

IB Transport Specific Extensions for DAT 2.0

Final Draft

Contents

1.	Data Structures and Types	1-1
1.1	IA specific attributes	1-1
1.2	Definitions (dat_ib_extensions.h)	1-1
1.2.1	DAT_IB_EVENT_NUMBER	1-1
1.2.2	DAT_IB_OP.....	1-2
1.2.3	DAT_IB_EXT_TYPE	1-2
1.2.4	DAT_IB_STATUS.....	1-3
1.2.5	DAT_IB_RETURN.....	1-3
1.2.6	DAT_IB_DTOS.....	1-3
1.2.7	DAT_IB_HANDLE_TYPE.....	1-4
1.2.8	DAT_IB_EVD_EXTENSION_FLAGS.....	1-4
1.2.9	DAT_IB_MEM_PRIV_FLAGS	1-4
1.2.10	DAT_IB_IMMED_DATA	1-5
1.2.11	DAT_IB_EXTENSION_EVENT_DATA	1-5
2.	APIs	2-6
2.1	RDMA write with immediate data	2-7
2.1.1	Consumer Requirement	2-7
2.1.2	Transport Neutral Alternatives	2-7
2.1.3	Transport Requirements	2-7
2.1.4	Function Call	2-8
2.2	Atomic Operations	2-12
2.2.1	Consumer Requirement	2-12
2.2.2	Transport Neutral Alternatives	2-12
2.2.3	Transport Requirements	2-12
2.2.4	Atomicity Guarantees	2-13
2.2.5	Function Calls.....	2-14

1. Data Structures and Types

1.1 IA specific attributes

IA specific attributes for transport extension support are returned with `dat_ia_query()` using the proper `DAT_IA_ATTR_MASK` settings. With mask set to `DAT_IA_FIELD_IA_EXTENSION`, the attribute value will be set to `DAT_EXTENSION_IB` if the provider supports IB transport extensions. With the query mask set to `DAT_IA_FIELD_IA_EXTENSION_VERSION` the consumer can get the version number of the extension interface supported.

1.2 Definitions (`dat_ib_extensions.h`)

All IB prototypes, macros, types, and defines for provider specific extensions are defined in `~/include/dat_ib_extensions.h`. DAT 2.0 defines, in `~/include/dat.h`, a generic event data and the extended operations define the type of data provided with each event and operation type.

1.2.1 `DAT_IB_EVENT_NUMBER`

The `DAT_IB_EVENT_NUMBER` enum specifies the type of IB extension events.

All IB extended DTO events are reported with the single `DAT_IB_DTO_EVENT` type. The specific extended DTO operation is reported with a `DAT_IB_DTOS` type in the operation field of the base `DAT_EVENT` data structure. All other extended events are identified by unique `DAT_IB_EVENT_NUMBER` types.

```
typedef enum dat_ib_event_number
{
    DAT_IB_DTO_EVENT = DAT_IB_EXTENSION_RANGE_BASE,
    DAT_IB_EXT_EVENT1,
    DAT_IB_EXT_EVENT2
} DAT_IB_EVENT_NUMBER;
```

The enumeration will be extended as new extension events are added.

1.2.2 DAT_IB_OP

The DAT_IB_OP enum specifies the type of extension operation to perform. The IB operation type is provided as the DAT_EXTENDED_OP parameter via the DAT_EXTENSION_FUNC call to specify the IB extended operation to call. See section 2 for details on extended operation macros and API mappings.

```
typedef enum dat_ib_op
{
    DAT_IB_FETCH_AND_ADD_OP,
    DAT_IB_CMP_AND_SWAP_OP,
    DAT_IB_RDMA_WRITE_IMMED_OP,
    DAT_IB_NON_DTO_TYPE1_OP,
    DAT_IB_NON_DTO_TYPE2_OP
} DAT_IB_OP;
```

The enumeration will be extended as new extension operations are added.

1.2.3 DAT_IB_EXT_TYPE

The DAT_IB_EXT_TYPE enum specifies the type of extension operation that just completed. All IB extended completion types both, DTO and NON-DTO, are reported in the extended operation type with the single DAT_IB_DTO_EVENT type. The specific extended DTO operation is reported with a DAT_IB_DTOS type in the operation field of the base DAT_EVENT structure. All other extended events are identified by unique DAT_IB_EVENT_NUMBER types.

```
typedef enum dat_ib_ext_type
{
    DAT_IB_FETCH_AND_ADD,
    DAT_IB_CMP_AND_SWAP,
    DAT_IB_RDMA_WRITE_IMMED,
    DAT_IB_RECV_IMMED_DATA,
    DAT_IB_NON_DTO_TYPE1,
    DAT_IB_NON_DTO_TYPE2
} DAT_IB_EXT_TYPE;
```

The enumeration will be extended as new extension event types are added.

1.2.4 DAT_IB_STATUS

The DAT_IB_STATUS enum specifies the type of extension operation to call. All IB extended operations status is reported via the status field in the DAT_IB_EXTENSION_EVENT_DATA structure.

```
typedef enum dat_ib_status
{
    DAT_IB_OP_SUCCESS,
    DAT_IB_OP_ERR1,
    DAT_IB_OP_ERR2
} DAT_IB_STATUS;
```

The enumeration will be extended as new extension operations are added.

