

Unified Collective Communications (UCC) Specification

Version 1.1



Contents

1	Unified Collective Communications (UCC) Library Specification	1
2	Design	2
2.0.1	Component Diagram	2
3	Library Initialization and Finalization	3
4	Communication Context	4
5	Teams	5
6	Types of Collective Operations	7
7	Execution Engine and Events	12
7.0.1	Triggered Operations	12
7.0.2	Interaction between an User Thread and Event-driven UCC	12
8	Module Documentation	14
8.1	Library initialization data-structures	14
8.1.1	Detailed Description	15
8.1.2	Data Structure Documentation	15
8.1.2.1	struct ucc_lib_params	15
8.1.2.2	struct ucc_lib_attr	16
8.1.3	Typedef Documentation	16
8.1.3.1	ucc_lib_params_t	16
8.1.3.2	ucc_lib_attr_t	16
8.1.3.3	ucc_lib_h	17
8.1.3.4	ucc_lib_config_h	17
8.1.4	Enumeration Type Documentation	17
8.1.4.1	ucc_coll_type_t	17
8.1.4.2	ucc_reduction_op_t	18
8.1.4.3	ucc_thread_mode_t	18
8.1.4.4	ucc_coll_sync_type_t	18
8.1.4.5	ucc_lib_params_field	19
8.1.4.6	ucc_lib_attr_field	19
8.2	Datatypes data-structures and functions	19
8.2.1	Detailed Description	20
8.2.2	Data Structure Documentation	20
8.2.2.1	struct ucc_reduce_cb_params	20
8.2.3	Typedef Documentation	21
8.2.3.1	ucc_datatype_t	21
8.2.3.2	ucc_reduce_cb_params_t	21
8.2.3.3	ucc_generic_dt_ops_t	21
8.2.4	Enumeration Type Documentation	21
8.2.4.1	ucc_generic_dt_ops_field	21
8.2.4.2	ucc_generic_dt_ops_flags_t	21
8.2.5	Function Documentation	22

8.2.5.1	ucc_dt_create_generic()	22
8.2.5.2	ucc_dt_destroy()	22
8.2.6	Variable Documentation	22
8.2.6.1	start_pack	22
8.2.6.2	start_unpack	23
8.2.6.3	packed_size	23
8.2.6.4	pack	23
8.2.6.5	unpack	24
8.2.6.6	finish	24
8.2.6.7		24
8.2.6.8		24
8.2.6.9		24
8.3	Library initialization and finalization routines	25
8.3.1	Detailed Description	25
8.3.2	Function Documentation	25
8.3.2.1	ucc_lib_config_read()	25
8.3.2.2	ucc_lib_config_release()	26
8.3.2.3	ucc_lib_config_print()	26
8.3.2.4	ucc_lib_config_modify()	26
8.3.2.5	ucc_init()	26
8.3.2.6	ucc_finalize()	27
8.3.2.7	ucc_lib_get_attr()	27
8.4	Context abstraction data-structures	28
8.4.1	Detailed Description	28
8.4.2	Data Structure Documentation	29
8.4.2.1	struct ucc_mem_map	29
8.4.2.2	struct ucc_mem_map_params	30
8.4.2.3	struct ucc_context_params	30
8.4.2.4	struct ucc_context_attr	30
8.4.3	Typedef Documentation	30
8.4.3.1	ucc_oob_coll_t	31
8.4.3.2	ucc_mem_map_t	31
8.4.3.3	ucc_mem_map_params_t	31
8.4.3.4	ucc_context_params_t	31
8.4.3.5	ucc_context_attr_t	31
8.4.3.6	ucc_context_h	31
8.4.3.7	ucc_context_config_h	31
8.4.4	Enumeration Type Documentation	31
8.4.4.1	ucc_context_type_t	31
8.4.4.2	ucc_context_params_field	32
8.4.4.3	ucc_context_attr_field	32
8.5	Context abstraction routines	32
8.5.1	Detailed Description	33
8.5.2	Function Documentation	33
8.5.2.1	ucc_context_config_read()	33
8.5.2.2	ucc_context_config_release()	33
8.5.2.3	ucc_context_config_print()	33
8.5.2.4	ucc_context_config_modify()	34
8.5.2.5	ucc_context_create()	34
8.5.2.6	ucc_context_progress()	35
8.5.2.7	ucc_context_destroy()	35
8.5.2.8	ucc_context_get_attr()	35
8.6	Team abstraction data-structures	36
8.6.1	Detailed Description	37
8.6.2	Data Structure Documentation	37
8.6.2.1	struct ucc_ep_map_strided	37
8.6.2.2	struct ucc_ep_map_array	37
8.6.2.3	struct ucc_ep_map_t	37

8.6.2.4	struct ucc_team_params	37
8.6.2.5	struct ucc_team_attr	40
8.6.2.6	union ucc_ep_map_t. __unnamed2__	40
8.6.3	Typedef Documentation	40
8.6.3.1	ucc_team_p2p_conn_t	40
8.6.3.2	ucc_ep_map_t	40
8.6.3.3	ucc_team_params_t	40
8.6.3.4	ucc_team_attr_t	40
8.6.3.5	ucc_team_h	41
8.6.3.6	ucc_p2p_conn_t	41
8.6.3.7	ucc_context_addr_h	41
8.6.3.8	ucc_context_addr_len_t	41
8.6.4	Enumeration Type Documentation	41
8.6.4.1	ucc_team_params_field	41
8.6.4.2	ucc_team_attr_field	41
8.6.4.3	ucc_team_flags	42
8.6.4.4	ucc_post_ordering_t	42
8.6.4.5	ucc_ep_range_type_t	42
8.6.4.6	ucc_ep_map_type_t	43
8.7	Team abstraction routines	43
8.7.1	Detailed Description	43
8.7.2	Function Documentation	43
8.7.2.1	ucc_team_create_post()	43
8.7.2.2	ucc_team_create_test()	44
8.7.2.3	ucc_team_destroy()	44
8.7.2.4	ucc_team_get_attr()	45
8.7.2.5	ucc_team_create_from_parent()	45
8.7.2.6	ucc_team_get_size()	45
8.7.2.7	ucc_team_get_my_ep()	46
8.7.2.8	ucc_team_get_all_eps()	46
8.8	Collective operations data-structures	46
8.8.1	Detailed Description	47
8.8.2	Data Structure Documentation	47
8.8.2.1	struct ucc_coll_buffer_info_v	47
8.8.2.2	struct ucc_coll_buffer_info	48
8.8.3	Typedef Documentation	48
8.8.3.1	ucc_memory_type_t	48
8.8.3.2	ucc_coll_buffer_info_v_t	48
8.8.3.3	ucc_coll_buffer_info_t	48
8.8.3.4	ucc_coll_req_h	48
8.8.3.5	ucc_coll_callback_t	48
8.8.3.6	ucc_count_t	48
8.8.3.7	ucc_aint_t	49
8.8.3.8	ucc_coll_id_t	49
8.8.4	Enumeration Type Documentation	49
8.8.4.1	ucc_memory_type	49
8.8.4.2	ucc_coll_args_flags_t	49
8.8.4.3	ucc_error_type_t	50
8.8.4.4	ucc_coll_args_field	50
8.9	Collective Operations	50
8.9.1	Detailed Description	51
8.9.2	Data Structure Documentation	51
8.9.2.1	struct ucc_coll_args	51
8.9.2.2	union ucc_coll_args.src	51
8.9.2.3	union ucc_coll_args.dst	52
8.9.3	Typedef Documentation	52
8.9.3.1	ucc_coll_args_t	52
8.9.3.2	ucc_mem_h	52

8.9.4	Function Documentation	52
8.9.4.1	ucc_collective_init()	52
8.9.4.2	ucc_collective_post()	53
8.9.4.3	ucc_collective_init_and_post()	53
8.9.4.4	ucc_collective_test()	53
8.9.4.5	ucc_collective_finalize()	54
8.10	Events and Triggered operations' datastructures	54
8.10.1	Detailed Description	54
8.10.2	Data Structure Documentation	55
8.10.2.1	struct ucc_event	55
8.10.2.2	struct ucc_ee_params	55
8.10.3	Typedef Documentation	55
8.10.3.1	ucc_event_type_t	55
8.10.3.2	ucc_ee_type_t	55
8.10.3.3	ucc_ev_t	55
8.10.3.4	ucc_ee_params_t	55
8.10.4	Enumeration Type Documentation	55
8.10.4.1	ucc_event_type	55
8.10.4.2	ucc_ee_type	56
8.11	Events and Triggered Operations	56
8.11.1	Detailed Description	56
8.11.2	Function Documentation	56
8.11.2.1	ucc_ee_create()	56
8.11.2.2	ucc_ee_destroy()	57
8.11.2.3	ucc_ee_get_event()	57
8.11.2.4	ucc_ee_ack_event()	57
8.11.2.5	ucc_ee_set_event()	58
8.11.2.6	ucc_ee_wait()	58
8.11.2.7	ucc_collective_triggered_post()	59
8.12	Utility Operations	59
8.12.1	Detailed Description	59
8.12.2	Enumeration Type Documentation	59
8.12.2.1	ucc_config_print_flags_t	60
8.12.2.2	ucc_status_t	61
8.12.3	Function Documentation	61
8.12.3.1	ucc_status_string()	61
9	Data Structure Documentation	62
9.1	ucc_coll_callback Struct Reference	62
9.1.1	Detailed Description	62
9.1.2	Field Documentation	62
9.1.2.1	cb	62
9.1.2.2	data	62
9.2	ucc_ep_map_cb Struct Reference	62
9.2.1	Field Documentation	62
9.2.1.1	cb	62
9.2.1.2	cb_ctx	63
9.3	ucc_generic_dt_ops Struct Reference	63
9.3.1	Detailed Description	63
9.3.2	Field Documentation	63
9.3.2.1	mask	63
9.3.2.2	flags	63
9.3.2.3	contig_size	64
9.4	ucc_generic_dt_ops.reduce Struct Reference	64
9.4.1	Detailed Description	64
9.4.2	Field Documentation	64
9.4.2.1	cb	64
9.4.2.2	cb_ctx	64

9.5	ucc_oob_coll Struct Reference	64
9.5.1	Field Documentation	64
9.5.1.1	allgather	65
9.5.1.2	req_test	65
9.5.1.3	req_free	65
9.5.1.4	coll_info	65
9.5.1.5	n_oob_eps	65
9.5.1.6	oob_ep	65
9.6	ucc_team_p2p_conn Struct Reference	65
9.6.1	Field Documentation	65
9.6.1.1	conn_info_lookup	65
9.6.1.2	conn_info_release	65
9.6.1.3	conn_ctx	66
9.6.1.4	req_test	66
9.6.1.5	req_free	66
Index		67

Chapter 1

Unified Collective Communications (UCC) Library Specification

UCC is a collective communication operations API and library that is flexible, complete, and feature-rich for current and emerging programming models and runtimes.

Chapter 2

Design

- Highly scalable and performant collectives for HPC, AI/ML and I/O workloads
- Nonblocking collective operations that cover a variety of programming models
- Flexible resource allocation model
- Support for relaxed ordering model
- Flexible synchronous model
- Repetitive collective operations (init once and invoke multiple times)
- Hardware collectives are a first-class citizen

2.0.1 Component Diagram

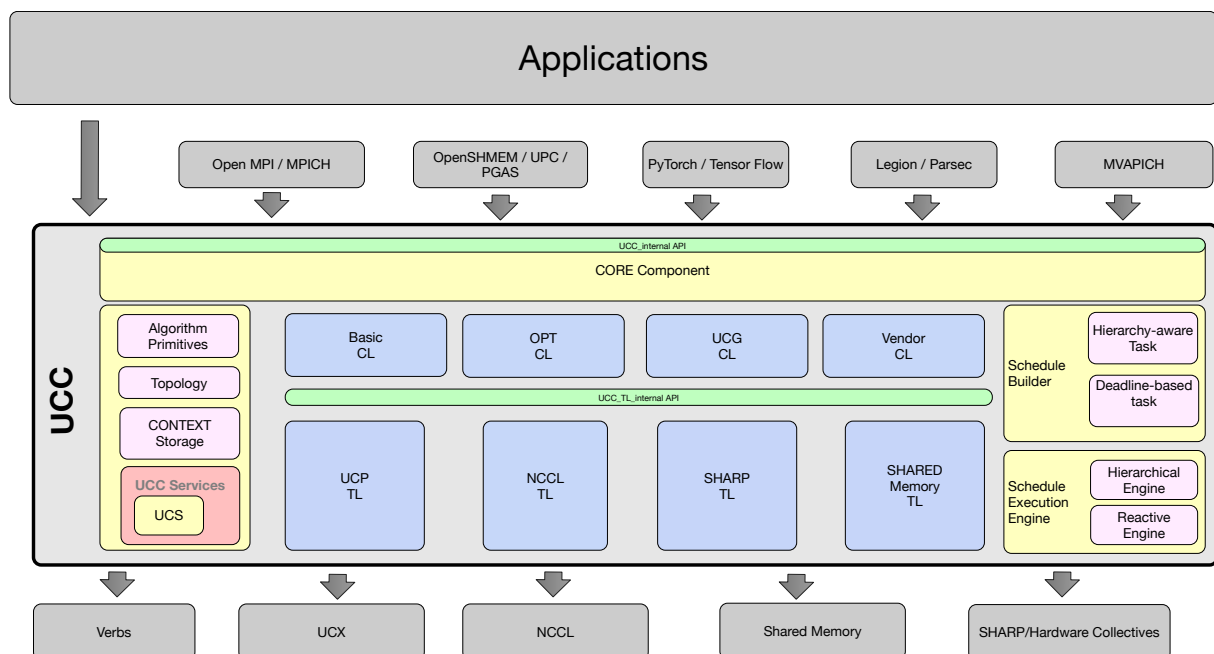


Figure 2.1: UCC Components and Usage

Chapter 3

Library Initialization and Finalization

These routines are responsible for allocating, initializing, and finalizing the resources for the library.

The UCC can be configured in three thread modes `UCC_THREAD_SINGLE`, `UCC_THREAD_FUNNELED`, and `UCC_LIB_THREAD_MULTIPLE`. In the `UCC_THREAD_SINGLE` mode, the user program must not be multithreaded. In the `UCC_THREAD_FUNNELED` mode, the user program may be multithreaded. However, all UCC interfaces should be invoked from the same thread. In the `UCC_THREAD_MULTIPLE` mode, the user program can be multithreaded and any thread may invoke the UCC operations.

The user can request different types of collective operations that vary in their synchronization models. The valid synchronization models are `UCC_NO_SYNC_COLLECTIVES` and `UCC_SYNC_COLLECTIVES`. The details of these synchronization models are described in the collective operation section.

The user can request the different collective operations and reduction operations required. The complete set of valid collective operations and reduction types are defined with the structures `ucc_coll_type_t` and `ucc_reduction_op_t`.

Chapter 4

Communication Context

The `ucc_context_h` is a communication context handle. It can encapsulate resources required for collective operations on team handles. The contexts are created by the `ucc_context_create` operation and destroyed by the `ucc_context_destroy` operation. The create operation takes in user-configured `ucc_context_params_t` structure to customize the context handle. The attributes of the context created can be queried using the `ucc_context_get_attrbs` operation.

When no out-of-band operation (OOB) is provided, the `ucc_context_create` operation is local requiring no communication with other participants. When OOB operation is provided, all participants of the OOB operation should participate in the create operation. If the context operation is a collective operation, the `ucc_context_destroy` operation is also a collective operation .i.e., all participants should call the destroy operation.

The context can be created as an exclusive type or shared type by passing constants `UCC_CONTEXT_EXCLUSIVE` and `UCC_CONTEXT_SHARED` respectively to the `ucc_context_params_t` structure. When context is created as a shared type, the same context handle can be used to create multiple teams. When context is created as an exclusive type, the context can be used to create multiple teams but the team handles cannot be valid at the same time; a valid team is defined as a team object where the user can post collective operations.

Notes : From the user perspective, the context handle represents a communication resource. The user can create one context and use it for multiple teams or use with a single team. This provides a finer control of resources for the user. From the library implementation perspective, the context could represent the network parallelism. The UCC library implementation can choose to abstract injection queues, network endpoints, GPU device context, UCP worker, or UCP endpoints using the communication context handles.

Chapter 5

Teams

The `ucc_team_h` is a team handle, which encapsulates the resources required for group operations such as collective communication operations. The participants of the group operations can either be an OS process, a control thread or a task.

