DOCA RDMA
Table of contents

Introduction
Prerequisites
Environment
Architecture
Objects
Device
Memory Map
Buffer Inventory and Buffers
Configuration Phase
Configurations
Mandatory Configurations
Optional Configurations
Device Support
Buffer Support
Establishing RDMA Connections
Exporting and Connecting RDMA
Connecting Using RDMA CM Connection Flow
Using Bridge Functions to Accept CM Connection
Execution Phase
Tasks
Receive Task
Send Task
Send With Immediate Task
Read Task
Write Task
Write With Immediate Task
Atomic Compare and Swap Task
Atomic Fetch and Add Task
Get Remote Sync Event Task
Set Remote Sync Event Task
Add Remote Sync Event Task
Events
State Machine
Idle
Starting
Running
Stopping
Alternative Datapath Options
DPA Datapath
GPU Datapath
DOCA RDMA Samples
Running the Samples
Samples
RDMA Read
RDMA Write

RDMA Write Immediate

RDMA Send and Receive

RDMA Send and Receive with Immediate

RDMA Remote Sync Event
This guide provides an overview and configuration instructions for the DOCA RDMA API.

**Introduction**

**Note**

This library is currently supported at beta level only.

DOCA RDMA enables direct access to the memory of remote machines, without interrupting the processing of their CPUs or operating systems. Avoiding CPU interruptions reduces context switching for I/O operations, leading to lower latency and higher bandwidth compared to traditional network communication methods.

DOCA RDMA library provides an API to execute the various RDMA operations.

This document is intended for software developers wishing to improve their applications by utilizing RDMA operations.

**Warning**

RDMA operations should be executed over a secure channel in a production deployment, given the sensitivity that arises from the nature of the protocol.

**Prerequisites**

This library follows the architecture of a DOCA Core Context, it is recommended read the following sections before proceeding:

- [DOCA Core Execution Model](#)
- [DOCA Core Device](#)
Environment

DOCA RDMA-based applications can run either on the host machine or on the NVIDIA® BlueField® networking platform (DPU or SuperNIC).

Architecture

DOCA RDMA is a DOCA Context as defined by DOCA Core. See NVIDIA DOCA Core Context for more information.

DOCA RDMA consists of two connected sides, passing data between one another. This includes the option for one side to access the remote side's memory if the granted permissions allow it.

The connection between the two sides can either be based on InfiniBand (IB) or based on Ethernet using RoCE. Currently, only reliable connection (RC) transport type is supported.

DOCA RDMA leverages the Core architecture to expose asynchronous tasks/events that are offloaded to hardware.

The supported operations that may be executed between the two sides, using DOCA RDMA, are:

- Receive
- Send
- Send with immediate
- Write
- Write with immediate
- Read
- Atomic compare and swap
- Atomic fetch and add
- Get remote DOCA Sync Event
- Set remote DOCA Sync Event
- Add remote DOCA Sync Event

**Objects**

**Device**

The RDMA library requires a DOCA device to operate. This device is used to utilize the connection between the peers in RDMA, access memory, and perform the different operations.

**Note**

The device must stay valid until the RDMA instance is destroyed.

**Memory Map**

Executing any DOCA RDMA operation in which data is passed between the peers requires creating a memory map (mmap) on each side.

- The mmap's permissions must include the relevant RDMA permission, according to the required RDMA operations. Tasks fail in case of insufficient permissions.

**Info**

Refer to section "Permissions" for more information.
• To allow the peer to execute RDMA operations, the mmap must be exported, using `doca_mmap_export_rdma()`, and passed to the peer (i.e., the side requesting the RDMA operation) where the remote mmap is created and used to access the memory.

**Buffer Inventory and Buffers**

Executing any DOCA RDMA operation, in which data is passed between the peers, requires using buffers, and thus requires a buffer inventory as well.

Each operation calls for a different set-up for the buffers in use, this is explicitly explained in the "Tasks" section.

**Configuration Phase**

To start using the library you need to first go through a configuration phase as described in [DOCA Core Context Configuration Phase](#).

This section describes how to configure and start the context, to allow execution of tasks and retrieval of events.

**Configurations**

The context can be configured to match the application use case.

**Mandatory Configurations**

These configurations are mandatory and must be set by the application before attempting to start the context:

**Task Configurations**

At least one task/event type must be configured. See configuration of [Tasks](#) and/or [Events](#).
Permissions

Different tasks require different permission to be set for both the RDMA and the mmap in use.

The following table summarizes the necessary RDMA and mmap permissions for each RDMA operation:

<table>
<thead>
<tr>
<th>DOCA RDMA task Types</th>
<th>Minimal Permissions</th>
<th>Should Export MMAP? (a)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The Side Submitting the Task</td>
<td>The Peer</td>
</tr>
<tr>
<td>Read Get Remote Sync Event</td>
<td>RDMA: Local read write</td>
<td>RDMA read</td>
</tr>
<tr>
<td>Write Write with Immediate Set Remote Sync Event</td>
<td>RDMA: Local read write</td>
<td>RDMA write</td>
</tr>
<tr>
<td>Atomic Compare and Swap Atomic Fetch and Add Atomic Fetch and Add Remote Sync Event</td>
<td>RDMA: Local read write</td>
<td>RDMA atomic</td>
</tr>
<tr>
<td>Send Send with Immediate</td>
<td>RDMA: Local read write</td>
<td>-</td>
</tr>
<tr>
<td>Receive</td>
<td>Depending on the received task</td>
<td>Local read write</td>
</tr>
</tbody>
</table>
Optional Configurations

If these configurations are not set, a default value is used.

Users may edit the default properties of the RDMA instance using the `doca_rdma_set_<property>()`. The user may also query the default/set properties using `doca_rdma_cap_get_<property>(struct doca_rdma *, ...)` functions.

Info

The number of tasks that can be submitted in bulk is dependent on the properties `max_send_buf_list_len` and `send_queue_size`.

Refer to `Library Capability` for querying valid property values when configuring the library context.

Device Support

DOCA RDMA requires a device to operate. For picking a device, see `DOCA Core Device Discovery`.
As device capabilities may change in the future, it is recommended to query each doca_devinfo for its capabilities relevant to RDMA operations, using doca_rdma_cap_*(struct doca_devinfo *, ...) functions, and check whether the device is suitable for the required RDMA task types, using doca_rdma_task_<task_type>_is_supported().

BlueField-2 and higher devices are supported:

- On the host, any doca_dev is supported
- On the BlueField Platform, applications must provide the library with SFs as a doca_dev. See NVIDIA OpenvSwitch Acceleration (OVS in DOCA) and BlueField DPU Scalable Function to see how to create SFs and connect them to the appropriate ports.

### Info

An exception to this is when running RDMA on the DPA datapath, which currently only supports PFs.

### Buffer Support

The DOCA RDMA library utilizes different buffer types, depending on the task and the buffer's purpose:

- Local mmap buffer
- Mmap from RDMA export buffer
- Mmap from PCIe export buffers

### Info
Establishing RDMA Connections

To establish the communication between the peers and allow the execution of different DOCA RDMA tasks, the RDMA instances must be connected.

There are two methods to establish RDMA connections as detailed in the following subsections.

Exporting and Connecting RDMA

Connecting the RDMA instances can be done by exporting each RDMA instance to the remote side to a blob by using `doca_rdma_export()`, transferring the blob to the opposite side, out-of-band (OOB), and providing it as input to the `doca_rdma_connect()` function on that side.

Note

This should be executed after `doca_ctx_start()` is called.

Info

Refer to section "State Machine" for more information.
All in all, the configuration flow should be as presented in the following image:

Warning
Connecting Using RDMA CM Connection Flow

Note

This connection method is not currently available for DPA/GPU data paths.

