DAPL IB collective extensions

High-Level Design and API

uDAPL provides a generalized abstraction to RDMA capable transports. As a generalized abstraction, it cannot exploit the unique properties that many of the underlying interconnects can provide. This paper documents an extension interface to uDAPL and specifies APIs that use the extension interface.

April 2011

Contents

Revision History

Revision Number	Description	Revision Date
Draft	Initial public release.	April 2011

1 Introduction

The introduction of a generalized extension interface to DAPL 2.0 gives DAPL providers an opportunity to offer value-added products to their customers. This document defines the extensions and their APIs for InfiniBand* operations and MPI-compatible collective operations.

As of this writing, not all details of the DAPL extension mechanism were defined for 2.0. For this reason, the APIs, structures, and enumerations detailed in this document may evolve as the DAPL 2.0 specification solidifies.

2 Data Structures and Types

2.1 DAT_IB_OP

The DATA_IB_OP enumeration specifies the type of extension operation. The DAPL specification assigns extension ranges to be used by extension providers. Although the mnemonics are defined by this document, the initial value of the extension operations will be assigned by the DAPL specification.

```
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_UD_SEND_OP,
      DAT_IB_QUERY_COUNTERS_OP,
      DAT_IB_PRINT_COUNTERS_OP,
      DAT_IB_COLLECTIVE_CREATE_GROUP_OP,
      DAT_IB_COLLECTIVE_FREE_GROUP_OP,
      DAT_IB_COLLECTIVE_SET_CLOCK_OP,
      DAT_IB_COLLECTIVE_READ_CLOCK_OP,
      DAT_IB_COLLECTIVE_SCATTER_OP,
      DAT_IB_COLLECTIVE_SCATTERV_OP,
      DAT_IB_COLLECTIVE_GATHER_OP,
      DAT_IB_COLLECTIVE_GATHERV_OP,
      DAT_IB_COLLECTIVE_ALLGATHER_OP,
      DAT_IB_COLLECTIVE_ALLGATHERV_OP,
      DAT_IB_COLLECTIVE_ALLTOALL_OP,
      DAT_IB_COLLECTIVE_ALLTOALLV_OP,
      DAT_IB_COLLECTIVE_REDUCE_OP,
      DAT_IB_COLLECTIVE_ALLREDUCE_OP,
      DAT_IB_COLLECTIVE_REDUCE_SCATTER_OP,
      DAT_IB_COLLECTIVE_SCAN_OP,
      DAT_IB_COLLECTIVE_BROADCAST_OP,
      DAT_IB_COLLECTIVE_BARRIER_OP
} DAT_IB_OP;
```

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

2.2 DAT_IB_EXTENSION_DATA

When a DAT_EVENT specifies an event type of DAT_IB_EXTENSION_EVENT, event data is defined as follows:

```
typedef struct dat_ib_extension_data {
      DAT_IB_EXT_TYPE   type;
      DAT_IB_STATUS     status;
      union {
          DAT_IB_IMMED_DATA              immed;
          DAT_IB_COLLECTIVE_EVENT_DATA   coll;
      } val;
      DAT_IB_ADDR_HANDLE    remote_ah;
} DAT_IB_EXTENSION_DATA;
```

The DAT_IB_EXTENSION_DATA structure is mapped over the *extension_data* array field of DAT_EVENT_EXTENSION_DATA type.

2.3 DAT_IB_EXT_STATUS

The DATA_IB_STATUS enumeration specifies the status of extension data operation contained in the DAT_EVENT.

```
typedef enum dat_ib_status
{
      DAT_OP_SUCCESS = DAT_SUCCESS,
      DAT_IB_OP_ERR,
      DAT_IB_COLL_COMP_ERR,

} DAT_IB_STATUS;
```

2.4 DAT_IB_EXT_TYPE

The DATA_IB_EXT_TYPE enumeration specifies the type of extension data contained in the DAT_EVENT.

```
typedef enum dat_ib_ext_type
{
      DAT_IB_FETCH_AND_ADD,         // 0
      DAT_IB_CMP_AND_SWAP,          // 1
      DAT_IB_RDMA_WRITE_IMMED,      // 2
      DAT_IB_RDMA_WRITE_IMMED_DATA, // 3
      DAT_IB_RECV_IMMED_DATA,       // 4
      DAT_IB_UD_CONNECT_REQUEST,    // 5
      DAT_IB_UD_REMOTE_AH,          // 6
      DAT_IB_UD_PASSIVE_REMOTE_AH,  // 7
      DAT_IB_UD_SEND,               // 8
      DAT_IB_UD_RECV,               // 9
      DAT_IB_UD_CONNECT_REJECT,     // 10
      DAT_IB_UD_CONNECT_ERROR,      // 11
      DAT_IB_COLLECTIVE_CREATE_STATUS,     // 12
      DAT_IB_COLLECTIVE_CREATE_DATA,       // 13
      DAT_IB_COLLECTIVE_CLOCK_SET_STATUS,  // 14
      DAT_IB_COLLECTIVE_SCATTER_STATUS,    // 15
      DAT_IB_COLLECTIVE_SCATTERV_STATUS,   // 16
      DAT_IB_COLLECTIVE_GATHER_STATUS,     // 17
      DAT_IB_COLLECTIVE_GATHERV_STATUS,    // 18
      DAT_IB_COLLECTIVE_ALLGATHER_STATUS,  // 19
      DAT_IB_COLLECTIVE_ALLGATHERV_STATUS,// 20
      DAT_IB_COLLECTIVE_ALLTOALL_STATUS,   // 21
      DAT_IB_COLLECTIVE_ALLTOALLV_STATUS,  // 22
      DAT_IB_COLLECTIVE_REDUCE_STATUS,     // 23
      DAT_IB_COLLECTIVE_ALLREDUCE_STATUS,  // 24
      DAT_IB_COLLECTIVE_REDUCE_SCATTER_STATUS,// 25
      DAT_IB_COLLECTIVE_SCAN_STATUS,      // 26
      DAT_IB_COLLECTIVE_BROADCAST_STATUS, // 27
      DAT_IB_COLLECTIVE_BARRIER_STATUS,   // 28

} DAT_IB_TYPE;
```