1.2.5 DAT_IB_RETURN

The DAT_IB_RETURN enum specifies the extended return codes for IB extension calls that do not map directly to existing DAT_RETURN definitions.

```
typedef enum dat_ib_return
{
    DAT_IB_ERR = DAT_EXTENSION_BASE
} DAT_IB_RETURN;
```

The enumeration will be extended as new extension operation return codes are added.

1.2.6 DAT_IB_DTOS

The DAT_IB_DTOS enum specifies the types of extended DTO operations.

```
typedef enum dat_ib_dtos
{
    DAT_IB_DTO_RDMA_WRITE_IMMED = DAT_DTO_EXTENSION_BASE,
    DAT_IB_DTO_RECV_IMMED,
    DAT_IB_DTO_FETCH_AND_ADD,
    DAT_IB_DTO_CMP_AND_SWAP
} DAT_IB_DTOS;
```

The enumeration will be extended as new extended DTO operations are added.

1.2.7 DAT_IB_HANDLE_TYPE

The DAT_IB_HANDLE_TYPE enum specifies the types of extended handles that do not map directly to existing DAT_HANDLE_TYPE definitions.

```
typedef enum dat_ib_handle_type
{
    DAT_IB_HANDLE_TYPE_EXT = DAT_HANDLE_TYPE_EXTENSION_BASE

} DAT_IB_HANDLE_TYPE;
```

The enumeration will be extended as new extended handle types are added.

1.2.8 DAT_IB_EVD_EXTENSION_FLAGS

The DAT_IB_EVD_EXTENSION_FLAGS enum specifies the EVD extension flags that do not map directly to existing DAT_EVD_FLAGS. This new EVD flag has been added to identify an extended EVD that does not fit the existing stream types.

```
typedef enum dat_ib_evd_extension_flags
{
    DAT_IB_EVD_EXTENSION_FLAG = DAT_EVD_EXTENSION_BASE

} DAT_IB_EVD_EXTENSION_FLAGS;
```

The enumeration will be extended as new extended handle types are added.

1.2.9 DAT_IB_MEM_PRIV_FLAGS

The DAT_IB_MEM_PRIV_FLAGS enum specifies the memory privilege extension flags that do not map directly to existing DAT_MEM_PRIV_FLAGS. New privilege flags have been added for atomic operations.

```
typedef enum dat_ib_mem_priv_flags
{
    DAT_IB_MEM_PRIV_REMOTE_ATOMIC = DAT_MEM_PRIV_EXTENSION_BASE

} DAT_IB_MEM_PRIV_FLAGS;
```

Definition will be extended as new extension memory privilege flags are added.

1.2.10 DAT_IB_IMMEDIATE_DATA

The DAT_IB_DTO_DATA enum specifies the extension data reference by the DAT_EXTENSION_EVENT_DATA structure field when event number is set to DAT_IB_DTO_EVENT and the extension type is set to DAT_IB_RECV_IMMEDIATE_DATA. All DTO information, including an operation set to DAT_IB_DTO_RECV_IMMEDIATE, will be specified in the DAT_EVENT_DATA.DAT_DTO_COMPLETION_EVENT_DATA structure.

```
typedef struct dat_ib_immediate_data
{
    DAT_UINT32      data;

} DAT_IB_IMMEDIATE_DATA;
```

New data structures will be added as new extension data types are defined.

1.2.11 DAT_IB_EXTENSION_EVENT_DATA

```
/*
 * Definitions for extended event data:
 *   When dat_event->event_number >= DAT_IB_EXTENSION_BASE_RANGE
 *   then dat_event->extension_data == DAT_EXTENSION_EVENT_DATA type
 *   and ((DAT_EXTENSION_EVENT_DATA*)dat_event->extension_data)->type
 *   specifies extension data values.
 * NOTE: DAT_EXTENSION_EVENT_DATA cannot exceed 64 bytes as defined by
 *       "DAT_UINT64 extension_data[8]" in DAT_EVENT (dat.h)
 */
typedef struct dat_ib_extension_event_data
{
    DAT_IB_EXT_TYPE      type;
    DAT_IB_STATUS        status;
    union {
        DAT_IB_IMMEDIATE_DATA immed;
    } val;

} DAT_IB_EXTENSION_EVENT_DATA;
```

2. APIs

The following function prototypes are actually implemented as pre-processor macros. The macro validates that extensions are supported and then calls the `DAT_EXTENSION_FUNC` vector in the `dat_provider` structure. The type definition for the core extension call is as follows:

```
typedef DAT_RETURN (*DAT_EXTENSION_FUNC) (
    IN    DAT_HANDLE,          /* DAT handle          */
    IN    DAT_EXTENDED_OP,    /* DAT extension operation */
    IN    va_list);          /* va_list, variable arguments*/
```

Each API below details input/output arguments and completion semantics. Explicit return codes are not given but they can be assumed to be logical uses of existing DAT return codes.

A uDAPL application can determine which extensions and versions are supported by a uDAPL provider by making the `ep_ia_query()` call and iterating the `DAT_NAMED_ATTR` array pointed to by the `provider_specific_attr` member in `DAT_PROVIDER_ATTR`. The `DAT_NAMED_ATTR` type contains two string pointers of *name* and *value*. The table below specifies the *name*/extension relationship. In most cases, simply having the name defined implies support and the *string* value does not supply additional context.