The RDMA CM (communication manager) flow uses the server/client scheme where one of the RDMA instances acts as a server for the second RDMA instance (client). The process for both RDMA instances is non-blocking, event driven, and governed by the progress engine (PE). The connection process is reported to both instances by callbacks which should be set with `doca_rdma_set_connection_state_callbacks()`.

There are four state callbacks:

- **Connection request callback** – This function is called by `doca_pe_progress()` when a connection request is received by an RDMA instance acting as server

- **Connection established callback** – This function is called by `doca_pe_progress()` when a connection is successfully established between server/client RDMA instances

- **Connection failure callback** – This function is called by `doca_pe_progress()` when a connection fails to be established between server/client RDMA instances

- **Connection disconnection callback** – This function is called by `doca_pe_progress()` when a connection disconnects either server/client RDMA instances

A typical connection flow would be as follows:

The exported data contains sensitive information. Make sure to pass this data through a secure channel!
1. Prior to initiating a connection, the RDMA instance acting as server (i.e., RDMA server) must start active listening for a connection from a remote RDMA peer (using RDMA CM) to a specific port using `doca_rdma_start_listen_to_port()`. An RDMA server can stop listening for a connection from a remote RDMA peer (using RDMA CM) by using `doca_rdma_stop_listen_to_port()`.

2. The RDMA CM instance acting as client (i.e., RDMA client) can now perform an RDMA connection to the RDMA server. As first step it must create an address object by using `doca_rdma_addr_create()`. The parameters to this function correspond to the RDMA server details required to perform a connection. This object can be destroyed by using `doca_rdma_addr_destroy()`, and retrieve it from a connection with `doca_rdma_connection_get_addr()`.

3. The RDMA client can set the connection user data to include in each connection using `doca_rdma_connection_set_user_data()`, and retrieve it from a connection using `doca_rdma_connection_get_user_data()`.

4. The RDMA client can now perform a connection to the RDMA server using `doca_rdma_connect_to_addr()`. Depending on the network topology and configuration, the connection can be established with IPv4, IPv6, or GID.

5. The RDMA server receives a notification with a connection request through the previously set connection request callback function. The RDMA server can decide to accept the connection with `doca_rdma_connection_accept()` or reject the connection with `doca_rdma_connection_reject()`.

   - If the RDMA server rejects the connection or the connection cannot be successfully established, the RDMA server and RDMA client receive a notification through the connection failure callback function.
   
   - If the RDMA server accepts the connection and the connection can be successfully established, the RDMA server and RDMA client receive a notification through the connection established callback function.

6. After the RDMA operation is complete, either side can perform the disconnection process using `doca_rdma_connection_disconnect()`. The RDMA instance that did not initiate the disconnection process receives a notification through the disconnection callback function.
Using Bridge Functions to Accept CM Connection

DOCA RDMA offers connection functionality for user RDMA CM applications acting as a server that maintains a CM event channel and performs the listen process itself (i.e., not using DOCA RDMA connection flow functions).

The functionality must be executed as follows:

1. Server user application, using RDMA CM, must create an RDMA CM event channel, start active listening for a connection from a remote RDMA peer, and monitor the created CM event channel. These functions are performed without the use of the DOCA RDMA connection flow functions explained in section "Connecting Using RDMA CM Connection Flow".

Note

The connection process involves resolving the RDMA server connection address. This process is limited to 5 seconds by default, but it can be set using `doca_rdma_set_connection_request_timeout()` and retrieved using `doca_rdma_get_connection_request_timeout()`.
2. Once the server user application received a connection request from a remote RDMA peer acting as client (using RDMA CM), it can call `doca_rdma_bridge_accept()`. This method acts as a bridge to accept a connection request from an application that performs the listen process by itself. The previously explained `doca_rdma_connection_accept()` cannot be used for this connection step as the user application needs to provide the RDMA CM id to accept the connection.

3. After the server side calls `doca_rdma_bridge_accept()` and confirms the client connection is successfully established, it should call `doca_rdma_bridge_established()` to finish the connection process from the server side. Only after a connection is established can DOCA RDMA tasks be allocated and submitted.

**Step 1:** Initiate the RDMA instance, server user application creates CM event channel and start listening for a remote RDMA peer connection request

**RDMA Server**

```c
doca_rdma_create();
doca_rdma_as_ctxt();
(set properties)
...
doca_ctxt_start();
```

(Create and monitor RDMA CM event channel and start listening)
(Wait for connection request)

**Step 2:** Server receives connection request from a remote RDMA peer

**RDMA Server**

```c
(connection request)
doca_rdma_bridge_accept();
```

**Step 3:** Server confirms connection establish and finalize bridge connection process

**RDMA Server**

```c
(confirm connection established with peer)
doca_rdma_bridge_established();
```

(Finish configuration and start executing RDMA operations)

---

**Execution Phase**
This section describes execution on CPU using DOCA Core PE. For additional execution environments refer to section "Alternative Datapath Options".

**Tasks**

DOCA RDMA exposes asynchronous tasks that leverage the DPU hardware according to the DOCA Core architecture. See DOCA Core Task.

**Note**

Most DOCA RDMA operations are not atomic and therefore it is imperative that the application handle synchronization appropriately. Moreover, successful completion of a write task, with or without immediate, does not guarantee data has been fully written to the remote address.

**Note**

All buffers used in DOCA RDMA tasks must remain valid until the task result is retrieved.

**Receive Task**

This task should be submitted prior to an expected submission of a send/send with immediate/write with immediate task on the remote side.

**Task Configuration**

<table>
<thead>
<tr>
<th>Description</th>
<th>API to Set the Configuration</th>
<th>API to Query Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable the task</td>
<td>doca_rdma_task_receive_set_conf</td>
<td>doca_rdma_cap_task_receive_is_supported</td>
</tr>
<tr>
<td>Number of tasks</td>
<td>doca_rdma_task_receive_set_conf</td>
<td>–</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Notes</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| Destination buffer               | Buffer pointing to a local memory address. The data is written to the buffer upon successful completion of the task. | • Linked list buffers are supported  
• The given destination buffer/list of buffers (given in dst_buf) must have a total length sufficient for the expected message size or the task would fail  
• The destination buffer is not mandatory and may be NULL when the requested DOCA RDMA task on the remote side is "write with immediate" or when the remote side is sending an empty message, with or without immediate (these tasks are presented later on in the "Tasks" section)  
• For the DOCA RDMA receive task, the length of each buffer is considered as the length from the end of the data section until the end of the buffer, as this is the available memory that can be written to in each buffer. The data length is increased in each buffer if data is written to it once the task is successfully completed. |

**Task Input**

Common input as described in *DOCA Core Task*.

**Task Output**

Common output as described in *DOCA Core Task*. 