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

2.5 DAT_IB_IMMED_DATA

The DAT_IB_IMMED_DATA type contains the 32 bits of unsigned immediate data associated with an RDMA write.

```
typedef DAT_UINT32 DAT_IB_IMMED_DATA;
```

2.6 DAT_IB_ADDR_HANDLE_DATA

The DAT_IB_ADDR_HANDLE type contains the UD address information required for UD sends. This information is returned during connection establishment events with UD service type endpoints.

```
typedef struct dat_ib_addr_handle
{
    struct ibv_ah       *ah;
    DAT_UINT32          qpn;
    DAT_SOCK_ADDR6      ia_addr;

} DAT_IB_ADDR_HANDLE;
```

2.7 DAT_IB_COLLECTIVE_MEMBER

The DAT_IB_COLLECTIVE_MEMBER type contains the addressing information for a collective member. Provider member address information is opaque to consumer and is returned during the synchronous create member call along with its size.

```
typedef    void * DAT_IB_COLLECTIVE_MEMBER;
```

2.8 DAT_IB_COLLECTIVE_GROUP

The DAT_IB_COLLECTIVE_GROUP type contains detailed information regarding rank distribution within each node (intranode) and across all nodes (internode) in the group for some providers that require details per node within the group.

```
typedef struct dat_ib_collective_group
{
      int   local_size;   /* # processes on this node */
      int   local_rank;   /* my rank within node */
      int   *local_ranks; /* global rank, each local process */
      int   external_size;/* # nodes, each node=one process */

      /* rank among ext processes if one of them, else -1 */
      int   external_rank;

      /* global rank for each external process */
      int   *external_ranks;

      /* mapping from global rank to local rank. -1 if the
      process is on a different node */
      int   *intranode_table;

      /* mapping from global rank to external rank. -1 if the
      process is not external */
      int   *internode_table;

      int   is_comm_world;

} DAT_IB_COLLECTIVE_GROUP;
```

2.9 DAT_IB_COLLECTIVE_CREATE_EVENT_DATA

The DAT_COLLECTIVE_CREATE_DATA type contains the handle of a newly created collective group that this endpoint has become a member of and context provided by consumer.

```
typedef struct dat_ib_collective_create_event_data {

      DAT_CONTEXT               user_context;
      DAT_COLLECTIVE_HANDLE     collective_handle;

} DAT_IB_COLLECTIVE_CREATE_EVENT_DATA;
```

2.9.1 DAT_IB_COLLECTIVE_DATA_TYPE

In general, uDAPL treats all data as a byte stream. Some collective operations, such as reductions, require knowledge of the data type to properly perform the operation. The DAT_COLLECTIVE_DATA_TYPE enumerates the possible data types used in a collective.

```
typedef enum dat_ib_collective_data_type {
      DAT_IB_COLLECTIVE_TYPE_INT8,  /* signed 8 bit type */
      DAT_IB_COLLECTIVE_TYPE_UINT8, /* unsigned 8 bit type */
      DAT_IB_COLLECTIVE_TYPE_INT16, /* signed 16 bit type */
      DAT_IB_COLLECTIVE_TYPE_UINT16,/* unsigned 16 bit type */
      DAT_IB_COLLECTIVE_TYPE_INT32, /* signed 32 bit type */
      DAT_IB_COLLECTIVE_TYPE_UINT32,/* unsigned 32 bit type */
      DAT_IB_COLLECTIVE_TYPE_INT64, /* signed 64 bit type */
      DAT_IB_COLLECTIVE_TYPE_UINT64,/* unsigned 64 bit type */
      DAT_IB_COLLECTIVE_TYPE_FLOAT, /* single-precision FP */
      DAT_IB_COLLECTIVE_TYPE_DOUBLE /* double-precision FP */
} DAT_COLLECTIVE_DATA_TYPE;
```