Extension	Name Attribute
Indicates general support for extensions	DAT_EXTENSION_INTERFACE
Indicates version of extended API	DAT_EXTENSION_VERSION
<code>dat_ib_post_fetch_and_add</code>	DAT_IB_FETCH_AND_ADD_OP
<code>dat_ib_post_cmp_and_swap</code>	DAT_IB_CMP_AND_SWAP_OP
<code>dat_ib_post_rdma_write_immed_data</code>	DAT_IB_IMMED_DATA_OP

2.1 RDMA write with immediate data

2.1.1 Consumer Requirement

Applications need an optimized mechanism to notify the receiving end that RDMA write data has completed beyond the two operation method currently required (RDMA write followed by message send). IB provides a RDMA write operation that will support 4-bytes of inline data that will be delivered at the time of remote completion of RDMA write operation is complete. It avoids any latency penalties normally associated with a two operation method.

The initiating side exposes a 4-byte immediate data parameter for the application to set the inline data. On the receiving side, the write with immediate completion notification must be indicated through a receive completion with the 4-byte immediate data placed in the event. The consumer is responsible for the byte order of the immediate data since it is completely opaque to the provider.

2.1.2 Transport Neutral Alternatives

RDMA providers supporting RDMA writes and message sends could collectively group the two operations together to provide similar functionality. A bundled single door-bell mechanism could be used that would optimize the work request operation on the initiator side. It is a little more difficult on the receiving side where the transport provider has to distinguish between 4 bytes of normal message data and 4 bytes of immediate data that belongs to the RDMA write. This requires cooperation with the application so that receive buffers of the appropriate size are allocated and managed on behalf of the transport provider.

2.1.3 Transport Requirements

Additional transport requirements for DAT Provider-to- Provider interaction above the standard requirements stated in Chapter 4:

1. There is a one-to-one correspondence between send and RDMA Write with Immediate Data operations on one Endpoint of the Connection and receive operations on the other Endpoint of the Connection.
2. There is no correspondence between RDMA operations on one Endpoint of the Connection and receive or send data transfer operation on the other Endpoint of the Connection with exception of RDMA Write with Immediate Data.
3. Receive operations on a Connection must be completed in the order of posting of their corresponding sends and RDMA Write with Immediate Data.
4. RDMA Write with Immediate Data operation posted on a Connection must have its data payload delivered to the target memory region and Immediate Data delivered to the matching receive operation without errors prior to the successful receive completion.

2.1.4 Function Call

Synopsis:

```
DAT_RETURN dat_ib_post_rdma_write_with_immed (
    IN DAT_EP_HANDLE          ep_handle,
    IN DAT_COUNT              num_segments
    IN DAT_LMR_TRIPLET        *local_iov,
    IN DAT_DTO_COOKIE         user_cookie,
    IN DAT_RMR_TRIPLET        *remote_iov,
    IN DAT_UINT32             immediate_data,
    IN DAT_COMPLETION_FLAGS  completion_flags);
```

Parameters:

ep_handle	Handle for an instance of the Endpoint
num_segments	Number of <i>lmr_triplets</i> in <i>local_iov</i>
local_iov:	I/O Vector specifying the local buffer from which the data is transferred.
user_cookie	User-provided cookie that is returned to a consumer at the completion of the RDMA write with immediate
remote_iov	I/O Vector specifying the remote buffer to which the data shall be written.
immediate_data	Immediate data to be transferred to the remote side with the RDMA write data.
completion_flags	Flags for posted RDMA Write. The default DAT_COMPLETION_DEFAULT_FLAG is 0 (see Dat 2.0 specification, Appendix A.4 for definitions.

RDMA Write with Immediate Data DTO Flag Definitions

Features	Definition/Bit	Value	Description	Caveat
Completion Suppression		0x00	Generate Completion	
	DAT_COMPLETION_SUPPRESS_FLAG	0x01	Suppress successful Completion	
Solicited Wait		0x00	No request for notification completion for matching receive on the other side of the connection	
	DAT_COMPLETION_SOLICITED_WAIT_FLAG	0x02	Request for notification completion for matching receive on the other side of the connection.	
Notification of Completion		0x00	Notification Completion	Local Endpoint must be

	DAT_COMPLETION_UNSIGNALLED_FLAG	0x04	Non-notification Completion	configured for Notification Suppression.
Barrier Fence		0x00	No request for RDMA Read Barrier Fence	
	DAT_COMPLETION_BARRIER_FENCE_FLAG	0x08	Request for RDMA Read Barrier Fence	

Description:

dat_ib_post_rdma_write_with_immed requests a transfer of all the data from the *local_iov* over the connection of the *ep_handle* Endpoint into the *remote_buffer* and transfer of the *immediate_data* to the remote end of the connection. The *dat_ib_post_rdma_write_with_immed* will consume a Recv buffer on the remote side of the connection. The matching Recv operation will complete successfully only if both RDMA data and Immediate data were successfully delivered into specified locations.

num_segments specifies the number of segments in the *local_iov*. The *local_iov* segments are traversed in the I/O Vector order until all the data is transferred. The actual order of transfer of the data from the segments is left to the implementation. The *local_iov* and the *remote_buffer* specifications should adhere to the rules defined in Appendix A.4.

The requested length of the data transfer is specified by the local buffer length. That is the sum of the *segment_lengths* of *local_iov*. This does not include Immediate Data.

