
Application Note

# Mellanox IB-Verbs API (VAPI)

## Mellanox Software Programmer's

## Interface for InfiniBand Verbs

Mellanox IB-Verbs API (VAPI)

Document Number: 2088AN

Mellanox Technologies, Inc.
2900 Stender Way
Santa Clara, CA 95054
U.S.A.
www.Mellanox.com

Tel: (408) 970-3400
Fax: (408) 970-3403

Mellanox Technologies Ltd.
POB 586 Yokneam 20692
Israel

Tel: +972-4-909-7200
Fax: +972-4959-3245

# *Contents*

# List of Tables

# 1 Introduction

This document provides a C-Language definition of the VERBS abstraction defined in the IBTA specification Volume 1, chapter 11.

This interface will be supported by all Mellanox HCA devices and is designed to easily absorb HW changes and support different OS without exposing those changes to the application running above the VERBs.

**Table 1 VAPI Calls List**

| *VAPI CALL* | *Description* | *Status* | *Page* |
|---|---|---|---|
| **VAPI_open_hca** | **Open HCA** | Supported | p.7 |
| **EVAPI_get_hca_hndl** | **Get HCA Handle** | Supported | p.9 |
| **VAPI_query_hca_cap** | **Query HCA Capabilities** | Supported | p.10 |
| **VAPI_query_hca_port_prop** | **Query HCA Port Properties** | Supported | p.13 |
| **VAPI_query_hca_gid_tbl** | **Query HCA Gid Tbl** | Supported | p.15 |
| **VAPI_query_hca_pkey_tbl** | **Query HCA Pkey Tbl** | Supported | p.16 |
| **VAPI_modify_hca_attr** | **Modify HCA Attributes** | Supported | p.17 |
| **VAPI_close_hca** | **Close HCA** | Supported | p.20 |
| **VAPI_alloc_pd** | **Allocate Protection Domain** | Supported | p.21 |
| **VAPI_dealloc_pd** | **Deallocate Protection Domain** | Supported | p.22 |
| **VAPI_alloc_rdd** | **Allocate Reliable Datagram Domain** | None | p.23 |
| **VAPI_dealloc_rdd** | **Deallocate Reliable Datagram Domain** | None | p.24 |
| **VAPI_create_addr_hndl** | **Create Address Handle** | Supported | p.25 |
| **VAPI_modify_addr_hndl** | **Modify Address Handle** | Supported | p.27 |
| **VAPI_query_addr_hndl** | **Query Address Handle** | Supported | p.28 |
| **VAPI_destroy_addr_hndl** | **Destroy Address Handle** | Supported | p.29 |
| **VAPI_create_qp** | **Create Queue Pair** | Supported | p.31 |
| **VAPI_modify_qp** | **Modify Queue Pair** | Supported | p.34 |
| **VAPI_query_qp** | **Query Queue Pair** | Supported | p.37 |
| **VAPI_destroy_qp** | **Destroy Queue Pair** | Supported | p.38 |
| **VAPI_get_special_qp** | **Get Special QP** | Supported | p.39 |
| **VAPI_create_cq** | **Create Completion Queue** | Supported | p.41 |
| **VAPI_query_cq** | **Query Completion Queue** | Supported | p.43 |

**Table 1 VAPI Calls List (Continued)**

| *VAPI CALL* | *Description* | *Status* | *Page* |
|---|---|---|---|
| VAPI_resize_cq | Resize Completion Queue | None | p.44 |
| VAPI_destroy_cq | Destroy Completion Queue | Supported | p.45 |
| VAPI_create_eec | Create EE Context | None | p.47 |
| VAPI_modify_eec_attr | Modify EE Context Attributes | None | p.48 |
| VAPI_query_eec_attr | Query EE Context | None | p.49 |
| VAPI_destroy_eec | Destroy EE Context | None | p.50 |
| VAPI_register_mr | Register Memory Region | Supported | p.51 |
| VAPI_query_mr | Query Memory Region | Supported | p.53 |
| VAPI_deregister_mr | Deregister Memory Region | Supported | p.54 |
| VAPI_reregister_mr | Reregister Memory Region | None | p.55 |
| VAPI_register_smr | Register Shared Memory Region | None | p.56 |
| VAPI_alloc_mw | Allocate Memory Window | None | p.57 |
| VAPI_query_mw | Query Memory Window | None | p.58 |
| VAPI_bind_mw | Bind Memory Window | None | p.59 |
| VAPI_dealloc_mw | Deallocate Memory Window | None | p.60 |
| VAPI_attach_to_multicast | Attach QP to Multicast Group | Supported | p.61 |
| VAPI_detach_from_multicast | Detach QP from Multicast Group | Supported | p.62 |
| VAPI_post_sr | Post Send Request | Supported | p.63 |
| VAPI_post_rr | Post Receive Request | Supported | p.66 |
| VAPI_poll_cq | Poll for Completion | Supported | p.67 |
| VAPI_req_comp_notif | Request Completion Notification | Supported | p.69 |
| EVAPI_set_comp_eventh | Request Completion Notification for a Completion Queue | Supported | p.70 |
| EVAPI_clear_comp_eventh | Clear Completion Notification for a Completion Queue | Supported | p.71 |
| VAPI_set_comp_event_handler | Set Completion Event Handler | Supported | p.73 |
| VAPI_set_async_event_handler | VAPI_query_hca_cap | Supported | p.74 |

# 2 HCA Verbs

This chapter describes the Mellanox implementation of the following verbs handling basic HCA functionality:



## 2.1 Open HCA

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_open_hca
(
        IN      VAPI_hca_id_t       hca_id,
        OUT     VAPI_hca_hndl_t     *hca_hndl_p

)
```

**ARGUMENTS:** **hca_id:** HCA identifier.
**hca_hndl_p:** Pointer to the HCA object handle.

**RETURNS:** **VAPI_OK**
**VAPI_EAGAIN:** Insufficient resources**.**
**VAPI_EINVAL_HCA_ID**: Invalid HCA identifier.
**VAPI_EBUSY**: HCA already in use.

**DESCRIPTION:**

Creates a new HCA Object.
The newly created object is assigned to the device described by hca_id_num.

## 2.2 Get HCA Handle

**SYNOPSIS:**

```
VAPI_ret_t
EVAPI_get_hca_hndl
(
        IN      VAPI_hca_id_t       hca_id,
        OUT     VAPI_hca_hndl_t     *hca_hndl_p

)
```

**ARGUMENTS:** **hca_id:** HCA identifier.
**hca_hndl_p:** Pointer to the HCA object handle.

**RETURNS:** **VAPI_OK**
**VIP_EINVAL_HCA_HNDL :** No such opened HCA.

**DESCRIPTION:**

Gets the handle of an already opened HCA. All applications should use this function to get the HCA handle to work with.

## 2.3 Query HCA Capabilities

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_query_hca_cap
(
        IN      VAPI_hca_hndl_t         hca_hndl,
        OUT     VAPI_hca_vendor_t       *hca_vendor_p
        OUT     VAPI_hca_cap_t          *hca_cap_p

)
```

**ARGUMENTS:**

**hca_hndl**: HCA object handle.
**hca_vendor_p**: Pointer to HCA vendor-specific information object**.**
**hca_cap_p**: Pointer to HCA capabilities object

**RETURNS:**

**VAPI_OK**
**VAPI_EINVAL_HCA_HNDL:** Invalid HCA handle.
**VAPI_EAGAIN:** Insufficient resources.
**VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**

Query HCA capabilities retrieves:

- A structure of type **VAPI_hca_vendor_t**, providing a list of the vendor-specific information about the HCA,
- A structure of type **VAPI_hca_cap_t**, providing a detailed list of the HCA capabilities.

Further information on the hca ports can be retrieved using the verbs **VAPI_query_hca_port_prop**, **VAPI_query_hca_gid_tbl** and **VAPI_query_hca_pkey_tbl**.

The following table lists the vendor-specific parameters, which are described in the structure **VAPI_hca_vendor_t**:

**Table 2 VAPI_hca_vendor_t**

| | |
|---|---|
| vendor_id | Vendor ID |
| vendor_part_id | Vendor supplied part ID |
| hw_ver | Hardware Version |

The **hca_cap** structure describes the HCA capabilities and is of type **VAPI_hca_cap_t**, as listed in the following table:

**Table 3 VAPI_hca_cap_t**

| | |
|---|---|
| max_num_qp | Maximum number of QP supported. |
| max_qp_ous_wr | Maximum number of outstanding work requests per QP supported<br>does not appear in VIP. |
| flags | Different capabilities:<br>- VAPI_RESIZE_OUS_WQE_CAP - ability to resize maximum outstanding WR per QP.<br>- VAPI_BAD_PKEY_COUNT_CAP - bad p_key counter support indication.<br>- VAPI_BAD_QKEY_COUNT_CAP - q_key violation counter support indication.<br>- VAPI_RAW_MULTI_CAP - supports raw packets multicast.<br>- VAPI_AUTO_PATH_MIG_CAP - supports automatic path migration<br>- VAPI_CHANGE_PHY_PORT_CAP - Ability to change the primary physical port when transitioning from SQD to RTS state.<br>- VAPI_UD_AV_PORT_ENFORCE_CAP - Port check enforced for UD address vectors. |
| max_num_sg_ent | Maximum number of scatter gather entries for descriptors other than reliable datagram. |
| max_num_sg_ent_rd | Maximum number of scatter gather entries for descriptors reliable datagram. |
| max_num_cq | Maximum number of completion queues supported by this HCA. |
| max_num_ent_cq | Maximum number of completion queue entries in each CQ. |
| max_num_mr | Maximum number of memory regions supported. |
| max_mr_size | Maximum size of contiguous memory region. |
| max_pd_num | Maximum number of protection domains supported. |
| page_size_cap | Memory maximum page size. |
| phys_port_num | Number of physical ports. |
| max_pkeys | Number of partitions supported. |
| node_guid | Node GUID for this HCA. |
| local_ca_ack_delay | Max expected time between message receive and ACK or NAK transmission. |
| max_qp_ous_rd_atom | Maximum number of read or atomic operation requests that can be outstanding on a Queue Pair. |
| max_ee_ous_rd_atom | Maximum number of read or atomic operation requests that can be outstanding on an EE. |
| max_res_rd_atom | Maximum number of resources that must be allocated to incoming RDMA Read or Atomic. |
| max_qp_init_rd_atom | Maximum number of RDMA Reads or Atomic operations that can be initiated on a specific QP. |
| max_ee_init_rd_atom | Maximum number of RDMA Reads or Atomic operations that can be initiated on a specific QP (hard coded to 1 ***on IB v1.0***) |