2.9.2 DAT_IB_REDUCE_OP

The DAT_REDUCE_TYPE enumerates the possible reduction operations.

```
typedef enum dat_ib_collective_reduce_data_op {
      DAT_IB_COLLECTIVE_REDUCE_OP_MAX,    /* maximum */
      DAT_IB_COLLECTIVE_REDUCE_OP_MIN,    /* minimum */
      DAT_IB_COLLECTIVE_REDUCE_OP_SUM,    /* sum */
      DAT_IB_COLLECTIVE_REDUCE_OP_PROD,   /* product */
      DAT_IB_COLLECTIVE_REDUCE_OP_LAND,   /* logical AND */
      DAT_IB_COLLECTIVE_REDUCE_OP_BAND,   /* bit-wise AND */
      DAT_IB_COLLECTIVE_REDUCE_OP_LOR,    /* logical OR */
      DAT_IB_COLLECTIVE_REDUCE_OP_BOR,    /* bit-wise OR */
      DAT_IB_COLLECTIVE_REDUCE_OP_LXOR,   /* logical XOR */
      DAT_IB_COLLECTIVE_REDUCE_OP_BXOR,   /* bit-wise XOR */
      DAT_IB_COLLECTIVE_REDUCE_OP_MAXLOC, /* max, location */
      REDUCE_OP_MINLOC  /* min value and location */
} DAT_REDUCE_OP;
```

2.9.3 DAT_IB_COLLECTIVE_RANK

The `DAT_IB_COLLECTIVE_RANK` is a collective index of the endpoints associated with a collective group.

```
typedef unsigned int DAT_IB_COLLECTIVE_RANK;
```

2.9.4 DAT_IB_COLLECTIVE_ID

The `DAT_IB_COLLECTIVE_ID` is a network-unique identifier for a collective group.

```
typedef unsigned int DAT_IB_COLLECTIVE_ID;
```

3 APIs

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 the existing DAT return codes.

A uDAPL application can determine which extensions are supported by a uDAPL provider by making the *dat_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_INTERFFACE
dat_ib_post_rdma_write_immed	DAT_IB_IMMED_DATA
dat_ib_post_cmp_and_swap	DAT_IB_CMP_AND_SWAP
dat_ib_post_fetch_and_add	DAT_IB_FETCH_AND_ADD
dat_ib_post_send_ud	DAT_IB_UD
dat_ib_collective_create_member	Implied if any collective is supported
dat_ib_collective_free_member	Implied if any collective is supported
dat_ib_collective_create_group	Implied if any collective is supported
dat_ib_collective_free_group	Implied if any collective is supported
dat_ib_collective_scatter	DAT_COLL_SCATTER
dat_ib_collective_scatterv	DAT_COLL_SCATTERV
dat_ib_collective_gather	DAT_COLL_GATHER
dat_ib_collective_gatherv	DAT_COLL_GARHERV
dat_ib_collective_allgather	DAT_COLL_ALLGATHER
dat_ib_collective_allgatherv	DAT_COLL_ALLGATHERV
dat_ib_collective_alltoall	DAT_COLL_ALLTOALL
dat_ib_collective_alltoallv	DAT_COLL_ALLTOALLV
dat_ib_collective_reduce	DAT_COLL_REDUCE
dat_ib_collective_allreduce	DAT_COLL_ALLREDUCE
dat_ib_collective_reduce_scatter	DAT_COLL_REDUCESCATTER
dat_ib_collective_broadcast	DAT_COLL_BROADCAST
dat_ib_collective_scan	DAT_COLL_SCAN
dat_ib_collective_barrier	DAT_COLL_BARRIER

The *value* attribute for DAT_COLL_PORT_REDUCE will contain one or more of the space separated names defined in the DAT_REDUCE_TYPE (that is REDUCE_OP_MAX REDUCE_OP_MIN, and so forth).

3.1 IB RDMA Extensions

3.1.1 dat_ib_post_rdma_write_immed()

```
DAT_RETURN
dat_ib_post_rdma_write_immed(
      IN DAT_EP_HANDLE           ep_handle,
      IN DAT_COUNT               num_segments,
      IN DAT_LMR_TRIPLET         *local_iov,
      IN DAT_CONTEXT             user_context,
      IN DAT_RMR_TRIPLE          *remote_iov,
      IN DAT_UINT32              immediate_data,
      IN DAT_COMPLETION_FLAGS    completion_flags
      );
```

This asynchronous call performs a normal RDMA write to the specified remote endpoint. When the write completes, an extended DAT_IB_RDMA_WRITE_IMMED_DATA completion event containing the unsigned 32 bit immediate data value is posted to the receive EVD on the remote endpoint. Event completion for the request completes as a normal RDMA write.