---

**Table:**

<table>
<thead>
<tr>
<th>Description</th>
<th>API to Set the Configuration</th>
<th>API to Query Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination buffer list length</td>
<td>doca_rdma_task_receive_set_dst_buf_list_len</td>
<td>doca_rdma_cap_task_receive_get_max_dst_buf_list_len</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Notes</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>Result length</td>
<td>The length of data received by the task</td>
<td>Valid only on successful completion of the task</td>
</tr>
<tr>
<td>Result opcode</td>
<td>The opcode of the operation executed by the peer and received by the task</td>
<td>Valid only after task completion, irrespective of success</td>
</tr>
</tbody>
</table>
| Result immediate data | The immediate data received by the task                        | • Valid only on successful completion of the task  
                        |                                                               | • Valid only when an immediate value was received (i.e. when the result opcode is DOCA_RDMA_OPCODE_RECV_SEND_WITH_IMM or DOCA_RDMA_OPCODE_RECV_WRITE_WITH_IMM) – may be retrieved using doca_rdma_task_receive_get_result_opcode() |

**Task Completion Success**

After the task completes successfully, the following happens:

- The received data is copied to the tail segment extending the original data segment
- The data length is increased by the received data length

**Task Completion Failure**

If the task fails midway:

- If a fatal error occurs, the context is stopped, and the task should be freed by the user
- If a non-fatal error occurs, the task status is updated. Some buffers may be updated and some may remain unchanged.
Task Limitations

- The operation is not atomic and therefore it is imperative that the application handle synchronization appropriately.
- The destination buffer must remain valid until task is completed.
- The total length of the message must not exceed the `max_message_size` device capability.
- The buffer list length must not exceed the `dst_buf_list_len` property of the DOCA RDMA receive task.
- Other limitations are described in DOCA Core Task.

Send Task

This task should be submitted to transfer a message to the remote side, and while the remote side is expecting a message and had submitted a receive task beforehand.

Task Configuration

<table>
<thead>
<tr>
<th>Description</th>
<th>API to Set the Configuration</th>
<th>API to Query Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable the task</td>
<td><code>doca_rdma_task_send_set_conf</code></td>
<td><code>doca_rdma_cap_task_send_is_supported</code></td>
</tr>
<tr>
<td>Number of tasks</td>
<td><code>doca_rdma_task_send_set_conf</code></td>
<td>–</td>
</tr>
<tr>
<td>Source buffer list length</td>
<td><code>doca_rdma_set_max_send_buf_list_len</code></td>
<td><code>doca_rdma_cap_get_max_send_buf_list_len</code></td>
</tr>
</tbody>
</table>

1. This configuration affects other tasks as well.

Task Input
Common input as described in DOCA Core Task.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
</table>
| Source buffer      | Buffer pointing to a local memory address and holds the data to be sent to the remote peer | • Linked list buffers are supported  
• The total length of the given source buffer/list of buffers (in src_buf) may not exceed the expected message size on the remote side or the task fails  
• The source buffer is not mandatory and may be NULL when wishing to send an empty message  
• For the DOCA RDMA send task, the length of each buffer is considered as its data length |

**Task Output**

Common output as described in DOCA Core Task.

**Task Completion Success**

After the task completes successfully, the following happens:

- On successful completion of the task, the data in the source buffer will be sent to the remote side.

- It doesn't indicate that the data is received by the remote side.

**Task Completion Failure**

If the task fails midway:

- If a fatal error occurs, the context is stopped, and the task should be freed by the user
If a non-fatal error occurs, the task status is updated

**Task Limitations**

- The operation is not atomic. Therefore, it is imperative for the application to handle synchronization appropriately.
- The source buffer must remain valid until the task completes.
- The total length of the message must not exceed the `max_message_size` device capability.
- The buffer list length must not exceed the `max_send_buf_list_len` property of the DOCA RDMA instance.
- Other limitations are described in [DOCA Core Task](#).

**Send With Immediate Task**

This task should be submitted to transfer a message to the remote side with immediate data (a 32-bit value sent to the remote side, out-of-band), and while the remote side is expecting a message and had submitted a receive task beforehand.

**Task Configuration**

<table>
<thead>
<tr>
<th>Description</th>
<th>API to Set the Configuration</th>
<th>API to Query Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable the task</td>
<td><code>doca rdma_task_send_imm_set_conf</code></td>
<td><code>doca rdma_cap_task_send_imm_is_supported</code></td>
</tr>
<tr>
<td>Number of tasks</td>
<td><code>doca rdma_task_send_imm_set_conf</code></td>
<td><code>--</code></td>
</tr>
<tr>
<td>Source buffer list length</td>
<td><code>doca rdma_set_max_send_buf_list_len</code></td>
<td><code>doca rdma_cap_get_max_send_buf_list_len</code></td>
</tr>
</tbody>
</table>

1. This configuration affects other tasks as well.
**Task Input**

Common input as described in [DOCA Core Task](#).

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source buffer</td>
<td>Buffer pointing to a local memory address and holding the data to be sent to the remote peer</td>
<td>- Linked list buffers are supported.&lt;br&gt;- The total length of the given source buffer/list of buffers (in <code>src_buf</code>) may not exceed the expected message size on the remote side or the task fails.&lt;br&gt;- The source buffer is not mandatory and may be NULL when wishing to send an empty message (may be relevant when wishing to keep a connection alive)&lt;br&gt;- For the DOCA RDMA send task, the length of each buffer is considered as its data length</td>
</tr>
<tr>
<td>Immediate data</td>
<td>32-bit value sent to the remote side, out-of-band</td>
<td>- The <code>immediate_data</code> field should be in Big-Endian format. This value is received by the remote side only once a receive task is completed successfully.</td>
</tr>
</tbody>
</table>

**Task Output**

Common output as described in [DOCA Core Task](#).

**Task Completion Success**

After the task completes successfully, the following happens:

- The data in the source buffer is sent to the remote side
- It does not indicate that the data is received by the remote side
Task Completion Failure

If the task fails midway:

- If a fatal error occurs, the context is stopped, and the task should be freed by the user.
- If a non-fatal error occurs, the task status is updated.

Task Limitations

- The operation is not atomic. Therefore, it is imperative for the application to handle synchronization appropriately.
- The source buffer must remain valid until the task completes.
- The total length of the message must not exceed the `max_message_size` device capability.
- The buffer list length must not exceed the `max_send_buf_list_len` property of the DOCA RDMA instance.
- Other limitations are described in DOCA Core Task.

Read Task

This task should be submitted when wishing to read data from remote memory (i.e., the memory on the remote side of the connection).

Task Configuration

<table>
<thead>
<tr>
<th>Description</th>
<th>API to Set the Configuration</th>
<th>API to Query Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable the task</td>
<td><code>doca_rdma_task_read_set_conf</code></td>
<td><code>doca_rdma_cap_task_read_is_supported</code></td>
</tr>
</tbody>
</table>
Description | API to Set the Configuration | API to Query Support
---|---|---
Number of tasks | doca_rdma_task_read_set_conf | -
Destination buffer list length | doca_rdma_set_max_send_buf_list_len | doca_rdma_cap_get_max_send_buf_list_len

1. This configuration affects other tasks as well.

**Task Input**

Common input as described in **DOCA Core Task**.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
</table>
| Source buffer | Points to a remote memory address and holds the data to be read | • Linked list buffers are not supported  
• The source buffer (src_buf) is not mandatory and may be NULL when wishing to read zero bytes (may be relevant when wishing to keep a connection alive)  
• The data is read only from the data section of the source buffer  
• The length of the source buffer is considered its data length. The length of data read from the source buffer depends on its data length yet can not exceed the total length of the given destination buffer/list of buffers. That is, the actual length read depends on the minimal length between the source and destination. |
| Destination buffer | Points to a local memory address. The data is written to the buffer upon successful completion of the task | • Linked list buffers are supported  
• The length of each destination buffer is considered as the length from the end of the data section until the end of the buffer, as this is the available memory that can be written to in each buffer  
• May be NULL if the source buffer has been set to NULL |
### Task Output

Common output as described in **DOCA Core Task**.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result length</td>
<td>The length of data read by the task</td>
<td>Valid only on successful completion of the task</td>
</tr>
</tbody>
</table>

### Task Completion Success

After the task completes successfully, the following happens:

- The read data is appended after the data section in the destination buffer, as it was prior to the task submission
- The data length is increased by the read data length

### Task Completion Failure

If the task fails midway:

- If a fatal error occurs, the context is stopped, and the task should be freed by the user
- If a non-fatal error occurs, the task status is updated. Some destination buffers may be updated and some may remain unchanged.

