

NAME

`ibv_create_cq_ex` - create a completion queue (CQ)

SYNOPSIS

```
#include <infiniband/verbs.h>
```

```
struct ibv_cq_ex *ibv_create_cq_ex(struct ibv_context *context,
                                  struct ibv_create_cq_attr_ex *cq_attr);
```

DESCRIPTION

`ibv_create_cq_ex()` creates a completion queue (CQ) for RDMA device context *context*. The argument *cq_attr* is a pointer to struct `ibv_cq_init_attr_ex` as defined in `<infiniband/verbs.h>`.

```
struct ibv_cq_init_attr_ex {
    int          cqe;          /* Minimum number of entries required for CQ */
    void         *cq_context; /* Consumer-supplied context returned for completion events */
    struct ibv_comp_channel *channel; /* Completion channel where completion events will be queued. M
    int          comp_vector; /* Completion vector used to signal completion events. Must be >= 0 and <
    uint64_t     wc_flags;    /* The wc_flags that should be returned in ibv_poll_cq_ex. Or'ed bit of en
    uint32_t     comp_mask;   /* compatibility mask (extended verb). */
    uint32_t     flags       /* One or more flags from enum ibv_create_cq_attr_flags */
};
```

```
enum ibv_wc_flags_ex {
    IBV_WC_EX_WITH_BYTE_LEN      = 1 << 0, /* Require byte len in WC */
    IBV_WC_EX_WITH_IMM          = 1 << 1, /* Require immediate in WC */
    IBV_WC_EX_WITH_QP_NUM       = 1 << 2, /* Require QP number in WC */
    IBV_WC_EX_WITH_SRC_QP       = 1 << 3, /* Require source QP in WC */
    IBV_WC_EX_WITH_SLID         = 1 << 4, /* Require slid in WC */
    IBV_WC_EX_WITH_SL           = 1 << 5, /* Require sl in WC */
    IBV_WC_EX_WITH_DLID_PATH_BITS = 1 << 6, /* Require dlid path bits in WC */
    IBV_WC_EX_WITH_COMPLETION_TIMESTAMP = 1 << 7, /* Require completion timestamp in WC */
    IBV_WC_EX_WITH_CVLAN        = 1 << 8, /* Require VLAN info in WC */
    IBV_WC_EX_WITH_FLOW_TAG      = 1 << 9, /* Require flow tag in WC */
};
```

```
enum ibv_cq_init_attr_mask {
    IBV_CQ_INIT_ATTR_MASK_FLAGS = 1 << 0,
};
```

```
enum ibv_create_cq_attr_flags {
    IBV_CREATE_CQ_ATTR_SINGLE_THREADED    = 1 << 0, /* This CQ is used from a single threaded, thus
};
```

Polling an extended CQ

In order to poll an extended CQ efficiently, a user could use the following functions.

Completion iterator functions

int ibv_start_poll(struct ibv_cq_ex *cq, struct ibv_poll_cq_attr *attr)

Start polling a batch of work completions. *attr* is given in order to make this function easily extensible in the future. This function either returns 0 on success or an error code otherwise. When no completions are available on the CQ, ENOENT is returned, but the CQ remains in a valid state. On success, querying the completion's attribute could be done using the query functions described below. If an error code is given, *end_poll* shouldn't be called.

int ibv_next_poll(struct ibv_cq_ex *cq)

This function is called in order to get the next work completion. It has to be called after *start_poll* and before *end_poll* are called. This function either returns 0 on success or an error code otherwise. When no completions are available on the CQ, ENOENT is returned, but the CQ remains in a valid state. On success, querying the completion's attribute could be done using the query functions described below. If an error code is given, *end_poll* should still be called, indicating this is the end of the polled batch.

void ibv_end_poll(struct ibv_cq_ex *cq)

This function indicates the end of polling batch of work completions. After calling this function, the user should start a new batch by calling *start_poll*.

Polling fields in the completion

Below members and functions are used in order to poll the current completion. The current completion is the completion which the iterator points to (*start_poll* and *next_poll* advances this iterator). Only fields that the user requested via *wc_flags* in *ibv_create_cq_ex* could be queried. In addition, some fields are only valid in certain opcodes and status codes.

uint64_t wr_id - Can be accessed directly from struct *ibv_cq_ex*.

enum ibv_wc_status - Can be accessed directly from struct *ibv_cq_ex*.

enum ibv_wc_opcode ibv_wc_read_opcode(struct ibv_cq_ex *cq); Get the opcode from the current completion.

uint32_t ibv_wc_read_vendor_err(struct ibv_cq_ex *cq); Get the vendor error from the current completion.

uint32_t ibv_wc_read_byte_len(struct ibv_cq_ex *cq); Get the vendor error from the current completion.

uint32_t ibv_wc_read_imm_data(struct ibv_cq_ex *cq); Get the immediate data field from the current completion.

uint32_t ibv_wc_read_qp_num(struct ibv_cq_ex *cq); Get the QP number field from the current completion.

uint32_t ibv_wc_read_src_qp(struct ibv_cq_ex *cq); Get the source QP number field from the current completion.

int ibv_wc_read_wc_flags(struct ibv_cq_ex *cq); Get the QP flags field from the current completion.

uint16_t ibv_wc_read_pkey_index(struct ibv_cq_ex *cq); Get the pkey index field from the current completion.

uint32_t ibv_wc_read_slid(struct ibv_cq_ex *cq); Get the slid field from the current completion.

uint8_t ibv_wc_read_sl(struct ibv_cq_ex *cq); Get the sl field from the current completion.

uint8_t ibv_wc_read_dlid_path_bits(struct ibv_cq_ex *cq); Get the dlid_path_bits field from the current completion.

uint64_t ibv_wc_read_completion_ts(struct ibv_cq_ex *cq); Get the completion timestamp from the current completion.

uint16_t ibv_wc_read_cvlan(struct ibv_cq_ex *cq); Get the CVLAN field from the current completion.

uint32_t ibv_wc_read_flow_tag(struct ibv_cq_ex *cq); Get flow tag from the current completion.

RETURN VALUE

ibv_create_cq_ex() returns a pointer to the CQ, or NULL if the request fails.

NOTES

ibv_create_cq_ex() may create a CQ with size greater than or equal to the requested size. Check the `cqe` attribute in the returned CQ for the actual size.

CQ should be destroyed with `ibv_destroy_cq`.

SEE ALSO

ibv_create_cq(3), **ibv_destroy_cq(3)**, **ibv_resize_cq(3)**, **ibv_req_notify_cq(3)**, **ibv_ack_cq_events(3)**,
ibv_create_qp(3)

AUTHORS

Matan Barak <matanb@mellanox.com>