Create and destroy routines: `ucc_team_create_post` routine is used to create the team handle and `ucc_team_create_test` routine for learning the status of the create operation. The team handle is destroyed by the `ucc_team_destroy` operation. A team handle is customized using the user configured `ucc_team_params_t` structure.

Invocation semantics: The `ucc_team_create_post` is a nonblocking collective operation, in which the participants are determined by the user-provided OOB collective operation. Overlapping of multiple `ucc_team_create_post` operations are invalid. Posting a collective operation before the team handle is created is invalid. The team handle is destroyed by a blocking collective operation; the participants of this collective operation are the same as the create operation. When the user does not provide an OOB collective operation, all participants calling the `ucc_create_post` operation will be part of a new team created.

Communication Contexts: Each process or a thread participating in the team creation operation contributes one or more communication contexts to the operation. The number of contexts provided by all participants should be the same and each participant should provide the same type of context. The newly created team uses the context for collective operations. If the communication context abstracts the resources for the library, the collective operations on this team uses the resources provided by the context.

Endpoints: That participants to the `ucc_team_create_post` operation can provide an endpoint, a 64-bit unsigned integer. The endpoint is an address for communication. Each participant of the team has a unique integer as endpoint .i.e., the participants of the team do not share the same endpoint. For example, the user can bind the endpoint to the parallel programming model's index such as OpenSHMEM PE, an OS process ID, or a thread ID. The UCC implementation can use the endpoint as an index to identify the resources required for communication such as communication contexts. When the user does not provide the endpoint, the library generates the endpoint, which can be queried by the user. In addition to the endpoint, the user can provide information about the endpoints such as whether the endpoint is a continuous range or not.

Ordering: The collective operations on the team can either be ordered or unordered. In the ordered model, the UCC collectives are invoked in order .i.e., on a given team, each of the participants of the collective operation invokes the operation in the same order. In the unordered model, the collective operations are not necessarily invoked in the same order.

Interaction with Threads: The team can be created in either mode .i.e., the library initialized by `UCC_LIB_THREAD_MULTIPLE`, `UCC_LIB_THREAD_SINGLE`, or `UCC_LIB_THREAD_FUNNELED`. In the `UCC_LIB_THREAD_MULTIPLE` mode, each of the user threads can post a collective operation. However, it is not valid to post concurrent collectives operations from multiple threads to the same team.

Memory per Team: A team can be configured by a memory descriptor described by `ucc_mem_map_params_t` structure. The memory can be used as an input and output buffers for the collective operation. This is particularly useful for PGAS programming models, where the input and output buffers are defined before the invocation operation. For example, the input and output buffers in the OpenSHMEM programming model are defined during the programming model initialization.

Synchronization Model: The team can be configured to support either synchronized collectives or non-synchronized collectives. If the UCC library is configured with synchronized collective operations and the team is configured with non-synchronized collective operations, the library might not be able to provide any optimizations and might support only synchronized collective operations.

Outstanding Calls: The user can configure maximum number of outstanding collective operations of any type for a given team. This is represented by an unsigned integer. This is provided as a hint to the library for resource management.

Team ID: The team identifier is a unique 64-bit unsigned integer for the given process .i.e, the team identifier should be unique for all teams it creates or participates. If the team identifier is provided by the user, it should be passed as a configuration parameter to the team create operation.

Split Team Operations

The team split routines provide an alternate way to create teams. All split routines require a parent team and all participants of the parent team call the split operation. The participants of the new team may include some or all participants of the parent team.

The newly created team shares the communication contexts with the parent team. The endpoint of the new team is contiguous and is not related to the parent team. It inherits the thread model, synchronization model, collective ordering model, outstanding collectives configuration, and memory descriptor from the parent team.

The split operation can be called by multiple threads, if the parent team to the split operations are different and if it agrees with the thread model of the UCC library.

Notes: The rationale behind requiring all participants of the parent team to participate in the split operation is to avoid overlapping participants between multiple split operations, which is known to increase the implementation complexity. Also, currently, higher-level programming models do not require these semantics.

Chapter 6

Types of Collective Operations

A UCC collective operation is a group communication operation among the participants of the team. All participants of the team are required to call the collective operation. Each participant is represented by the endpoint that is unique to the team used for the collective operation. This section provides a set of routines for launching, progressing, and completing the collective operations.

Invocation semantics: The `ucc_collective_init` routine is a non-blocking collective operation to initialize the buffers, operation type, reduction type, and other information required for the collective operation. All participants of the team should call the initialize operation. The collective operation is invoked using a `ucc_collective_post` operation. `ucc_collective_init_and_post` operation initializes as well as post the collective operation.

Collective Type: The collective operation supported by UCC is defined by the enumeration `ucc_coll_type_t`. The semantics are briefly described here, however in most cases it agrees with the semantics of collective operations in the popular programming models such as MPI and OpenSHMEM. When they differ, the semantics changes are documented. All collective operations execute on the team. For the collective operations defined by `ucc_coll_type_t`, all participants of the team are required to participate in the collective operations. Further the team should be created with endpoints, where the “eps” should be ordered and contiguous.

UCC supports three types of collective operations: (a) `UCC_{ALLTOALL, ALLTOALLV, ALLGATHER, ALLGATHERV, ALLREDUCE, REDUCE_SCATTER, REDUCE_SCATTERV, BARRIER}` operations where all participants contribute to the results and receive the results (b) `UCC_{REDUCE, GATHER, GATHERV, FANIN}` where all participants contribute to the result and one participant receives the result. The participant receiving the result is designated as root. (c) `UCC_{BROADCAST, SCATTER, SCATTERV, FANOUT}` where one participant contributes to the result, and all participants receive the result. The participant contributing to the result is designated as root.

- The `UCC_COLL_TYPE_BCAST` operation moves the data from the root participant to all participants in the team.
- The `UCC_COLL_TYPE_BARRIER` synchronizes all participants of the collective operation. In this routine, first, each participant waits for all other participants to enter the operation. Then, once it learns the entry of all other participants into the operation, it exits the operation completing it locally.
- In the `UCC_COLL_TYPE_FAN_IN` operation, the root participant synchronizes with all participants of the team. The non-root completes when it sends synchronizing message to the root. Unlike `UCC_COLL_TYPE_BARRIER`, it doesn't have to synchronize with the rest of the non-root participants. The root participant completes the operation when it receives synchronizing messages from all non-root participants of the team.
- The `UCC_COLL_TYPE_FAN_OUT` operation is a synchronizing operation like `UCC_COLL_TYPE_FAN_IN`. In this operation, the root participant sends a synchronizing message to all non-root participants and completes. The non-root participant completes once it receives a message from the root participant.
- In the `UCC_COLL_TYPE_GATHER` operation, each participant of the collective operation sends data to the root participant. All participants send the same amount of data (`block_size`) to the root. The

size of the block is $\text{dt_elem_size} * \text{count}$. The total amount of data received by the root is equal to $\text{block_size} * \text{num_participants}$. Here, the "count" represents the number of data elements. The "dt_elem_size" represents the size of the data element in bytes. The "num_participants" represents the number of participants in the team. The data on the root is placed in the receive buffer ordered by the "ep" ordering. For example, if the participants' endpoints are ordered as "ep_a" to "ep_n", the data from the participant with ep_i is placed as an "ith" block on the receive buffer.

- The UCC_COLL_TYPE_ALLGATHER operation is similar to UCC_COLL_TYPE_GATHER with one exception. Unlike in GATHER operation, the result is available at all participants' receive buffer instead of only at the root participant.

Each participant sends the data of size "block_size" to all other participants in the collective operation. The size of the block is $\text{dt_elem_size} * \text{count}$. Here, the "count" represents the number of data elements. The "dt_elem_size" represents the size of the data element in bytes. The data on each participant is placed in the receive buffer ordered by the "ep" ordering. For example, if the participants' endpoints are ordered as "ep_a" to "ep_n", the data from the participant with ep_i is placed as an "ith" block on the receive buffer.

- In the UCC_COLL_TYPE_SCATTER operation, the root participant of the collective operation sends data to all other participants. It sends the same amount of data (block_size) to all participants. The size of the block (block_size) is $\text{dt_elem_size} * \text{count}$. The total amount of data sent by the root is equal to $\text{block_size} * \text{num_participants}$. Here, the "count" represents the number of data elements. The "dt_elem_size" represents the size of the data element in bytes. The "num_participants" represents the number of participants in the team.
- In the UCC_COLL_TYPE_ALLTOALL collective operation, all participants exchange a fixed amount of the data. For a given participant, the size of data in src buffer is "size", where $\text{size} = \text{dt_elem_size} * \text{count} * \text{num_participants}$. Here, the "count" represents the number of data elements per destination. The "dt_elem_size" represents the size of the data element in bytes. The "num_participants" represents the number of participants in the team. The size of src buffer is the same as the dest buffer, and it is the same across all participants. Each participant exchanges $\text{dt_elem_size} * \text{count}$ data with every participant of the collective.
- In UCC_COLL_TYPE_REDUCE collective the element-wise reduction operation is performed on the src buffer of all participants in the collective operation. The result is stored on the dst buffer of the root. The size of src buffer and dst buffer is the same, which is equal to $\text{dt_elem_size} * \text{count}$. Here, the "count" represents the number of data elements. The "dt_elem_size" represents the size of the data element in bytes.
- The UCC_COLL_TYPE_ALLREDUCE first performs an element-wise reduction on the src buffers of all participants. Then the result is distributed to all participants. After the operation, the results are available on the dst buffer of all participants. The size of src buffer and dst buffer is the same for all participants. The size of src buffer and dst buffer is the same, which is equal to $\text{dt_elem_size} * \text{count}$. Here, the "count" represents the number of data elements. The "dt_elem_size" represents the size of the data element in bytes.
- The UCC_COLL_TYPE_REDUCE_SCATTER first performs an element-wise reduction on the src buffer and then scatters the result to the dst buffer. The "size" of src buffer is $\text{count} * \text{dt_elem_size}$, where dt_elem_size is the number of bytes for the data type element and count is the number of elements of that datatype. It is the user's responsibility to ensure that data and the result are equally divisible among the participants. Assuming that the result is divided into "n" blocks, the ith block is placed in the receive buffer of endpoint "i". Like other collectives, for this collective, the "ep" should be ordered and contiguous.

INPLACE: When INPLACE is set for UCC_COLL_TYPE_REDUCE_SCATTER, UCC_COLL_TYPE_REDUCE, UCC_COLL_TYPE_ALLREDUCE, UCC_COLL_TYPE_SCATTER, and UCC_COLL_TYPE_ALLTOALL the receive buffers act as both send and receive buffer.

For UCC_COLL_TYPE_BCAST operation, setting INPLACE flag has no impact.

The "v" Variant Collective Types: The UCC_COLL_TYPE_{ALLTOALLV, SCATTERV, GATHERV, and REDUCE_SCATTERV} operations add flexibility to their counter parts (i.e., ALLTOALL, SCATTER,

GATHER, and REDUCE_SCATTER) in that the location of data for the send and receive are specified by displacement arrays.

Reduction Types: The reduction operation supported by UCC_{ALLREDUCE, REDUCE, REDUCE_SCATTER, REDUCE_SCATTERV} operation is defined by the enumeration ucc_reduction_op_t. The valid datatypes for the reduction is defined by the enumeration ucc_datatype_t.

Ordering: The team can be configured for ordered collective operations or unordered collective operations. For unordered collectives, the user is required to provide the “tag”, which is an unsigned 64-bit integer.

Synchronized and Non-Synchronized Collectives: In the synchronized collective model, on entry, the participants cannot read or write to other participants without ensuring all participants have entered the collective operation. On the exit of the collective operation, the participants may exit after all participants have completed the reading or writing to the buffers.

In the non-synchronized collective model, on entry, the participants can read or write to other participants. If the input and output buffers are defined on the team and RMA operations are used for data transfer, it is the responsibility of the user to ensure the readiness of the buffer. On exit, the participants may exit once the read and write to the local buffers are completed.

Buffer Ownership: The ownership of input and output buffers are transferred from the user to the library after invoking the ucc_collective_init routine. On return from the routine, the ownership is transferred back to the user on ucc_collective_finalize. However, after invoking and returning from ucc_collective_post or ucc_collective_init_and_post routines, the ownership stays with the library and it is returned to the user, when the collective is completed.

The table below lists the necessary fields that user must initialize depending on the collective operation type.

			allgather	allgatherv	allreduce	alltoall	alltoallv	barrier	bcast	fanin	fanout
SRC	info	buffer	✓	✓	✓	✓			✓		
		count	✓	✓	✓	✓			✓		
		datatype	✓	✓	✓	✓			✓		
		mem_type	✓	✓	✓	✓			✓		
	info_v	buffer					✓				
		counts					✓				
		displacements					✓				
		datatype					✓				
	mem_type					✓					
DST	info	buffer	✓		✓	✓					
		count	✓		✓	✓					
		datatype	✓		✓	✓					
		mem_type	✓		✓	✓					
	info_v	buffer		✓			✓				
		counts		✓			✓				
		displacements		✓			✓				
		datatype		✓			✓				
	mem_type		✓			✓					
root								✓	✓	✓	
INPLACE			src is ignored	src is ignored	src is ignored	src is ignored	src is ignored	N/A	N/A	N/A	N/A
comments											

			gather	gatherv	reduce	reduce_scatter	reduce_scatterv	scatter	scatterv
SRC	info	buffer	v	v	v	v	v	v	
		count	v	v	v	v	v	v	
		datatype	v	v	v	v	v	v	
		mem_type	v	v	v	v	v	v	
	info_v	buffer							v
		counts							v
		displacements							v
		datatype							v
DST	info	buffer	v		v	v		v	v
		count	v		v	v		v	v
		datatype	v		v	v		v	v
		mem_type	v		v	v		v	v
	info_v	buffer		v			v		
		counts		v			v		
		displacements		v					
		datatype		v			v		
root			v	v	v			v	v
	INPLACE		src is ignored at root	src is ignored at root	src is ignored at root	src is ignored	src is ignored	dst is ignored at root	dst is ignored at root
	comments		dst only at root	dst only at root	dst only at root			src only at root	src only at root

Chapter 7

Execution Engine and Events

The execution engine is an execution context that supports event-driven network execution on the CUDA streams, CPU threads, and DPU threads. It is intended to interact with execution threads that are asynchronous (offloaded collective execution) which can be implemented on GPUs, DPUs, or remote CPUs.

UCC supports triggering collective operations by library-generated and user-generated events. The library events are generated on posting or completion of operations. The user-generated events include the completion of compute or communication operations. With a combination of library-generated and user-generated events, one can build dependencies between compute and collective operations, or between the collective operations.

Besides the execution engine, events are key for event-driven execution. The operations on the execution engines generate events that are stored internally on the execution engines. The valid events are defined by [ucc_event_type_t](#). If the underlying hardware doesn't support event-driven execution, the implementations can implement this with the event queues or lists.

The interaction between the user and library is through the UCC interfaces. [ucc_ee_create](#) creates execution engines. The user or library can generate an event and post it to the execution engines using [ucc_ee_set_event](#) interface. The user can wait on the events with the [ucc_ee_wait](#) interface. The user can get the event from the ee using [ucc_ee_get_event](#) interface and acknowledge the event with [ucc_ee_ack_event](#) interface. Once acknowledged, the library destroys the event.

Thread Mode: While in the UCC_THREAD_MULTIPLE mode, the execution engine and operations can be invoked from multiple threads.

Order: All non-triggered operations posted to the execution engine are executed in-order. However, there are no ordering guarantees between the execution engines.

7.0.1 Triggered Operations

Triggered operations enable the posting of operations on an event. For triggered operations, the team should be configured with event-driven execution. The collection operations is defined by the interface [ucc_collective_triggered_post](#).

The operations are launched on the event. So, there is no order established by the library. If user desires an order for the triggered operations, the user should provide the tag for matching the collective operations.

7.0.2 Interaction between an User Thread and Event-driven UCC

The figure shows the interaction between application threads and the UCC library configured with event-driven teams. In this example scenario, we assume that the UCC team are configured with two events queues - one for post operations and one for completions.

(1) The application initializes the collective operation when it knows the control parameters of the collective such as buffer addresses, lengths, and participants of the collective. The data need not be ready as it posts

the collective operation which will be triggered on an event. For example, the event here is the completion of compute by the application.