### Task Limitations

- The operation is not atomic. Therefore, it is imperative for the application to handle synchronization appropriately.
- The task buffers must remain valid until task is completed
The given source buffer length must not exceed the `max_message_size` device capability.

The destination buffer list length must not exceed the `max_send_buf_list_len` property of the DOCA RDMA instance.

Other limitations are described in DOCA Core Task.

Write Task

This task should be submitted when wishing to write data to remote memory (i.e., the memory on the remote side of the connection).

Task Configuration

<table>
<thead>
<tr>
<th>Description</th>
<th>API to Set the Configuration</th>
<th>API to Query Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable the task</td>
<td><code>doca_rdma_task_write_set_conf</code></td>
<td><code>doca_rdma_cap_task_write_is_supported</code></td>
</tr>
<tr>
<td>Number of tasks</td>
<td><code>doca_rdma_task_write_set_conf</code></td>
<td><code>-</code></td>
</tr>
<tr>
<td>Source buffer list length</td>
<td><code>doca_rdma_set_max_send_buf_list_len</code></td>
<td><code>doca_rdma_cap_get_max_send_buf_list_len</code></td>
</tr>
</tbody>
</table>

1. This configuration affects other tasks as well.

Task Input

Common input as described in DOCA Core Task.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
</table>
| Source buffer         | Buffer pointing to a local memory address and holding the data to be written to the remote peer. | • Linked list buffers are supported  
• The source buffer should point to a local memory address from which the data should be read. The data is read only from the data section of the source buffer. |
### Task Output

Common output as described in [DOCA Core Task](#).

### Task Completion Success

After the task completes successfully, the following happens:

- The written data is appended after the data section in the destination buffer, as it was prior to the task submission.

- The data length is increased by the written data length.
Task Completion Failure

If the task fails midway:

- If a fatal error occurs, the context is stopped, and the task should be freed by the user
- If a non-fatal error occurs, the task status is updated. Some destination buffers may be updated and some may remain unchanged.

Task Limitations

- The operation is not atomic. Therefore, it is imperative for the application to handle synchronization appropriately.
- The task buffers must remain valid until task is completed
- The total length of the given source buffer/list of buffers must be not exceed the max_message_size device capability
- The source buffer list length must not exceed the max_send_buf_list_len property of the DOCA RDMA instance
- Other limitations are described in DOCA Core Task

Write With Immediate Task

This task should be submitted when wishing to write data to remote memory (i.e., the memory on the remote side of the connection).

Task Configuration
<table>
<thead>
<tr>
<th>Description</th>
<th>API to Set the Configuration</th>
<th>API to Query Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable the task</td>
<td><code>doca_rdma_task_write_imm_set_conf</code></td>
<td><code>doca_rdma_cap_task_write_imm_is_supported</code></td>
</tr>
<tr>
<td>Number of tasks</td>
<td><code>doca_rdma_task_write_imm_set_conf</code></td>
<td>-</td>
</tr>
<tr>
<td>Source buffer list length</td>
<td><code>doca_rdma_set_max_send_buf_list_length</code></td>
<td><code>doca_rdma_cap_get_max_send_buf_list_length</code></td>
</tr>
</tbody>
</table>

1. This configuration affects other tasks as well.

**Task Input**

Common input as described in DOCA Core Task.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
</table>
| Source buffer | Buffer pointing to a local memory address and holding the data to be written to the remote peer | • Linked list buffers are supported  
• The source buffer should point to a local memory address from which the data should be read. The data is read only from the data section of the source buffer.  
• The source buffer (`src_buf`) is not mandatory and may be NULL when wishing to write zero bytes  
• The length of the buffer is considered as its data length |
| Destination buffer | Points to a remote memory address. The data is written to the buffer upon successful completion of the task. | • Linked list buffers are not supported  
• The destination buffer (`dst_buf`) should point to a remote memory address  
• The length of the buffer is considered as its data length  
• The length of the destination buffer is considered as the length from the end of the data section until the end of the buffer, as this is the available memory that can be written to |
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immediate data</td>
<td>32-bit value sent to the remote side, out-of-band</td>
<td>• Should be in a Big-Endian format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Value is received by the remote side only once a receive task completes successfully</td>
</tr>
</tbody>
</table>

**Task Output**

Common output as described in [DOCA Core Task](#).

**Task Completion Success**

A write with immediate task succeeds only if the remote side is expecting the immediate and had submitted a receive task beforehand.

After the task completes successfully, the following happens:

- The written data is appended after the data section in the destination buffer, as it was prior to the task submission
- The data length is increased by the written data length.

**Task Completion Failure**

If the task fails midway:

- If a fatal error occurs, the context is stopped, and the task should be freed by the user
- If a non-fatal error occurs, the task status is updated. Some destination buffers may be updated and some may remain unchanged.

**Task Limitations**

- The operation is not atomic. Therefore, it is imperative for the application to handle synchronization appropriately.

- The tasks buffers must remain valid until task is completed.

- The total length of the given source buffer/list of buffers must be not exceed the `max_message_size` device capability.

- The source buffer list length must not exceed the `max_send_buf_list_len` property of the DOCA RDMA instance.

- Other limitations are described in DOCA Core Task.

**Atomic Compare and Swap Task**

This task should be submitted when wishing to execute an 8-byte atomic read-modify-write operation on the remote memory (i.e., the memory on the remote side of the connection), in which the remote value is retrieved and updated if it is equal to a given value.

**Task Configuration**

<table>
<thead>
<tr>
<th>Description</th>
<th>API to Set the Configuration</th>
<th>API to Query Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable the task</td>
<td><code>doca_rdma_task_atomic_cmp_swpswp_set_conf</code></td>
<td><code>doca_rdma_cap_task_atomic_cmp_swpswp_is_supported</code></td>
</tr>
<tr>
<td>Number of tasks</td>
<td><code>doca_rdma_task_atomic_cmp_swpswp_set_conf</code></td>
<td>-</td>
</tr>
</tbody>
</table>
Task Input

Common input as described in DOCA Core Task.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
</table>
| Destination buffer | Buffer pointing to a remote memory address                                  | • Linked list buffers are not supported  
• The destination buffer's data section must begin in a memory address aligned to 8-bytes  
• Only the first 8-bytes following the data address are considered for atomic operations |
| Compare data     | 64-bit value to be compared with the value in the destination buffer         |                                                                                                                                         |
| Swap data        | 64-bit value to be swapped with the value in the destination buffer          | • The value in the destination buffer is only swapped if the compared data value is equal to the value in the destination buffer. Otherwise, the destination buffer remains unchanged. |
| Result buffer    | Buffer pointing to a local memory address. The original value of the destination buffer (before executing the atomic operation) is written to the buffer upon success. | • Linked list buffers are not supported  
• The result is written to the first 8-bytes following the data address |

Task Output

Common output as described in DOCA Core Task.

Task Completion Success
After the task completes successfully, the following happens:

- If the compared values are equal, the value in the destination is swapped with the 64-bit value in the task's swap data field (swap_data)

- If the compared values are not equal, the value in the destination value remains unchanged

- The original value of the destination buffer (before executing the atomic operation) is written to the result buffer

**Task Completion Failure**

If the task fails midway:

- The context is stopped, and the task should be freed by the user

**Task Limitations**

- Task buffers must remain valid until task is completed

- Other limitations are described in **DOCA Core Task**

**Atomic Fetch and Add Task**

This task should be submitted when wishing to execute an 8-byte atomic read-modify-write operation on the remote memory (i.e., the memory on the remote side of the connection), in which the remote value is retrieved and increased by a given value.