**Table 3 VAPI_hca_cap_t (Continued)**

| | |
|---|---|
| atomic_cap | Level of atomicity supported:<br>- VAPI_NOT_SUPPORTED - atomic operations not supported.<br>- VAPI_LOCAL_QP - atomicity guaranteed only between QPs on this HCA.<br>- VAPI_ALL - atomicity is guaranteed between this HCA and any other device. |
| max_ee_num | Maximum number of EE contexts supported. |
| max_rdd_num | Maximum number of RDDs supported. |
| max_mw_num | Maximum Number of Memory Windows supported. |
| max_raw_ipv6_qp | Maximum number of raw IPV6 QPs supported. |
| max_raw_ethy_qp | Maximum number of raw Ethertype packets supported by this QP. |
| max_mcast_grp_num | Maximum number of multicast groups supported. |
| max_mcast_qp_attach_num | Maximum number of QPs that can be attached to a multicast group. |
| max_total_mcast_qp_attach_num | Maximum number of QPs that can be attached to any multicast group. |
| max_ah_num | Maximum number of address handles. |

## 2.4 Query HCA Port Properties

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_query_hca_port_prop
(
        IN      VAPI_hca_hndl_t     hca_hndl,
        IN      IB_port_t           port_num,
        OUT     VAPI_hca_port_t     *hca_port_p

)
```

**ARGUMENTS:** **hca_hndl**: HCA object handle.
**port_num**: Port number
**hca_port_p**: HCA port object describing the port properties**.**

**RETURNS:** **VAPI_OK**
**VAPI_EINVAL_HCA_HNDL:** Invalid HCA handle.
**VAPI_EAGAIN:** Insufficient resources.
**VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**

Query HCA port properties retrieves a structure of type **VAPI_hca_port_t** for the port specified in **port_num**. The number of the HCA physical ports can be obtained using the verb **VAPI_query_hca_cap**. Further information about the port p-key table and gid table can be obtained using the verbs **VAPI_query_hca_gid_tbl** and **VAPI_query_hca_pkey_tbl**.

Upon successful completion, the verb returns a **hca_port_p** structure of type **VAPI_hca_port_t**, which is described in the following table:

**Table 4 VAPI_hca_port_t**

| | |
|---|---|
| max_mtu | Max MTU supported on this port. |
| max_msg_sz | Max Message size supported on this port. |
| lid | Base LID |
| lmc | LMC |
| state | Port State |
| capability_mask | Various bits as defined in PortInfo CapabilityMask in IB spec. (IB_capability_mask_bits_t) |
| max_vl_num | Maximum number of VL supported by this port. |
| bad_pkey_counter | Bad PKey counter (if supported). |

**Table 4 VAPI_hca_port_t (Continued)**

| | |
|---|---|
| qkey_viol_counter | QKey violation counter. |
| sm_lid | Subnet Manager LID on this port. |
| sm_sl | Subnet Manager SL on this port. |
| pkey_tbl_len | P-Key table length. |
| gid_tbl_len | GID table length. |

## 2.5 Query HCA Gid Tbl

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_query_hca_gid_tbl
(
        IN      VAPI_hca_hndl_t     hca_hndl,
        IN      IB_port_t           port_num,
        IN      u_int16_t           tbl_len_in,
        OUT     u_int16_t           *tbl_len_out,
        OUT     IB_gid_t            *gid_tbl_p

)
```

**ARGUMENTS:**

**hca_hndl**: HCA object handle.
**port_num**: Port number.
**tbl_len_in**: Number of entries in given gid_tbl_p buffer.
**tbl_len_out**: Actual number of entries in this port GID table.
**gid_tbl_p**: The GID table buffer to return result in.

**RETURNS:**

**VAPI_OK**
**VAPI_EINVAL_HCA_HNDL:** Invalid HCA handle.
**VAPI_EAGAIN:** tbl_len_out > tbl_len_in.
**VAPI_EINVAL_PARAM:** Invalid port number.

**DESCRIPTION:**

The GID table of the given port is returned in gid_tbl_p.
If tbl_len_out (actual number of entries) is more than tbl_len_in, the function should be recalled with a larger buffer.

## 2.6 Query HCA Pkey Tbl

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_query_hca_pkey_tbl
(
        IN      VAPI_hca_hndl_t     hca_hndl,
        IN      IB_port_t           port_num,
        IN      u_int16_t           tbl_leb_in,
        OUT     u_int16_t           *tbl_leb_out,
        OUT     VAPI_pkey_t         *pkey_tbl_p

)
```

**ARGUMENTS:**

**hca_hndl**: HCA object handle.
**port_num**: Port number.
**tbl_len_in**: Number of entries in given gid_tbl_p buffer.
**tbl_len_out**: Actual number of entries in this port GID table.
**pkey_tbl_p**: The PKEY table buffer to return result in.

**RETURNS:**

**VAPI_OK**
**VAPI_EINVAL_HCA_HNDL:** Invalid HCA handle.
**VAPI_EAGAIN:** tbl_len_out > tbl_len_in.
**VAPI_EINVAL_PARAM:** .Invalid port number.

**DESCRIPTION:**

## 2.7 Modify HCA Attributes

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_modify_hca_attr
(
        IN      VAPI_hca_hndl_t         hca_hndl,
        IN      IB_port_t               port_num,
        IN      VAPI_hca_attr_t         *hca_attr_p,
        IN      VAPI_hca_attr_mask_t    *hca_attr_mask_p

)
```

**ARGUMENTS:** **hca_hndl**: Handle to HCA.
**port_num**: Port number.
**hca_attr_p**: Pointer to the HCA attributes structure.
**hca_attr_mask_p**: Pointer to the HCA attributes mask

**ARGUMENTS:** **VAPI_OK**
**VAPI_EINVAL_HCA_HNDL:** Invalid HCA handle.
**VAPI_EINVAL_COUNTER:** Invalid counter.
**VAPI_EINVAL_COUNT_VAL:** Invalid counter value.
**VAPI_EPERM:** Insufficient permissions to do this operation.

**DESCRIPTION:**

Sets the HCA attributes specified in **hca_attr_p** to port number **port_num**. Only the values specified in **hca_attr_mask_p** are modified. **hca_attr_p** is a pointer to a structure of type **VAPI_hca_attr_t,** which is specified in the following table:

**Table 5 VAPI_hca_attr_t**

| | |
|---|---|
| is_sm | Is Subnet manager |
| is_snmp_tun_sup | Is SMNP tunneling supported |
| is_dev_mgt_sup | Is device management supported |
| is_vendor_cls_sup | Is Vendor Class supported |

The attributes to be changed are specified using a mask of type **VAPI_hca_attr_mask_t**. The mask can be any combination of the following flags:

**Table 6 HCA Attributes Flags**

| | |
|---|---|
| HCA_ATTR_IS_SM | Is Subnet manager |
| HCA_ATTR_IS_SNMP_TUN_SUP | Is SMNP tunneling supported |
| HCA_ATTR_IS_DEV_MGT_SUP | Is device management supported |
| HCA_ATTR_IS_VENDOR_CLS_SUP | Is Vendor Class supported |

To set up the attributes mask, the following macros should be used:

**Table 7 HCA Attributes Mask Macros**

| | |
|---|---|
| HCA_ATTR_MASK_SET(hca_attr, flag) | set a specific flag in attributes mask |
| HCA_ATTR_MASK_CLR(hca_attr, flag) | clear a specific flag in attributes mask |
| HCA_ATTR_MASK_IS_SET(hca_attr, flag) | check if a flag is set in the mask |
| HCA_ATTR_MASK_CLR_ALL(hca_attr) | clear all flags in the attributes mask. |
| HCA_ATTR_MASK_SET_ALL(hca_attr) | set all flags in the attributes mask |
| HCA_ATTR_IS_FLAGS_SET(hca_attr) | check any of the capability_mask bits is set |

## 2.8 Release HCA Handle

**Note**: This is an Extended VAPI function

**SYNOPSIS:**

```
VAPI_ret_t
EVAPI_release_hca_hndl
(
        IN      VAPI_hca_hndl_t     hca_hndl

)
```

**ARGUMENTS:** **hca_hndl**: Handle for which process resources are released.

**RETURNS:** **VAPI_OK**
**VAPI_EINVAL_HCA_HNDL:** No such opened HCA.

**DESCRIPTION:**

Releases all resources used by this process for an opened HCA. To release all resources, applications *must* use this function.

## 2.9 Close HCA

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_close_hca
(
        IN      VAPI_hca_hndl_t     hca_hndl

)
```

**ARGUMENTS:** **hca_hndl**: Handle to HCA.

**RETURNS:** **VAPI_OK**
**VAPI_EINVAL_HCA_HNDL:** Invalid HCA handle.
**VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**

This call deallocates all the structures allocated during the call to **VAPI_open_hca** and any other resource in the domain of the CI.

It is the responsibility of the consumers to free resources allocated for the HCA that are under its scope.

## 2.10 Allocate Protection Domain

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_alloc_pd
(
        IN      VAPI_hca_hndl_t     hca_hndl,
        OUT     VAPI_pd_hndl_t      *pd_hndl_p

)
```

**ARGUMENTS:**

**hca_hndl**: Handle to HCA.
**pd_hndl_p**: Pointer to handle to Protection Domain object.

**RETURNS:**

**VAPI_OK**
**VAPI_EINVAL_HCA_HNDL:** Invalid HCA handle.
**VAPI_EAGAIN:** Insufficient resources.
**VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**

This call registers a new protection domain.