- (2) When the application completes the compute, it posts the `UCC_EVENT_COMPUTE_COMPLETE` event to the execution engine.
- (3) The library thread polls the event queue and triggers the operations that are related to the compute event.
- (4) The library posts the `UCC_EVENT_POST_COMPLETE` event to the event queue.
- (5) On completion of the collective operation, the library posts `UCC_EVENT_COLLECTIVE_COMPLETE` event to the completion event queue.

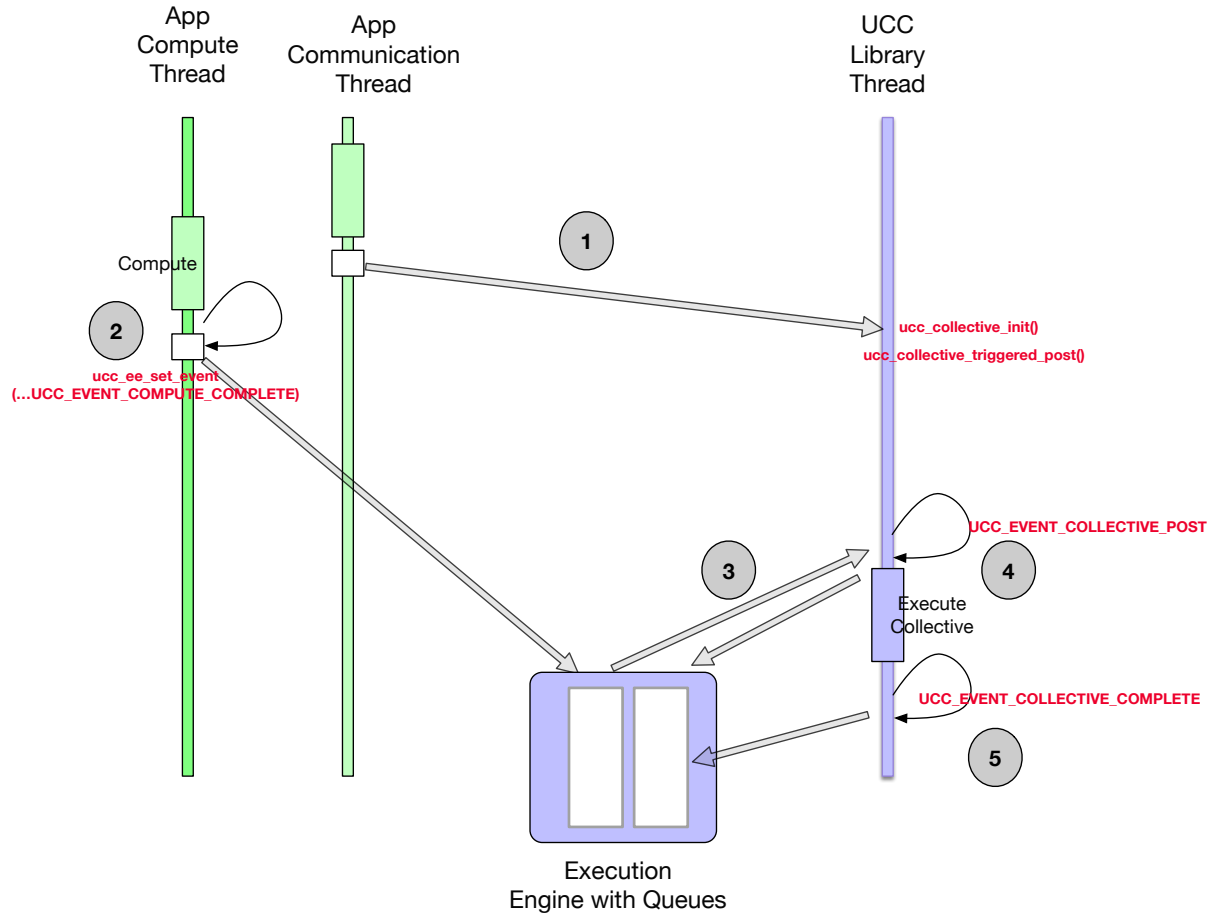


Figure 7.1: UCC Execution Engine and Events

Chapter 8

Module Documentation

8.1 Library initialization data-structures

Data Structures

- struct `ucc_lib_params`
Structure representing the parameters to customize the library. [More...](#)
- struct `ucc_lib_attr`
Structure representing the attributes of the library. [More...](#)

Typedefs

- typedef struct `ucc_lib_params` `ucc_lib_params_t`
Structure representing the parameters to customize the library.
- typedef struct `ucc_lib_attr` `ucc_lib_attr_t`
Structure representing the attributes of the library.
- typedef struct `ucc_lib_info` * `ucc_lib_h`
UCC library handle.
- typedef struct `ucc_lib_config` * `ucc_lib_config_h`
UCC library configuration handle.

Enumerations

- enum `ucc_coll_type_t` {
 `UCC_COLL_TYPE_ALLGATHER` = `UCC_BIT(0)` ,
 `UCC_COLL_TYPE_ALLGATHERV` = `UCC_BIT(1)` ,
 `UCC_COLL_TYPE_ALLREDUCE` = `UCC_BIT(2)` ,
 `UCC_COLL_TYPE_ALLTOALL` = `UCC_BIT(3)` ,
 `UCC_COLL_TYPE_ALLTOALLV` = `UCC_BIT(4)` ,
 `UCC_COLL_TYPE_BARRIER` = `UCC_BIT(5)` ,
 `UCC_COLL_TYPE_BCAST` = `UCC_BIT(6)` ,
 `UCC_COLL_TYPE_FANIN` = `UCC_BIT(7)` ,
 `UCC_COLL_TYPE_FANOUT` = `UCC_BIT(8)` ,
 `UCC_COLL_TYPE_GATHER` = `UCC_BIT(9)` ,
 `UCC_COLL_TYPE_GATHERV` = `UCC_BIT(10)` ,
 `UCC_COLL_TYPE_REDUCE` = `UCC_BIT(11)` ,
 `UCC_COLL_TYPE_REDUCE_SCATTER` = `UCC_BIT(12)` ,
 `UCC_COLL_TYPE_REDUCE_SCATTERV` = `UCC_BIT(13)` ,
 `UCC_COLL_TYPE_SCATTER` = `UCC_BIT(14)` ,
}

```
UCC_COLL_TYPE_SCATTERV = UCC_BIT(15) ,
UCC_COLL_TYPE_LAST }
```

Enumeration representing the collective operations.

- enum `ucc_reduction_op_t` {
`UCC_OP_SUM` ,
`UCC_OP_PROD` ,
`UCC_OP_MAX` ,
`UCC_OP_MIN` ,
`UCC_OP_LAND` ,
`UCC_OP_LOR` ,
`UCC_OP_LXOR` ,
`UCC_OP_BAND` ,
`UCC_OP BOR` ,
`UCC_OP_BXOR` ,
`UCC_OP_MAXLOC` ,
`UCC_OP_MINLOC` ,
`UCC_OP_AVG` ,
`UCC_OP_LAST` }

Enumeration representing the UCC reduction operations.

- enum `ucc_thread_mode_t` {
`UCC_THREAD_SINGLE` = 0 ,
`UCC_THREAD_FUNNELED` = 1 ,
`UCC_THREAD_MULTIPLE` = 2 }

Enumeration representing the UCC library's thread model.

- enum `ucc_coll_sync_type_t` {
`UCC_NO_SYNC_COLLECTIVES` = 0 ,
`UCC_SYNC_COLLECTIVES` = 1 }

Enumeration representing the collective synchronization model.

- enum `ucc_lib_params_field` {
`UCC_LIB_PARAM_FIELD_THREAD_MODE` = UCC_BIT(0) ,
`UCC_LIB_PARAM_FIELD_COLL_TYPES` = UCC_BIT(1) ,
`UCC_LIB_PARAM_FIELD_REDUCTION_TYPES` = UCC_BIT(2) ,
`UCC_LIB_PARAM_FIELD_SYNC_TYPE` = UCC_BIT(3) }

UCC library initialization parameters.

- enum `ucc_lib_attr_field` {
`UCC_LIB_ATTR_FIELD_THREAD_MODE` = UCC_BIT(0) ,
`UCC_LIB_ATTR_FIELD_COLL_TYPES` = UCC_BIT(1) ,
`UCC_LIB_ATTR_FIELD_REDUCTION_TYPES` = UCC_BIT(2) ,
`UCC_LIB_ATTR_FIELD_SYNC_TYPE` = UCC_BIT(3) }

8.1.1 Detailed Description

Unified Collective Communications (UCC) Library Specification

UCC is a collective communication operations API and library that is flexible, complete, and feature-rich for current and emerging programming models and runtimes.

Library initialization parameters and data-structures

8.1.2 Data Structure Documentation

8.1.2.1 struct `ucc_lib_params`

Description

`ucc_lib_params_t` defines the parameters that can be used to customize the library. The bits in "mask" bit array is defined by `ucc_lib_params_field`, which correspond to fields in structure `ucc_lib_params_t`. The

valid fields of the structure is specified by the setting the bit to "1" in the bit-array "mask". When bits corresponding to the fields is not set, the fields are not defined.

Data Fields

uint64_t	mask	
ucc_thread_mode_t	thread_mode	
uint64_t	coll_types	
uint64_t	reduction_types	
ucc_coll_sync_type_t	sync_type	

8.1.2.2 struct ucc_lib_attr

Description

[ucc_lib_attr_t](#) defines the attributes of the library. The bits in "mask" bit array is defined by [ucc_lib_attr_field](#), which correspond to fields in structure [ucc_lib_attr_t](#). The valid fields of the structure is specified by the setting the bit to "1" in the bit-array "mask". When bits corresponding to the fields is not set, the fields are not defined.

Data Fields

uint64_t	mask	
ucc_thread_mode_t	thread_mode	
uint64_t	coll_types	
uint64_t	reduction_types	
ucc_coll_sync_type_t	sync_type	

8.1.3 Typedef Documentation

8.1.3.1 ucc_lib_params_t

```
typedef struct ucc_lib_params ucc_lib_params_t
```

Description

[ucc_lib_params_t](#) defines the parameters that can be used to customize the library. The bits in "mask" bit array is defined by [ucc_lib_params_field](#), which correspond to fields in structure [ucc_lib_params_t](#). The valid fields of the structure is specified by the setting the bit to "1" in the bit-array "mask". When bits corresponding to the fields is not set, the fields are not defined.

8.1.3.2 ucc_lib_attr_t

```
typedef struct ucc_lib_attr ucc_lib_attr_t
```

Description

[ucc_lib_attr_t](#) defines the attributes of the library. The bits in "mask" bit array is defined by [ucc_lib_attr_field](#), which correspond to fields in structure [ucc_lib_attr_t](#). The valid fields of the structure

is specified by the setting the bit to "1" in the bit-array "mask". When bits corresponding to the fields is not set, the fields are not defined.

8.1.3.3 ucc_lib_h

```
typedef struct ucc_lib_info* ucc_lib_h
```

The ucc library handle is an opaque handle created by the library. It abstracts the collective library. It holds the global information and resources associated with the library. The library handle cannot be passed from one library instance to another.

8.1.3.4 ucc_lib_config_h

```
typedef struct ucc_lib_config* ucc_lib_config_h
```

8.1.4 Enumeration Type Documentation

8.1.4.1 ucc_coll_type_t

```
enum ucc_coll_type_t
```

Library initialization and finalize
Description

[ucc_coll_type_t](#) represents the collective operations supported by the UCC library. Currently, it supports barrier, broadcast, all-reduce, reduce, alltoall, all-gather, gather, scatter, fan-in and fan-out operations.

Enumerator

UCC_COLL_TYPE_ALLGATHER	
UCC_COLL_TYPE_ALLGATHERV	
UCC_COLL_TYPE_ALLREDUCE	
UCC_COLL_TYPE_ALLTOALL	
UCC_COLL_TYPE_ALLTOALLV	
UCC_COLL_TYPE_BARRIER	
UCC_COLL_TYPE_BCAST	
UCC_COLL_TYPE_FANIN	
UCC_COLL_TYPE_FANOUT	
UCC_COLL_TYPE_GATHER	
UCC_COLL_TYPE_GATHERV	
UCC_COLL_TYPE_REDUCE	
UCC_COLL_TYPE_REDUCE_SCATTER	
UCC_COLL_TYPE_REDUCE_SCATTERV	
UCC_COLL_TYPE_SCATTER	
UCC_COLL_TYPE_SCATTERV	
UCC_COLL_TYPE_LAST	

8.1.4.2 ucc_reduction_op_t

enum `ucc_reduction_op_t`

Description

`ucc_reduction_op_t` represents the UCC reduction operations. It is used by the library initialization routine `ucc_init` to request the operations expected by the user. It is used by the `ucc_lib_attr_t` to communicate the operations supported by the library.

Enumerator

UCC_OP_SUM	
UCC_OP_PROD	
UCC_OP_MAX	
UCC_OP_MIN	
UCC_OP_LAND	
UCC_OP_LOR	
UCC_OP_LXOR	
UCC_OP_BAND	
UCC_OP_BOR	
UCC_OP_BXOR	
UCC_OP_MAXLOC	
UCC_OP_MINLOC	
UCC_OP_AVG	
UCC_OP_LAST	

8.1.4.3 ucc_thread_mode_t

enum `ucc_thread_mode_t`

Description

`ucc_thread_mode_t` is used to initialize the UCC library's thread mode. The UCC library can be configured in three thread modes `UCC_THREAD_SINGLE`, `UCC_THREAD_FUNNELED`, and `UCC_LIB_THREAD_MULTIPLE`. In the `UCC_THREAD_SINGLE` mode, the user program must not be multithreaded. In the `UCC_THREAD_FUNNELED` mode, the user program may be multithreaded. However, all UCC interfaces should be invoked from the same thread. In the `UCC_THREAD_MULTIPLE` mode, the user program can be multithreaded and any thread may invoke the UCC operations.

Enumerator

UCC_THREAD_SINGLE	Single-threaded library model
UCC_THREAD_FUNNELED	Funnel thread model
UCC_THREAD_MULTIPLE	Multithread library model

8.1.4.4 ucc_coll_sync_type_t

enum `ucc_coll_sync_type_t`

Description

`ucc_coll_sync_type_t` represents the collective synchronization models. Currently, it supports two synchronization models synchronous and non-synchronous collective models. In the synchronous collective model, the collective communication is not started until participants have not entered the collective operation, and it is not completed until all participants have not completed the collective. In the non-synchronous collective

model, collective communication can be started as soon as the participant enters the collective operation and is completed as soon as it completes locally.

Enumerator

UCC_NO_SYNC_COLLECTIVES	Synchornous collectives
UCC_SYNC_COLLECTIVES	Non-synchronous collectives

8.1.4.5 ucc_lib_params_field

enum `ucc_lib_params_field`

Enumerator

UCC_LIB_PARAM_FIELD_THREAD_MODE	
UCC_LIB_PARAM_FIELD_COLL_TYPES	
UCC_LIB_PARAM_FIELD_REDUCTION_TYPES	
UCC_LIB_PARAM_FIELD_SYNC_TYPE	

8.1.4.6 ucc_lib_attr_field

enum `ucc_lib_attr_field`

Enumerator

UCC_LIB_ATTR_FIELD_THREAD_MODE	
UCC_LIB_ATTR_FIELD_COLL_TYPES	
UCC_LIB_ATTR_FIELD_REDUCTION_TYPES	
UCC_LIB_ATTR_FIELD_SYNC_TYPE	

8.2 Datatypes data-structures and functions

Data Structures

- struct `ucc_reduce_cb_params`
Descriptor of user-defined reduction callback. [More...](#)
- struct `ucc_generic_dt_ops`
UCC generic data type descriptor.
- struct `ucc_generic_dt_ops.reduce`
User-defined reduction callback.

Typedefs

- typedef uint64_t `ucc_datatype_t`
Enumeration representing the UCC library's datatype.
- typedef struct `ucc_reduce_cb_params ucc_reduce_cb_params_t`
Descriptor of user-defined reduction callback.
- typedef struct `ucc_generic_dt_ops ucc_generic_dt_ops_t`
UCC generic data type descriptor.

Enumerations

- enum `ucc_generic_dt_ops_field` { `UCC_GENERIC_DT_OPS_FIELD_FLAGS` = `UCC_BIT(0)` }
- enum `ucc_generic_dt_ops_flags_t` {
`UCC_GENERIC_DT_OPS_FLAG_CONTIG` = `UCC_BIT(0)` ,
`UCC_GENERIC_DT_OPS_FLAG_REDUCE` = `UCC_BIT(1)` }

Flags that can be specified for generic datatype.