**Task Configuration**

<table>
<thead>
<tr>
<th>Description</th>
<th>API to Set the Configuration</th>
<th>API to Query Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable the task</td>
<td><code>doca_rdma_task_atomic_fetch_add_set_conf</code></td>
<td><code>doca_rdma_cap_task_atomic_fetch_add_is_supported</code></td>
</tr>
<tr>
<td>Description</td>
<td>API to Set the Configuration</td>
<td>API to Query Support</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------------------------------------------------------------------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>Number of tasks</td>
<td>doca_rdma_task_atomic_fetch_add_set_conf</td>
<td>-</td>
</tr>
</tbody>
</table>

**Task Input**

Common input as described in **DOCA Core Task**.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination buffer</td>
<td>Buffer that points to a remote memory address</td>
<td>- Linked list buffers are not supported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The destination buffer's data section must begin in a memory address aligned to 8-bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Only the first 8-bytes following the data address are considered for atomic operations</td>
</tr>
<tr>
<td>Add data</td>
<td>64-bit value to be added to the value in the destination buffer</td>
<td></td>
</tr>
<tr>
<td>Result buffer</td>
<td>Buffer pointing to a local memory address. The original value of the destination buffer (before executing the atomic operation) is written to the buffer upon success.</td>
<td>- Linked list buffers are not supported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The result is written to the first 8-bytes following the data address</td>
</tr>
</tbody>
</table>

**Task Output**

Common output as described in **DOCA Core Task**.

**Task Completion Success**
After the task completes successfully, the following happens:

- The value in the destination is increased by the 64-bit value in the task's add data field
- The original value of the destination buffer (before executing the atomic operation) is written to the result buffer

**Task Completion Failure**

If the task fails midway:

- The context is stopped, and the task should be freed by the user

**Task Limitations**

- Task buffers must remain valid until task is completed
- Other limitations are described in [DOCA Core Task](#)

**Get Remote Sync Event Task**

This task should be submitted when wishing to get the value of a remote sync event.

**Task Configuration**

<table>
<thead>
<tr>
<th>Description</th>
<th>API to Set the Configuration</th>
<th>API to Query Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable the task</td>
<td><code>doca_rdma_task_remote_net_sync_event_get_set_conf</code></td>
<td><code>doca_rdma_cap_task_remote_net_sync_event_get_is_supported</code></td>
</tr>
<tr>
<td>Number of tasks</td>
<td><code>doca_rdma_task_remote_net_sync_event_get_set_conf</code></td>
<td>-</td>
</tr>
<tr>
<td>Destination buffer list length</td>
<td><code>doca_rdma_set_max_send_buf_list_len</code></td>
<td><code>doca_rdma_cap_get_max_send_buf_list_len</code></td>
</tr>
</tbody>
</table>
1. This configuration affects other tasks as well.

**Task Input**

Common input as described in DOCA Core Task.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync Event</td>
<td>The remote DOCA Sync Event to get its value</td>
<td></td>
</tr>
</tbody>
</table>
| Destination buffer | Points to a local memory address. The Sync Event value is written to the buffer upon successful completion of the task. | • Linked list buffers are supported  
• The length of the each buffer is considered as the length from the end of the data section until the end of the buffer, as this is the available memory that can be written to in each buffer |

**Task Output**

Common output as described in DOCA Core Task.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result length</td>
<td>The length of data received by the task</td>
<td>Valid only on successful completion of the task</td>
</tr>
</tbody>
</table>

**Task Completion Success**

After the task completes successfully, the following happens:

- The remote Sync Event value is appended after the data section in the destination buffer, as it was prior to the task submission
The data length is increased by the retrieved data length

**Task Completion Failure**

If the task fails midway:

- If a fatal error occurs, the context is stopped, and the task should be freed by the user.
- If a non-fatal error occurs, the task status is updated. Some destination buffers may be updated and some may remain unchanged.

**Task Limitations**

- The operation is not atomic. Therefore, it is imperative for the application to handle synchronization appropriately.
- The destination buffer must remain valid until the task is completed.
- The destination buffer list length must not exceed the `max_send_buf_list_len` property of the DOCA RDMA instance.
- Other limitations are described in [DOCA Core Task](#).

**Set Remote Sync Event Task**

This task should be submitted when wishing to set a remote sync event to a given value.

**Task Configuration**

<table>
<thead>
<tr>
<th>Description</th>
<th>API to Set the Configuration</th>
<th>API to Query Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable the task</td>
<td><code>doca_rdma_task_remote_net_sync_event_notify_set_set_conf</code></td>
<td><code>doca_rdma_cap_task_remote_net_sync_event_notify_set_is_supported</code></td>
</tr>
<tr>
<td>Description</td>
<td>API to Set the Configuration</td>
<td>API to Query Support</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------</td>
<td>-------------------------------------------------------------</td>
</tr>
<tr>
<td>Number of tasks</td>
<td><code>doca_rdma_task_remote_net_sync_event_notify_set_set_conf</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source buffer list length</td>
<td><code>doca_rdma_set_max_send_buf_list_len 7</code></td>
<td><code>doca_rdma_cap_get_max_send_buf_list_len</code></td>
</tr>
</tbody>
</table>

1. This configuration affects other tasks as well.

**Task Input**

Common input as described in **DOCA Core Task**.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source buffer</td>
<td>Points to a local memory address from which the Sync Event should be retrieved</td>
<td>• Linked list buffers are supported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The data is retrieved only from the buffer data section, until 8-bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The length of the source buffer is considered its data length. The length of data retrieved from the source buffer will not exceed the Sync Event value length (8-bytes). Thus, the actual length retrieved depends on the minimal length between the source buffer and Sync Event value length.</td>
</tr>
<tr>
<td>Sync Event</td>
<td>The remote DOCA Sync Event to get its value</td>
<td></td>
</tr>
</tbody>
</table>

**Task Output**

Common output as described in **DOCA Core Task**.
**Task Completion Success**

After the task completes successfully, the following happens:

- The remote sync event value is set to the data in the source buffer

**Task Completion Failure**

If the task fails midway:

- If a fatal error occurs, the context is stopped, and the task should be freed by the user
- If a non-fatal error occurs, the task status is updated, and the Sync Event value is undefined

**Task Limitations**

- The operation is not atomic. Therefore, it is imperative for the application to handle synchronization appropriately.
- The source buffer must remain valid until the task completes
- The source buffer list length must not exceed the `max_send_buf_list_len` property of the DOCA RDMA instance
- Other limitations are described in DOCA Core Task

**Add Remote Sync Event Task**

This task should be submitted when wishing to atomically increase a remote sync event by a given value.

**Task Configuration**
<table>
<thead>
<tr>
<th>Description</th>
<th>API to Set the Configuration</th>
<th>API to Query Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable the task</td>
<td><code>doca_rdma_task_remote_net_sync_event_notify_add_set_conf</code></td>
<td><code>doca_rdma_cap_task_remote_net_sync_event_notify_add_is_supported</code></td>
</tr>
<tr>
<td>Number of tasks</td>
<td><code>doca_rdma_task_remote_net_sync_event_notify_add_set_conf</code></td>
<td>-</td>
</tr>
</tbody>
</table>

**Task Input**

Common input as described in **DOCA Core Task**.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync event</td>
<td>A remote Sync Event</td>
<td></td>
</tr>
<tr>
<td>Add data</td>
<td>64-bit value that is added to the Sync Event value</td>
<td></td>
</tr>
</tbody>
</table>
| Result buffer | Buffer pointing to a local memory address. The original Sync Event value of the destination buffer (before executing the atomic operation) is written to the buffer upon success. | • Linked list buffers are not supported  
• The result is written to the first 8-bytes following the data address |

**Task Output**

Common output as described in **DOCA Core Task**.