A Consumer shall not modify the *local_iov* or its content until the DTO is completed. When Consumer does not adhere to this rule, the behavior of the Provider and the underlying Transport is not defined. Providers that allow Consumers to get ownership of the *local_iov* but not the memory it specifies back after the *dat_ib_post_rdma_write_with_immed* returns, should document this behavior and also specify its support in Provider attributes. This behavior allows Consumers full control of the *local_iov* after *dat_ib_post_rdma_write_with_immed* returns. Because this behavior is not guaranteed by all Providers, portable Consumers shall not rely on this behavior. Consumers shall not rely on the Provider copying *local_iov* information.

The DAT_SUCCESS return of the *dat_ib_post_rdma_write_with_immed* is at least the equivalent of posting an RDMA Write with Immediate Data operation directly by native Transport. Providers shall avoid resource allocation as part of *dat_ib_post_rdma_write_with_immed* to ensure that this operation is non-blocking.

The completion of the posted *dat_ib_post_rdma_write_with_immed* is reported to the Consumer asynchronously through a DTO Completion event based on the specified *completion_flags* value. The value of *DAT_COMPLETION_UNSIGNALLED_FLAG* is only valid if the Endpoint Request Completion Flags *DAT_COMPLETION_UNSIGNALLED_FLAG*. Otherwise, *DAT_INVALID_PARAMETER* is returned.

The user_cookie allows Consumers to have unique identifiers for each DTO. These identifiers are completely under user control and are opaque to the Provider. There is no requirement on the Consumer that the value *user_cookie* should be unique for each DTO. The *user_cookie* is returned to the Consumer in the Completion event for the posted RDMA Write.

DAPL Extension Design and API

The operation is valid for the Endpoint in the *DAT_EP_STATE_CONNECTED* and *DAT_EP_STATE_DISCONNECTED* states. If the operation returns successfully for the Endpoint in the *DAT_EP_STATE_DISCONNECTED* state, the posted *dat_ib_post_rdma_write_with_immed* is immediately flushed to *request_evd_handle*.

If the reported *status* of the Completion DTO event corresponding to the posted *dat_ib_post_rdma_write_with_immed* DTO is not *DAT.DTO_SUCCESS*, the *transferred_length* in the DTO Completion event is not defined.

dat_ib_post_rdma_write_with_immed is asynchronous and non-blocking. Its thread safety is Provider-dependent. This routine is always thread safe with respect to *dat_ib_post_recv*.

Event Type and Data:

EP	Event Number	Extended DTOS	DTO and EXTENSION DATA
Initiator	DAT_IB.DTO.EVENT	DAT_IB.DTO.RDMA.WRITE.IMMED	N/A
Remote	DAT_IB.DTO.EVENT	DAT_IB.DTO.RECV.IMMED	<i>transferred_length</i> is set to <i>rdma write length</i> and <i>DAT_IB.IMMED_DATA</i> contains 32 bit immediate data

Return Codes:

DAT_SUCCESS	The operation was successful.
DAT_INSUFFICIENT_RESOURCES	The operation failed due to resource limitations.
DAT_INVALID_PARAMETER	Invalid parameter; For example, one of the IOV segments pointed to a memory outside its LMR, or the number of IOVs specified exceeds EP capacity.
DAT_INVALID_HANDLE	Invalid DAT handle; <i>ep_handle</i> is invalid
DAT_INVALID_STATE	Endpoint was not in the <i>DAT_EP_STATE_CONNECTED</i> or <i>DAT_EP_STATE_DISCONNECTED</i> state
DAT_LENGTH_ERROR	The size of the receiving buffer was too small for sending buffer data. The size of the remote buffer was too small for the data of the local buffer.
DAT_PROTECTION_VIOLATION	remote memory access. Protection Zone mismatch between either an LMR of one of the <i>local_iov</i> segments and the local Endpoint or the <i>rmr_context</i> and the remote Endpoint.
DAT_PRIVILEGES_VIOLATION	Privileges violation for local or remote memory access. Either one of the LMRs used in <i>local_iov</i> was invalid or did not have the local read privileges, or <i>rmr_context</i> did not have the remote write privileges.

DAT_MODEL_NOT_SUPPORTED	The requested Model was not supported by the Provider.
-------------------------	--

Usage:

For the best *dat_ib_post_rdma_write_with_immed* operation performance, the Consumer should align each buffer segment of *local_iov* to the *Optimal Buffer Alignment* attribute of the Provider. For portable applications, the Consumer should align each buffer segment of *local_iov* to *DAT_OPTIMAL_ALIGNMENT*.

DAT does not guarantee any ordering between multiple RDMA DTOs even over the same connection to the same remote memory.

The pipeline of RDMA DTOs over a single connection can proceed simultaneously. Thus, if they access the same remote memory the result of the remote buffer is indeterminate. The result of multiple *dat_ib_post_rdma_write_with_immed* operations accessing the same buffer simultaneously can range from data in the buffer from any one of those RDMA Write operations, to data in the buffer being a mixture from multiple *dat_ib_post_rdma_write_with_immed* operations. Consumer can control RDMA Read ordering with respect to other RDMA Writes via *DAT_COMPLETION_BARRIER_FENCE_FLAG*.

If Consumer desires a deterministic result they should use ULP protocol to ensure that only one RDMA Write with immediate operation accesses remote buffer at a time. For example, they can use 0-size RDMA Read between a pair of RDMA Writes that access the same remote location.

Rationale:

Each instance of multiple *dat_ib_post_rdma_write_with_immed* operations accessing the same remote location generates a return code the same as if it were a single *dat_ib_post_rdma_write_with_immed* accessing that memory location. In other words, no error will be generated because multiple *dat_ib_post_rdma_write_with_immed* operations access the same memory location.