## 2.11 Deallocate Protection Domain

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_dealloc_pd
(
        IN      VAPI_hca_hndl_t     hca_hndl,
        IN      VAPI_pd_hndl_t      pd_hndl

)
```

**ARGUMENTS:** **hca_hndl**: Handle to HCA.
**pd_hndl:** Handle to Protection Domain Object.

**RETURNS:** **VAPI_OK**
**VAPI_EINVAL_HCA_HNDL:** Invalid HCA handle.
**VAPI_EINVAL_PD_HNDL:** Invalid Protection Domain
**VAPI_EBUSY:** Protection Domain in use.
**VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**

Deregisters the Protection Domain from the **PDA**. It is the **PDA's** responsibility to validate that there are no objects associated with the Protection Domain being deallocated.

## 2.12 Allocate Reliable Datagram Domain

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_alloc_rdd
(
        IN      VAPI_hca_hndl_t     hca_hndl,
        OUT     VAPI_rdd_hndl_t     *rdd_hndl_p

)
```

**ARGUMENTS:** **hca_hndl**: HCA Handle.
**rdd_hndl_p**: Pointer to Reliable Datagram Domain object handle.

**RETURNS:** **VAPI_OK**
**VAPI_EINVAL_HCA_HNDL:** Invalid HCA handle.
**VAPI_EAGAIN:** Out of resources.
**VAPI_EINVAL_RD_UNSUPPORTED**: RD is not supported.
**VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**

Allocates an RD domain.

## 2.13 Deallocate Reliable Datagram Domain

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_dealloc_rdd
(
        IN      VAPI_hca_hndl_t     hca_hndl,
        IN      VAPI_rdd_hndl_t     rdd_hndl

)
```

**ARGUMENTS:** **hca_hndl**: Handle to HCA.
**rdd_hndl**: Reliable Datagram Domain object.

**RETURNS:** **VAPI_OK**
**VAPI_EINVAL_HCA_HNDL:** Invalid HCA handle.
**VAPI_EINVAL_RDD_HNDL:** Invalid RDD handle.
**VAPI_EAGAIN:** Out of resources.
**VAPI_EBUSY**: RDD is busy.
**VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**

Deallocates an RD domain.

# 3 Address Management Verbs

This chapter describes the Mellanox implementation of the following verbs handling basic address management functionality:

- Create Address Handle (p.25)
- Modify Address Handle (p.27)
- Query Address Handle (p.28)
- Destroy Address Handle (p.29)

## 3.1 Create Address Handle

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_create_addr_hndl
(
        IN      VAPI_hca_hndl_t         hca_hndl,
        IN      VAPI_pd_hndl_t          pd_hndl,
        IN      VAPI_ud_av_t            *av_p,
        OUT     VAPI_ud_av_hndl_t       *av_hndl_p

)
```

**ARGUMENTS:**
**hca_hndl**: Handle to HCA.
**pd_hndl:** Protection domain handle
**av_p**: Pointer to Address Vector structure.
**av_hndl_p**: Handle of Address Vector.

**RETURNS:**
**VAPI_OK**
**VAPI_EINVAL_HCA_HNDL:** Invalid HCA handle.
**VAPI_EINVAL_PD_HNDL:** Invalid Protection Domain handle.
**VAPI_EAGAIN:** Insufficient resources.
**VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**

Creates a new Address Vector Handle that can be used later when posting a WR to a UD QP.

The fields of the address vector are specified in the following table:

**Table 8 VAPI_ud_av_t**

| | |
|---|---|
| sl | Service Level |
| grh_flag | Send GRH flag |
| dlid | Destination LID |
| traffic_class | TClass (for global routing) |
| flow_label | Flow Label (for global routing) |
| hop_limit | Hop Limit (for global routing) |
| sgid_index | SGID index in SGID table (for global routing) |
| dgid | Destination GID |
| static_rate | Maximum static rate:<br>- 1 - 1 Gb/s<br>- 2 - 2.5 Gb/s<br>- 3 - 10 Gb/s<br>- 4 - 30 Gb/s |
| src_path_bits | Source Path Bits (with which to create the src LID) |
| port | Egress port |

## 3.2 Modify Address Handle

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_modify_addr_hndl
(
        IN      VAPI_hca_hndl_t         hca_hndl,
        IN      VAPI_ud_av_hndl_t       av_hndl,
        IN      VAPI_ud_av_t            *av_p

)
```

**ARGUMENTS:**

**hca_hndl**: Handle to HCA.
**av_hndl** : Handle of Address Vector handle
**av_p**: Pointer to Address Vector structure.

**RETURNS:**

**VAPI_OK**
**VAPI_EINVAL_HCA_HNDL:** Invalid HCA handle.
**VAPI_EINVAL_AV_HNDL:** Invalid Address Vector handle.
**VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**

Modifies existing address vector handle to point to a different new address vector. For address vector fields, refer to **Table 8, "VAPI_ud_av_t," on page 26**.

## 3.3 Query Address Handle

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_query_addr_hndl
(
        IN      VAPI_hca_hndl_t         hca_hndl,
        IN      VAPI_ud_av_hndl_t       av_hndl,
        OUT     VAPI_ud_av_t            *av_p

)
```

**ARGUMENTS:**

**hca_hndl**: Handle to HCA.
**av_hndl** : Handle of Address Vector .
**av_p**: Pointer to Address Vector structure.

**RETURNS:**

**VAPI_OK**
**VAPI_EINVAL_HCA_HNDL:** Invalid HCA handle.
**VAPI_EINVAL_AV_HNDL:** Invalid address vector handle.
**VAPI_EPERM:** Insufficient permission.

**DESCRIPTION:**

Returns pointer to ADDR_VECP with information about the UD Address Vector represented by AddrVecHandle. For address vector fields, refer to **Table 8, “VAPI_ud_av_t,” on page 26**.

## 3.4 Destroy Address Handle

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_destroy_addr_hndl
(
        IN      VAPI_hca_hndl_t     hca_hndl,
        IN      VAPI_ud_av_hndl_t   av_hndl

)
```

**ARGUMENTS:** **hca_hndl**: Handle to HCA.
**av_hndl**: Handle to Address Vector

**RETURNS:** **VAPI_OK**
**VAPI_EINVAL_HCA_HNDL:** Invalid HCA handle.
**VAPI_EINVAL_AV_HNDL:** Invalid address vector handle.
**VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**

Destroys address vector structure.

# 4 Queue Pair Verbs

This chapter describes the Mellanox implementation of the following verbs handling basic Queue Pair functionality:



## 4.1 Create Queue Pair

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_create_qp
(
        IN      VAPI_hca_hndl_t         hca_hndl,
        IN      VAPI_qp_init_attr_t     *qp_init_attr_p,
        OUT     VAPI_qp_hndl_t          *qp_hndl_p,
        OUT     VAPI_qp_prop_t          *qp_prop_p

)
```

**ARGUMENTS:**

**hca_hndl** : HCA Hande.
**qp_init_attr_p:** Pointer to QP attribute to used for initialization.
**qp_hndl_p:** Pointer to returned QP Handle number.
**qp_prop_p:** Pointer to properties of created QP.

**RETURNS:**

**VAPI_OK**
**VAPI_EAGAIN:** Insufficient resources.
**VAPI_EINVAL_HCA_HNDL:** Invalid HCA handle.
**VAPI_EINVAL_CQ_HNDL:** Invalid CQ handle.
**VAPI_E2BIG_WR_NUM:** Number of WR exceeds HCA cap.
**VAPI_E2BIG_SG_NUM:** Number of SG exceeds HCA cap.
**VAPI_EINVAL_PD_HNDL:** Invalid protection domain handle.
**VAPI_EINVAL_SERVICE_TYPE:** Invalid service type for this QP.
**VAPI_EINVAL_RDD:**Invalid RD domain handle.
**VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**

This call interacts with the **QPM** in order to allocate a **QPO (Queue Pair Object)**. The **QPO** is responsible for the resource allocation and management of the **Queue Pair Context**.

When **VAPI_create_qp** is invoked, the QP will be in the reset state. To make the Queue Pair useful to the consumer, it has to modify its state so reception and/or transmission is enabled, e.g., RTS. This is done by calling to **VAPI_modify_qp**.