**Task Completion Success**

After the task completes successfully, the following happens:
• The value of the remote sync event is increased by the 64-bit value in the task's add data field

• The original value of the remote sync event (before executing the operation) is written to the result buffer

Task Completion Failure

If the task fails midway:

• The context is stopped, and the task should be freed by the user

Task Limitations

• Result buffer must remain valid until task is completed

• Other limitations are described in DOCA Core Task

Events

DOCA RDMA exposes asynchronous events to notify about changes that happen unexpectedly, according to DOCA Core architecture.

The only event DOCA RDMA exposes is common events as described in DOCA Core Event.

State Machine

The DOCA RDMA library follows the Context state machine as described in DOCA Core Context State Machine.

The following section describes how to move states and what is allowed in each state.
Idle

In this state, it is expected that application either:

- Destroys the context
- Starts the context

Allowed operations:

- Configuring the context according to section "Configurations"
- Starting the context

It is possible to reach this state as follows:

<table>
<thead>
<tr>
<th>Previous State</th>
<th>Transition Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>Create the context</td>
</tr>
<tr>
<td>Running</td>
<td>Call stop after making sure all tasks have been freed</td>
</tr>
<tr>
<td>Stopping</td>
<td>Call progress until all tasks are completed and freed</td>
</tr>
</tbody>
</table>

Starting

This state cannot be reached.

Running

In this state, it is expected that application:

1. Connects the RDMA instances on both peers. Refer to section "Establishing RDMA Connections" for more information.

2. Performs an RDMA instance disconnection process if the connection was established using the RDMA CM flow. Refer to section "Connecting Using RDMA CM Connection Flow" for more information.
3. Performs a new connection of the RDMA instances on both peers after an RDMA instance disconnection process if the connection was established using the RDMA CM flow. Refer to section "Connecting Using RDMA CM Connection Flow" for more information.

4. Accepts and indicates an established RDMA connection if the listening and CM channel monitoring was done by the user application. Refer to section "Connecting Using RDMA CM Connection Flow" for more information.

5. Allocates and submits tasks.

6. Calls progress to complete tasks and/or receive events.

Allowed operations:

- Performing a connection between 2 peers
- Allocating previously configured task
- Submitting an allocated task
- Calling stop

It is possible to reach this state as follows:

<table>
<thead>
<tr>
<th>Previous State</th>
<th>Transition Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>Call start after configuration</td>
</tr>
</tbody>
</table>

**Stopping**

In this state, it is expected that application:

1. Calls progress to complete all inflight tasks (tasks complete with failure)

2. Frees any completed tasks

3. Performs an RDMA instance disconnection process if the connection was established using the RDMA CM flow. Refer to section "Connecting Using RDMA CM Connection Flow" for more information.
Allowed operations:

- Call progress

It is possible to reach this state as follows:

<table>
<thead>
<tr>
<th>Previous State</th>
<th>Transition Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running</td>
<td>Call progress and fatal error occurs</td>
</tr>
<tr>
<td>Running</td>
<td>Call stop without freeing all tasks</td>
</tr>
</tbody>
</table>

**Alternative Datapath Options**

DOCA RDMA allows data path to be run on DPA or GPU.

**DPA Datapath**

DOCA offers the DOCA DPA library which provides a programming model for offloading communication-centric user code to run on the DPA processor on the BlueField DPU. For additional information on the [DOCA DPA library](#).

**Note**

DOCA RDMA on DPA datapath supports local networks only (i.e., cross-network or routing is not supported).

The user can choose to run an RDMA operation on the DPA datapath by configuring the DOCA RDMA context used by the application in the following manner:

1. Obtain DOCA CTX by calling `doca_rdma_as_ctx()`.

2. Set the datapath of the context to DPA by calling `doca_ctx_set_datapath_on_dpa()`. For additional information, refer to [DOCA Core Alternative Data Path](#).
3. Finish context configuration and start the context by calling `doca_ctx_start()`. For additional information, refer to DOCA Context.

After configuring the datapath, the user can obtain a DPA handle for the DOCA RDMA context by calling `doca_rdma_get_dpa_handle()`.

The DPA handle can be used by the DOCA DPA library for datapath operations. For additional information, refer to DOCA DPA Communication Model.

**GPU Datapath**

DOCA offers the DOCA GPUNetIO library which provides a programming model for offloading the orchestration of the communication to a GPU CUDA kernel. For additional information on the DOCA GPUNetIO library.

The user can choose to run an RDMA operation on the GPU datapath by configuring the DOCA RDMA context used by the application in the following manner:

1. Obtain DOCA CTX by calling `doca_rdma_as_ctx()`.

2. Set the datapath of the context to GPU by calling `doca_ctx_set_datapath_on_gpu()`. For additional information, refer to DOCA Core Alternative Data Path.

3. Finish context configuration and start the context by calling `doca_ctx_start()`. For additional information, refer to DOCA Core Context.

After configuring the datapath, the user can obtain a GPU handle for the DOCA RDMA context by calling `doca_rdma_get_gpu_handle()`.

The GPU handle must be passed to a GPU CUDA kernel so the DOCA GPUNetIO CUDA device functions can execute datapath operations. For additional information, refer to DOCA GPUNetIO device functions.

**DOCA RDMA Samples**

These samples illustrate how to use the DOCA RDMA API to execute DOCA RDMA operations.
Running the Samples

1. Refer to the following documents:
   - NVIDIA DOCA Installation Guide for Linux for details on how to install BlueField-related software.
   - NVIDIA DOCA Troubleshooting Guide for any issue you may encounter with the installation, compilation, or execution of DOCA samples.

2. To build a given sample:

   ```
   cd /opt/mellanox/doca/samples/doca_rdma/<sample_name>
   meson /tmp/build
   ninja -C /tmp/build
   ```

3. Sample usage:
   - Common arguments
<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-d, --device</td>
<td>IB device name (optional). If not provided, a random IB device is assigned.</td>
</tr>
<tr>
<td>-ld, --local-descriptor-path</td>
<td>Local descriptor file path that includes the local connection information to be copied to the remote program</td>
</tr>
<tr>
<td>-re, --remote-descriptor-path</td>
<td>Remote descriptor file path that includes the remote connection information to be copied from the remote program</td>
</tr>
<tr>
<td>-m, --mmap-descriptor-path</td>
<td>Remote descriptor file path that includes the remote mmap connection information to be copied from the remote program</td>
</tr>
<tr>
<td>-g, --gid-index</td>
<td>GID index for DOCA RDMA (optional)</td>
</tr>
</tbody>
</table>

- Sample-specific arguments

<table>
<thead>
<tr>
<th>Sample</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDMA Read Responder</td>
<td>-r, --read-string</td>
<td>String to read (optional). If not provided, &quot;Hi DOCA RDMA!&quot; is defined.</td>
</tr>
<tr>
<td>RDMA Send Immediate</td>
<td>-s, --send-string</td>
<td></td>
</tr>
<tr>
<td>RDMA Write Requester</td>
<td>-w, --write-string</td>
<td></td>
</tr>
<tr>
<td>RDMA Write Immediate Requester</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4. For additional information per sample, use the -h option:

   /tmp/build/<sample_name> -h

**Samples**
Each sample presents a connection between two peers, transferring data from one to another, using a different RDMA operation in each sample. For more information on the available RDMA operations, refer to section "Tasks".