Functions

- `ucc_status_t ucc_dt_create_generic` (const `ucc_generic_dt_ops_t` *ops, void *context, `ucc_datatype_t` *datatype_p)

Create a generic datatype.

- void `ucc_dt_destroy` (`ucc_datatype_t` datatype)

Destroy generic datatype.

Variables

- void *(* `ucc_generic_dt_ops::start_pack`)(void *context, const void *buffer, size_t count)
Start a packing request.
- void *(* `ucc_generic_dt_ops::start_unpack`)(void *context, void *buffer, size_t count)
Start an unpacking request.
- size_t(* `ucc_generic_dt_ops::packed_size`)(void *state)
Get the total size of packed data.
- size_t(* `ucc_generic_dt_ops::pack`)(void *state, size_t offset, void *dest, size_t max_length)
Pack data.
- `ucc_status_t`(* `ucc_generic_dt_ops::unpack`)(void *state, size_t offset, const void *src, size_t length)
Unpack data.
- void(* `ucc_generic_dt_ops::finish`)(void *state)
Finish packing/unpacking.
- struct {
`ucc_status_t`(* `cb`)(const `ucc_reduce_cb_params_t` *params)
void * `cb_ctx`
} `ucc_generic_dt_ops::reduce`

User-defined reduction callback.

8.2.1 Detailed Description

Datatypes data-structures and functions

8.2.2 Data Structure Documentation

8.2.2.1 struct `ucc_reduce_cb_params`

This structure is the argument to the `reduce.cb` callback. It must implement the reduction of `n_vectors + 1` data vectors each containing "count" elements. First vector is "src1", other `n_vectors` have start address `v_j = src2 + count * dt_extent * stride * j`. The result is stored in `dst`, so that `dst[i] = src1[i] + v0[i] + v1[i] + ... + v_nvectors[i]`, for `i` in `[0:count)`, where "+" represents user-defined reduction of 2 elements

Data Fields

<code>uint64_t</code>	<code>mask</code>	
<code>void *</code>	<code>src1</code>	
<code>void *</code>	<code>src2</code>	

Data Fields

void *	dst	
size_t	n_vectors	
size_t	count	
size_t	stride	
ucc_dt_generic_t *	dt	
void *	cb_ctx	

8.2.3 Typedef Documentation

8.2.3.1 ucc_datatype_t

```
typedef uint64_t ucc_datatype_t
```

Description

[ucc_datatype_t](#) represents the datatypes supported by the UCC library's collective and reduction operations. The predefined operations are signed and unsigned integers of various sizes, float 16, 32, and 64, and user-defined datatypes. User-defined datatypes are created using [ucc_dt_create_generic](#) interface and can support user-defined reduction operations. Predefined reduction operations can be used only with predefined datatypes.

8.2.3.2 ucc_reduce_cb_params_t

```
typedef struct ucc_reduce_cb_params ucc_reduce_cb_params_t
```

This structure is the argument to the `reduce.cb` callback. It must implement the reduction of `n_vectors + 1` data vectors each containing "count" elements. First vector is "src1", other `n_vectors` have start address `v_j = src2 + count * dt_extent * stride * j`. The result is stored in `dst`, so that `dst[i] = src1[i] + v0[i] + v1[i] + ... + v_nvectors[i]`, for `i` in `[0:count)`, where "+" represents user-defined reduction of 2 elements

8.2.3.3 ucc_generic_dt_ops_t

```
typedef struct ucc_generic_dt_ops ucc_generic_dt_ops_t
```

This structure provides a generic datatype descriptor that is used to create user-defined datatypes.

8.2.4 Enumeration Type Documentation

8.2.4.1 ucc_generic_dt_ops_field

```
enum ucc_generic_dt_ops_field
```

Enumerator

UCC_GENERIC_DT_OPS_FIELD_FLAGS	
--------------------------------	--

8.2.4.2 ucc_generic_dt_ops_flags_t

```
enum ucc_generic_dt_ops_flags_t
```

Enumerator

UCC_GENERIC_DT_OPS_FLAG_CONTIG	If set, the created datatype represents a contiguous memory region with the size specified in <code>ucc_generic_dt_ops::contig_size</code> field of <code>ucc_generic_dt_ops</code>
UCC_GENERIC_DT_OPS_FLAG_REDUCE	If set, the created datatype has user-defined reduction operation associated with it. <code>reduce.cb</code> and <code>reduce.ctx</code> fields of <code>ucc_generic_dt_ops</code> must be initialized. Collective operations that involve reduction (<code>allreduce</code> , <code>reduce</code> , <code>reduce_scatter/v</code>) can use user-defined data-types only when this flag is set.

8.2.5 Function Documentation

8.2.5.1 ucc_dt_create_generic()

```
ucc_status_t ucc_dt_create_generic (
    const ucc_generic_dt_ops_t * ops,
    void * context,
    ucc_datatype_t * datatype_p )
```

This routine creates a generic datatype object. The generic datatype is described by the `ops` object which provides a table of routines defining the operations for generic datatype manipulation. Typically, generic datatypes are used for integration with datatype engines provided with MPI implementations (MPICH, Open MPI, etc). The application is responsible for releasing the `datatype_p` object using `ucc_dt_destroy()` routine.

Parameters

in	<code>ops</code>	Generic datatype function table as defined by <code>ucc_generic_dt_ops_t</code> .
in	<code>context</code>	Application defined context passed to this routine. The context is passed as a parameter to the routines in the <code>ops</code> table.
out	<code>datatype_p</code>	A pointer to datatype object.

Returns

Error code as defined by `ucc_status_t`

8.2.5.2 ucc_dt_destroy()

```
void ucc_dt_destroy (
    ucc_datatype_t datatype )
```

8.2.6 Variable Documentation

8.2.6.1 start_pack

```
void (* ucc_generic_dt_ops::start_pack) (void *context, const void *buffer, size_t count)
```

The pointer refers to application defined start-to-pack routine.

Parameters

in	<i>context</i>	User-defined context.
in	<i>buffer</i>	Buffer to pack.
in	<i>count</i>	Number of elements to pack into the buffer.

Returns

A custom state that is passed to the subsequent [pack\(\)](#) routine.

8.2.6.2 start_unpack

```
void>(*ucc_generic_dt_ops::start_unpack)(void *context, void *buffer, size_t count)
```

The pointer refers to application defined start-to-unpack routine.

Parameters

in	<i>context</i>	User-defined context.
in	<i>buffer</i>	Buffer to unpack to.
in	<i>count</i>	Number of elements to unpack in the buffer.

Returns

A custom state that is passed later to the subsequent [unpack\(\)](#) routine.

8.2.6.3 packed_size

```
size_t(*ucc_generic_dt_ops::packed_size)(void *state)
```

The pointer refers to user defined routine that returns the size of data in a packed format.

Parameters

in	<i>state</i>	State as returned by start_pack() routine.
----	--------------	--

Returns

The size of the data in a packed form.

8.2.6.4 pack

```
size_t(*ucc_generic_dt_ops::pack)(void *state, size_t offset, void *dest, size_t max_length)
```

The pointer refers to application defined pack routine.

Parameters

in	<i>state</i>	State as returned by start_pack() routine.
in	<i>offset</i>	Virtual offset in the output stream.
in	<i>dest</i>	Destination buffer to pack the data.
in	<i>max_length</i>	Maximum length to pack.

Returns

The size of the data that was written to the destination buffer. Must be less than or equal to *max_length*.

8.2.6.5 unpack

```
ucc_status_t(* ucc_generic_dt_ops::unpack) (void *state, size_t offset, const void *src, size_t length)
```

The pointer refers to application defined unpack routine.

Parameters

in	<i>state</i>	State as returned by start_unpack() routine.
in	<i>offset</i>	Virtual offset in the input stream.
in	<i>src</i>	Source to unpack the data from.
in	<i>length</i>	Length to unpack.

Returns

UCC_OK or an error if unpacking failed.

8.2.6.6 finish

```
void(* ucc_generic_dt_ops::finish) (void *state)
```

The pointer refers to application defined finish routine.

Parameters

in	<i>state</i>	State as returned by start_pack() and start_unpack() routines.
----	--------------	--

8.2.6.7

```
struct { ... } ucc_generic_dt_ops::reduce
```

The pointer refers to user-defined reduction routine.

Parameters

in	<i>params</i>	reduction descriptor
----	---------------	----------------------

8.2.6.8

```
ucc_status_t(* { ... } ::cb) (const ucc_reduce_cb_params_t *params)
```

8.2.6.9

```
void* { ... } ::cb_ctx
```

8.3 Library initialization and finalization routines

Functions

- `ucc_status_t ucc_lib_config_read` (`const char *env_prefix`, `const char *filename`, `ucc_lib_config_h *config`)
The `ucc_lib_config_read` routine provides a method to read library configuration from the environment and create configuration descriptor.
- `void ucc_lib_config_release` (`ucc_lib_config_h config`)
The `ucc_lib_config_release` routine releases the configuration descriptor.
- `void ucc_lib_config_print` (`const ucc_lib_config_h config`, `FILE *stream`, `const char *title`, `ucc_config_print_flags_t print_flags`)
The `ucc_lib_config_print` routine prints the configuration information.
- `ucc_status_t ucc_lib_config_modify` (`ucc_lib_config_h config`, `const char *name`, `const char *value`)
The `ucc_lib_config_modify` routine modifies the runtime configuration as described by the descriptor.
- `static ucc_status_t ucc_init` (`const ucc_lib_params_t *params`, `const ucc_lib_config_h config`, `ucc_lib_h *lib_p`)
The `ucc_init` initializes the UCC library.
- `ucc_status_t ucc_finalize` (`ucc_lib_h lib_p`)
The `ucc_finalize` routine finalizes the UCC library.
- `ucc_status_t ucc_lib_get_attr` (`ucc_lib_h lib_p`, `ucc_lib_attr_t *lib_attr`)
The `ucc_lib_get_attr` routine queries the library attributes.

8.3.1 Detailed Description

Library initialization and finalization routines

8.3.2 Function Documentation

8.3.2.1 `ucc_lib_config_read()`

```
ucc_status_t ucc_lib_config_read (
    const char * env_prefix,
    const char * filename,
    ucc_lib_config_h * config )
```

Parameters

out	<code>env_prefix</code>	If not NULL, the routine searches for the environment variables with the prefix UCC_<env_prefix>. Otherwise, the routines search for the environment variables that start with the prefix @ UCC_.
in	<code>filename</code>	If not NULL, read configuration values from the file defined by <code>filename</code> . If the file does not exist, it will be ignored and no error will be reported to the user.
out	<code>config</code>	Pointer to configuration descriptor as defined by <code>ucc_lib_config_h</code> .

Description

`ucc_lib_config_read` allocates the `ucc_lib_config_h` handle and fetches the configuration values from the run-time environment. The run-time environment supported are environment variables or a configuration file.

Returns

Error code as defined by `ucc_status_t`

8.3.2.2 ucc_lib_config_release()

```
void ucc_lib_config_release (
    ucc_lib_config_h config )
```

Parameters

in	<i>config</i>	Pointer to the configuration descriptor to be released. Configuration descriptor as defined by ucc_lib_config_h .
----	---------------	---

Description

The routine releases the configuration descriptor that was allocated through [ucc_lib_config_read\(\)](#) routine.

8.3.2.3 ucc_lib_config_print()

```
void ucc_lib_config_print (
    const ucc_lib_config_h config,
    FILE * stream,
    const char * title,
    ucc_config_print_flags_t print_flags )
```

Parameters

in	<i>config</i>	ucc_lib_config_h "Configuration descriptor" to print.
in	<i>stream</i>	Output stream to print the configuration to.
in	<i>title</i>	Configuration title to print.
in	<i>print_flags</i>	Flags that control various printing options.

Description

The routine prints the configuration information that is stored in ucc_lib_config_h "configuration" descriptor.

8.3.2.4 ucc_lib_config_modify()

```
ucc_status_t ucc_lib_config_modify (
    ucc_lib_config_h config,
    const char * name,
    const char * value )
```

Parameters

in	<i>config</i>	Pointer to the configuration descriptor to be modified
in	<i>name</i>	Configuration variable to be modified
in	<i>value</i>	Configuration value to set

Description

The [ucc_lib_config_modify](#) routine sets the value of identifier "name" to "value".

Returns

Error code as defined by [ucc_status_t](#)

8.3.2.5 ucc_init()

```
static ucc_status_t ucc_init (
    const ucc_lib_params_t * params,
```



```
const ucc_lib_config_h config,
ucc_lib_h * lib_p ) [inline], [static]
```

Parameters

in	<i>params</i>	user provided parameters to customize the library functionality
in	<i>config</i>	UCC configuration descriptor allocated through ucc_config_read() routine.
out	<i>lib_p</i>	UCC library handle

Description

A local operation to initialize and allocate the resources for the UCC operations. The parameters passed using the `ucc_lib_params_t` and `ucc_lib_config_h` structures will customize and select the functionality of the UCC library. The library can be customized for its interaction with the user threads, types of collective operations, and reductions supported. On success, the library object will be created and `ucc_status_t` will return `UCC_OK`. On error, the library object will not be created and corresponding error code as defined by `ucc_status_t` is returned.

Returns

Error code as defined by `ucc_status_t`

8.3.2.6 ucc_finalize()

```
ucc_status_t ucc_finalize (
    ucc_lib_h lib_p )
```

Parameters

in	<i>lib_p</i>	Handle to <code>ucc_lib_h</code> "UCC library".
----	--------------	---

Description

A local operation to release the resources and cleanup. All participants that invoked `ucc_init` should call this routine.

Returns

Error code as defined by `ucc_status_t`

8.3.2.7 ucc_lib_get_attr()

```
ucc_status_t ucc_lib_get_attr (
    ucc_lib_h lib_p,
    ucc_lib_attr_t * lib_attr )
```

Parameters

out	<i>lib_attr</i>	Library attributes
in	<i>lib_p</i>	Input library object

Description

A query operation to get the attributes of the library object. The attributes are library configured values and reflect the choices made by the library implementation.

Returns

Error code as defined by `ucc_status_t`

8.4 Context abstraction data-structures

Data Structures

- struct `ucc_oob_coll`
OOB collective operation for creating the context.
- struct `ucc_mem_map`
- struct `ucc_mem_map_params`
- struct `ucc_context_params`
Structure representing the parameters to customize the context. [More...](#)
- struct `ucc_context_attr`
Structure representing context attributes. [More...](#)

Typedefs

- typedef struct `ucc_oob_coll` `ucc_oob_coll_t`
OOB collective operation for creating the context.
- typedef struct `ucc_mem_map` `ucc_mem_map_t`
- typedef struct `ucc_mem_map_params` `ucc_mem_map_params_t`
- typedef struct `ucc_context_params` `ucc_context_params_t`
Structure representing the parameters to customize the context.
- typedef struct `ucc_context_attr` `ucc_context_attr_t`
Structure representing context attributes.
- typedef struct `ucc_context` * `ucc_context_h`
UCC context.
- typedef struct `ucc_context_config` * `ucc_context_config_h`
UCC context configuration handle.

Enumerations

- enum `ucc_context_type_t` {
 `UCC_CONTEXT_EXCLUSIVE` = 0 ,
 `UCC_CONTEXT_SHARED` }
- enum `ucc_context_params_field` {
 `UCC_CONTEXT_PARAM_FIELD_TYPE` = `UCC_BIT(0)` ,
 `UCC_CONTEXT_PARAM_FIELD_SYNC_TYPE` = `UCC_BIT(1)` ,
 `UCC_CONTEXT_PARAM_FIELD_OOB` = `UCC_BIT(2)` ,
 `UCC_CONTEXT_PARAM_FIELD_ID` = `UCC_BIT(3)` ,
 `UCC_CONTEXT_PARAM_FIELD_MEM_PARAMS` = `UCC_BIT(4)` }
- enum `ucc_context_attr_field` {
 `UCC_CONTEXT_ATTR_FIELD_TYPE` = `UCC_BIT(0)` ,
 `UCC_CONTEXT_ATTR_FIELD_SYNC_TYPE` = `UCC_BIT(1)` ,
 `UCC_CONTEXT_ATTR_FIELD_CTX_ADDR` = `UCC_BIT(2)` ,
 `UCC_CONTEXT_ATTR_FIELD_CTX_ADDR_LEN` = `UCC_BIT(3)` ,
 `UCC_CONTEXT_ATTR_FIELD_WORK_BUFFER_SIZE` = `UCC_BIT(4)` }

8.4.1 Detailed Description

Data-structures associated with context creation and management routines

8.4.2 Data Structure Documentation

8.4.2.1 struct ucc_mem_map

Data Fields

void *	address	the address of a buffer to be attached to a UCC context
size_t	len	the length of the buffer

8.4.2.2 struct ucc_mem_map_params

Data Fields

ucc_mem_map_t *	segments	array of ucc_mem_map elements
uint64_t	n_segments	the number of ucc_mem_map elements

8.4.2.3 struct ucc_context_params

Description

ucc_context_params_t defines the parameters that can be used to customize the context. The "mask" bit array fields are defined by ucc_context_params_field. The bits in "mask" bit array is defined by ucc_context_params_field, which correspond to fields in structure ucc_context_params_t. The valid fields of the structure is specified by the setting the bit to "1" in the bit-array "mask". When bits corresponding to the fields is not set, the fields are not defined.