When a QP is opened, the properties specified by qp_init_attr_p are applied to the QP. qp_init_attr_p type **VAPI_qp_init_attr_t** and includes the following fields:

**Table 9 VAPI_qp_init_attr_t**

| | |
|---|---|
| sq_cq_hndl | CQ handle to be associated with the SQ. |
| rq_cq_hndl | CQ handle to be associated with the RQ. |
| cap | QP Capabilities (see **Table 10, "VAPI_qp_cap_t," on page 32**). |
| rdd_hndl | RD Domain handle. |
| sq_sig_type | SQ Signaling type:<br>- VAPI_SIGNAL_ALL_WR - signal every submitted WR.<br>- VAPI_SIGNAL_REQ_WR - consumer should specify on a WR if it wants signalling |
| rq_sig_type | RQ Signaling type[1]. |
| pd_hndl | Protection Domain handle. |
| ts_type | Transport Service Type<br>- VAPI_TS_RC<br>- VAPI_TS_RD<br>- VAPI_TS_UC<br>- VAPI_TS_UD |

1. Signaling type on RQ is Mellanox specific.

The QP capability structure, VAPI_qp_cap_t, contains the following fields:

**Table 10 VAPI_qp_cap_t**

| | |
|---|---|
| max_oust_wr_sq | Maximum number of outstanding WRs in the SQ. |
| max_oust_wr_rq | Maximum number of outstanding WRs in the RQ. |
| max_sg_size_sq | Maximum number of scatter/gather elements in a WR in the SQ. |
| max_sg_size_rq | Maximum number of scatter/gather elements in a WR in the RQ. |

If completed successfully, the verb returns the handle to the QP and the actual properties that were assigned to the QP in **qp_prop_p**, of type **VAPI_qp_prop_t**. The fields of **VAPI_qp_prop_t** are specified in the following table:

**Table 11 VAPI_qp_prop_t**

| | |
|---|---|
| qp_num | QP number |
| cap | Actual QP capabilities (see **Table 10, "VAPI_qp_cap_t," on page 32**) |

## 4.2 Modify Queue Pair

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_modify_qp
(
        IN      VAPI_hca_hndl_t         hca_hndl,
        IN      VAPI_qp_hndl_t          qp_hndl,
        IN      VAPI_qp_attr_t          *qp_attr_p,
        IN      VAPI_qp_attr_mask_t     *qp_attr_mask_p,
        OUT     VAPI_qp_cap_t           *qp_cap_p

)
```

**ARGUMENTS:**

**hca_hndl**: HCA handle.
**qp_hndl**: QP handle.
**qp_attr_p**: Pointer to QP attributes to be modified.
**qp_attr_mask_p**: Pointer to the attributes mask to be modified.
**qp_cap_p**: Pointer to QP actual capabilities returned.

**RETURNS:**

**VAPI_OK**
**VAPI_EAGAIN:** Out of resources.
**VAPI_EINVAL_HCA_HNDL:** Invalid HCA handle.
**VAPI_EINVAL_QP_HNDL:** Invalid QP handle.
**VAPI_ENOSYS_ATTR:** QP attribute is not supported.
**VAPI_EINVAL_ATTR**: Cannot change QP attribute.
**VAPI_ENOSYS_ATOMIC:** Atomic operation not supported.
**VAPI_EINVAL_PKEY_IX:** PKey index out of range.
**VAPI_EINVAL_PKEY_TBL_ENTRY:** Pkey index points to an invalid entry in pkey table.
**VAPI_EINVAL_QP_STATE:** Invalid QP state.
**VAPI_EINVAL_RDD_HNDL:** Invalid RDD domain handle.
**VAPI_EINVAL_MIG_STATE:** Invalid path migration state.
**VAPI_E2BIG_MTU**: MTU exceeds HCA port capabilities.
**VAPI_EINVAL_PORT**: Invalid port.
**VAPI_EINVAL_SERVICE_TYPE**: Invalid service type.
**VAPI_E2BIG_WR_NUM**: Maximum number of WRs requested exceeds HCA capabilities.
**VAPI_EINVAL_RNR_NAK_TIMER**: Invalid RNR NAK timer value.
**VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**

Modifies the QP attributes and transitions the QP to a new state. Note that only a subset of all the attributes can be modified while changing the QP state. The **qp_attr_mask_p** specifies the actual attributes to be modified.

The QP attributes specified are of type **VAPI_qp_attr_t** and are specified in the following table:

**Table 12 VAPI_qp_attr_t**

| | | |
|---|---|---|
| qp_state | Next QP State:<br>RST, INIT, RTR, RTS, SQD, SQER, ERR | Always |
| en_sqd_asyn_notif | Enable SQD asynchronous affiliated event notification. | Any->SQD |
| qp_num | QP Number. This is relevant only for query_qp, and not modify_qp. | |
| remote_atomic_flags | Enable/Disable RDMA and Atomic Operations:<br>- VAPI_EN_REM_WRITE - enable RDMA WR<br>- VAPI_EN_REM_READ - enable RDMA RD<br>- VAPI_EN_REM_ATOMIC_OP - enable RDMA atomic operations | RST->INIT<br>INIT->RTR<br>RTR->RTS<br>RTS->RTS<br>SQER->RTS |
| pkey_ix | PKey Index | RST->INIT<br>INIT->RTR |
| port | Physical Port | RST->INIT |
| qkey | QKey (for unconnected service types) | RST->INIT<br>INIT->RTR<br>RTR->RTS<br>RTS->RTS<br>SQER->RTS<br>SQD->RTS |
| path_mtu | | |
| av | Remote Node Address Vector (connected service types), see **Section Table 8“VAPI_ud_av_t,” on page 26**. | INIT->RTR<br>SQD->RTS |
| timeout | Timeout (RC only) | |
| retry_count | Retry count (RC only) | |
| rnr_retry | RNR retry count (RC only) | |
| rq_psn | RQ PSN | INIT->RTR |
| qp_ous_rd_atom | Number of responder resources for RDMA read/atomic | INIT->RTR<br>SQD->RTS |
| alt_av | Alternate Destination Node Address Vector (RC, UC only), see **Section Table 8“VAPI_ud_av_t,” on page 26**. | INIT->RTR<br>RTR->RTS<br>RTS->RTS<br>SQD->RTS |
| alt_timeout | Timeout (RC only) | |
| alt_retry_count | Retry count (RC only) | |
| alt_rnr_retry | RNR retry count (RC only) | |
| alt_pkey_ix | Alternate Path PKey index | |
| min_rnr_timer | Minimum RNR NAK timer field (RC only) | INIT->RTR<br>RTR->RTS<br>RTS->RTS<br>SQER->RTS<br>SQD->RTS |

**Table 12 VAPI_qp_attr_t (Continued)**

| | | |
|---|---|---|
| timeout | Timeout (RC only) | RTR->RTS<br>SQD->RTS |
| retry_count | Retry Count (RC Only) | RTR->RTS<br>SQD->RTS |
| rnr_retry | RNR Retry Count (RC only) | RTR->RTS |
| sq_psn | SQ PSN | RTR->RTS |
| ous_dst_rd_atom | Number of outstanding RDMA read/atomic operations at destination. | RTR->RTS<br>SQD->RTS |
| path_mig_state | Path migration state:<br>- VAPI_MIGRATED<br>- VAPI_REARM | RTR->RTS<br>RTS->RTS<br>SQER->RTS<br>SQD->RTS |
| cap | Required QP capabilities (only max_sq_ous_wr, max_rq_ous_wr are valid). For more detail, see **Table 10, "VAPI_qp_cap_t," on page 32**). | |
| dest_qp_num | Destination QP number. | |

When successful, the output of this verb is the actual QP capability of type **VAPI_qp_cap_t** (for more detail, see **Table 10, "VAPI_qp_cap_t," on page 32**). Only the **max_sq_ous_wr** and **max_rq_ous_wr** fields are valid.

The QP attributes to be changed are specified using a mask of type **VAPI_qp_attr_mask_t**. The mask can be any combination of the flags, which are denoted by the **HCA_ATTR_** prefix and the parameter name listed in **Table 12, "VAPI_qp_attr_t," on page 35** in capital letters (for example, **HCA_ATTR_QP_STATE, HCA_ATTR_SQD_ASYN_NOTIF, HCA_ATTR_REMOTE_ATOMIC_FLAGS**, etc.).

To set up the attributes mask, the following macros should be used:

**Table 13 QP Attributes Mask Macros**

| | |
|---|---|
| QP_ATTR_MASK_SET(hca_attr, flag) | Set a specific flag in the attributes mask. |
| QP_ATTR_MASK_CLR(hca_attr, flag) | Clear a specific flag in the attributes mask. |
| QP_ATTR_MASK_IS_SET(hca_attr, flag) | Check whether a flag is set in the mask. |
| QP_ATTR_MASK_CLR_ALL(hca_attr) | Clear all flags in the attributes mask. |
| QP_ATTR_MASK_SET_ALL(hca_attr) | Set all flags in the attributes mask. |

## 4.3 Query Queue Pair

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_query_qp
(
        IN      VAPI_hca_hndl_t         hca_hndl,
        IN      VAPI_qp_hndl_t          qp_hndl,
        OUT     VAPI_qp_attr_t          *qp_attr_p,
        OUT     VAPI_qp_attr_mask_t     *qp_attr_mask_p,
        OUT     VAPI_qp_init_attr_t     *qp_init_attr_p

)
```

**ARGUMENTS:**

**hca_hndl**: HCA handle.
**qp_hndl**: QP handle.
**qp_attr_p**: Pointer to QP attributes.
**qp_attr_mask_p**: Pointer to QP attributes mask.
**qp_init_attr_p**: Pointer to init attributes

**RETURNS:**

**VAPI_OK**
**VAPI_EINVAL_HCA_HNDL:** Invalid HCA handle.
**VAPI_EINVAL_QP_HNDL:** Invalid QP handle.
**VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**

Returns a VAPI_qp_attr_t structure to the application with all the relevant information that applies to the QP matching qp_hndl, see **Table 12, “VAPI_qp_attr_t,” on page 35**. The verb also returns a pointer to VAPI_qp_init_attr_t, which includes further information of the QP init attributes, see **Table 9, “VAPI_qp_init_attr_t,” on page 32**.

Note that only the relevant fields in **qp_attr_p** and **qp_init_attr_p** are valid. The valid fields in **qp_attr_p** are marked in the mask returned by **qp_attr_mask_p**. For further description of the **VAPI_qp_attr_mask_t**, see **Section “VAPI_modify_qp,” on page 34**.

## 4.4 Destroy Queue Pair

**SYNOPSIS:**

```
VAPI_destroy_qp
(
        IN      VAPI_hca_hndl_t     hca_hndl,
        IN      VAPI_qp_hndl_t      qp_hndl

)
```

**ARGUMENTS:** **hca_hndl**: HCA handle.
**qp_hndl**: QP handle.

**RETURNS:** **VAPI_OK**
**VAPI_EINVAL_HCA_HNDL:** Invalid HCA handle.
**VAPI_EINVAL_QP_HNDL:** Invalid QP handle.
**VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**

Releases all the resources allocated by the CI for this QP.

It is the responsibility of the consumers to release all the resources allocated for all the WRs posted to the QP that are no longer under the responsibility of the CI.