Each sample is comprised of two executables, each running on a peer.

The samples can run on either DPU or host, as long as the chosen peers have a connection between them.

Note

Prior to running the samples, ensure that the chosen devices, selected by the device name and the GID index, are set correctly and have a connection between one another. In each sample, it is the user's responsibility to copy the descriptors between the peers.

Most of the samples follow the following main basic steps:

1. Allocating resources:
   
   1. Locating and opening a device. The chosen device is one that supports the tasks relevant for the sample. If the sample requires no task, any device may be chosen.
   
   2. Creating a local MMAP and configuring it (including setting the MMAP memory range and relevant permissions)
   
   3. Creating a DOCA PE
   
   4. Creating an RDMA instance and configuring it (including setting the relevant permissions)
   
   5. Connecting the RDMA context to the PE

2. Sample-specific configurations:
   
   1. Configuring the tasks relevant to the sample, if any. Including:
1. Setting the number of tasks for each task type.

2. Setting callback functions for each task type, with the following logic:
   
   1. Successful completion callback:
      
      1. Verifying the data received from the remote, if any, is valid.
      
      2. Printing the transferred data.
      
      3. Freeing the task and task-specific resources (such as source/destination buffers).
      
      4. If an error occurs in steps a. and b., update the error that was encountered.

   2. Failed completion callback:
      
      1. Update the error that was encountered.

   Note
   
   If the context is not in idle state, only the first error in the flow is saved.

   5. Decreasing the number of remaining tasks and stopping the context once it reaches 0.

   Note
   
   If the context is not in idle state, only the first error in the flow is saved.
2. Freeing the task and task-specific resources (such as source/destination buffers).

3. Decreasing the number of remaining tasks and stopping the context once it reaches 0.

2. Setting a state change callback function, with the following logic:

- Once the context moves to Starting state (can only be reached from Idle), export and connect the RDMA and, in some samples, export the local mmap or the sync event.

   ★ **Note**

   During this step, the user is responsible for copying the descriptors between the two peers.

   ★ **Note**

   The descriptors are to be read and used only by the peer, using the relevant DOCA functions (the descriptors contain encoded data).

- Once the context moves to Running state (can only be reached from Starting state in RDMA samples):

  ★ In some samples, only print a log and wait for the peer, or synchronize events

  ★ In other samples, prepare and submit a task:

  1. If needed, create an mmap from the received exported mmap descriptor, passed from the peer.
2. Request the required buffers from the buffer inventory.

3. Allocate and initiate the required task, together with setting the number of remaining tasks parameter as the task's user data.

4. Submit the task.

   - Once the context moves to Stopping state, print a relevant log.
   - Once the context moves to Idle state:
     1. Print a relevant log.
     2. Send update that the main loop may be stopped.

3. Setting the program's resources as the context user data to be used in callbacks.

4. Creating a buffer inventory and starting it.

5. Starting the context.

![Info]

*Info*

After starting the context, the state change callback function is called by the PE which executes the relevant steps.

![Info]

*Info*

In a successful run, each section is executed in the order they are presented in section 2.b.

6. Progressing the PE until the context returns to Idle state and the main loop may be stopped, either because of a run in which all tasks have been completed, or due to a
fatal error.

7. Cleaning up the resources.

RDMA Read

RDMA Read Requester

This sample illustrates how to read from a remote peer (the responder) using DOCA RDMA.

The sample logic is as presented in the General Sample Steps, with attention to the following:

1. The permissions for the local mmap in this sample are set to local read and write.

2. A read task is configured for this sample.

3. In this sample, data is read from the peer, verified to be valid, and printed in the successful task completion callback.

4. The local mmap is not exported as the peer does not intend to access it.

5. To read from the peer, a remote mmap is created from the peer's exported mmap.

Reference:

- /opt/mellanox/doca/samples/doca_rdma/rdma_read_requester/rdma_read_requester_sample.c
- /opt/mellanox/doca/samples/doca_rdma/rdma_read_requester/rdma_read_requester_main.c
- /opt/mellanox/doca/samples/doca_rdma/rdma_read_requester/meson.build

RDMA Read Responder

This sample illustrates how to set up a remote peer for a DOCA RDMA read request.

The sample logic is as presented in the General Sample Steps, with attention to the following:
1. The permissions for both the local mmap and the RDMA instance in this sample allow for RDMA read.

2. No tasks are configured for this sample, and thus no tasks are prepared and submitted, nor are there task completion callbacks.

3. The local mmap is exported to the remote memory to allow it to be used by the peer for RDMA read.

4. No remote mmap is created as there is no intention to access the remote memory in this sample.

Reference:

- /opt/mellanox/doca/samples/doca_rdma/rdma_read_responder/rdma_read_responder_sample.c
- /opt/mellanox/doca/samples/doca_rdma/rdma_read_responder/rdma_read_responder_main.c
- /opt/mellanox/doca/samples/doca_rdma/rdma_read_responder/meson.build

**RDMA Write**

**RDMA Write Requester**

This sample illustrates how to write to a remote peer (the responder) using DOCA RDMA.

The sample logic is as presented in the General Sample Steps, with attention to the following:

1. The permissions for the local mmap in this sample are set to local read and write.

2. A write task is configured for this sample.

3. In this sample, data is written to the peer and printed in the successful task completion callback.

4. The local mmap is not exported as the peer does not intend to access it.

5. To write to the peer, a remote mmap is created from the peer’s exported mmap.
**RDMA Write Responder**

This sample illustrates how to set up a remote peer for a DOCA RDMA write request.

The sample logic is as presented in the [General Sample Steps](#), with attention to the following:

1. The permissions for both the local mmap and the RDMA instance in this sample allow for RDMA write.

2. No tasks are configured for this sample, and thus no tasks are prepared and submitted, nor are there task completion callbacks. In this sample, the data written to the memory of the responder is printed once the context state is changed to Running, using the state change callback. This is done only after receiving input from the user, indicating that the requester had finished writing.

3. The local mmap is exported to the remote memory to allow it to be used by the peer for RDMA write.

4. No remote mmap is created as there is no intention to access the remote memory in this sample.

Reference:

- `/opt/mellanox/doca/samples/doca_rdma/rdma_write_responder/rdma_write_responder_sample.c`
- `/opt/mellanox/doca/samples/doca_rdma/rdma_write_responder/rdma_write_responder_main.c`
- `/opt/mellanox/doca/samples/doca_rdma/rdma_write_responder/meson.build`


**RDMA Write Immediate**

**RDMA Write Immediate Requester**

This sample illustrates how to write to a remote peer (the responder) using DOCA RDMA along with a 32-bit immediate value which is sent OOB.

The sample logic is as presented in the General Sample Steps, with attention to the following:

1. The permissions for the local mmap in this sample is set to local read and write.
2. A write with immediate task is configured for this sample.
3. In this sample, data is written to the peer and printed in the successful task completion callback.
4. The local mmap is not exported as the peer does not intend to access it.
5. To write to the peer, a remote mmap is created from the peer's exported mmap.

Reference:

- /opt/mellanox/doca/samples/doca_rdma/rdma_writeImmediate_requester/rdma_writeImmediate_req
- /opt/mellanox/doca/samples/doca_rdma/rdma_writeImmediate_requester/rdma_writeImmediate_req
- /opt/mellanox/doca/samples/doca_rdma/rdma_writeImmediate_requester/meson.build

**RDMA Write Immediate Responder**

This sample illustrates how the set up a remote peer for a DOCA RDMA write request whilst receiving a 32-bit immediate value from the peer's OOB.