Model Implications:

The error behavior for the case when remote buffer is too small for transferred data may be transport specific. The remote buffer size is defined the size of the RMR and not necessarily the *segment_length* of the *DAT_RMR_TRIPLET* specified locally.

The error can be provided synchronously or asynchronously. If the error is return synchronously then *DAT_LENGTH_ERROR* is returned. A synchronously returned error has no effect on the state of the Endpoint to which operation was posted or any other posted operations. A behavior of the connection as well as the type of the asynchronous error return when an error is return asynchronously is defined by the underlying RDMA transport. For example, a connection may be broken as the result of the asynchronous error. An asynchronous error may be return locally, remotely or both.

2.2 Atomic Operations

2.2.1 Consumer Requirement

Cluster applications need an optimized mechanism to synchronize data across the fabric. Atomic operations such as `compare_swap` and `fetch_add` which execute a 64-bit operation at a specific address on a remote node can be used for such a purpose. These operations provide the consumer the ability to read, modify and write the destination address while at the same time guarantee that no other read or write operation will occur across any other EP on the same HA. The scope may optionally extend to HA capabilities of the provider. The atomic operation is expected to use the same remote memory addressing mechanism as RDMA Reads and Writes. Atomic operations will be supported on reliable connection services, will be naturally aligned on an 8 byte boundary, does not need immediate data support, and will always return the original pre-operation remote data into a local 64-bit memory address. All operations on the responder's HA memory is done in the native endian format of that memory system. All operations on the requestor's memory are done in the native endian format of the requestor.

2.2.2 Transport Neutral Alternatives

The feature is specific to IB and strongly recommends hardware support. There is no clear high-performing transport neutral solution based on the requirement to atomically read, modify, and write the 64-bit remote memory location while at the same time guarantee that no other EP will write or read this address between the read and the write.

To perform this operation in software with a set of ULP messages or RDMA reads and writes would adversely affect applications.

2.2.3 Transport Requirements

Additional transport requirements for DAT Provider-to-Provider interaction above the standard requirements stated in Chapter 4:

1. There is no correspondence between ATOMIC operations on one Endpoint of the Connection and receive or send data transfer operation on the other Endpoint of the Connection.
2. If a RDMA READ work request is posted before an ATOMIC Operation work request then the atomic may execute its remote memory operations before the previous RDMA READ has read its data. This can occur because the responder is allowed to delay execution of the RDMA READ. Strict ordering can be assured by posting the ATOMIC Operation work request with the fence modifier. The fence modifier causes the requestor to wait till the RDMA READ completes before issuing the ATOMIC Operation.
3. When a sequence of requests arrives at an EP, the ATOMIC Operation only accesses memory after prior (non-RDMA READ) requests access memory and before subsequent requests access memory. Since the responder takes time to issue the response to the atomic request, and this response takes more time to reach the requestor and even more time for the requestor to create a completion queue entry, requests after the atomic may access the responders memory before the requestor writes the completion queue entry for the ATOMIC Operation request.
4. Each ATOMIC Operation request requires an explicit response and acknowledge message. An ATOMIC Operation response.

2.2.4 Atomicity Guarantees

Atomicity of the read/modify/write on the responder's node by the ATOMIC Operation shall be assured in the presence of concurrent atomic accesses by other EPs on the same provider HA.

A provider may optionally assure atomicity of ATOMIC Operations in the presence of concurrent memory accesses from other provider HA's, IO devices, and CPUs.

2.2.5 Function Calls

2.2.5.1 dat_ib_post_cmp_and_swap()

Synopsis:

```

DAT_RETURN
dat_ib_post_cmp_and_swap(
    IN DAT_EP_HANDLE      ep_handle,
    IN DAT_COUNT          num_segments
    IN DAT_UINT64         cmp_value,
    IN DAT_UINT64         swap_value,
    IN DAT_LMR_TRIPLET    *local_iov,
    IN DAT_DTO_COOKIE     user_cookie,
    IN DAT_RMR_TRIPLET    *remote_iov,
    IN DAT_COMPLETION_FLAGS completion_flags);
    
```

Parameters:

ep_handle	Handle for an instance of the Endpoint
num_segments	Number of <i>lmr_triplets</i> in local_iov
cmp_value	64 bit value used to compare with the remote memory location
swap_value	64 bit value to swap remote memory if cmp_value matches
local_iov:	I/O Vector specifying the local buffer to which the results of the atomic operation is transferred. Memory privilege MUST be set to DAT_IB_MEM_PRIV_REMOTE_ATOMIC when registering this memory.
user_cookie	User-provided cookie that is returned to a consumer at the completion of the operation
remote_iov	I/O Vector specifying the remote buffer to which the data shall be written. Memory privilege MUST be set to DAT_IB_MEM_PRIV_REMOTE_ATOMIC when registering this memory.
completion_flags	Flags for posted operation. The default DAT_COMPLETION_DEFAULT_FLAG is 0 (see Dat 2.0 specification, Appendix A.4 for definitions).