Data Fields

uint64_t	mask	
ucc_context_type_t	type	
ucc_coll_sync_type_t	sync_type	
ucc_context_oob_coll_t	oob	
uint64_t	ctx_id	
ucc_mem_map_params_t	mem_params	

8.4.2.4 struct ucc_context_attr

Description

ucc_context_attr_t defines the attributes of the context. The bits in "mask" bit array is defined by ucc_context_attr_field, which correspond to fields in structure ucc_context_attr_t. The valid fields of the structure is specified by the setting the bit to "1" in the bit-array "mask". When bits corresponding to the fields is not set, the fields are not defined.

Data Fields

uint64_t	mask	
ucc_context_type_t	type	
ucc_coll_sync_type_t	sync_type	
ucc_context_addr_h	ctx_addr	
ucc_context_addr_len_t	ctx_addr_len	
uint64_t	global_work_buffer_size	

8.4.3 Typedef Documentation

8.4.3.1 ucc_oob_coll_t

```
typedef struct ucc_oob_coll ucc_oob_coll_t
```

8.4.3.2 ucc_mem_map_t

```
typedef struct ucc_mem_map ucc_mem_map_t
```

8.4.3.3 ucc_mem_map_params_t

```
typedef struct ucc_mem_map_params ucc_mem_map_params_t
```

8.4.3.4 ucc_context_params_t

```
typedef struct ucc_context_params ucc_context_params_t
```

Description

[ucc_context_params_t](#) defines the parameters that can be used to customize the context. The "mask" bit array fields are defined by [ucc_context_params_field](#). The bits in "mask" bit array is defined by [ucc_context_params_field](#), which correspond to fields in structure [ucc_context_params_t](#). The valid fields of the structure is specified by the setting the bit to "1" in the bit-array "mask". When bits corresponding to the fields is not set, the fields are not defined.

8.4.3.5 ucc_context_attr_t

```
typedef struct ucc_context_attr ucc_context_attr_t
```

Description

[ucc_context_attr_t](#) defines the attributes of the context. The bits in "mask" bit array is defined by [ucc_context_attr_field](#), which correspond to fields in structure [ucc_context_attr_t](#). The valid fields of the structure is specified by the setting the bit to "1" in the bit-array "mask". When bits corresponding to the fields is not set, the fields are not defined.

8.4.3.6 ucc_context_h

```
typedef struct ucc_context* ucc_context_h
```

The UCC context is an opaque handle to abstract the network resources for collective operations. The network resources could be either software or hardware. Based on the type of the context, the resources can be shared or either be exclusively used. The UCC context is required but not sufficient to execute a collective operation.

8.4.3.7 ucc_context_config_h

```
typedef struct ucc_context_config* ucc_context_config_h
```

8.4.4 Enumeration Type Documentation**8.4.4.1 ucc_context_type_t**

```
enum ucc_context_type_t
```

Enumerator

UCC_CONTEXT_EXCLUSIVE	
UCC_CONTEXT_SHARED	

8.4.4.2 ucc_context_params_field

enum `ucc_context_params_field`

Enumerator

UCC_CONTEXT_PARAM_FIELD_TYPE	
UCC_CONTEXT_PARAM_FIELD_SYNC_TYPE	
UCC_CONTEXT_PARAM_FIELD_OOB	
UCC_CONTEXT_PARAM_FIELD_ID	
UCC_CONTEXT_PARAM_FIELD_MEM_PARAMS	

8.4.4.3 ucc_context_attr_field

enum `ucc_context_attr_field`

Enumerator

UCC_CONTEXT_ATTR_FIELD_TYPE	
UCC_CONTEXT_ATTR_FIELD_SYNC_TYPE	
UCC_CONTEXT_ATTR_FIELD_CTX_ADDR	
UCC_CONTEXT_ATTR_FIELD_CTX_ADDR_LEN	
UCC_CONTEXT_ATTR_FIELD_WORK_BUFFER_SIZE	

8.5 Context abstraction routines

Functions

- `ucc_status_t ucc_context_config_read (ucc_lib_h lib_handle, const char *filename, ucc_context_config_h *config)`
Routine reads the configuration information for contexts from the runtime environment and creates the configuration descriptor.
- `void ucc_context_config_release (ucc_context_config_h config)`
The `ucc_context_config_release` routine releases the configuration descriptor.
- `void ucc_context_config_print (const ucc_context_config_h config, FILE *stream, const char *title, ucc_config_print_flags_t print_flags)`
The `ucc_context_config_print` routine prints the configuration information.
- `ucc_status_t ucc_context_config_modify (ucc_context_config_h config, const char *cls, const char *name, const char *value)`
The `ucc_context_config_modify` routine modifies the runtime configuration of UCC context (optionally for a given CLS).
- `ucc_status_t ucc_context_create (ucc_lib_h lib_handle, const ucc_context_params_t *params, const ucc_context_config_h config, ucc_context_h *context)`
The `ucc_context_create` routine creates the context handle.
- `ucc_status_t ucc_context_progress (ucc_context_h context)`
The `ucc_context_progress` routine progresses the operations on the context handle.
- `ucc_status_t ucc_context_destroy (ucc_context_h context)`
The `ucc_context_destroy` routine frees the context handle.

- `ucc_status_t ucc_context_get_attr (ucc_context_h context, ucc_context_attr_t *context_attr)`

The routine queries the attributes of the context handle.

8.5.1 Detailed Description

Context create and management routines

8.5.2 Function Documentation

8.5.2.1 ucc_context_config_read()

```
ucc_status_t ucc_context_config_read (
    ucc_lib_h lib_handle,
    const char * filename,
    ucc_context_config_h * config )
```

Parameters

in	<i>lib_handle</i>	Library handle
in	<i>filename</i>	If not NULL, read configuration values from the file defined by <i>filename</i> . If the file does not exist, it will be ignored and no error will be reported to the user.
out	<i>config</i>	Pointer to configuration descriptor as defined by <code>ucc_context_config_h</code> .

Description

`ucc_context_config_read` allocates the `ucc_lib_config_h` handle and fetches the configuration values from the run-time environment. The run-time environment supported are environment variables or a configuration file. It uses the `env_prefix` from `ucc_lib_config_read`. If `env_prefix` is not NULL, the routine searches for the environment variables with the prefix `UCC_<env_prefix>`. Otherwise, the routines search for the environment variables that start with the prefix `@ UCC_`.

Returns

Error code as defined by `ucc_status_t`

8.5.2.2 ucc_context_config_release()

```
void ucc_context_config_release (
    ucc_context_config_h config )
```

Parameters

in	<i>config</i>	Pointer to the configuration descriptor to be released. Configuration descriptor as defined by <code>ucc_context_config_h</code>
----	---------------	--

Description

The routine releases the configuration descriptor that was allocated through `ucc_context_config_read()` routine.

8.5.2.3 ucc_context_config_print()

```
void ucc_context_config_print (
    const ucc_context_config_h config,
    FILE * stream,
```

```
const char * title,
ucc_config_print_flags_t print_flags )
```

Parameters

in	<i>config</i>	ucc_context_config_h "Configuration descriptor" to print.
in	<i>stream</i>	Output stream to print the configuration to.
in	<i>title</i>	Configuration title to print.
in	<i>print_flags</i>	Flags that control various printing options.

Description

The routine prints the configuration information that is stored in ucc_context_config_h "configuration" descriptor.

8.5.2.4 ucc_context_config_modify()

```
ucc_status_t ucc_context_config_modify (
    ucc_context_config_h config,
    const char * cls,
    const char * name,
    const char * value )
```

Parameters

in	<i>config</i>	Pointer to the configuration descriptor to be modified
in	<i>cls</i>	Comma separated list of CLS or NULL. If NULL then core context config is modified.
in	<i>name</i>	Configuration variable to be modified
in	<i>value</i>	Configuration value to set

Description

The [ucc_context_config_modify](#) routine sets the value of identifier "name" to "value" for a specified CL.

Returns

Error code as defined by [ucc_status_t](#)

8.5.2.5 ucc_context_create()

```
ucc_status_t ucc_context_create (
    ucc_lib_h lib_handle,
    const ucc_context_params_t * params,
    const ucc_context_config_h config,
    ucc_context_h * context )
```

Parameters

in	<i>lib_handle</i>	Library handle
in	<i>params</i>	Customizations for the communication context
in	<i>config</i>	Configuration for the communication context to read from environment
out	<i>context</i>	Pointer to the newly created communication context

Description

The [ucc_context_create](#) creates the context and [ucc_context_destroy](#) releases the resources and destroys

the context state. The creation of context does not necessarily indicate its readiness to be used for collective or other group operations. On success, the context handle will be created and `ucc_status_t` will return `UCC_OK`. On error, the library object will not be created and corresponding error code as defined by `ucc_status_t` is returned.

Returns

Error code as defined by `ucc_status_t`

8.5.2.6 `ucc_context_progress()`

```
ucc_status_t ucc_context_progress (
    ucc_context_h context )
```

Parameters

in	<i>context</i>	Communication context handle to be progressed
----	----------------	---

Description

The `ucc_context_progress` routine progresses the operations on the content handle. It does not block for lack of resources or communication.

Returns

Error code as defined by `ucc_status_t`

8.5.2.7 `ucc_context_destroy()`

```
ucc_status_t ucc_context_destroy (
    ucc_context_h context )
```

Parameters

in	<i>context</i>	Communication context handle to be released
----	----------------	---

Description

`ucc_context_destroy` routine releases the resources associated with the handle *context*. All teams associated with the team should be released before this. It is invalid to associate any team with this handle after the routine is called.

Returns

Error code as defined by `ucc_status_t`

8.5.2.8 `ucc_context_get_attr()`

```
ucc_status_t ucc_context_get_attr (
    ucc_context_h context,
    ucc_context_attr_t * context_attr )
```

Parameters

in	<i>context</i>	Communication context
out	<i>context_attr</i>	Attributes of the communication context

Description

`ucc_context_get_attr` routine queries the context handle attributes described by `ucc_context_attr`.

Returns

Error code as defined by `ucc_status_t`

8.6 Team abstraction data-structures

Data Structures

- struct `ucc_team_p2p_conn`
- struct `ucc_ep_map_strided`
- struct `ucc_ep_map_array`
- struct `ucc_ep_map_cb`
- struct `ucc_ep_map_t`
- struct `ucc_team_params`
Structure representing the parameters to customize the team. [More...](#)
- struct `ucc_team_attr`
Structure representing the team attributes. [More...](#)
- union `ucc_ep_map_t.__unnamed2__`

Typedefs

- typedef struct `ucc_team_p2p_conn` `ucc_team_p2p_conn_t`
- typedef struct `ucc_ep_map_t` `ucc_ep_map_t`
- typedef struct `ucc_team_params` `ucc_team_params_t`
Structure representing the parameters to customize the team.
- typedef struct `ucc_team_attr` `ucc_team_attr_t`
Structure representing the team attributes.
- typedef struct `ucc_team` * `ucc_team_h`
UCC team handle.
- typedef void * `ucc_p2p_conn_t`
- typedef void * `ucc_context_addr_h`
- typedef size_t `ucc_context_addr_len_t`

Enumerations

- enum `ucc_team_params_field` {
`UCC_TEAM_PARAM_FIELD_ORDERING` = `UCC_BIT(0)` ,
`UCC_TEAM_PARAM_FIELD_OUTSTANDING_COLLIS` = `UCC_BIT(1)` ,
`UCC_TEAM_PARAM_FIELD_EP` = `UCC_BIT(2)` ,
`UCC_TEAM_PARAM_FIELD_EP_LIST` = `UCC_BIT(3)` ,
`UCC_TEAM_PARAM_FIELD_EP_RANGE` = `UCC_BIT(4)` ,
`UCC_TEAM_PARAM_FIELD_TEAM_SIZE` = `UCC_BIT(5)` ,
`UCC_TEAM_PARAM_FIELD_SYNC_TYPE` = `UCC_BIT(6)` ,
`UCC_TEAM_PARAM_FIELD_OOB` = `UCC_BIT(7)` ,
`UCC_TEAM_PARAM_FIELD_P2P_CONN` = `UCC_BIT(8)` ,
`UCC_TEAM_PARAM_FIELD_MEM_PARAMS` = `UCC_BIT(9)` ,
`UCC_TEAM_PARAM_FIELD_EP_MAP` = `UCC_BIT(10)` ,
`UCC_TEAM_PARAM_FIELD_ID` = `UCC_BIT(11)` ,
`UCC_TEAM_PARAM_FIELD_FLAGS` = `UCC_BIT(12)` }
- enum `ucc_team_attr_field` {
`UCC_TEAM_ATTR_FIELD_POST_ORDERING` = `UCC_BIT(0)` ,
`UCC_TEAM_ATTR_FIELD_OUTSTANDING_CALLS` = `UCC_BIT(1)` ,
`UCC_TEAM_ATTR_FIELD_EP` = `UCC_BIT(2)` ,
`UCC_TEAM_ATTR_FIELD_EP_RANGE` = `UCC_BIT(3)` ,
`UCC_TEAM_ATTR_FIELD_SYNC_TYPE` = `UCC_BIT(4)` ,
`UCC_TEAM_ATTR_FIELD_MEM_PARAMS` = `UCC_BIT(5)` }

- enum `ucc_team_flags` { `UCC_TEAM_FLAG_COLL_WORK_BUFFER` = `UCC_BIT(0)` }
- enum `ucc_post_ordering_t` {
`UCC_COLLECTIVE_POST_ORDERED` = 0 ,
`UCC_COLLECTIVE_POST_UNORDERED` = 1 ,
`UCC_COLLECTIVE_INIT_ORDERED` = 2 ,
`UCC_COLLECTIVE_INIT_UNORDERED` = 3 ,
`UCC_COLLECTIVE_INIT_AND_POST_ORDERED` = 4 ,
`UCC_COLLECTIVE_INIT_AND_POST_UNORDERED` = 5 }
- enum `ucc_ep_range_type_t` {
`UCC_COLLECTIVE_EP_RANGE_CONTIG` = 0 ,
`UCC_COLLECTIVE_EP_RANGE_NONCONTIG` = 1 }
- enum `ucc_ep_map_type_t` {
`UCC_EP_MAP_FULL` = 1 ,
`UCC_EP_MAP_STRIDED` = 2 ,
`UCC_EP_MAP_ARRAY` = 3 ,
`UCC_EP_MAP_CB` = 4 }

8.6.1 Detailed Description

Data-structures associated with team create and management routines

8.6.2 Data Structure Documentation

8.6.2.1 struct `ucc_ep_map_strided`

Data Fields

<code>uint64_t</code>	<code>start</code>	
<code>int64_t</code>	<code>stride</code>	

8.6.2.2 struct `ucc_ep_map_array`

Data Fields

<code>void *</code>	<code>map</code>	
<code>size_t</code>	<code>elem_size</code>	4 if array is int, 8 if e.g. <code>uint64_t</code>

8.6.2.3 struct `ucc_ep_map_t`

Data Fields

<code>ucc_ep_map_type_t</code>	<code>type</code>	
<code>uint64_t</code>	<code>ep_num</code>	number of eps mapped to ctx
union <code>ucc_ep_map_t.__unnamed2__</code>	<code>__unnamed__</code>	

8.6.2.4 struct `ucc_team_params`

Description

`ucc_team_params_t` defines the parameters that can be used to customize the team. The "mask" bit array fields are defined by `ucc_team_params_field`. The bits in "mask" bit array is defined by `ucc_team_params_field`, which correspond to fields in structure `ucc_team_params_t`. The valid fields of the structure is specified by the setting the bit to "1" in the bit-array "mask". When bits corresponding to the fields is not set, the fields are not defined.