## 4.5 Get Special QP

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_get_special_qp
(
        IN      VAPI_hca_hndl_t         hca_hndl,
        IN      IB_port_t               port,
        IN      VAPI_special_qp_t       qp,
        IN      VAPI_qp_init_attr_t     *qp_init_attr_p,
        OUT     VAPI_qp_hndl_t          *qp_hndl_p,
        OUT     VAPI_qp_cap_t           *qp_cap_p

)
```

**ARGUMENTS:**

**hca_hndl**: HCA Handle.
**phy_port**: Physical port (valid only for QP0 and QP1).
**qp**: QP type.
**qp_init_attr_p**: Pointer to init attributes structure.
**qp_hndl_p**: Pointer to the QP handle.
**qp_cap_p**: Pointer to the actual QP capabilities.

**RETURNS:**

**VAPI_OK**
**VAPI_EAGAIN**: Out of resources.
**VAPI_EINVAL_HCA_HNDL**: Invalid HCA handle.
**VAPI_EINVAL_QP_TYPE**: Invalid special QP type.
**VAPI_EBUSY**: QP already in use (GSI, SMI QPs only).
**VAPI_E2BIG_RAW_DGRAM_NUM**: number of raw. datagram QPs exceeded.
**VAPI_EINVAL_PORT**: Invalid port number.
**VAPI_EINVAL_CQ_HNDL**: Invalid CQ handle
**VAPI_E2BIG_WR_NUM**: Maximum number of work requests exceeds HCA capabilities.
**VAPI_E2BIG_SG_NUM**: Maximum number of scatter/gather elements exceeds HCA capabilities.
**VAPI_EINVAL_PD_HNDL**: Invalid protection domain handle.
**VAPI_ENOSYS_RAW**: Raw datagram QPs are not supported.
**VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**

Creates a special QP that can be used to generate MADs, RAW IPV6 or Ethertype packets.

The valid QP types are (defined in VAPI_special_qp_t):

- VAPI_SMI_QP - QP 0
- VAPI_GSI_QP - QP 1
- VAPI_RAW_ETY_QP - Raw ethertype QP – not supported

- VAPI_RAW_IPV6_QP - Raw IPv6 QP

The physical port is valid only  for QP 0 and QP 1, and is otherwise ignored.

The **VAPI_qp_init_attr_t** is described in **Table 9, "VAPI_qp_init_attr_t," on page 32**. Note that the **rdd_hndl** and the transport service (**ts_type)** are not used.

Upon successful completion, the QP returns the QP handle and the actual QP capabilities. The **VAPI_qp_cap_t** is described in **Table 10, "VAPI_qp_cap_t," on page 32**.

# 5 Completion Queue Verbs

This chapter describes the Mellanox implementation of the following verbs handling completion queue functionality:



## 5.1 Create Completion Queue

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_create_cq
(
        IN      VAPI_hca_hndl_t     hca_hndl,
        IN      VAPI_cqe_num_t      cqe_num,
        OUT     VAPI_cq_hndl_t      *cq_hndl_p,
        OUT     VAPI_cqe_num_t      *num_of_entries_p

)
```

**ARGUMENTS:**
**hca_hndl:** HCA handle.
**cqe_num:** Minimum number of entries required in CQ.
**cq_hndl_p:** Pointer to the created CQ handle.
**num_of_entries_p:** Actual number of entries in CQ.

**RETURNS:**
**VAPI_OK**
**VAPI_EAGAIN**: Out of resources.
**VAPI_EINVAL_HCA_HNDL**: Invalid HCA handle.
**VAPI_E2BIG_CQ_NUM**: Number of entries in CQ exceeds HCA capabilities.
**VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**

Allocates the required data structures for administration of a completion queue, including completion queue buffer space sufficient for the maximum number of entries in the completion.

Completion queue entries are accessed directly by the application.

## 5.2 Query Completion Queue

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_query_cq
(
        IN      VAPI_hca_hndl_t     hca_hndl,
        IN      VAPI_cq_hndl_t      cq_hndl,
        OUT     VAPI_cqe_num_t      *num_of_entries_p

)
```

**ARGUMENTS:** **hca_hndl**: HCA handle.
**cq_hndl**: Completion Queue handle.
**num_of_entries_p**: Pointer to actual number of entries in CQ**.**

**RETURNS:** **VAPI_OK**
**VAPI_EINVAL_HCA_HNDL**: Invalid HCA handle
**VAPI_EINVAL_CQ_HNDL**: Invalid CQ handle.
**VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**

Retrieves the number of entries in the CQ.

## 5.3 Resize Completion Queue

**Note**: not currently supported.

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_resize_cq
(
        IN      VAPI_hca_hndl_t     hca_hndl,
        IN      VAPI_cq_hndl_t      cq_hndl,
        IN      VAPI_cqe_num_t      cqe_num,
        OUT     VAPI_cqe_num_t      *num_of_entries_p


)
```

**ARGUMENTS:**

**hca_hndl**: HCA handle.
**cq_hndl**: CQ handle.
**cqe_num**: Minimim number of entries required in CQ.
**num_of_entries_p**: Pointer to actual number of entries in CQ.

**RETURNS:**

**VAPI_OK**
**VAPI_EAGAIN**: out of resources
**VAPI_EINVAL_HCA_HNDL**: Invalid HCA handle.
**VAPI_EINVAL_CQ_HNDL**: Invalid CQ handle.
**VAPI_E2BIG_CQ_NUM**: Number of entries in CQ exceeds HCA capabilities.
**VAPI_E2BIG_OUS_ENT**: Number of current outstanding entries in CQ exceeds required size.
**VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**

Requests the CEM to increase or decrease the size of the buffer that holds the completion queue entries for a specific CQ.

## 5.4 Destroy Completion Queue

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_destroy_cq
(
        IN      VAPI_hca_hndl_t     hca_hndl,
        IN      VAPI_cq_hndl_t      cq_hndl

)
```

**ARGUMENTS:** **hca_hndl**: HCA handle.
**cq_hndl**: CQ handle.

**RETURNS:** **VAPI_OK**
**VAPI_EINVAL_HCA_HNDL**: Invalid HCA handle.
**VAPI_EINVAL_CQ_HNDL**: Invalid CQ handle.
**VAPI_EBUSY**: one or more work queues are still associated with this CQ.
**VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**

Destroys a CQ and releases all resources associated with it.

Destruction of the completion is not allowed if there are any QPs still associated to the CQ. The CQM holds a list of QP's associated with each completion queue.

# 6 EE Context Verbs

This chapter describes the Mellanox implementation of the following verbs handling basic address management functionality:

- Create EE Context (p.47)
- Modify EE Context Attributes (p.48)
- Query EE Context (p.49)
- Destroy EE Context (p.50)

## 6.1 Create EE Context

**Note**: not currently supported.

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_create_eec
(
        IN      VAPI_hca_hndl_t     hca_hndl,
        IN      VAPI_rdd_t          rdd,
        OUT     VAPI_eec_hndl_t     *eec_hndl_p

)
```

**ARGUMENTS:** **hca_hndl** : HCA handle.
**rdd:** RD domain.
**eec_hndl_p**: Pointer to EE Context Handle.

**RETURNS:** **VAPI_OK**
**VAPI_EAGAIN**: out of resources.
**VAPI_EINVAL_HCA_HNDL**: Invalid HCA handle.
**VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**
Creates an EE context.

## 6.2 Modify EE Context Attributes

**Note**: not currently supported.

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_modify_eec_attr
(
        IN      VAPI_hca_hndl_t     hca_hndl,
        IN      VAPI_eec_hndl_t     eec_hndl,
        IN      VAPI_eec_attr_t     *eec_attr_p

)
```

**ARGUMENTS:**
**hca_hndl**: HCA handle.
**eec_hndl**: EE Context handle
**eec_attr_p**: Pointer to EE Context Attributes structure.

**RETURNS:**
**VAPI_OK**
**VAPI_EAGAIN**: out of resources.
**VAPI_EINVAL_HCA_HNDL**: Invalid HCA handle.
**VAPI_EINVAL_EEC_HNDL**: Invalid EEC handle.
**VAPI_EINVAL_EEC_STATE**: Invalid EEC state.
**VAPI_EINVAL_RDD**: Invalid RD domain.
**VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**

Modifies attributes of an EE context.

## 6.3 Query EE Context

**Note**: not currently supported.

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_query_eec_attr
(
        IN      VAPI_hca_hndl_t     hca_hndl,
        IN      VAPI_eec_hndl_t     eec_hndl,
        OUT     VAPI_eec_attr_t     *eec_attr_p

)
```

**ARGUMENTS:** **hca_hndl**: HCA handle.
**eec_hndl**: EE context handle.
**eec_attr_p**: Pointer to EE Context Attributes structure.

**RETURNS:** **VAPI_OK**
**VAPI_EINVAL_HCA_HNDL**: Invalid HCA handle.
**VAPI_EINVAL_EEC_HNDL**: Invalid EEC handle.
**VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**

Returns attributes of an EE context.

## 6.4 Destroy EE Context

**Note**: not currently supported.

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_destroy_eec
(
        IN      VAPI_hca_hndl_t     hca_hndl,
        IN      VAPI_eec_hndl_t     eec_hndl

)
```

**ARGUMENTS:** **hca_hndl**: HCA handle.
**eec_hndl**: EE context handle.

**RETURNS:** **VAPI_OK**
**VAPI_EINVAL_HCA_HNDL**: Invalid HCA handle.
**VAPI_EINVAL_EEC_HNDL**: Invalid EEC handle.
**VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**

Destroys an EE context.

# 7 Memory Management Verbs

This chapter describes the Mellanox implementation of the following verbs handling basic memory management functionality:



## 7.1 Register Memory Region

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_register_mr
(
        IN      VAPI_hca_hndl_t     hca_hndl,
        IN      VAPI_mrw_t          *req_mrw_p,
        OUT     VAPI_mr_hndl_t      *mr_hndl_p,
        OUT     VAPI_mrw_t          *rep_mrw_p

)
```

**ARGUMENTS:**

**hca_hndl** : HCA handle.
**req_mrw_p**: Pointer to the requested memory region properties.
**mr_hndl_p**: Pointer to the memory region handle.
**rep_mrw**: Pointer to the responded memory region properties.