Compare and Swap DTO Flag Definitions

Features	Definition/Bit	Value	Description	Caveat
Completion		0x00	Generate Completion	

Suppression	DAT_COMPLETION_SUPPRESS_FLAG	0x01	Suppress successful Completion	
	DAT_COMPLETION_SOLICITED_WAIT_FLAG	0x02	Request for notification completion for matching receive on the other side of the connection.	
Notification of Completion		0x00	Notification Completion	Local Endpoint must be configured for Notification Suppression.
	DAT_COMPLETION_UNSIGNALLED_FLAG	0x04	Non-notification Completion	
Barrier Fence		0x00	No request for RDMA Read and Atomic Barrier Fence	
	DAT_COMPLETION_BARRIER_FENCE_FLAG	0x08	Request for RDMA Read and Atomic Barrier Fence	

Description:

This call is modeled after the InfiniBand atomic Compare and Swap operation. The *cmp_value* is compared to the 64 bit value stored at the remote memory location specified in *remote_iov*. If the two values are equal, the 64 bit *swap_value* is stored in the remote memory location. In all cases, the original 64-bit value stored in the remote memory location is copied to the *local_iov*. All operations on the responder’s HA memory is done in the native endian format of that memory system. All operations on the requestor’s memory are done in the native endian format of the requestor.

The *local_iov* and the *remote_iov* specifications should adhere to the rules defined in Appendix A.4. The virtual address shall be naturally aligned to an 8 byte boundary.

Providers shall not allow Consumers ownership of the *local_iov* or its memory after the *dat_ib_post_cmp_and_swap* returns. A Consumer shall not read or modify the *local_iov* or its content until the DTO is completed.

The DAT_SUCCESS return of the *dat_ib_post_cmp_and_swap* is at least the equivalent of posting an atomic operation directly by native Transport. Providers shall avoid resource allocation as part of *dat_ib_post_cmp_and_swap* to ensure that this operation is nonblocking.

The completion of the posted *dat_ib_post_cmp_and_swap* is reported to the Consumer asynchronously through a DTO Completion event based on the specified *completion_flags* value. The value of *DAT_COMPLETION_UNSIGNALLED_FLAG* is only valid if the Endpoint Request Completion Flags *DAT_COMPLETION_UNSIGNALLED_FLAG*. Otherwise, *DAT_INVALID_PARAMETER* is returned.

The *user_cookie* allows Consumers to have unique identifiers for each DTO. These identifiers are completely under user control and are opaque to the Provider. There is no requirement on the Consumer that the value *user_cookie* should be unique for each DTO. The *user_cookie* is returned to the Consumer in the Completion event for the posted *dat_ib_post_cmp_and_swap*.

The operation is valid for the Endpoint in the *DAT_EP_STATE_CONNECTED* and *DAT_EP_STATE_DISCONNECTED* states. If the operation returns successfully for the Endpoint in the *DAT_EP_STATE_DISCONNECTED* state, the posted *dat_ib_post_cmp_and_swap* is immediately flushed to *request_evd_handle*.

If the reported *status* of the Completion DTO event corresponding to the posted *dat_ib_post_cmp_and_swap* DTO is *DAT.DTO_SUCCESS*, the original 64-bit value stored in the remote memory location is copied to the *local_iov* and if the *cmp_value* is equal to the 64 bit value stored at the remote memory location specified in *remote_iov* then the 64 bit *swap_value* is stored in the remote memory location. If the *cmp_value* is not equal to the 64 bit value stored at the remote memory location specified in *remote_iov* the value at the remote memory location remains unchanged.

If the reported *status* of the Completion DTO event corresponding to the posted *dat_ib_post_cmp_and_swap* DTO is not *DAT.DTO_SUCCESS*, the contents of the memory specified by IO vectors *local_iov* and the *remote_iov* are not defined.

dat_ib_post_cmp_and_swap is asynchronous and non-blocking. Its thread safety is Provider-dependent.

Event Type and Data:

Endpoint	Event Number	Extended DTOS	Extended Event Data Type
Initiator	DAT_IB.DTO.EVENT	DAT_IB.DTO.CMP_SWAP	n/a
Remote	n/a	n/a	n/a

Return Codes:

DAT_SUCCESS	The operation was successful.
DAT_INSUFFICIENT_RESOURCES	The operation failed due to resource limitations.
DAT_INVALID_PARAMETER	Invalid parameter; For example, one of the IOV segments pointed to a memory outside its LMR, or the number of IOVs specified exceeds EP capacity.
DAT_INVALID_HANDLE	Invalid DAT handle; <i>ep_handle</i> is invalid
DAT_INVALID_STATE	Endpoint was not in the <i>DAT_EP_STATE_CONNECTED</i> or <i>DAT_EP_STATE_DISCONNECTED</i> state
DAT_LENGTH_ERROR	The size of the receiving buffer was too small for sending buffer data. The size of the remote buffer was too small for the data of the local buffer.
DAT_PROTECTION_VIOLATION	remote memory access. Protection Zone mismatch between either an LMR of one of the <i>local_iov</i> segments and the local Endpoint or the <i>rmr_context</i> and the remote Endpoint.
DAT_PRIVILEGES_VIOLATION	Privileges violation for local or remote memory access. Either one of the LMRs used in <i>local_iov</i> was invalid or did not have the local read privileges, or <i>rmr_context</i> did

	not have the remote write privileges.
DAT_MODEL_NOT_SUPPORTED	The requested Model was not supported by the Provider.

Usage:

The EP attributes max_rdma_read_out and max_rdma_read_in include both outstanding RDMA Read and IB atomic operations support by the underlying Transport.