Data Fields

uint64_t	mask	
uint64_t	flags	
ucc_post_ordering_t	ordering	ucc_team_params::ordering is set to one the values defined by ucc_post_ordering_t
uint64_t	outstanding_colls	ucc_team_params::outstanding_colls represents the number of outstanding non-blocking calls the user expects to post to the team. If the user posts more non-blocking calls than set, the behavior is undefined. If not set, there is no limit on the number of outstanding calls to be posted.
uint64_t	ep	ucc_team_params::ep The endpoint is a non-negative unique integer identifying the participant in the collective. If ep is not set, and ucc_team_params::oob is not set, the library generates the ep. The generated ep can be queried using the ucc_team_get_attr interface.
uint64_t *	ep_list	ucc_team_params::ep_list The endpoint list provides the list of eps participating to create the team.
ucc_ep_range_type_t	ep_range	ucc_team_params::ep_range can be either contiguous or not contiguous. It is a hint to the library.
uint64_t	team_size	ucc_team_params::team_size The team size is the number of participants in the team. If ucc_team_params::oob is provided, the team size and ucc_oob_coll::n_oob_eps should be the same.
ucc_coll_sync_type_t	sync_type	ucc_team_params::sync_type The options for sync_type are provided by ucc_coll_sync_type_t

Data Fields

<code>ucc_team_oob_coll_t</code>	<code>oob</code>	<p><code>ucc_team_params::oob</code> The signature of the function is defined by <code>ucc_oob_coll_t</code>. The oob is used for exchanging information between the team participants during team creation. The user is responsible for implementing the oob operation. The relation between <code>ucc_team_params::ep</code> and <code>ucc_oob_coll::oob_ep</code> is defined as below:</p> <ul style="list-style-type: none"> • When both are not provided. The library is responsible for generating the ep, which can be then queried via the <code>ucc_team_get_attr</code> interface. This requires, however, <code>ucc_params_t ep_map</code> to be set and context created by <code>ucc_oob_coll</code>. The behavior is undefined, when neither <code>ucc_team_params::ep</code> or <code>ucc_team_params::ep_map</code>, or <code>ucc_team_params::oob</code> is not set. • When <code>ucc_team_params::ep</code> is provided and <code>ucc_team_params::oob</code> is not provided. The “ep” is the unique integer for the participant. • When <code>ucc_oob_coll::oob_ep</code> is provided and <code>ucc_team_params::ep</code> is not provided. The “ep” will be equivalent to <code>ucc_oob_coll::oob_ep</code>. • When both are provided, the <code>ucc_oob_coll::oob_ep</code> and <code>ucc_team_params_t::ep</code> should be same. Otherwise, it is undefined.
<code>ucc_team_p2p_conn_t</code>	<code>p2p_conn</code>	<code>ucc_team_params::p2p_conn</code> is a callback function for the gathering the point-to-point communication information.
<code>ucc_mem_map_params_t</code>	<code>mem_params</code>	<code>ucc_team_params::mem_params</code> provides an ability to attach a buffer to the team. This can be used as input/output or control buffer for the team. Typically, it can be useful for one-sided collective implementation.
<code>ucc_ep_map_t</code>	<code>ep_map</code>	<code>ucc_team_params::ep_map</code> provides a mapping between <code>ucc_oob_coll::oob_ep</code> used by the team and <code>ucc_oob_coll::oob_ep</code> used by the context. The mapping options are defined by <code>ucc_ep_map_t</code> . The definition is valid only when context is created with an <code>ucc_oob_coll</code> .
<code>uint64_t</code>	<code>id</code>	<code>ucc_team_params::id</code> The team id is a unique integer identifying the team that is active. The integer is unique within the process and not the job .i.e., any two active non-overlapping teams can have the same id. This semantic helps to avoid a global information exchange .i.e, the processes or threads not participating in the particular, need not participate in the team creation. If not provided, the team id is created internally. For the MPI programming model, this can be inherited from the MPI communicator id.

8.6.2.5 struct ucc_team_attr

Description

[ucc_team_attr_t](#) defines the attributes of the team. The bits in "mask" bit array is defined by [ucc_team_attr_field](#), which correspond to fields in structure [ucc_team_attr_t](#). The valid fields of the structure is specified by the setting the bit to "1" in the bit-array "mask". When bits corresponding to the fields is not set, the fields are not defined.

Data Fields

uint64_t	mask	
ucc_post_ordering_t	ordering	
uint64_t	outstanding_colls	
uint64_t	ep	
ucc_ep_range_type_t	ep_range	
ucc_coll_sync_type_t	sync_type	
ucc_mem_map_params_t	mem_params	

8.6.2.6 union ucc_ep_map_t. __unnamed2__

Data Fields

struct ucc_ep_map_strided	strided	
struct ucc_ep_map_array	array	
struct ucc_ep_map_cb	cb	

8.6.3 Typedef Documentation

8.6.3.1 ucc_team_p2p_conn_t

```
typedef struct ucc_team_p2p_conn ucc_team_p2p_conn_t
```

8.6.3.2 ucc_ep_map_t

```
typedef struct ucc_ep_map_t ucc_ep_map_t
```

8.6.3.3 ucc_team_params_t

```
typedef struct ucc_team_params ucc_team_params_t
```

Description

[ucc_team_params_t](#) defines the parameters that can be used to customize the team. The "mask" bit array fields are defined by [ucc_team_params_field](#). The bits in "mask" bit array is defined by [ucc_team_params_field](#), which correspond to fields in structure [ucc_team_params_t](#). The valid fields of the structure is specified by the setting the bit to "1" in the bit-array "mask". When bits corresponding to the fields is not set, the fields are not defined.

8.6.3.4 ucc_team_attr_t

```
typedef struct ucc_team_attr ucc_team_attr_t
```

Description

`ucc_team_attr_t` defines the attributes of the team. The bits in "mask" bit array is defined by `ucc_team_attr_field`, which correspond to fields in structure `ucc_team_attr_t`. The valid fields of the structure is specified by the setting the bit to "1" in the bit-array "mask". When bits corresponding to the fields is not set, the fields are not defined.

8.6.3.5 ucc_team_h

```
typedef struct ucc_team* ucc_team_h
```

The UCC team handle is an opaque handle created by the library. It abstracts the group resources required for the collective operations and participants of the collective operation. The participants of the collective operation can be an OS process or thread.

8.6.3.6 ucc_p2p_conn_t

```
typedef void* ucc_p2p_conn_t
```

8.6.3.7 ucc_context_addr_h

```
typedef void* ucc_context_addr_h
```

8.6.3.8 ucc_context_addr_len_t

```
typedef size_t ucc_context_addr_len_t
```

8.6.4 Enumeration Type Documentation

8.6.4.1 ucc_team_params_field

```
enum ucc_team_params_field
```

Enumerator

UCC_TEAM_PARAM_FIELD_ORDERING	
UCC_TEAM_PARAM_FIELD_OUTSTANDING_COLL	
UCC_TEAM_PARAM_FIELD_EP	
UCC_TEAM_PARAM_FIELD_EP_LIST	
UCC_TEAM_PARAM_FIELD_EP_RANGE	
UCC_TEAM_PARAM_FIELD_TEAM_SIZE	
UCC_TEAM_PARAM_FIELD_SYNC_TYPE	
UCC_TEAM_PARAM_FIELD_OOB	
UCC_TEAM_PARAM_FIELD_P2P_CONN	
UCC_TEAM_PARAM_FIELD_MEM_PARAMS	
UCC_TEAM_PARAM_FIELD_EP_MAP	
UCC_TEAM_PARAM_FIELD_ID	
UCC_TEAM_PARAM_FIELD_FLAGS	

8.6.4.2 ucc_team_attr_field

```
enum ucc_team_attr_field
```

Enumerator

UCC_TEAM_ATTR_FIELD_POST_ORDERING	
UCC_TEAM_ATTR_FIELD_OUTSTANDING_CALLS	
UCC_TEAM_ATTR_FIELD_EP	
UCC_TEAM_ATTR_FIELD_EP_RANGE	
UCC_TEAM_ATTR_FIELD_SYNC_TYPE	
UCC_TEAM_ATTR_FIELD_MEM_PARAMS	

8.6.4.3 ucc_team_flags

enum `ucc_team_flags`

Enumerator

UCC_TEAM_FLAG_COLL_WORK_BUFFER	
--------------------------------	--

8.6.4.4 ucc_post_ordering_t

enum `ucc_post_ordering_t`

Enumerator

UCC_COLLECTIVE_POST_ORDERED	When set to this value, the collective participants shall post the operation in the same order.
UCC_COLLECTIVE_POST_UNORDERED	When set to this value, the collective participants shall post the operation in any order.
UCC_COLLECTIVE_INIT_ORDERED	When set to this value, the collective participants shall initialize the operation in the same order.
UCC_COLLECTIVE_INIT_UNORDERED	When set to this value, the collective participants shall initialize the operation in any order.
UCC_COLLECTIVE_INIT_AND_POST_ORDERED	When set to this value, the collective participants shall initialize and post the operation in the same order.
UCC_COLLECTIVE_INIT_AND_POST_UNORDERED	When set to this value, the collective participants shall initialize and post the operation in any order.

8.6.4.5 ucc_ep_range_type_t

enum `ucc_ep_range_type_t`

Enumerator

UCC_COLLECTIVE_EP_RANGE_CONTIG	
UCC_COLLECTIVE_EP_RANGE_NONCONTIG	

8.6.4.6 ucc_ep_map_type_t

enum `ucc_ep_map_type_t`

Enumerator

<code>UCC_EP_MAP_FULL</code>	The ep range of the team spans all eps from a context
<code>UCC_EP_MAP_STRIDED</code>	The ep range of the team can be described by the 2 values: start, stride.
<code>UCC_EP_MAP_ARRAY</code>	The ep range is given as an array of intergers that map the ep in the team to the team_context rank.
<code>UCC_EP_MAP_CB</code>	The ep range mapping is defined as callback provided by the UCC user.

8.7 Team abstraction routines

Functions

- `ucc_status_t ucc_team_create_post (ucc_context_h *contexts, uint32_t num_contexts, const ucc_team_params_t *team_params, ucc_team_h *new_team)`
The routine is a method to create the team.
- `ucc_status_t ucc_team_create_test (ucc_team_h team)`
The routine queries the status of the team creation operation.
- `ucc_status_t ucc_team_destroy (ucc_team_h team)`
The team frees the team handle.
- `ucc_status_t ucc_team_get_attr (ucc_team_h team, ucc_team_attr_t *team_attr)`
The routine returns the attributes of the team.
- `ucc_status_t ucc_team_create_from_parent (uint64_t my_ep, uint32_t included, ucc_team_h parent_team, ucc_team_h *new_team)`
The routine creates a new team from the parent team.
- `ucc_status_t ucc_team_get_size (ucc_team_h team, uint32_t *size)`
The routine returns the size of the team.
- `ucc_status_t ucc_team_get_my_ep (ucc_team_h team, uint64_t *ep)`
The routine returns the endpoint of the calling participant.
- `ucc_status_t ucc_team_get_all_eps (ucc_team_h team, uint64_t **ep, uint64_t *num_eps)`
The routine queries all endpoints associated with the team handle.

8.7.1 Detailed Description

Team create and management routines

8.7.2 Function Documentation

8.7.2.1 ucc_team_create_post()

```
ucc_status_t ucc_team_create_post (
    ucc_context_h * contexts,
    uint32_t num_contexts,
    const ucc_team_params_t * team_params,
    ucc_team_h * new_team )
```

Parameters

in	<code>contexts</code>	Communication contexts abstracting the resources
----	-----------------------	--

Parameters

in	<i>num_contexts</i>	Number of contexts passed for the create operation
in	<i>team_params</i>	User defined configurations for the team
out	<i>new_team</i>	Team handle

Description

[ucc_team_create_post](#) is a nonblocking collective operation to create the team handle. It takes in parameters [ucc_context_h](#) and [ucc_team_params_t](#). The [ucc_team_params_t](#) provides user configuration to customize the team and, [ucc_context_h](#) provides the resources for the team and collectives. The routine returns immediately after posting the operation with the new team handle. However, the team handle is not ready for posting the collective operation. [ucc_team_create_test](#) operation is used to learn the status of the new team handle. On error, the team handle will not be created and corresponding error code as defined by [ucc_status_t](#) is returned.

Returns

Error code as defined by [ucc_status_t](#)

8.7.2.2 ucc_team_create_test()

```
ucc_status_t ucc_team_create_test (
    ucc_team_h team )
```

Parameters

in	<i>team</i>	Team handle to test
----	-------------	---------------------

Description

[ucc_team_create_test](#) routines tests the status of team handle. If required it can progress the communication but cannot block on the communications.

Returns

Error code as defined by [ucc_status_t](#)

8.7.2.3 ucc_team_destroy()

```
ucc_status_t ucc_team_destroy (
    ucc_team_h team )
```

Parameters

in	<i>team</i>	Destroy previously created team and release all resources associated with it.
----	-------------	---

Description

[ucc_team_destroy](#) is a nonblocking collective operation to release all resources associated with the team handle, and destroy the team handle. It is invalid to post a collective operation after the [ucc_team_destroy](#) operation. It is invalid to call [ucc_team_destroy](#) operation while [ucc_team_create_post](#) is in progress. It is the user's responsibility to ensure there is one outstanding [ucc_team_create_post](#) or [ucc_team_destroy](#) operation is in progress.

Returns

Error code as defined by [ucc_status_t](#)

8.7.2.4 ucc_team_get_attr()

```
ucc_status_t ucc_team_get_attr (
    ucc_team_h team,
    ucc_team_attr_t * team_attr )
```

Parameters

in	<i>team</i>	Team handle
out	<i>team_attr</i>	Attributes of the team

Description

[ucc_team_get_attr](#) routine queries the team handle attributes. The attributes of the team handle are described by the team attributes [ucc_team_attr_t](#)

Returns

Error code as defined by [ucc_status_t](#)

8.7.2.5 ucc_team_create_from_parent()

```
ucc_status_t ucc_team_create_from_parent (
    uint64_t my_ep,
    uint32_t included,
    ucc_team_h parent_team,
    ucc_team_h * new_team )
```

Parameters

in	<i>my_ep</i>	Endpoint of the process/thread calling the split operation
in	<i>parent_team</i>	Parent team handle from which a new team handle is created
in	<i>included</i>	Variable indicating whether a process/thread participates in the newly created team; value 1 indicates the participation and value 0 indicates otherwise
out	<i>new_team</i>	Pointer to the new team handle

Description

[ucc_team_create_from_parent](#) is a nonblocking collective operation, which creates a new team from the parent team. If a participant intends to participate in the new team, it passes a TRUE value for the "included" parameter. Otherwise, it passes FALSE. The routine returns immediately after the post-operation. To learn the completion of the team create operation, the [ucc_team_create_test](#) operation is used.

Returns

Error code as defined by [ucc_status_t](#)

8.7.2.6 ucc_team_get_size()

```
ucc_status_t ucc_team_get_size (
    ucc_team_h team,
    uint32_t * size )
```

Parameters

in	<i>team</i>	Team handle
out	<i>size</i>	The size of team as number of endpoints

Description

`ucc_team_get_size` routine queries the size of the team. It reflects the number of unique endpoints in the team.

Returns

Error code as defined by `ucc_status_t`

8.7.2.7 `ucc_team_get_my_ep()`

```
ucc_status_t ucc_team_get_my_ep (
    ucc_team_h team,
    uint64_t * ep )
```

Parameters

out	<i>ep</i>	Endpoint of the participant calling the routine
in	<i>team</i>	Team handle

Description

`ucc_team_get_my_ep` routine queries and returns the endpoint of the participant invoking the interface.

Returns

Error code as defined by `ucc_status_t`

8.7.2.8 `ucc_team_get_all_eps()`

```
ucc_status_t ucc_team_get_all_eps (
    ucc_team_h team,
    uint64_t ** ep,
    uint64_t * num_eps )
```

Parameters

out	<i>ep</i>	List of endpoints
out	<i>num_eps</i>	Number of endpoints
in	<i>team</i>	Team handle

Description

`ucc_team_get_all_eps` routine queries and returns all endpoints of all participants in the team.

Returns

Error code as defined by `ucc_status_t`

8.8 Collective operations data-structures

Data Structures

- struct `ucc_coll_buffer_info_v`
- struct `ucc_coll_buffer_info`
- struct `ucc_coll_callback`

UCC collective completion callback.