Endpoint	EVD	Extension Type	Extension Event Data Type
Initiator	Request	DAT_IB_RDMA_WRITE_IMMED_STATUS	N/A
Remote	Receive	DAT_IB_RDMA_WRITE_IMMED_DATA	DAT_IB_IMMEDIATE_DATA

3.2 IB Atomic Extensions

3.2.1 dat_ib_post_cmp_and_swap()

```
DAT_RETURN
dat_ib_post_cmp_and_swap(
      IN DAT_EP_HANDLE          ep_handle,
      IN DAT_UINT64             cmp_value,
      IN DAT_UINT64             swap_value,
      IN DAT_LMR_TRIPLET        *local_iov,
      IN DAT_CONTEXT            user_context,
      IN DAT_RMR_TRIPLE         *remote_iov,
      IN DAT_COMPLETION_FLAGS   completion_flags
      );
```

This asynchronous call is modeled after the InfiniBand atomic Compare and Swap operation. The *cmp_value* is compared to the unsigned 64 bit value stored at the remote memory location specified in *remote_iov*. If the two values are equal, the unsigned 64 bit *swap_value* is stored in the remote memory location. Both the compare and potential swap are performed atomically with respect to other memory operations performed by the Interface Adapter (IA) associated with the endpoint. In all cases, the original unsigned 64 bit value stored in the remote

memory location is returned to local memory specified by the *local_iov* parameter. Completion status is posted as a send EVD completion event.

Endpoint	EVD	Extension Type	Extension Event Data Type
Initiator	Request	DAT_IB_CMP_AND_SWAP_STATUS	N/A
Remote	N/A		

3.2.2 dat_ib_post_fetch_and_add()

```
DAT_RETURN
dat_ib_post_fetch_and_add(
      IN DAT_EP_HANDLE           ep_handle,
      IN DAT_UINT64              add_value,
      IN DAT_LMR_TRIPLET*        local_iov,
      IN DAT_CONTEXT             user_context,
      IN DAT_RMR_TRIPLE          *remote_iov,
      IN DAT_COMPLETION_FLAGS    completion_flags
      );
```

This asynchronous call is modeled after the InfiniBand atomic Fetch and Add operation. The *add_value* is added to the unsigned 64 bit value stored at the remote memory location specified in *remote_iov*. The fetch and add are performed atomically with respect to other memory operations performed by the Interface Adapter (IA) associated with the endpoint. The original pre-added unsigned 64 bit value stored in the remote memory location is retuned to local memory specified by the *local_iov* parameter. Completion status is posted as a send EVD completion event.

Endpoint	EVD	Extension Type	Extension Event Data Type
Initiator	Request	DAT_IB_FETCH_AND_ADD_STATUS	N/A
Remote	N/A		

3.3 IB Collective Extensions

The collective extensions described in the following sections are designed to support MPI and general multicast operations over IB fabrics. Where feasible, they come as close to MPI semantics as possible. Unless otherwise stated, all members participating in a data collective operation must call the associated collective routine for the data transfer operation to complete. Unless otherwise stated, the root collective member of a data operation will receive its own portion of the collective data. In most cases, the root member can prevent sending/receiving data when such operations would be redundant. When root data is already "in place", the root member may set the send and/or receive buffer pointer argument to NULL.

Unlike standard DAPL movement operations that require registered memory and LMR objects, collective data movement operations employ pointers to user-virtual address space that do not require pre-registration by the application. From a resource usage point of view, the API user should consider that the provider implementation my perform memory registrations/deregistration on behalf of the application to accomplish a data transfer.

Most collective calls are asynchronous. Upon completion, an event will be posted to the EVD specified when the collective was created.

3.3.1 dat_ib_collective_create_group()

```
DAT_RETURN
dat_collective_create_group(
      IN DAT_COLLECTIVE_MEMBER      *collective_group,
      IN DAT_COUNT                   group_size,
      IN DAT_COLLECTIVE_RANK         self,
      IN DAT_COLLECTIVE_ID           group_id,
      IN DAT_EVD_HANDLE              evd_handle,
      IN DAT_PZ_HANDLE               pz_handle,
      IN DAT_CONTEXT                 user_context
      );
```

This asynchronous call initiates the process of creating a collective group and must be called by all group members. The *collective_group* argument points to an array of address/connection qualifier pairs that identify the members of the group in rank order. The *group_size* argument specifies the size of the group and therefore the size of the *collective_group* array. The *self* argument identifies the rank of the caller. The *group_id* argument specifies a network-unique identifier for this instance of the collective group. All members of the group must specify the same *group_id* value for the same collective instance. The *evd_handle* argument specifies the EVD used for all asynchronous collective completions including this call. The *pz_handle* specifies the protection zone for the group. The *user_context* argument will be returned in the DAT_IB_COLLECTIVE_CREATE_DATA event.

On a successful completion, each group member will receive a DAT_IB_COLLECTIVE_CREATE_DATA event on the EVD specified by *evd_handle*. The event contains the collective handle, the rank of the receiving endpoint within the collective group, the size of the group, and the caller specified *user_context.* The returned collective handle can be used in network clock, multicast, and other collective operations.

All members of the collective group must create a PSP, through `dat_psp_create()`, with the connection qualifier value associated with their rank in the *`collective_group`* array before making this call.