**RETURNS:**

**VAPI_OK**
**VAPI_EAGAIN**: out of resources.
**VAPI_EINVAL_HCA_HNDL**: Invalid HCA handle.
**VAPI_EINVAL_PD_HNDL**: Invalid PD handle.
**VAPI_EINVAL_VA**: Invalid virtual address.
**VAPI_EINVAL_LEN**: Invalid length.

**VAPI_EINVAL_ACL**: Invalid ACL specifier.
**VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**

Registers a memory region.

The caller should fill the **req_mrw_p** structure fields with the **type**, **virtual start address**, **size**, protection domain handle (**pd_hndl**) and the access control list (**acl**). Upon successful completion, the **rep_mrw_p** will include the **l_key**, the **r_key** (if remote access was requested). The memory region handle is returned in **mr_hndl_p**. Note that the **type** in **req_mrw_p** should be initialized with **MR** (Memory Region). The **VAPI_mrw_t** is described in the following table:

**Table 14 VAPI_mrw_t**

| FIELD | DESCRIPTION |
|---|---|
| type | Memory Region Type:<br>- VAPI_MR = Memory Region<br>- VAPI_MW = Memory Windows<br>- VAPI_MPR = Memory Physical Region<br>- VAPI_MSHAR = Memory Shared Region |
| l_key | For regions: Local protection key.<br>For windows: l_key of region to bind to. |
| r_key | Remote protection key given for this window/region |
| start | Start address of region/window (64 bit address).[1] For physical region, this is the start IOVA. |
| size | Length of the region/window (64 bits).[2] |
| pd_hndl | Protection domain handle (regions only) |
| acl | Region access control list.<br>- VAPI_EN_LOCAL_WRITE<br>- VAPI_EN_REMOTE_WRITE<br>- VAPI_EN_REMOTE_READ<br>- VAPI_EN_REMOTE_ATOM<br>- VAPI_EN_MEMREG_BIND |
| pbuf_list_len | Physical buffers list (pbuf_list_p) length. For physical memory region only (type==VAPI_MPR) |
| *pbuf_list_p | Physical buffers addresses list. For physical memory region only (type==VAPI_MPR) |
| iova_offset | Offset of "start" in first buffer. For physical memory region only (type==VAPI_MPR) |

1. May differ from requested address by aligning down to page boundary.
2. May differ from requested address by aligning up to page boundary.

.

## 7.2 Query Memory Region

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_query_mr
(
        IN      VAPI_hca_hndl_t     hca_hndl,
        IN      VAPI_mr_hndl_t      mr_hndl,
        OUT     VAPI_mrw_t          *rep_mrw_p,
        OUT     VAPI_virt_addr_t    *remote_start_p,
        OUT     VAPI_virt_addr_t    *remote_size_p

)
```

**ARGUMENTS:**

**hca_hndl**: HCA handle.
**mr_hndl**: Memory Region handle.
**rep_mrw_p**: Pointer to Memory Region attributes
**remote_start_p**: Pointer to returned value of the remote start address.
**remote_size_p**: Pointer to returned value of the size of the remote region.

**RETURNS:**

**VAPI_OK**
**VAPI_EINVAL_HCA_HNDL**: Invalid HCA handle.
**VAPI_EINVAL_MR_HNDL**:Invalid Memory Region handle.
**VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**

Queries a memory region handle and returns a **VAPI_mrw_t**, which includes all the memory region properties: protection domain handle, ACL, LKey, RKey and actual protection bounds. The protection bounds returned in rep_mrw_p are the **local** protection bounds enforced by the HCA. The **remote** protection bounds are returned in **remote_start_p** and **remote_size_p** and are valid only when remote access is requested.

## 7.3 Deregister Memory Region

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_deregister_mr
(
        IN      VAPI_hca_hndl_t     hca_hndl,
        IN      VAPI_mr_hndl_t      mr_hndl

)
```

**ARGUMENTS:** **hca_hndl**: HCA handle.
**mr_hndl**: Memory Region Handle

**RETURNS:** **VAPI_OK**
**VAPI_EINVAL_HCA_HNDL**: Invalid HCA handle
**VAPI_EINVAL_MR_HNDL**: Invalid memory region handle
**VAPI_EBUSY**: memory region still has bound window(s)
**VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**

Destroys a registered memory region. The memory region deregistering has to be invalidated from the CI.

## 7.4 Reregister Memory Region

**Note**: not currently supported.

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_reregister_mr
(
        IN      VAPI_hca_hndl_t         hca_hndl,
        IN      VAPI_mr_hndl_t          mr_hndl,
        IN      VAPI_mr_change_t        change_type,
        IN      VAPI_mrw_t              *req_mrw_p,
        OUT     VAPI_mr_hndl_t          *rep_mr_hndl_p,
        OUT     VAPI_mrw_t              *rep_mrw_p

)
```

**ARGUMENTS:**

**hca_hndl**: HCA handle.
**mr_hndl**: Old Memory Region handle.
**change_type**: Requested change type.
**req_mrw_p**: Pointer to the requested memory region properties.
**rep_mr_hndl_p**: Pointer to the returned new memory region handle.
**rep_mrw_p**: Pointer to the returned memory region properties.

**RETURNS:** **VAPI_OK**

**VAPI_EAGAIN**: out of resources.
**VAPI_EINVAL_HCA_HNDL**: Invalid HCA handle.
**VAPI_EINVAL_MR_HNDL**: Invalid memory region handle.
**VAPI_EINVAL_VA**: Invalid virtual address.
**VAPI_EINVAL_LEN**: Invalid length.
**VAPI_EINVAL_PD_HNDL**: Invalid protection domain handle.
**VAPI_EINVAL_ACL**: Invalid ACL specifier.
**VAPI_EBUSY**: memory region still has bound window(s).
**VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**

Reregisters the memory region associated with the **mr_hndl**.

## 7.5 Register Shared Memory Region

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_register_smr
(
        IN      VAPI_hca_hndl_t     hca_hndl,
        IN      VAPI_mr_hndl_t      orig_mr_hndl,
        IN      VAPI_mrw_t          *req_mrw_p,
        OUT     VAPI_mr_hndl_t      *mr_hndl_p,
        OUT     VAPI_mrw_t          *rep_mrw_p

)
```

**ARGUMENTS:**

**hca_hndl** : HCA handle.
**orig_mr_hndl**: Original memory region handle.
**req_mrw_p**: Pointer to the requested memory region properties.
**mr_hndl_p**: Pointer to the returned memory region handle.
**rep_mrw**: Pointer to the returned memory region properties.

**RETURNS:**

**VAPI_OK**
**VAPI_EAGAIN**: out of resources.
**VAPI_EINVAL_HCA_HNDL**: Invalid HCA handle.
**VAPI_EINVAL_MR_HNDL**: Invalid MR handle.
**VAPI_EINVAL_PD_HNDL**: Invalid PD handle.
**VAPI_EINVAL_ACL**: Invalid ACL specifier.
**VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**

Registers a shared memory region associated with the physical buffers of an existing memory region referenced by **orig_mr_hndl**. The **req_mrw_p** is a pointer to the requested memory region properties and is of type **VAPI_mrw_t**. The requested type should be set to **MSHAR** (shared memory region), and the struct should contain the requested start virtual address (**start** field), the protection domain handle and the ACL.

## 7.6 Allocate Memory Window

**Note**: not currently supported.

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_alloc_mw
(
        IN      VAPI_hca_hndl_t     hca_hndl,
        IN      VAPI_pd_hndl_t      pd,
        OUT     VAPI_mw_hndl_t      *mw_hndl_p,
        OUT     VAPI_rkey_t         *rkey_p

)
```

**ARGUMENTS:**

**hca_hnd:** HCA handle.
**pd:** Protection Domain handle.
**mw_hndl_p**: Pointer to new allocated windows handle.
**rkey_p**: Pointer to windows unbounded Rkey

**RETURNS:**

**VAPI_OK**
**VAPI_EAGAIN**
**VAPI_EINVAL_HCA_HNDL**
**VAPI_EINVAL_PD_HNDL**
**VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**

Allocates an MWO object that later can be bound to an RKey.

## 7.7 Query Memory Window

**Note**: not currently supported.

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_query_mw
(
        IN      VAPI_hca_hndl_t     hca_hndl,
        IN      VAPI_mw_hndl_t      mw_hndl,
        OUT     VAPI_rkey_t         *rkey_p,
        OUT     VAPI_pd_hndl_t      *pd_p

)
```

**ARGUMENTS:**
**hca_hndl**: HCA handle.
**mw_hndl**: Windows handle.
***rkey_p**: Pointer to unbound Rkey of window.
***pd_p**: Pointer to Protection Domain handle of window.

**RETURNS:**
**VAPI_OK**
**VAPI_EINVAL_HCA_HNDL**
**VAPI_EINVAL_PD_HNDL**
**VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**

Returns the current PD associated with the memory domain retrieved from the PDA (no access to HW required).