If connection was established without outstanding RDMA Read/Atomic attributes matching on Endpoints on both sides (outstanding RDMA Read/Atomic outgoing on one end is larger than the outstanding RDMA Read/Atomic incoming on the other end), connection is broken when the number of incoming RDMA Read/Atomic exceeds the outstanding RDMA Read/Atomic incoming attribute of the Endpoint.

The local and remote atomic buffer memory requires atomic memory privileges of DAT_IB_MEM_PRIV_REMOTE_ATOMIC. Failure to set up local and remote buffer memory privileges for these Providers will result in asynchronous DTO completion error and connection being broken.

DAT does not guarantee any ordering between multiple RDMA Read/atomic DTO even over the same connection to the same remote memory. The pipeline of RDMA Read/atomic DTOs over a single connection can proceed simultaneously. Thus, if they access the same remote memory the result of the remote buffer is indeterminate. Consumer can control RDMA Read/atomic ordering with respect to other RDMA Reads or Writes or Sends or Atomics via DAT_COMPLETION_BARRIER_FENCE_FLAG.

Rationale:

Model Implications:

The number of posted RDMA Read/atomic operations on Send WQ can exceed max_rdma_read_out attribute of the EP. DAT Provider ensures that the number of outstanding atomic operations on the remote endpoint of the connection does not exceed the EP attribute. Consumer should rely on its own atomic operation flow control to ensure that the number of atomic operations for which completions have not been generated does not exceed the EP max_rdma_read_out attribute value.

While Provider does guarantee flow control for RDMA Read/atomic DTOs (the maximum number of RDMA Reads/atomics reaching the remote host simultaneously over a single connection), Consumer should avoid posting more than max_rdma_read_out RDMA Read/atomic operations to the connection. Since all DTOs posted to the Send WQ of the EP are processed in order, inability to process an RDMA Read/Atomic that exceeds max_rdma_read_out will stall processing of all other DTOs of the Send WQ of the EP. This is irrespective of the Consumer specification of DAT_COMPLETION_BARRIER_FENCE_FLAG value that is Consumer requested stalling of the Send WQ processing.

2.2.5.2 dat_ib_post_fetch_and_add()

Synopsis:

```

DAT_RETURN
dat_ib_post_fetch_and_add(
    IN DAT_EP_HANDLE          ep_handle,
    IN DAT_COUNT              num_segments
    IN DAT_UINT64             add_value,
    IN DAT_LMR_TRIPLET        *local_iov,
    IN DAT_DTO_COOKIE         user_cookie,
    IN DAT_RMR_TRIPLET        *remote_iov,
    IN DAT_COMPLETION_FLAGS  completion_flags);
    
```

Parameters:

ep_handle	Handle for an instance of the Endpoint
num_segments	Number of <i>lmr_triplets</i> in local_iov
add_value	64 bit value used to compare with the remote memory location
local_iov:	I/O Vector specifying the local buffer to which the results of the atomic operation is transferred. Memory privilege MUST be set to DAT_IB_MEM_PRIV_REMOTE_ATOMIC when registering this memory.
user_cookie	User-provided cookie that is returned to a consumer at the completion of the operation
remote_iov	I/O Vector specifying the remote buffer to which the data shall be written. Memory privilege MUST be set to DAT_IB_MEM_PRIV_REMOTE_ATOMIC when registering this memory.
completion_flags	Flags for posted operation. The default DAT_COMPLETION_DEFAULT_FLAG is 0 (see Dat 2.0 specification, Appendix A.4 for definitions).

Compare and Swap DTO Flag Definitions

Features	Definition/Bit	Value	Description	Caveat
Completion Suppression		0x00	Generate Completion	
	DAT_COMPLETION_SUPPRESS_FLAG	0x01	Suppress successful Completion	
Notification of Completion		0x00	Notification Completion	Local Endpoint must be configured for Notification Suppression.
	DAT_COMPLETION_UNSIGNALLED_FLAG	0x04	Non-notification Completion	

Barrier Fence		0x00	No request for RDMA Read Barrier Fence
	DAT_COMPLETION_ BARRIER_FENCE_ FLAG	0x08	Request for RDMA Read and Atomic Barrier Fence

Description:

This call is modeled after the InfiniBand atomic Fetch and Add operation. The *add_value* is added to the 64 bit value stored at the remote memory location specified in *remote_iov*. The original pre-added 64 bit value stored in the remote memory location is copied to the *local_iov*. All operations on the responder's HA memory is done in the native endian format of that memory system. All operations on the requestor's memory are done in the native endian format of the requestor.

The *local_iov* and the *remote_iov* specifications should adhere to the rules defined in Appendix A.4. The virtual address shall be naturally aligned to an 8 byte boundary.

Providers shall not allow Consumers ownership of the *local_iov* or its memory after the *dat_ib_post_fetch_and_add* returns. A Consumer shall not read or modify the *local_iov* or its content until the DTO is completed.

The DAT_SUCCESS return of the *dat_ib_post_fetch_and_add* is at least the equivalent of posting an atomic operation directly by native Transport. Providers shall avoid resource allocation as part of *dat_ib_post_fetch_and_add* to ensure that this operation is nonblocking.

The completion of the posted *dat_ib_post_fetch_and_add* is reported to the Consumer asynchronously through a DTO Completion event based on the specified *completion_flags* value. The value of *DAT_COMPLETION_UNSIGNALLED_FLAG* is only valid if the Endpoint Request Completion Flags *DAT_COMPLETION_UNSIGNALLED_FLAG*. Otherwise, *DAT_INVALID_PARAMETER* is returned.