Multiple collective groups can be defined and an endpoint may belong to more than one collective group.

Endpoint	EVD	Extension Type	Extension Event Data Type
All - Failure	Collective	DAT_IB_COLLECTIVE_CREATE_STATUS	N/A
All - Success	Collective	DAT_IB_COLLECTIVE_CREATE_DATA	DAT_COLLECTIVE_CREATE_DATA

3.3.2 dat_ib_collective_free_group()

```
DAT_RETURN
dat_collective_free_group(
      IN DAT_COLLECTIVE_HANDLE      collective_handle
      );
```

This synchronous call destroys a previously created collective group associated with the *`collective_handle`* argument. Any pending or in-process requests associated with the collective group will be terminated and be posted to the appropriate EVD.

3.3.3 dat_ib_collective_broadcast()

```
DAT_RETURN
dat_collective_broadcast(
      IN DAT_COLLECTIVE_HANDLE      collective_handle,
      IN DAT_PVOID                  buffer,
      IN DAT_COUNT                  byte_count,
      IN DAT_COLLECTIVE_RANK        root,
      IN DAT_CONTEXT                user_context,
      IN DAT_COMPLETION_FLAGS       completion_flags
      );
```

This asynchronous call performs a broadcast send operation that transfers data specified by the *`buffer`* argument of the *root* into the *`buffer`* argument of all other endpoints in the collective group specified by *`collective_handle`*. The operation is completed on the collective EVD unless completions are suppressed through the completion flags. All broadcasts are considered "in place" transfers. The tables below show the result of a broadcast operation.

	Root Send Buffer Elements		
Members	A[0]	n/a	n/a
	n/a	n/a	n/a
	n/a	n/a	n/a

	Receive Buffer Elements for Members on Completion		
Members	A[0]	n/a	n/a
	A[0]	n/a	n/a
	A[0]	n/a	n/a

Endpoint	EVD	Extension Type	Extension Event Data Type
All	Collective	DAT_IB_COLLECTIVE_BROADCAST_STATUS	N/A

3.3.4 dat_ib_collective_scatter()

```
DAT_RETURN
dat_collective_scatter(
      IN DAT_COLLECTIVE_HANDLE     collective_handle,
      IN DAT_PVOID                 send_buffer,
      IN DAT_COUNT                 send_byte_count,
      IN DAT_PVOID                 recv_buffer,
      IN DAT_COUNT                 recv_byte_count,
      IN DAT_COLLECTIVE_RANK       root,
      IN DAT_CONTEXT               user_context,
      IN DAT_COMPLETION_FLAGS      completion_flags
      );
```

This asynchronous call performs a scatter of the data specified by the *send_buffer* argument to the collective group specified by *collective._handle*. = Data is received in the buffer specified by the *recv_buffer* argument. The *recv_byte_count* argument specifies the size of the receive buffer. Data from the root *send_buffer* will be divided by the number of members in the collective group to form equal and contiguous memory partitions. Each member of the collective group will receive its rank relative partition. An error is returned if the *send_byte_count* does not describe memory that can be evenly divided by the size of the collective group. An "in place" transfer for the root rank can be indicated by passing NULL as the *recv_buffer* argument. The *send_buffer* and *send_byte_count* arguments are ignored on non-root members. The operation is completed on the collective EVD unless completions are suppressed through the completion flags. The tables below show the result of a scatter operation.

	Root Send Buffer Elements		
Members	A[0]	A[1]	A[2]
	n/a	n/a	n/a
	n/a	n/a	n/a

	Receive Buffer Elements for Members on Completion		
Members	A[0]	n/a	n/a
	A[1]	n/a	n/a
	A[2]	n/a	n/a

Endpoint	EVD	Extension Type	Extension Event Data Type
All	Collective	DAT_IB_COLLECTIVE_SCATTER_STATUS	N/A

3.3.5 dat_ib_collective_scatterv()

```
DAT_RETURN
dat_collective_scatterv(
      IN DAT_COLLECTIVE_HANDLE      collective_handle,
      IN DAT_PVOID                  *send_buffers,
      IN DAT_COUNT                  *send_byte_counts,
      IN DAT_PVOID                  recv_buffer,
      IN DAT_COUNT                  recv_byte_count,
      IN DAT_COLLECTIVE_RANK        root,
      IN DAT_CONTEXT                user_context,
      IN DAT_COMPLETION_FLAGS       completion_flags
      );
```

This asynchronous call performs a non-uniform scatter of the data specified by the *send_buffers* array argument to the collective group specified by *collective._handle*. The *send_buffers* array contains one buffer pointer for each member of the collective group, in rank order. The *send_byte_counts* array contains a byte count for each corresponding send buffer pointer. The *recv_buffer* and *recev_byte_count* arguments specify where received portions of the scatter are to be received. An "in place" transfer for the root rank can be indicated by passing NULL as the *recv_buffer* argument. The *send_buffers* and *send_byte_counts* arguments are ignored on non-root members. The operation is completed on the collective EVD unless completions are suppressed through the completion flags.