## 7.8 Bind Memory Window

**Note**: not currently supported.

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_bind_mw
(
        IN      VAPI_hca_hndl_t         hca_hndl,
        IN      VAPI_mw_hndl_t          mw_hndl,
        IN      const VAPI_mrw_t        *bind_prop_p,
        IN      VAPI_qp_hndl_t          qp_hndl,
        IN      VAPI_wr_id_t            id,
        IN      VAPI_comp_type_t        comp_type,
        OUT     VAPI_rkey_t             *new_rkey_p

)
```

**ARGUMENTS:**

**hca_hndl**: HCA handle.
**mw_hndl**: Handle of memory window.
**bind_prop_p**: Pointer to binding properties.
**qp_hndl**: QP handle for binding this window.
**id**: WQE ID for binding operation (to track on completion).
**comp_type**: Completion type for binding WQE.
**new_rkey_p**: Pointer to RKey of bound window.

**RETURNS:**

**VAPI_OK**
**OUT_OF_RESOURCES**
**INVALID_HCA_hndl**
**INVALID_QP_hndl**
**INVALID_MEMORY_WINDOWS_hndl**
**INVALID_R_KEY**
**INVALID_MEMORY_REGION_hndl**
**INVALID_L_KEY**
**INVALID_VIRTUAL_ADRESS**
**INVALID_LENGTH**
**INVALID_ACCESS_REQUEST**
**INVALID_COMP_REQUEST_STATUS**
**WORK_REQUEST_OK**
**WORK_REQUEST_PROTECTION_ERROR**
**VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**

Binds a memory window. This call is performed entirely in user mode. The posted descriptor returns an RKey that can be used in subsequent remote access to the bound memory region.

## 7.9 Deallocate Memory Window

**Note**: not currently supported.

**SYNOPSIS:**

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_dealloc_mw
(
        IN      VAPI_hca_hndl_t     hca_hndl,
        IN      VAPI_mw_hndl_t      mw_hndl

)
```

**ARGUMENTS:** **HCAHndl**: HCA handle.
**MEMWWINHandle**: Memory Windows handle.

**RETURNS:** **VAPI_OK**
**INVALID_HCA_hndl**
**INVALID_MEMORY_WINDOWS_hndl**
**VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**

Destroys memory windows.

# 8 Multicast Verbs

This chapter describes the Mellanox implementation of the following verbs handling basic multicast functionality:

- Attach Qp to Multicast Group (p.61)
- Detach QP from Multicast Group (p.62)

## 8.1 Attach QP to Multicast Group

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_attach_to_multicast
(
        IN    VAPI_hca_hndl_t    hca_hndl,
        IN    IB_gid_t           mcg_dgid,
        IN    VAPI_qp_hndl_t     qp_hndl,
        IN    IB_lid_t           mcg_dlid

)
```

**ARGUMENTS:**

**hca_hndl**: HCA handle.
**mcg_dgid**: DGID of multicast group.
**qp_hndl**: QP handle.
**mcg_dlid:** DLID of multicast group - currently ignored

**RETURNS:**

**VAPI_OK**
**VAPI_EAGAIN**: out of resources.
**VAPI_EINVAL_HCA_HNDL**: Invalid HCA handle.
**VAPI_EINVAL_QP_HNDL:** Invalid QP handle.
**VAPI_E2BIG_MCG_SIZE:** Too many QPs attached to multicast group.
**VAPI_EINVAL_MCG_GID:** Invalid multicast DGID.
**VAPI_EINVAL_SERVICE_TYPY:** Invalid Service Type for this QP.

**DESCRIPTION:**

Attaches QP to multicast group.

## 8.2 Detach QP from Multicast Group

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_detach_from_multicast
(
        IN    VAPI_hca_hndl_t    hca_hndl,
        IN    IB_gid_t           mcg_dgid,
        IN    VAPI_qp_hndl_t     qp_hndl,
        IN    IB_lid_t           mcg_dlid

)
```

**ARGUMENTS:**

**hca_hndl**: HCA handle.
**mcg_dgid**: DGID of multicast group.
**qp_hndl**: QP handle.
**mcg_dlid:** DLID of multicast group - currently ignored

**RETURNS:**

**VAPI_OK**
**VAPI_EAGAIN**: Out of resources.
**VAPI_EINVAL_HCA_HNDL**: Invalid HCA handle.
**VAPI_EINVAL_QP_HNDL:** Invalid QP handle.
**VAPI_E2BIG_MCG_SIZE:** Number of QPs attached to multicast groups exceeded.
**VAPI_EINVAL_MCG_GID:** Invalid multicast DGID.
**VAPI_EINVAL_SERVICE_TYPY:** Invalid Service Type for this QP.

**DESCRIPTION:**

Detaches QP from a multicast group.

# 9 Work Request Verbs

This chapter describes the Mellanox implementation of the following verbs handling work requests:

- Post Send request (p.63)
- Post Receive Request (p.66)
- Poll for Completion (p.67)
- Request Completion Notification (p.69)
- Request Completion Notification for a Completion Queue (p.70)
- Clear Completion Notification for a Completion Queue (p.71)

## 9.1 Post Send Request

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_post_sr
(
        IN      VAPI_hca_hndl_t     hca_hndl,
        IN      VAPI_qp_hndl_t      qp_hndl,
        IN      VAPI_sr_desc_t      *sr_desc_p

)
```

**ARGUMENTS:**

**hca_hndl**: HCA handle.
**qp_hndl**: QP handle.
**sr_desc_p**: Pointer to the send request descriptor attributes structure.

**RETURNS:**

**VAPI_OK**
**VAPI_EINVAL_HCA_HNDL**: Invalid HCA handle.
**VAPI_EINVAL_QP_HNDL**: Invalid QP handle.
**VAPI_E2BIG_WR_NUM**: Too many posted work requests.
**VAPI_EINVAL_OP**: Invalid operation.
**VAPI_EINVAL_QP_STATE**: Invalid QP state.
**VAPI_EINVAL_NOTIF_TYPE**: Invalid completion notification type.
**VAPI_EINVAL_SG_FMT**: Invalid scatter/gather list format.
**VAPI_EINVAL_SG_NUM**: Invalid scatter/gather list length.
**VAPI_ENOSYS_ATOMIC**: Atomic operations not supported.

**VAPI_EINVAL_AH**: Invalid address handle.
**VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**

Posts a send queue work request, the properties of which are specified in the structure pointed to by **sr_desc_p**, which is of type **VAPI_sr_desc_t**:

**Table 15 VAPI_sr_desc_t**

| | |
|---|---|
| id | User defined work request ID |
| opcode | Transaction OPCODE:<br>- RDMA_WRITE<br>- RDAM_WRITE_WITH_IMM<br>- SEND<br>- SEND_WITH_IMM<br>- RDMA_READ<br>- ATOMIC_CMP_AND_SWAP<br>- ATOMIC_FETCH_AND_ADD |
| comp_type | Set if completion will be generated:<br>- SIGNALED<br>- UNSIGNALED (default) |
| sg_lst_p | Pointer to the gather list. See **Table 16, "VAPI_sg_lst_entry_t," on page 65**. |
| sg_lst_len | Length of the gather list. |
| imm_data | dword immediate data. |
| fence | Complete execution before next descriptor is executed. |
| remote_ah | Remote node address handle (valid only for datagrams). |
| remote_qp | Remote node QP (valid only for datagrams). |
| remote_qkey | Remote node QKey (valid only for datagrams). |
| ethertype | Ethertype associated with the WR (valid only for RAW datagrams). |
| eec_hndl | EEC handle (valid only for UD). |
| set_se | Set solicited bit (SE) on last packet of message:<br>- true<br>- false - default |
| remote_addr | Remote Address (valid only for RDMA write). |
| r_key | Remote Address Rkey (valid only for RDMA write). |

**Table 15 VAPI_sr_desc_t (Continued)**

| | |
|---|---|
| operand1 | First 64 bit operand for atomic operation. |
| operand2 | Second 64 bit operand for atomic operation. |
| local_data_seg | A local data segment. |
| if_gent | Generate event upon description completion:<br>- DO_EVENT<br>- NO_EVENT (default) |
| ack_req | Set ack-required bit in last packet:<br>- SET_ACK<br>- NO_ACK |

The **sg_lst_p** points to a gather list (of length **sg_lst_len**), which is an array of local buffers used as the source of the data to be transmitted in this Work Request. Each entry in this array has the following format.

**Table 16 VAPI_sg_lst_entry_t**

| | |
|---|---|
| src_addr | 64 bit source data address |
| len | Length of buffer |
| l_key | Lkey used by the CI when accessing this array. |

## 9.2 Post Receive Request

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_post_rr
(
        IN      VAPI_hca_hndl_t     hca_hndl,
        IN      VAPI_QP_hndl_t      qp_hndl,
        IN      VAPI_rr_desc_t      *rr_desc_p

)
```

**ARGUMENTS:**

**hca_hndl**: HCA handle.
**qp_hndl**: QP handle.
**rr_desc_p**: Pointer to the receive request descriptor attributes structure.

**RETURNS:**

**VAPI_OK**
**VAPI_EINVAL_HCA_HNDL**: Invalid HCA handle.
**VAPI_EINVAL_QP_HNDL**: Invalid QP handle.
**VAPI_E2BIG_WR_NUM**: Too many posted work requests.
**VAPI_EINVAL_OP**: Invalid operation.
**VAPI_EINVAL_QP_STATE**: Invalid QP state.
**VAPI_EINVAL_SG_FMT**: Invalid scatter/gather list format.
**VAPI_EINVAL_SG_NUM**: Invalid scatter/gather list length.
**VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**

Posts a receive request descriptor to the receive queue. The receive request descriptor is of type **VAPI_rr_desc_t** and is described in the following table:

**Table 17 VAPI_rr_desc_t**

| | |
|---|---|
| id | User defined work request ID. |
| opcode | Transaction OPCODE:<br>- VAPI_RECEIVE |
| comp_type | Set if completion will be generated:[1]<br>- VAPI_SIGNALED<br>- VAPI_UNSIGNALED (default) |
| sg_lst_p | Pointer to the scatter list. See **Table 16, "VAPI_sg_lst_entry_t," on page 65**. |
| sg_lst_len | Length of the scatter list. |

1. This is Mellanox specific

## 9.3 Poll for Completion

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_poll_cq
(
        IN      VAPI_hca_hndl_t     hca_hndl,
        IN      VAPI_cq_hndl_t      cq_hndl,
        OUT     VAPI_wc_desc_t      *comp_desc_p