The *user_cookie* allows Consumers to have unique identifiers for each DTO. These identifiers are completely under user control and are opaque to the Provider. There is no requirement on the Consumer that the value *user_cookie* should be unique for each DTO. The *user_cookie* is returned to the Consumer in the Completion event for the posted *dat_ib_post_fetch_and_add*.

The operation is valid for the Endpoint in the *DAT_EP_STATE_CONNECTED* and *DAT_EP_STATE_DISCONNECTED* states. If the operation returns successfully for the Endpoint in the *DAT_EP_STATE_DISCONNECTED* state, the posted *dat_ib_post_fetch_and_add* is immediately flushed to *request_evd_handle*.

If the reported *status* of the Completion DTO event corresponding to the posted *dat_ib_post_fetch_and_add* DTO is *DAT.DTO_SUCCESS*, the *add_value* is added to the 64 bit value stored at the remote memory location specified in *remote_iov* and stored in the same *remote_iov* location. The original pre-added 64 bit value stored in the remote memory location is copied to the *local_iov*.

If the reported *status* of the Completion DTO event corresponding to the posted *dat_ib_post_fetch_and_add* DTO is not *DAT.DTO.SUCCESS*, the contents of the memory specified by IO vectors *local_iov* and the *remote_iov* are not defined.

dat_ib_post_fetch_and_add is asynchronous and non-blocking. Its thread safety is Provider-dependent.

Event Type and Data:

Endpoint	Event Number	Extended DTOS	Extended Event Data Type
Initiator	DAT_IB.DTO.EVENT	DAT_IB.DTO.FETCH_AND_ADD	n/a
Remote	n/a	n/a	n/a

Return Codes:

DAT.SUCCESS	The operation was successful.
DAT.INSUFFICIENT_RESOURCES	The operation failed due to resource limitations.
DAT.INVALID_PARAMETER	Invalid parameter; For example, one of the IOV segments pointed to a memory outside its LMR, or the number of IOVs specified exceeds EP capacity.
DAT.INVALID_HANDLE	Invalid DAT handle; <i>ep_handle</i> is invalid
DAT.INVALID_STATE	Endpoint was not in the <i>DAT_EP_STATE.CONNECTED</i> or <i>DAT_EP_STATE.DISCONNECTED</i> state
DAT.LENGTH_ERROR	The size of the receiving buffer was too small for sending buffer data. The size of the remote buffer was too small for the data of the local buffer.
DAT.PROTECTION_VIOLATION	remote memory access. Protection Zone mismatch between either an LMR of one of the <i>local_iov</i> segments and the local Endpoint or the <i>rmr_context</i> and the remote Endpoint.
DAT.PRIVILEGES_VIOLATION	Privileges violation for local or remote memory access. Either one of the LMRs used in <i>local_iov</i> was invalid or did not have the local read privileges, or <i>rmr_context</i> did not have the remote write privileges.
DAT.MODEL_NOT_SUPPORTED	The requested Model was not supported by the Provider.

Usage:

The EP attributes `max_rdma_read_out` and `max_rdma_read_in` include both outstanding RDMA Read and IB atomic operations support by the underlying Transport.

If connection was established without outstanding RDMA Read/Atomic attributes matching on Endpoints on both sides (outstanding RDMA Read/Atomic outgoing on one end is larger than the outstanding RDMA Read/Atomic incoming on the other end), connection is broken when the number of incoming RDMA Read/Atomic exceeds the outstanding RDMA Read/Atomic incoming attribute of the Endpoint.

The local and remote atomic buffer memory requires atomic memory privileges of `DAT_IB_MEM_PRIV_REMOTE_ATOMIC`. Failure to set up local and remote buffer memory privileges for these Providers will result in asynchronous DTO completion error and connection being broken.

DAT does not guarantee any ordering between multiple RDMA Read/atomic DTO even over the same connection to the same remote memory. The pipeline of RDMA Read/atomic DTOs over a single connection can proceed simultaneously. Thus, if they access the same remote memory the result of the remote buffer is indeterminate. Consumer can control RDMA Read/atomic ordering with respect to other RDMA Reads or Writes or Sends or Atomics via `DAT_COMPLETION_BARRIER_FENCE_FLAG`.

Rationale:**Model Implications:**

The number of posted RDMA Read/atomic operations on Send WQ can exceed `max_rdma_read_out` attribute of the EP. DAT Provider ensures that the number of outstanding atomic operations on the remote endpoint of the connection does not exceed the EP attribute. Consumer should rely on its own atomic operation flow control to ensure that the number of atomic operations for which completions have not been generated does not exceed the EP `max_rdma_read_out` attribute value.

While Provider does guarantee flow control for RDMA Read/atomic DTOs (the maximum number of RDMA Reads/atomics reaching the remote host simultaneously over a single connection), Consumer should avoid posting more than `max_rdma_read_out` RDMA Read/atomic operations to the connection. Since all DTOs posted to the Send WQ of the EP are processed in order, inability to process an RDMA Read/Atomic that exceeds `max_rdma_read_out` will stall processing of all other DTOs of the Send WQ of the EP. This is irrespective of the Consumer specification of `DAT_COMPLETION_BARRIER_FENCE_FLAG` value that is Consumer requested stalling of the Send WQ processing.