Endpoint	EVD	Extension Type	Extension Event Data Type
All	Receive	DAT_IB_COLLECTIVE_SCATTERV_STATUS	N/A

3.3.6 dat_ib_collective_gather()

```
DAT_RETURN
dat_collective_gather(
      IN DAT_COLLECTIVE_HANDLE      collective_handle,
      IN DAT_PVOID                   send_buffer,
      IN DAT_COUNT                   send_byte_count,
      IN DAT_PVOID                   recv_buffer,
      IN DAT_COUNT                   recv_byte_count,
      IN DAT_COLLECTIVE_RANK         root,
      IN DAT_CONTEXT                 user_context,
      IN DAT_COMPLETION_FLAGS        completion_flags
      );
```

This asynchronous call performs a gather of the data sent by all members of the collective specified by the *collective_handle* argument. The data to be sent is specified by the *send_buffer* and *send_byte_count* arguments. Data is received by the collective member specified by the *root* argument in the buffer specified by the *recv_buffer* and *recv_byte_count* arguments. Data is placed into the receive buffer in collective rank order. An "in place" transfer for the root rank can be indicated by passing NULL as the *send_buffer* argument. The *recv_buffer* and *recv_byte_count* arguments are ignored on non-root members. The operation is completed on the collective EVD unless completions are suppressed through the completion flags. The tables below show the result of a gather operation.

	Send Buffer Elements for all Members		
Members	A[0]	n/a	n/a
	B[0]	n/a	n/a
	C[0]	n/a	n/a

	Root Receive Buffer Elements on Completion		
Members	A[0]	B[0]	C[0]
	n/a	n/a	n/a
	n/a	n/a	n/a

Endpoint	**EVD**	**Extension Type**	**Extension Event Data Type**
All	Collective	DAT_IB_COLLECTIVE_GATHER_STATUS	N/A

3.3.7 dat_ib_collective_gatherv()

```
DAT_RETURN
dat_collective_gatherv(
      IN DAT_COLLECTIVE_HANDLE      collective_handle,
      IN DAT_PVOID                   send_buffer,
      IN DAT_COUNT                   send_byte_count,
      IN DAT_PVOID                   *recv_buffers,
      IN DAT_COUNT                   *recv_byte_counts,
      IN DAT_COLLECTIVE_RANK         root,
      IN DAT_CONTEXT                 user_context,
      IN DAT_COMPLETION_FLAGS        completion_flags
      );
```

This asynchronous call performs a non-uniform gather of the data sent by all members of the collective specified by the *collective_handle* argument. The data to be sent is specified by the *send_buffer* and *send_byte_count* arguments. Data is received by the collective member specified by the *root* argument into the buffers specified by the *recv_buffers* and *recv_byte_counts* array arguments. Data is placed into the receive buffer associated with the rank that sent it. An "in place" transfer for the root rank can be indicated by passing NULL as the *send_buffer* argument. The *recv_buffers* and *recv_byte_counts* arguments are ignored on non-root members. The operation is completed on the collective EVD unless completions are suppressed through the completion flags.

Endpoint	EVD	Extension Type	Extension Event Data Type
All	Collective	DAT_IB_COLLECTIVE_GATHERV_STATUS	N/A

3.3.8 dat_ib_collective_allgather()

```
DAT_RETURN
dat_collective_allgather(
      IN DAT_COLLECTIVE_HANDLE      collective_handle,
      IN DAT_PVOID                   send_buffer,
      IN DAT_COUNT                   send_byte_count,
      IN DAT_PVOID                   recv_buffer,
      IN DAT_COUNT                   recv_byte_count,
      IN DAT_CONTEXT                 user_context,
      IN DAT_COMPLETION_FLAGS        completion_flags
      );
```

This asynchronous call is equivalent to having all members of a collective group perform a `dat_collective_gather()` as the root. This results in all members of the collective having identical contents in their receive buffer. Refer to section 3.3.6 for the semantics of the gather API. The tables below show the result of an allgather operation.

	Send Buffer Elements for all Members		
Members	A[0]	n/a	n/a
	B[0]	n/a	n/a
	C[0]	n/a	n/a

	Receive Buffer Elements for all Members on Completion		
Members	A[0]	B[0]	C[0]
	A[0]	B[0]	C[0]
	A[0]	B[0]	C[0]

Endpoint	EVD	Extension Type	Extension Event Data Type
All	collective	DAT_IB_COLLECTIVE_ALLGATHER_STATUS	N/A

3.3.9 dat_ib_collective_allgatherv()

```
DAT_RETURN
dat_collective_allgatherv(
      IN DAT_COLLECTIVE_HANDLE      collective_handle,
      IN DAT_PVOID                  send_buffer,
      IN DAT_COUNT                  send_byte_count,
      IN DAT_PVOID                  *recv_buffers,
      IN DAT_COUNT                  *recv_byte_counts,
      IN DAT_CONTEXT                user_context,
      IN DAT_COMPLETION_FLAGS       completion_flags
      );
```

This `asynchronous` call performs a non-uniform `dat_collective_allgather()` operation. It is equivalent to having all members of a collective group perform a `dat_collective_gatherv()` as the root. This results in all members of the collective having identical contents in their receive buffer. Refer to section 3.3.7 for the semantics of the gatherv API.