)
```

**ARGUMENTS:** **hca_hndl**: Handle to HCA.
**cq_hndl**: CQ handle.
***comp_desc_p**: Pointer to work completion descriptor structure.

**RETURNS:** **VAPI_OK**
**VAPI_EINVAL_HCA_HNDL**: Invalid HCA handle.
**VAPI_EINVAL_CQ_HNDL**: Invalid CQ handle.
**VAPI_EAGAIN**: CQ is empty.
**VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**

Retrieves an ICQE (Independent Completion Queue Entry), which is a device independent structure used to retrieve completion status of a WR posted to the Send/Receive Queue including **VAPI_bind_mw**.

The verb retrieves a completion queue entry into the descriptor pointed by **wc_desc_p**, which is of type **VAPI_wc_desc_t** and described in the following table:

**Table 18 VAPI_wc_desc_t**

| | |
|---|---|
| id | User defined work request ID |
| opcode | Transaction OPCODE:<br>- VAPI_RDMA_WRITE<br>- VAPI_RDAM_WRITE_WITH_IMM<br>- VAPI_SEND<br>- VAPI_SEND_WITH_IMM<br>- VAPI_RDMA_READ<br>- VAPI_ATOMIC_CMP_AND_SWAP<br>- VAPI_ATOMIC_FETCH_AND_ADD<br>- VAPI_MEM_WND_BIND<br>- VAPI_SEND_DATA_RCV (for post_receive requests)<br>- VAPI_RDMA_DATA_RCV (for RDMA with imm) |
| len | The actual number of bytes transferred[1]. |

**Table 18 VAPI_wc_desc_t (Continued)**

| | |
|---|---|
| imm_data | dword immediate data (valid only when **RDMA_DATA_RCV** is indicated in the opcode field). |
| remote_node_addr | Remote node address (datagram services only), see **Table 19, "VAPI_remote_node_addr_t," on page 68**. |
| grh_flag | GRH present indication. |
| pkey_ix | PKey index. |
| status | Completion status:<br>- VAPI_SUCCESS<br>- VAPI_LOC_LEN_ERR - local length error<br>- VAPI_LOC_OP_ERR - local operation error<br>- VAPI_LOC_PROT_ERR - local protection error<br>- VAPI_WR_FLUSH_ERR - WR flushed error<br>- VAPI_MW_BIND_ERR - memory window bind error<br>- VAPI_REM_INV_REQ_ERR - remote invalid request error<br>- VAPI_REM_ACCESS_ERR - remote access error<br>- VAPI_REM_OP_ERR -remote operation error<br>- VAPI_RETRY_EXC_ERR - transport retry exceeded error<br>- VAPI_RNR_RETRY_EXC_ERR - RNR retry exceeded error<br>- VAPI_REM_INV_RD_REQ_ERR - remote invalid RD request<br>- VAPI_INV_EECN_ERR - invalid EEC number error<br>- VAPI_INV_EEC_STATE_ERR - invalid EEC state error |
| free_res_count | Freed resource count (RD RQ only). |

1. In cases of a RAW datagram or UD, the **len** includes 40 extra bytes, regardless of whether a GRH is included.

The **remote_node_address** is of type **VAPI_remote_node_addr_t** and is valid only for Datagram services. It is specified in the following table:

**Table 19 VAPI_remote_node_addr_t**

| | |
|---|---|
| type | Remote node address type:<br>- VAPI_RNA_RD<br>- VAPI_RNA_UD<br>- VAPI_RNA_RAW_IPV6<br>- VAPI_RNA_RAW_ETY |
| slid | Source LID |
| sl | Service Level |
| qp_ety (Union)<br>- qp<br>- ety | <br>- Queue pair (RD and UD)<br>- Ethertype (RAW Ethertype only) |
| ee_dlid (Union)<br>- loc_eecn<br>- dst_path_bits | <br>- Local EEC number (RD)<br>- DLID path bits (UD, RAW IPv6 and RAW Ethertype) |

## 9.4 Request Completion Notification

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_req_comp_notif
(
        IN      VAPI_hca_hndl_t         hca_hndl,
        IN      VAPI_cq_hndl_t          cq_hndl,
        IN      VAPI_cq_notif_type_t    notif_type

)
```

**ARGUMENTS:**

**hca_hndl**: HCA handle.
**cq_hndl**: CQ handle.
**notif_type**: CQ Notification Type.

**RETURNS:**

**VAPI_OK**
**VAPI_EINVAL_HCA_HNDL**: Invalid HCA handle.
**VAPI_EINVAL_CQ_HNDL**: Invalid CQ handle.
**VAPI_EINVAL_NOTIF_TYPE**: Invalid notification type.
**VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**

Requests one of the following types of notification (specified in **notif_type**):

- VAPI_NEXT_COMP - notify on next completion.
- VAPI_SOLIC_COMP - notify on solicited completion.

## 9.5 Request Completion Notification for a Completion Queue

**Note**: This is an Extended VAPI function

**SYNOPSIS:**

```
VAPI_ret_t
EVAPI_set_comp_eventh
(
        IN      VAPI_hca_hndl_t                 hca_hndl,
        IN      VAPI_cq_hndl_t                  cq_hndl,
        IN      VAPI_completion_event_handler_t  completion_handler,
        IN      void *                          private_data,
        OUT     EVAPI_compl_handler_hndl_t      *completion_handler_hndl

)
```

**ARGUMENTS:**

**hca_hndl**: HCA handle.
**cq_hndl**: CQ handle.
**completion_handler:** handler to call for completions on Completion Queue cq_hndl.
**private_data:** pointer to data for completion handler.
**completion_handler_hndl:** returned handle to use for clearing this completion handler.

**RETURNS:**

**VAPI_OK**
**VAPI_EINVAL_HCA_HNDL**: Invalid HCA handle.
**VAPI_EINVAL_CQ_HNDL**: Invalid CQ handle.

**DESCRIPTION:**

Registers a specific completion handler to handle completions for a specific completion queue. The private data specified here is provided to the completion callback when a completion occurs on the specified CQ. If the private data is a pointer, it should point to static or "malloc'ed" data; the private data must be available until this completion handler instance is cleared (with EVAPI_clear_comp_eventh).

## 9.6 Clear Completion Notification for a Completion Queue

**Note**: This is an Extended VAPI function

**SYNOPSIS:**

```
VAPI_ret_t
EVAPI_clear_comp_eventh
(
        IN    VAPI_hca_hndl_t                hca_hndl,
        OUT   EVAPI_compl_handler_hndl_t     completion_handler_hndl

)
```

**ARGUMENTS:**

**hca_hndl**: HCA handle.
**completion_handler_hndl:** handle to use for clearing this completion handler.

**RETURNS:**

**VAPI_OK**
**VAPI_EINVAL_HCA_HNDL**: Invalid HCA handle.
**VAPI_EINVAL_CQ_HNDL**: Invalid CQ handle.

**DESCRIPTION:**

Clears a completion handler that was registered to handle completions for a specific completion queue. If a handler was not registered, returns OK anyway.

# 10 Event Handling Verbs

This chapter describes the Mellanox implementation of the following verbs handling events:

- Set Completion Event Handler (p.73)
- Set Asynchronous Event Handler (‘p.74)

## 10.1 Set Completion Event Handler

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_set_comp_event_handler
(
        IN      VAPI_hca_hndl_t                 hca_hndl,
        IN      VAPI_completion_event_handler_t handler,
        IN      void                           *private_data

)
```

**ARGUMENTS:** **hca_hndl**: HCA handle.
**handler**: Completion Event Handler function address.
**private_data**: Pointer to handler context (handler specific).

**RETURNS:** **VAPI_OK**
**VAPI_EINVAL_HCA_HNDL**: Invalid HCA handle.
**VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**

Registers a completion event handler. Only one CQ event handler can be registered per HCA.

The CQ event handler function prototype is as follows:

```
void
VAPI_completion_event_handler
(
        IN      VAPI_hca_hndl_t         hca_hndl,
        IN      VAPI_cq_hndl_t          cq_hndl,
        IN      void                    *private_data

)
```

## 10.2 Set Asynchronous Event Handler

**SYNOPSIS:**

```
VAPI_ret_t
VAPI_set_async_event_handler
(
        IN      VAPI_hca_hndl_t                 hca_hndl,
        IN      VAPI_async_event_handler_t  handler,
        IN      void                            *private_data

)
```

**ARGUMENTS:** **hca_hndl**: HCA handler.
**handler**: Asynchronous event handler function address.
**private_data**: Pointer to handler context (handler specific).

**RETURNS:** **VAPI_OK**
**VAPI_EINVAL_HCA_HNDL**: Invalid HCA handle.
**‘VAPI_EPERM:** Insufficient permissions.

**DESCRIPTION:**

Sets the specified Asynchronous Event Handler.

The handler is a pointer to a function as follows:

```
void
VAPI_async_event_handler
(
        IN      VAPI_hca_hndl_t          hca_handle,
        IN      VAPI_event_record_t     *event_record_p,
        IN      void                    *private_data

)
```

The **event_record_p** is a pointer to a structure of type **VAPI_event_record_t**, which contains the following fields:

**Table 20 VAPI_event_record_t**

| | |
|---|---|
| type | Asynchronous event type:<br>- VAPI_QP_PATH_MIGRATED<br>- VAPI_EEC_PATH_MIGRATED<br>- VAPI_QP_COMM_ESTABLISHED<br>- VAPI_EEC_COMM_ESTABLISHED<br>- VAPI_SEND_QUEUE_DRAINED<br>- VAPI_CQ_ERROR<br>- VAPI_LOCAL_WQ_CATASTROPHIC_ERROR<br>- VAPI_PATH_MIG_REQ_ERROR<br>- VAPI_LOCAL_CATASTROPHIC_ERROR<br>- VAPI_PORT ERROR |
| modifier | Union of the following four fields: |
| qp_hndl | The affiliated QP handle (if applicable) |
| eec_hndl | The affiliated EEC handle (if applicable) |
| cq_hndl | The affiliated CQ handle (if applicable) |
| port_num | The affiliated port number (if applicable) |