Typedefs

- typedef enum `ucc_memory_type` `ucc_memory_type_t`
- typedef struct `ucc_coll_buffer_info_v` `ucc_coll_buffer_info_v_t`
- typedef struct `ucc_coll_buffer_info` `ucc_coll_buffer_info_t`
- typedef struct `ucc_coll_req` * `ucc_coll_req_h`
UCC collective request handle.
- typedef struct `ucc_coll_callback` `ucc_coll_callback_t`
UCC collective completion callback.
- typedef uint64_t `ucc_count_t`
Count datatype to support both small (32 bit) and large counts (64 bit)
- typedef uint64_t `ucc_aint_t`
Datatype to support both small (32 bit) and large address offsets (64 bit)
- typedef uint16_t `ucc_coll_id_t`
Datatype for collective tags.

Enumerations

- enum `ucc_memory_type` {
 `UCC_MEMORY_TYPE_HOST` ,
 `UCC_MEMORY_TYPE_CUDA` ,
 `UCC_MEMORY_TYPE_CUDA_MANAGED` ,
 `UCC_MEMORY_TYPE_ROCM` ,
 `UCC_MEMORY_TYPE_ROCM_MANAGED` ,
 `UCC_MEMORY_TYPE_LAST` ,
 `UCC_MEMORY_TYPE_UNKNOWN` = `UCC_MEMORY_TYPE_LAST` }
- enum `ucc_coll_args_flags_t` {
 `UCC_COLL_ARGS_FLAG_IN_PLACE` = `UCC_BIT(0)` ,
 `UCC_COLL_ARGS_FLAG_PERSISTENT` = `UCC_BIT(1)` ,
 `UCC_COLL_ARGS_FLAG_COUNT_64BIT` = `UCC_BIT(2)` ,
 `UCC_COLL_ARGS_FLAG_DISPLACEMENTS_64BIT` = `UCC_BIT(3)` ,
 `UCC_COLL_ARGS_FLAG_CONTIG_SRC_BUFFER` = `UCC_BIT(4)` ,
 `UCC_COLL_ARGS_FLAG_CONTIG_DST_BUFFER` = `UCC_BIT(5)` ,
 `UCC_COLL_ARGS_FLAG_TIMEOUT` = `UCC_BIT(6)` ,
 `UCC_COLL_ARGS_FLAG_MEM_MAPPED_BUFFERS` = `UCC_BIT(7)` }
- enum `ucc_error_type_t` {
 `UCC_ERR_TYPE_LOCAL` = 0 ,
 `UCC_ERR_TYPE_GLOBAL` = 1 }
- enum `ucc_coll_args_field` {
 `UCC_COLL_ARGS_FIELD_FLAGS` = `UCC_BIT(0)` ,
 `UCC_COLL_ARGS_FIELD_TAG` = `UCC_BIT(1)` ,
 `UCC_COLL_ARGS_FIELD_CB` = `UCC_BIT(2)` ,
 `UCC_COLL_ARGS_FIELD_GLOBAL_WORK_BUFFER` = `UCC_BIT(3)` }

8.8.1 Detailed Description

Data-structures associated with collective operation creation, progress, and finalize.

8.8.2 Data Structure Documentation

8.8.2.1 struct `ucc_coll_buffer_info_v`

Data Fields

<code>void *</code>	<code>buffer</code>	Starting address of the send/recv buffer
<code>ucc_count_t *</code>	<code>counts</code>	Array of counts of type <code>ucc_count_t</code> describing the total number of elements

Data Fields

ucc_aint_t *	displacements	Displacement array of team size and type ucc_aint_t . Entry i specifies the displacement relative to the start address for the incoming data(outgoing data) for the team member i. For send buffer the data is fetched from this displacement and for receive buffer the incoming data is placed at this displacement.
ucc_datatype_t	datatype	Datatype of each buffer element
ucc_memory_type_t	mem_type	Memory type of buffer as defined by ucc_memory_type

8.8.2.2 struct ucc_coll_buffer_info

Data Fields

void *	buffer	Starting address of the send/recv buffer
ucc_count_t	count	Total number of elements in the buffer
ucc_datatype_t	datatype	Datatype of each buffer element
ucc_memory_type_t	mem_type	Memory type of buffer as defined by ucc_memory_type

8.8.3 Typedef Documentation

8.8.3.1 ucc_memory_type_t

```
typedef enum ucc_memory_type ucc_memory_type_t
```

8.8.3.2 ucc_coll_buffer_info_v_t

```
typedef struct ucc_coll_buffer_info_v ucc_coll_buffer_info_v_t
```

8.8.3.3 ucc_coll_buffer_info_t

```
typedef struct ucc_coll_buffer_info ucc_coll_buffer_info_t
```

8.8.3.4 ucc_coll_req_h

```
typedef struct ucc_coll_req* ucc_coll_req_h
```

The UCC request handle is an opaque handle created by the library during the invocation of the collective operation. The request may be used to learn the status of the collective operation, progress, or complete the collective operation.

8.8.3.5 ucc_coll_callback_t

```
typedef struct ucc_coll_callback ucc_coll_callback_t
```

The callback is invoked whenever the collective operation is completed. It is not allowed to call UCC APIs from the completion callback except for [ucc_collective_finalize](#).

8.8.3.6 ucc_count_t

```
typedef uint64_t ucc_count_t
```

8.8.3.7 ucc_aint_t

```
typedef uint64_t ucc_aint_t
```

8.8.3.8 ucc_coll_id_t

```
typedef uint16_t ucc_coll_id_t
```

8.8.4 Enumeration Type Documentation**8.8.4.1 ucc_memory_type**

```
enum ucc_memory_type
```

Enumerator

UCC_MEMORY_TYPE_HOST	Default system memory
UCC_MEMORY_TYPE_CUDA	NVIDIA CUDA memory
UCC_MEMORY_TYPE_CUDA_MANAGED	NVIDIA CUDA managed memory
UCC_MEMORY_TYPE_ROCM	AMD ROCM memory
UCC_MEMORY_TYPE_ROCM_MANAGED	AMD ROCM managed system memory
UCC_MEMORY_TYPE_LAST	
UCC_MEMORY_TYPE_UNKNOWN	

8.8.4.2 ucc_coll_args_flags_t

```
enum ucc_coll_args_flags_t
```

Enumerator

UCC_COLL_ARGS_FLAG_IN_PLACE	If set, the output buffer is identical to the input buffer.
UCC_COLL_ARGS_FLAG_PERSISTENT	If set, the collective is considered persistent. Only, the persistent collective can be called multiple times with the same request.
UCC_COLL_ARGS_FLAG_COUNT_64BIT	If set, the count is 64bit, otherwise, it is 32 bit.
UCC_COLL_ARGS_FLAG_DISPLACEMENTS_64BIT ↔	If set, the displacement is 64bit, otherwise, it is 32 bit.
UCC_COLL_ARGS_FLAG_CONTIG_SRC_BUFFER ↔	If set, the src buffer is considered contiguous. Particularly, useful for alltoallv operation.
UCC_COLL_ARGS_FLAG_CONTIG_DST_BUFFER ↔	If set, the dst buffer is considered contiguous. Particularly, useful for alltoallv operation.
UCC_COLL_ARGS_FLAG_TIMEOUT	If set and the elapsed time after ucc_collective_post (or ucc_collective_triggered_post) is greater than ucc_coll_args_t::timeout , the library returns UCC_ERR_TIMED_OUT on the calling thread. Note, the status is not guaranteed to be global on all the processes participating in the collective.
UCC_COLL_ARGS_FLAG_MEM_MAPPED_BUFFERS ↔	If set, both src and dst buffers reside in a memory mapped region. Useful for one-sided collectives.

8.8.4.3 ucc_error_type_t

enum `ucc_error_type_t`

Enumerator

UCC_ERR_TYPE_LOCAL	
UCC_ERR_TYPE_GLOBAL	

8.8.4.4 ucc_coll_args_field

enum `ucc_coll_args_field`

Enumerator

UCC_COLL_ARGS_FIELD_FLAGS	
UCC_COLL_ARGS_FIELD_TAG	
UCC_COLL_ARGS_FIELD_CB	
UCC_COLL_ARGS_FIELD_GLOBAL_WORK_BUFFER	

8.9 Collective Operations

Data Structures

- struct `ucc_coll_args`
Structure representing arguments for the collective operations. [More...](#)
- union `ucc_coll_args.src`
- union `ucc_coll_args.dst`

Typedefs

- typedef struct `ucc_coll_args` `ucc_coll_args_t`
Structure representing arguments for the collective operations.
- typedef struct `ucc_mem_handle` * `ucc_mem_h`
UCC memory handle.

Functions

- `ucc_status_t ucc_collective_init` (`ucc_coll_args_t` *coll_args, `ucc_coll_req_h` *request, `ucc_team_h` team)
The routine to initialize a collective operation.
- `ucc_status_t ucc_collective_post` (`ucc_coll_req_h` request)
The routine to post a collective operation.
- `ucc_status_t ucc_collective_init_and_post` (`ucc_coll_args_t` *coll_args, `ucc_coll_req_h` *request, `ucc_team_h` team)
The routine to initialize and post a collective operation.
- static `ucc_status_t ucc_collective_test` (`ucc_coll_req_h` request)
The routine to query the status of the collective operation.
- `ucc_status_t ucc_collective_finalize` (`ucc_coll_req_h` request)
The routine to release the collective operation associated with the request object.

8.9.1 Detailed Description

Collective operations invocation and progress

8.9.2 Data Structure Documentation

8.9.2.1 struct ucc_coll_args

Description

`ucc_coll_args_t` defines the parameters that can be used to customize the collective operation. The "mask" bit array fields are defined by `ucc_coll_args_field`. The bits in "mask" bit array is defined by `ucc_coll_args_field`, which correspond to fields in structure `ucc_coll_args_t`. The valid fields of the structure are specified by setting the corresponding bit to "1" in the bit-array "mask".

The collective operation is selected by field "coll_type" which must be always set by user. If allreduce or * reduce operation is selected, the type of reduction is selected by the field * "predefined_reduction_op" or "custom_reduction_op". For unordered collective operations, the user-provided "tag" value orders the collective operation. For rooted collective operations such as reduce, scatter, gather, fan-in, and fan-out, the "root" field must be provided by user and specify the participant endpoint value. The user can request either "local" or "global" error information using the "error_type" field.

Information about user buffers used for collective operation must be specified according to the "coll_type".

Data Fields

<code>uint64_t</code>	mask	
<code>ucc_coll_type_t</code>	coll_type	Type of collective operation
union <code>ucc_coll_args.src</code>	src	
union <code>ucc_coll_args.dst</code>	dst	
<code>ucc_reduction_op_t</code>	op	Predefined reduction operation, if reduce, allreduce, reduce_scatter operation is selected. The field is only specified for collectives that use pre-defined datatypes
<code>uint64_t</code>	flags	
<code>uint64_t</code>	root	Root endpoint for rooted collectives
<code>ucc_error_type_t</code>	error_type	Error type
<code>ucc_coll_id_t</code>	tag	Used for ordering collectives
<code>void *</code>	global_work_buffer	User allocated scratchpad buffer for one-sided collectives. The buffer provided should be at least the size returned by <code>ucc_context_get_attr</code> with the field mask - <code>UCC_CONTEXT_ATTR_FIELD_WORK_BUFFER_SIZE</code> set to 1.
<code>ucc_coll_callback_t</code>	cb	
<code>double</code>	timeout	Timeout in seconds

8.9.2.2 union ucc_coll_args.src

Data Fields

<code>ucc_coll_buffer_info_t</code>	info	Buffer info for the collective
<code>ucc_coll_buffer_info_v_t</code>	info_v	Buffer info for the collective

8.9.2.3 union ucc_coll_args.dst

Data Fields

ucc_coll_buffer_info_t	info	Buffer info for the collective
ucc_coll_buffer_info_v_t	info_v	Buffer info for the collective

8.9.3 Typedef Documentation

8.9.3.1 ucc_coll_args_t

```
typedef struct ucc_coll_args ucc_coll_args_t
```

Description

[ucc_coll_args_t](#) defines the parameters that can be used to customize the collective operation. The "mask" bit array fields are defined by [ucc_coll_args_field](#). The bits in "mask" bit array is defined by [ucc_coll_args_field](#), which correspond to fields in structure [ucc_coll_args_t](#). The valid fields of the structure are specified by setting the corresponding bit to "1" in the bit-array "mask".

The collective operation is selected by field "coll_type" which must be always set by user. If allreduce or * reduce operation is selected, the type of reduction is selected by the field * "predefined_reduction_op" or "custom_reduction_op". For unordered collective operations, the user-provided "tag" value orders the collective operation. For rooted collective operations such as reduce, scatter, gather, fan-in, and fan-out, the "root" field must be provided by user and specify the participant endpoint value. The user can request either "local" or "global" error information using the "error_type" field.

Information about user buffers used for collective operation must be specified according to the "coll_type".

8.9.3.2 ucc_mem_h

```
typedef struct ucc_mem_handle* ucc_mem_h
```

The UCC memory handle is an opaque handle created by the library representing the buffer and address.

8.9.4 Function Documentation

8.9.4.1 ucc_collective_init()

```
ucc_status_t ucc_collective_init (
    ucc_coll_args_t * coll_args,
    ucc_coll_req_h * request,
    ucc_team_h team )
```

Parameters

out	<i>request</i>	Request handle representing the collective operation
in	<i>coll_args</i>	Collective arguments descriptor
in	<i>team</i>	Team handle

Description

[ucc_collective_init](#) is a collective initialization operation, where all participants participate. The user provides all information required to start and complete the collective operation, which includes the input and output

buffers, operation type, team handle, size, and any other hints for optimization. On success, the request handle is created and returned. On error, the request handle is not created and the appropriate error code is returned. On return, the ownership of buffers is transferred to the user. If modified, the results of collective operations posted on the request handle are undefined.

Returns

Error code as defined by [ucc_status_t](#)

8.9.4.2 ucc_collective_post()

```
ucc_status_t ucc_collective_post (
    ucc_coll_req_h request )
```

Parameters

in	<i>request</i>	Request handle
----	----------------	----------------

Description

[ucc_collective_post](#) routine posts the collective operation. It does not require synchronization between the participants for the post operation.

Returns

Error code as defined by [ucc_status_t](#)

8.9.4.3 ucc_collective_init_and_post()

```
ucc_status_t ucc_collective_init_and_post (
    ucc_coll_args_t * coll_args,
    ucc_coll_req_h * request,
    ucc_team_h team )
```

Parameters

out	<i>request</i>	Request handle representing the collective operation
in	<i>coll_args</i>	Collective arguments descriptor
in	<i>team</i>	Input Team

Description

[ucc_collective_init_and_post](#) initializes the collective operation and also posts the operation.

Note

: The [ucc_collective_init_and_post](#) can be implemented as a combination of [ucc_collective_init](#) and [ucc_collective_post](#) routines.

Returns

Error code as defined by [ucc_status_t](#)

8.9.4.4 ucc_collective_test()

```
static ucc_status_t ucc_collective_test (
    ucc_coll_req_h request ) [inline], [static]
```

Parameters

in	<i>request</i>	Request handle
----	----------------	----------------

Description

[ucc_collective_test](#) tests and returns the status of collective operation.

Returns

Error code as defined by [ucc_status_t](#)

8.9.4.5 ucc_collective_finalize()

```
ucc_status_t ucc_collective_finalize (
    ucc_coll_req_h request )
```

Parameters

in	<i>request</i>	- request handle
----	----------------	------------------

Description

[ucc_collective_finalize](#) operation releases all resources associated with the collective operation represented by the request handle.

