created and used for the process.

## 3.21. flexio_qmem Struct Reference

Describes queue memory, which may be either host memory or DPA memory

### flexio_uintptr_t flexio_qmem::daddr

DPA address of the queue memory (only valid for memtype FLEXIO_MEMTYPE_DPA).

### uint64_t flexio_qmem::humem_offset

Address offset in the umem of the queue memory (only valid for memtype FLEXIO_MEMTYPE_HOST).

### enumflexio_memtype flexio_qmem::memtype

Type of memory to use (FLEXIO_MEMTYPE_DPA or FLEXIO_MEMTYPE_HOST).

### uint32_t flexio_qmem::umem_id

UMEM ID of the queue memory.

## 3.22. flexio_qp_attr Struct Reference

Describes attributes for creating a Flex IO QP.

### uint8_t *flexio_qp_attr::dest_mac

Destination MAC address to set for the modified QP

### uint8_t flexio_qp_attr::fl

Indication to enable force loopback for the modified QP.

### uint8_t flexio_qp_attr::gid_table_index

GID table index to set for the modified QP

## uint8_t flexio_qp_attr::grh

GRH to set for the modified QP.

## int flexio_qp_attr::log_rq_depth

Log number of entries of the QP's RQ.

## uint8_t flexio_qp_attr::log_rra_max

Log of the number of allowed outstanding RDMA read/atomic operations

## int flexio_qp_attr::log_sq_depth

Log number of entries of the QP's SQ.

## uint8_t flexio_qp_attr::log_sra_max

Log of the number of allowed outstanding RDMA read/atomic operations as requester

## uint32_t flexio_qp_attr::min_rnr_nak_timer

Minimal RNR NACK timer to set for the modified QP.

## uint32_t flexio_qp_attr::next_rcv_psn

Next receive PSN to set for the modified QP.

## uint32_t flexio_qp_attr::next_send_psn

Next send PSN to set for the modified QP.

## flexio_qp_state flexio_qp_attr::next_state

QP state to move the QP to (reset, init, RTS, RTR).

## int flexio_qp_attr::no_sq

Indication to create the QP without an SQ.

## int flexio_qp_attr::ops_flag

Bitmask of enum flexio_qp_op_types, used to calculate WQE sizes.

## enumflexio_qp_qpc_mtu flexio_qp_attr::path_mtu

Path MTU to set for the modified QP.

## ibv_pd *flexio_qp_attr::pd

IBV protection domain information for the created QP.

## int flexio_qp_attr::qp_access_mask

QP's access permission (Expected values: IBV_ACCESS_REMOTE_WRITE, IBV_ACCESS_REMOTE_READ, IBV_ACCESS_REMOTE_ATOMIC, IBV_ACCESS_LOCAL_WRITE).

## struct flexio_qmem flexio_qp_attr::qp_wq_buff_qmem

Ring memory info for the created QP's WQ.

## struct flexio_qmem flexio_qp_attr::qp_wq_dbr_qmem

DBR memory info for the created QP's WQ.

## uint32_t flexio_qp_attr::remote_qp_num

Remote QP number to set for the modified QP.

## uint8_t flexio_qp_attr::retry_count

Retry count to set for the modified QP.

## ibv_gid flexio_qp_attr::rgid_or_rip

Remote GID or remote IP to set for the modified QP.

## uint16_t flexio_qp_attr::rlid

Remote LID to set for the modified QP.

### uint32_t flexio_qp_attr::rq_cqn

CQ number of the QP's RQ.

### int flexio_qp_attr::rq_type

QP's RQ type (regular, SRQ, zero-RQ)

### uint32_t flexio_qp_attr::sq_cqn

CQ number of the QP's SQ.

### uint32_t flexio_qp_attr::transport_type

QP's transport type (currently only FLEXIO_QPC_ST_RC is supported).

### uint32_t flexio_qp_attr::uar_id

QP UAR ID.

### uint16_t flexio_qp_attr::udp_sport

UDP port to set for the modified QP.

### uint32_t flexio_qp_attr::user_index

User defined user_index for the created QP.