The sample logic is as presented in the General Sample Steps, with attention to the following:

1. The permissions for both the local mmap and the RDMA instance in this sample allow for RDMA write.
2. A receive task is configured for this sample to retrieve the immediate value. Failing to submit a receive task prior to the write with immediate task results in a fatal failure.

3. In this sample, the successful task completion callback also includes:
   1. Checking the result opcode, to verify that the receive task has completed after receiving a write with immediate request.
   2. Verifying the data written to the memory of the responder is valid and printing it, along with the immediate data received.

4. The local mmap is exported to the remote memory, to allow it to be used by the peer for RDMA write.

5. No remote mmap is created as there is no intention to access the remote memory in this sample.

Reference:

- /opt/mellanox/doca/samples/doca_rdma/rdma_write_immediate_responder/rdma_write_immediate_responder
- /opt/mellanox/doca/samples/doca_rdma/rdma_write_immediate_responder/rdma_write_immediate_responder
- /opt/mellanox/doca/samples/doca_rdma/rdma_write_immediate_responder/meson.build

**RDMA Send and Receive**

**RDMA Send**

This sample illustrates how to send a message to a remote peer using DOCA RDMA.

The sample logic is as presented in the General Sample Steps, with attention to the following:

1. The permissions for the local mmap in this sample is set to local read and write.

2. A send task is configured for this sample.
3. In this sample, the data sent is printed during the task preparation, not in the successful task completion callback.

4. The local mmap is not exported as the peer does not intend to access it.

5. No remote mmap is created as there is no intention to access the remote memory in this sample.

Reference:

- /opt/mellanox/doca/samples/doca_rdma/rdma_send/rdma_send_sample.c
- /opt/mellanox/doca/samples/doca_rdma/rdma_send/rdma_send_main.c
- /opt/mellanox/doca/samples/doca_rdma/rdma_send/meson.build

**RDMA Receive**

This sample illustrates how the remote peer can receive a message sent by the peer (the sender).

The sample logic is as presented in the General Sample Steps, with attention to the following:

1. The permissions for the local mmap in this sample is set to local read and write.

2. A receive task is configured for this sample to retrieve the sent data. Failing to submit a receive task prior to the send task results in a fatal failure.

3. In this sample, data is received from the peer verified to be valid and printed in the successful task completion callback.

4. The local mmap is not exported as the peer does not intend to access it.

5. No remote mmap is created as there is no intention to access the remote memory in this sample.

Reference:

- /opt/mellanox/doca/samples/doca_rdma/rdma_receive/rdma_receive_sample.c
RDMA Send and Receive with Immediate

RDMA Send with Immediate

This sample illustrates how to send a message to a remote peer using DOCA RDMA along with a 32-bit immediate value which is sent OOB.

The sample logic is as presented in the General Sample Steps, with attention to the following:

1. The permissions for the local mmap in this sample is set to local read and write.

2. A send with immediate task is configured for this sample.

3. In this sample, the data sent is printed during the task preparation, not in the successful task completion callback.

4. The local mmap is not exported as the peer does not intend to access it.

5. No remote mmap is created as there is no intention to access the remote memory in this sample.

Reference:

- /opt/mellanox/doca/samples/doca_rdma/rdma_send_immediate/rdma_send_immediate_sample.c
- /opt/mellanox/doca/samples/doca_rdma/rdma_send_immediate/rdma_send_immediate_main.c
- /opt/mellanox/doca/samples/doca_rdma/rdma_send_immediate/meson.build

RDMA Receive with Immediate
This sample illustrates how the remote peer can receive a message sent by the peer (the sender) while also receiving a 32-bit immediate value from the peer's OOB.

The sample logic is as presented in the General Sample Steps, with attention to the following:

1. The permissions for the local mmap in this sample is set to local read and write.

2. A receive task is configured for this sample to retrieve the sent data and the immediate value. Failing to submit a receive task prior to the send with immediate task results in a fatal failure.

3. In this sample, the successful task completion callback also includes:
   1. Checking the result opcode, to verify that the receive task has completed after receiving a sent message with an immediate.
   2. Verifying the data received from the peer is valid and printing it along with the immediate data received.

4. In this sample, data is received from the peer verified to be valid and printed in the successful task completion callback.

5. The local mmap is not exported as the peer does not intend to access it.

6. No remote mmap is created as there is no intention to access the remote memory in this sample.

Reference:

- /opt/mellanox/doca/samples/doca_rdma/rdma_receive_immediate/rdma_receive_immediate_sample.c
- /opt/mellanox/doca/samples/doca_rdma/rdma_receive_immediate/rdma_receive_immediate_main.c
- /opt/mellanox/doca/samples/doca_rdma/rdma_receive_immediate/meson.build

**RDMA Remote Sync Event**
This sample illustrates how to synchronize between local sync event and a remote sync event DOCA RDMA.

**RDMA Remote Sync Event Requester**

The sample logic is as presented in the General Sample Steps, with attention to the following:

1. The permissions for the local mmap in this sample is set to local read and write.

2. A "remote net sync event notify set" task is configured for this sample.
   - For this task, the successful task completion callback has the following logic:
     1. Printing an info log saying the task was successfully completed and a specific successful completion log for the task.
     2. Decreasing the number of remaining tasks. Once 0 is reached:
       1. Freeing the task and task-specific resources.
       2. Stopping the context.
   - For this task, the failed task completion callback stops the context even when the number of remaining tasks is different than 0 (since the synchronization between the peers would fail).

3. A "remote net sync event get" task is configured for this sample.
   - For this task, the successful task completion callback also includes:
     1. Resubmitting the task, until a value greater than or equal to the expected value is retrieved.
     2. Once such value is retrieved, submitting a "remote net sync event notify set" task to signal sample completion, including:
       1. Updating the successful completion message accordingly.
       2. Increasing the number of submitted tasks.
3. If an error was encountered, and the "remote net sync event notify set" task was not submitted, the task and task resources are freed.
   - For this task, the failed task completion callback also includes freeing the "remote net sync event notify set" task and task resources.

4. The local mmap is not exported as the peer does not intend to access it.

5. No remote mmap is created as there is no intention to access the remote memory in this sample.

6. To synchronize events with the peer, a sync event remote net is created from the peer's exported sync event.

7. Both tasks are prepared and submitted in the state change callback, once the context moves from starting to running.

8. The user data of the "remote net sync event get" task points to the "remote net sync event notify set" task.

Reference:

- /opt/mellanox/doca/samples/doca_rdma/rdma_sync_event_requester/rdma_sync_event_requester_sam
- /opt/mellanox/doca/samples/doca_rdma/rdma_sync_event_requester/rdma_sync_event_requester_main
- /opt/mellanox/doca/samples/doca_rdma/rdma_sync_event_requester/meson.build

**RDMA Remote Sync Event Responder**

The sample logic is as presented in the General Sample Steps, with attention to the following:

1. The permissions for the local mmap in this sample is set to local read and write.

2. This sample includes creating a local sync event and exporting it to the remote memory to allow the peer to create a remote handle.

3. No tasks are configured for this sample, and thus no tasks are prepared and submitted, nor are there task completion callbacks. In this sample, the following
steps are executed once the context moves from starting to running, using the state change callback:

1. Waiting for the sync event to be signaled from the remote side.

2. Notifying the sync event from the local side.

3. Waiting for completion notification from the remote side.

Reference:

- /opt/mellanox/docs/samples/doca_rdma/rdma_sync_event_responder/rdma_sync_event_responder_sar
- /opt/mellanox/docs/samples/doca_rdma/rdma_sync_event_responder/rdma_sync_event_responder_main
- /opt/mellanox/docs/samples/doca_rdma/rdma_sync_event_responder/meson.build