Endpoint	EVD	Extension Type	Extension Event Data Type
All	collective	DAT_IB_COLLECTIVE_ALLGATHERV_STATUS	N/A

3.3.10 dat_ib_collective_alltoall()

```
DAT_RETURN
dat_collective_alltoall(
            IN DAT_COLLECTIVE_HANDLE     collective_handle,
            IN DAT_PVOID                  send_buffer,
            IN DAT_COUNT                  send_byte_count,
            IN DAT_PVOID                  recv_buffer,
            IN DAT_COUNT                  recv_byte_count,
            IN DAT_CONTEXT                user_context,
            IN DAT_COMPLETION_FLAGS       completion_flags
            );
```

This asynchronous call is an extension of *dat_collective_allgather()* to the case where each member sends distinct data specified by *send_buffer* to each of the other members. The *jth* block sent from rank *i* is received by rank *j* and is placed in the *ith* block of *recv_buffer*. The tables below show the result of an alltoall operation.

	Send Buffer Elements for all Members		
Members	A[0]	A[1]	A[2]
	B[0]	B[1]	B[2]
	C[0]	C[1]	C[2]

	Receive Buffer Elements for all Members on Completion		
Members	A[0]	B[0]	C[0]
	A[1]	B[1]	C[1]
	A[2]	B[2]	C[2]

Endpoint	**EVD**	**Extension Type**	**Extension Event Data Type**
All	collective	DAT_IB_COLLECTIVE_ALLTOALL_STATUS	N/A

3.3.11 dat_ib_collective_alltoallv()

```
DAT_RETURN
dat_collective_alltoallv(
      IN DAT_COLLECTIVE_HANDLE      collective_handle,
      IN DAT_PVOID                 *send_buffers,
      IN DAT_COUNT                 *send_byte_counts,
      IN DAT_PVOID                 *recv_buffers,
      IN DAT_COUNT                 *recv_byte_counts,
      IN DAT_CONTEXT               user_context,
      IN DAT_COMPLETION_FLAGS      completion_flags
      );
```

This asynchronous call performs a non-uniform `dat_collective_alltoall()` operation. Refer to the section 3.3.10 for the semantics of the alltoall API.

Endpoint	EVD	Extension Type	Extension Event Data Type
All	collective	DAT_IB_COLLECTIVE_ALLTOALLV_STATUS	N/A

3.3.12 dat_ib_collective_reduce()

```
DAT_RETURN
dat_collective_reduce(
      IN DAT_COLLECTIVE_HANDLE      collective_handle,
      IN DAT_PVOID                  send_buffer,
      IN DAT_COUNT                  send_byte_count,
      IN DAT_PVOID                  recv_buffer,
      IN DAT_COUNT                  recv_byte_count,
      IN DAT_REDUCE_OP              reduce_operation,
      IN DAT_COLLECTIVE_DATA_TYPE   data_type,
      IN DAT_COLLECTIVE_RANK        root,
      IN DAT_CONTEXT                user_context,
      IN DAT_COMPLETION_FLAGS            completion_flags
      );
```

This asynchronous call combines the elements of the data type specified by *data_type* from the buffer specified by *send_buffer* of all members of the collective by performing the operation specified by *reduce_operation* and placing the result into the buffer of the root member specified by *recv_buffer*. It is an error to specify a floating point type with any of the logical reduction operators.

When using the REDUCE_OP_MINLOC and REDUCE_OP _MAXLOC operations, it is assumed that the input and output buffers contain pair values where the first member of the pair is of the type specified by *data_type* followed by a COLLECTIVE_TYPE_UINT32 type. When the reduction is complete, the receive buffer will contain the MIN/MAX value in the first member of the pair with the first member rank that contained it in the second member of the pair. The tables below show the result of a REDUCE_OP_SUM reduce operation.

	Send Data Buffer Elements

Members			
	A[0]	A[1]	A[2]
	B[0]	B[1]	B[2]
	C[0]	C[1]	C[2]

	Receive Buffer Elements for Members on Completion		
Members	A[0]+B[0]+C[0]	A[1]+B[1]+C[1]	A[2]+B[2]+C[2]
	n/a	n/a	n/a
	n/a	n/a	n/a

Endpoint	EVD	Extension Type	Extension Event Data Type
All	Collective	DAT_IB_COLLECTIVE_REDUCE_STATUS	N/A

3.3.13 dat_ib_collective_allreduce()

```
DAT_RETURN
dat_collective_reduce(
      IN DAT_COLLECTIVE_HANDLE      collective_handle,
      IN DAT_PVOID                   send_buffer,
      IN DAT_COUNT                   send_byte_count,
      IN DAT_PVOID                   recv_buffer,
      IN DAT_COUNT                   recv_byte_count,
      IN DAT_REDUCE_OP               reduce_operation,
      IN DAT_COLLECTIVE_DATA_TYPE    data_type,
      IN DAT_CONTEXT                 user_context,
      IN DAT_COMPLETION_FLAGS        completion_flags
      );
```

This asynchronous call is identical to the `dat_collective_reduce()` call with the exception that the *`recv_buffer`* and *`recv_byte_count`* arguments are valid for all members of the collective and all members of will receive the reduction results. The tables below show the result of a `REDUCE_OP_SUM` allreduce operation. Refer to section 3.3.12 for the semantics of the reduce API.