### uint8_t flexio_qp_attr::vhca_port_num

VHCA port number to set for the modified QP.

## 3.23. flexio_qp_attr_opt_param_mask Struct Reference

Describes QP modify operation mask.

### bool flexio_qp_attr_opt_param_mask::min_rnr_nak_timer

Indication to modify the QP's min_rnr_nak_timer field.

### bool flexio_qp_attr_opt_param_mask::qp_access_mask

Indication to modify the QP's qp_access_mask field.

## 3.24. flexio_wq_attr Struct Reference

Describes attributes for creating a Flex IO WQ.

### uint8_t flexio_wq_attr::log_wq_depth

Log number of entries for the created WQ.

### ibv_pd *flexio_wq_attr::pd

IBV protection domain struct to use for creating the WQ.

### struct flexio_wq_sq_attr flexio_wq_attr::sq

SQ attributes (used only for SQs).

### uint32_t flexio_wq_attr::uar_id

WQ UAR ID.

### uint32_t flexio_wq_attr::user_index

User defined user_index for the created WQ.

### struct flexio_qmem flexio_wq_attr::wq_dbr_qmem

DBR memory address for the created WQ.

### struct flexio_qmem flexio_wq_attr::wq_ring_qmem

Ring memory info for the created WQ.

## 3.25. flexio_wq_sq_attr Struct Reference

Describes attributes for creating a Flex IO SQ.

### uint8_t flexio_wq_sq_attr::allow_multi_pkt_send_wqe

Indication enable multi packet send WQE for the created SQ.

## 3.26. spinlock_s Struct Reference

Describes Flex IO dev spinlock.

### uint32_t spinlock_s::locked

Indication for spinlock lock state.

# Chapter 4. Data Fields

Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

## A

**access**
flexio_mkey_attr
**addr**
flexio_dev_wqe_mem_ptr_send_data_seg
flexio_dev_wqe_rcv_data_seg
**affinity**
flexio_event_handler_attr
**allow_multi_pkt_send_wqe**
flexio_wq_sq_attr
**always_armed**
flexio_cq_attr
**app_bsize**
flexio_app_attr
**app_name**
flexio_app_attr
**app_ptr**
flexio_app_attr
**app_sig_sec_name**
flexio_app_attr
**arg**
flexio_event_handler_attr
**atomic**
flexio_dev_sqe_seg

## B

**batch_size**
flexio_cmdq_attr
**byte_cnt**
flexio_dev_cqe64

## C



## D

## E

**element_size**
flexio_dev_wqe_transpose_seg
**element_type**
flexio_cq_attr
**emulated_eqn**
flexio_cq_attr
**eth**
flexio_dev_sqe_seg
**event_data**
flexio_dev_eqe

## F

**fl**
flexio_qp_attr

## G

**general_id**
flexio_dev_wqe_ctrl_seg
**gid_table_index**
flexio_qp_attr
**grh**
flexio_qp_attr

## H

**host_stub_func**
flexio_event_handler_attr
**humem_offset**
flexio_qmem

## I

**id**
flexio_affinity
**idx_opcode**
flexio_dev_wqe_ctrl_seg
**inline_data**
flexio_dev_sqe_seg
flexio_dev_wqe_inline_data_seg
**inline_hdr_bsz**
flexio_dev_wqe_eth_seg
**inline_hdrs**
flexio_dev_wqe_eth_seg

## O



## P



## Q

## R

## V



## W

## Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation nor any of its direct or indirect subsidiaries and affiliates (collectively: “NVIDIA”) make no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assume no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer (“Terms of Sale”). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer’s own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer’s sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer’s product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, “MATERIALS”) ARE BEING PROVIDED “AS IS.” NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA’s aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

## Trademarks

NVIDIA, the NVIDIA logo, and Mellanox are trademarks and/or registered trademarks of Mellanox Technologies Ltd. and/or NVIDIA Corporation in the U.S. and in other countries. The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a world¬wide basis. Other company and product names may be trademarks of the respective companies with which they are associated.

## Copyright



NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
http://www.nvidia.com