	Send Buffer Elements		
Members	A[0]	A[1]	A[2]
	B[0]	B[1]	B[2]
	C[0]	C[1]	C[2]

	Receive Buffer Elements for Members on Completion		
Members	A[0]+B[0]+C[0]	A[1]+B[1]+C[1]	A[2]+B[2]+C[2]
	A[0]+B[0]+C[0]	A[1]+B[1]+C[1]	A[2]+B[2]+C[2]
	A[0]+B[0]+C[0]	A[1]+B[1]+C[1]	A[2]+B[2]+C[2]

Endpoint	EVD	Extension Type	Extension Event Data Type
All	Collective	DAT_IB_COLLECTIVE_ALLREDUCE_STATUS	N/A

3.3.14 dat_ib_collective_reduce_scatter()

```
DAT_RETURN
dat_collective_reduce_scatter(
      IN DAT_COLLECTIVE_HANDLE      collective_handle,
      IN DAT_PVOID                   send_buffer,
      IN DAT_COUNT                   send_byte_count,
      IN DAT_PVOID                   recv_buffer,
      IN DAT_COUNT                   *recv_byte_counts,
      IN DAT_REDUCE_OP               reduce_operation,
      IN DAT_COLLECTIVE_DATA_TYPE    data_type,
      IN DAT_CONTEXT                 user_context,
      IN DAT_COMPLETION_FLAGS        ompletion_flags
      );
```

This asynchronous call is identical to rank 0 of the collective calling `dat_collective_reduce()` followed by dat_collective_scatterv(). The number of bytes received in the scatter for each rank is determined by rank offset into the `recv_byte_counts` array. The tables below show the result of a `REDUCE_OP_SUM` reduce scatter operation.

	Send Data Buffer Elements		
Members	A[0]	A[1]	A[2]
	B[0]	B[1]	B[2]
	C[0]	C[1]	C[2]

	Receive Buffer Elements for Members on Completion		
Members	A[0]+B[0]+C[0]	n/a	n/a
	A[1]+B[1]+C[1]	n/a	n/a
	A[2]+B[2]+C[2]	n/a	n/a

Endpoint	EVD	Extension Type	Extension Event Data Type
All	Collective	DAT_IB_COLLECTIVE_REDUCE_SCATTER_STATUS	N/A

3.3.15 dat_ib_collective_scan()

```
DAT_RETURN
dat_collective_scan (
      IN DAT_COLLECTIVE_HANDLE      collective_handle,
      IN DAT_PVOID                   send_buffer,
      IN DAT_COUNT                   send_byte_count,
      IN DAT_PVOID                   recv_buffer,
      IN DAT_COUNT                   recv_byte_count,
      IN DAT_REDUCE_OP               reduce_operation,
      IN DAT_COLLECTIVE_DATA_TYPE    data_type,
      IN DAT_CONTEXT                 user_context,
      IN DAT_COMPLETION_FLAGS        completion_flags
      );
```

This asynchronous call is used to perform a prefix reduction on data distributed across the group. The operation returns, in *recv_buffer* of the member with rank *i*, the reduction of the values in *send_buffer* of members with ranks *0,...,i* (inclusive). The tables below show the result of a REDUCE_OP_SUM scan operation.

	Send Data Buffer Elements		
Members	A[0]	A[1]	A[2]
	B[0]	B[1]	B[2]
	C[0]	C[1]	C[2]

	Receive Buffer Elements for Members on Completion		
Members	A[0]	**A**[1]	A[2]
	A[0]+B[0]	A[1]+B[1]	A[2]+B[2]
	A[0]+B[0]+C[0]	A[1]+B[1]+C[1]	A[2]+B[2]+C[2]

Endpoint	EVD	Extension Type	Extension Event Data Type
All	Collective	DAT_IB_COLLECTIVE_SCAN_STATUS	N/A

3.3.16 dat_ib_collective_barrier()

```
DAT_RETURN
dat_collective_barrier(
      IN DAT_COLLECTIVE_HANDLE      collective_handle,
      IN DAT_CONTEXT                user_context,
      IN DAT_COMPLETION_FLAGS       completion_flags
      );
```

This call will synchronize all endpoints of the collective group specified by *collective_handle*. This is an asynchronous call that will post a completion to the collective EVD when all endpoints have synchronized.

Endpoint	EVD	Extension Type	Extension Event Data Type
All	Collective	DAT_IB_COLLECTIVE_BARRIER_STATUS	N/A