Returns

Error code as defined by [ucc_status_t](#)

8.10 Events and Triggered operations' datastructures

Data Structures

- struct [ucc_event](#)
- struct [ucc_ee_params](#)

Typedefs

- typedef enum [ucc_event_type](#) [ucc_event_type_t](#)
- typedef enum [ucc_ee_type](#) [ucc_ee_type_t](#)
- typedef struct [ucc_event](#) [ucc_ev_t](#)
- typedef struct [ucc_ee_params](#) [ucc_ee_params_t](#)

Enumerations

- enum [ucc_event_type](#) {
[UCC_EVENT_COLLECTIVE_POST](#) = UCC_BIT(0) ,
[UCC_EVENT_COLLECTIVE_COMPLETE](#) = UCC_BIT(1) ,
[UCC_EVENT_COMPUTE_COMPLETE](#) = UCC_BIT(2) ,
[UCC_EVENT_OVERFLOW](#) = UCC_BIT(3) }
- enum [ucc_ee_type](#) {
[UCC_EE_CUDA_STREAM](#) = 0 ,
[UCC_EE_CPU_THREAD](#) ,
[UCC_EE_LAST](#) ,
[UCC_EE_UNKNOWN](#) = UCC_EE_LAST }

8.10.1 Detailed Description

Data-structures associated with event-driven collective execution

8.10.2 Data Structure Documentation

8.10.2.1 struct ucc_event

Data Fields

ucc_event_type_t	ev_type	
void *	ev_context	
size_t	ev_context_size	
ucc_coll_req_h	req	

8.10.2.2 struct ucc_ee_params

Data Fields

ucc_ee_type_t	ee_type	
void *	ee_context	
size_t	ee_context_size	

8.10.3 Typedef Documentation

8.10.3.1 ucc_event_type_t

```
typedef enum ucc\_event\_type ucc_event_type_t
```

8.10.3.2 ucc_ee_type_t

```
typedef enum ucc\_ee\_type ucc_ee_type_t
```

8.10.3.3 ucc_ev_t

```
typedef struct ucc\_event ucc_ev_t
```

8.10.3.4 ucc_ee_params_t

```
typedef struct ucc\_ee\_params ucc_ee_params_t
```

8.10.4 Enumeration Type Documentation

8.10.4.1 ucc_event_type

```
enum ucc\_event\_type
```

Enumerator

UCC_EVENT_COLLECTIVE_POST	
UCC_EVENT_COLLECTIVE_COMPLETE	
UCC_EVENT_COMPUTE_COMPLETE	
UCC_EVENT_OVERFLOW	

8.10.4.2 ucc_ee_type

enum `ucc_ee_type`

Enumerator

UCC_EE_CUDA_STREAM	
UCC_EE_CPU_THREAD	
UCC_EE_LAST	
UCC_EE_UNKNOWN	

8.11 Events and Triggered Operations

Functions

- `ucc_status_t ucc_ee_create (ucc_team_h team, const ucc_ee_params_t *params, ucc_ee_h *ee)`
The routine creates the execution context for collective operations.
- `ucc_status_t ucc_ee_destroy (ucc_ee_h ee)`
The routine destroys the execution context created for collective operations.
- `ucc_status_t ucc_ee_get_event (ucc_ee_h ee, ucc_ev_t **ev)`
The routine gets the event from the event queue.
- `ucc_status_t ucc_ee_ack_event (ucc_ee_h ee, ucc_ev_t *ev)`
The routine acks the events from the event queue.
- `ucc_status_t ucc_ee_set_event (ucc_ee_h ee, ucc_ev_t *ev)`
The routine to set the event to the tail of the queue.
- `ucc_status_t ucc_ee_wait (ucc_ee_h ee, ucc_ev_t *ev)`
The routine blocks the calling thread until there is an event on the queue.
- `ucc_status_t ucc_collective_triggered_post (ucc_ee_h ee, ucc_ev_t *ee_event)`
The routine posts the collective operation on the execution engine, which is launched on the event.

8.11.1 Detailed Description

Event-driven Collective Execution

8.11.2 Function Documentation

8.11.2.1 ucc_ee_create()

```
ucc_status_t ucc_ee_create (
    ucc_team_h team,
    const ucc_ee_params_t * params,
    ucc_ee_h * ee )
```

Parameters

in	<i>team</i>	team handle
in	<i>params</i>	user provided params to customize the execution engine
out	<i>ee</i>	execution engine handle

Description

[ucc_ee_create](#) creates the execution engine. It enables event-driven collective execution. [ucc_ee_params_t](#) allows the execution engine to be configured to abstract either GPU and CPU threads. The execution engine is created and coupled with the team. There can be many execution engines coupled to the team. However, attaching the same execution engine to multiple teams is not allowed. The execution engine is created after the team is created and destroyed before the team is destroyed. It is the user's responsibility to destroy the execution engines before the team. If the team is destroyed before the execution engine is destroyed, the result is undefined.

Returns

Error code as defined by [ucc_status_t](#)

8.11.2.2 ucc_ee_destroy()

```
ucc_status_t ucc_ee_destroy (
    ucc_ee_h ee )
```

Parameters

in	ee	Execution engine handle
----	----	-------------------------

Description

[ucc_ee_destroy](#) releases the resources attached with the execution engine and destroys the execution engine. All events and triggered operations related to this ee are invalid after the destroy operation. To avoid race between the creation and destroying the execution engine, for a given ee, the [ucc_ee_create](#) and [ucc_ee_destroy](#) must be invoked from the same thread.

Returns

Error code as defined by [ucc_status_t](#)

8.11.2.3 ucc_ee_get_event()

```
ucc_status_t ucc_ee_get_event (
    ucc_ee_h ee,
    ucc_ev_t ** ev )
```

Parameters

in	ee	execution engine handle
out	ev	Event structure fetched from the event queue

Description

[ucc_ee_get_event](#) fetches the events from the execution engine. If there are no events posted on the ee, it returns immediately without waiting for events. All events must be acknowledged using the [ucc_ee_ack_event](#) interface. The event acknowledged is destroyed by the library. An event fetched with [ucc_ee_get_event](#) but not acknowledged might consume resources in the library.

Returns

Error code as defined by [ucc_status_t](#)

8.11.2.4 ucc_ee_ack_event()

```
ucc_status_t ucc_ee_ack_event (
```

```
ucc_ee_h ee,
ucc_ev_t * ev )
```

Parameters

in	ee	execution engine handle
in	ev	Event to be acked

Description

An event acknowledged by the user using `ucc_ee_ack_event` is destroyed by the library. Any triggered operations on the event should be completed before calling this interface. The behavior is undefined if the user acknowledges the event while waiting on the event or triggering operations on the event.

Returns

Error code as defined by `ucc_status_t`

8.11.2.5 ucc_ee_set_event()

```
ucc_status_t ucc_ee_set_event (
    ucc_ee_h ee,
    ucc_ev_t * ev )
```

Parameters

in	ee	execution engine handle
in	ev	Event structure fetched from the event queue

Description

`ucc_ee_set_event` sets the event on the execution engine. If the operations are waiting on the event when the user sets the event, the operations are launched. The events created by the user need to be destroyed by the user.

Returns

Error code as defined by `ucc_status_t`

8.11.2.6 ucc_ee_wait()

```
ucc_status_t ucc_ee_wait (
    ucc_ee_h ee,
    ucc_ev_t * ev )
```

Parameters

in	ee	execution engine handle
out	ev	Event structure fetched from the event queue

Description

The user thread invoking the `ucc_ee_wait` interface is blocked until an event is posted to the execution engine.

Returns

Error code as defined by `ucc_status_t`

8.11.2.7 ucc_collective_triggered_post()

```
ucc_status_t ucc_collective_triggered_post (
    ucc_ee_h ee,
    ucc_ev_t * ee_event )
```

Parameters

in	ee	execution engine handle
in	ee_event	Event triggering the post operation

Description

`ucc_collective_triggered_post` allow the users to schedule a collective operation that executes in the future when an event occurs on the execution engine.

Returns

Error code as defined by `ucc_status_t`

8.12 Utility Operations

Enumerations

- enum `ucc_config_print_flags_t` {
`UCC_CONFIG_PRINT_CONFIG` = `UCC_BIT(0)` ,
`UCC_CONFIG_PRINT_HEADER` = `UCC_BIT(1)` ,
`UCC_CONFIG_PRINT_DOC` = `UCC_BIT(2)` ,
`UCC_CONFIG_PRINT_HIDDEN` = `UCC_BIT(3)` }

Print configurations.

- enum `ucc_status_t` {
`UCC_OK` = 0 ,
`UCC_INPROGRESS` = 1 ,
`UCC_OPERATION_INITIALIZED` = 2 ,
`UCC_ERR_NOT_SUPPORTED` = -1 ,
`UCC_ERR_NOT_IMPLEMENTED` = -2 ,
`UCC_ERR_INVALID_PARAM` = -3 ,
`UCC_ERR_NO_MEMORY` = -4 ,
`UCC_ERR_NO_RESOURCE` = -5 ,
`UCC_ERR_NO_MESSAGE` = -6 ,
`UCC_ERR_NOT_FOUND` = -7 ,
`UCC_ERR_TIMED_OUT` = -8 ,
`UCC_ERR_LAST` = -100 }

Status codes for the UCC operations.

Functions

- const char * `ucc_status_string` (`ucc_status_t` status)

Routine to convert status code to string.

8.12.1 Detailed Description

Helper functions to be used across the library

8.12.2 Enumeration Type Documentation

8.12.2.1 ucc_config_print_flags_t

enum `ucc_config_print_flags_t`

Enumerator

UCC_CONFIG_PRINT_CONFIG	
UCC_CONFIG_PRINT_HEADER	
UCC_CONFIG_PRINT_DOC	
UCC_CONFIG_PRINT_HIDDEN	

8.12.2.2 ucc_status_t

enum `ucc_status_t`

Enumerator

UCC_OK	
UCC_INPROGRESS	Operation is posted and is in progress
UCC_OPERATION_INITIALIZED	Operation initialized but not posted
UCC_ERR_NOT_SUPPORTED	
UCC_ERR_NOT_IMPLEMENTED	
UCC_ERR_INVALID_PARAM	
UCC_ERR_NO_MEMORY	
UCC_ERR_NO_RESOURCE	
UCC_ERR_NO_MESSAGE	General purpose return code without specific error
UCC_ERR_NOT_FOUND	
UCC_ERR_TIMED_OUT	
UCC_ERR_LAST	

8.12.3 Function Documentation

8.12.3.1 ucc_status_string()

```
const char * ucc_status_string (
    ucc_status_t status )
```

Chapter 9

Data Structure Documentation

9.1 ucc_coll_callback Struct Reference

UCC collective completion callback.

Data Fields

- void(* [cb](#))(void *[data](#), [ucc_status_t](#) status)
- void * [data](#)

9.1.1 Detailed Description

The callback is invoked whenever the collective operation is completed. It is not allowed to call UCC APIs from the completion callback except for [ucc_collective_finalize](#).

9.1.2 Field Documentation

9.1.2.1 cb

```
void(* ucc_coll_callback::cb) (void *data, ucc\_status\_t status)
```

9.1.2.2 data

```
void* ucc_coll_callback::data
```

The documentation for this struct was generated from the following file:

- [ucc_def.h](#)

9.2 ucc_ep_map_cb Struct Reference

Data Fields

- [uint64_t](#)(* [cb](#))([uint64_t](#) ep, void *[cb_ctx](#))
- void * [cb_ctx](#)

9.2.1 Field Documentation

9.2.1.1 cb

```
uint64\_t(* ucc_ep_map_cb::cb) (uint64\_t ep, void *cb\_ctx)
```

9.2.1.2 cb_ctx

void* ucc_ep_map_cb::cb_ctx

The documentation for this struct was generated from the following file:

- ucc.h

9.3 ucc_generic_dt_ops Struct Reference

UCC generic data type descriptor.

Data Fields

- uint64_t [mask](#)
- uint64_t [flags](#)
- size_t [contig_size](#)
- void (*)([start_pack](#))(void *context, const void *buffer, size_t count)
Start a packing request.
- void (*)([start_unpack](#))(void *context, void *buffer, size_t count)
Start an unpacking request.
- size_t (*)([packed_size](#))(void *state)
Get the total size of packed data.
- size_t (*)([pack](#))(void *state, size_t offset, void *dest, size_t max_length)
Pack data.
- [ucc_status_t](#) (*)([unpack](#))(void *state, size_t offset, const void *src, size_t length)
Unpack data.
- void (*)([finish](#))(void *state)
Finish packing/unpacking.
- struct {
 [ucc_status_t](#) (*)([cb](#))(const [ucc_reduce_cb_params_t](#) *params)
 void * [cb_ctx](#)
} [reduce](#)

User-defined reduction callback.

9.3.1 Detailed Description

This structure provides a generic datatype descriptor that is used to create user-defined datatypes.

9.3.2 Field Documentation

9.3.2.1 mask

uint64_t ucc_generic_dt_ops::mask

9.3.2.2 flags

uint64_t ucc_generic_dt_ops::flags

9.3.2.3 contig_size

size_t ucc_generic_dt_ops::contig_size

size of the datatype if [UCC_GENERIC_DT_OPS_FLAG_CONTIG](#) is set

The documentation for this struct was generated from the following file:

- [ucc.h](#)

9.4 ucc_generic_dt_ops.reduce Struct Reference

User-defined reduction callback.

Data Fields

- [ucc_status_t](#)(* [cb](#))(const [ucc_reduce_cb_params_t](#) *params)
- void * [cb_ctx](#)

9.4.1 Detailed Description

The pointer refers to user-defined reduction routine.

Parameters

<i>in</i>	<i>params</i>	reduction descriptor
-----------	---------------	----------------------

9.4.2 Field Documentation

9.4.2.1 cb

9.4.2.2 cb_ctx

The documentation for this struct was generated from the following files:

9.5 ucc_oob_coll Struct Reference

OOB collective operation for creating the context.

Data Fields

- [ucc_status_t](#)(* [allgather](#))(void *src_buf, void *recv_buf, size_t size, void *allgather_info, void **request)
- [ucc_status_t](#)(* [req_test](#))(void *request)
- [ucc_status_t](#)(* [req_free](#))(void *request)
- void * [coll_info](#)
- uint32_t [n_oob_eps](#)
- uint32_t [oob_ep](#)

9.5.1 Field Documentation

9.5.1.1 allgather

```
ucc_status_t(* ucc_oob_coll::allgather) (void *src_buf, void *recv_buf, size_t size, void *allgather←
_info, void **request)
```

9.5.1.2 req_test

```
ucc_status_t(* ucc_oob_coll::req_test) (void *request)
```

9.5.1.3 req_free

```
ucc_status_t(* ucc_oob_coll::req_free) (void *request)
```

9.5.1.4 coll_info

```
void* ucc_oob_coll::coll_info
```

9.5.1.5 n_oob_eps

```
uint32_t ucc_oob_coll::n_oob_eps
```

Number of endpoints participating in the oob operation (e.g., number of processes representing a ucc team)

9.5.1.6 oob_ep

```
uint32_t ucc_oob_coll::oob_ep
```

Integer value that represents the position of the calling processes in the given oob op: the data specified by "src_buf" will be placed at the offset "oob_ep*size" in the "recv_buf". oob_ep must be uniq at every calling process and shuold be in the range [0:n_oob_eps).

The documentation for this struct was generated from the following file:

- ucc.h

9.6 ucc_team_p2p_conn Struct Reference

Data Fields

- int(* conn_info_lookup)(void *conn_ctx, uint64_t ep, ucc_p2p_conn_t **conn_info, void *request)
- int(* conn_info_release)(ucc_p2p_conn_t *conn_info)
- void * conn_ctx
- ucc_status_t(* req_test)(void *request)
- ucc_status_t(* req_free)(void *request)

9.6.1 Field Documentation

9.6.1.1 conn_info_lookup

```
int(* ucc_team_p2p_conn::conn_info_lookup) (void *conn_ctx, uint64_t ep, ucc_p2p_conn_t **conn←
_info, void *request)
```

9.6.1.2 conn_info_release

```
int(* ucc_team_p2p_conn::conn_info_release) (ucc_p2p_conn_t *conn_info)
```

9.6.1.3 conn_ctx

```
void* ucc_team_p2p_conn::conn_ctx
```

9.6.1.4 req_test

```
ucc_status_t(* ucc_team_p2p_conn::req_test) (void *request)
```

9.6.1.5 req_free

```
ucc_status_t(* ucc_team_p2p_conn::req_free) (void *request)
```

The documentation for this struct was generated from the following file:

- ucc.h

Index

- allgather
 - ucc_oob_coll, 64
- cb
 - Datatypes data-structures and functions, 24
 - ucc_coll_callback, 62
 - ucc_ep_map_cb, 62
 - ucc_generic_dt_ops.reduce, 64
- cb_ctx
 - Datatypes data-structures and functions, 24
 - ucc_ep_map_cb, 62
 - ucc_generic_dt_ops.reduce, 64
- coll_info
 - ucc_oob_coll, 65
- Collective Operations, 50
 - ucc_coll_args_t, 52
 - ucc_collective_finalize, 54
 - ucc_collective_init, 52
 - ucc_collective_init_and_post, 53
 - ucc_collective_post, 53
 - ucc_collective_test, 53
 - ucc_mem_h, 52
- Collective operations data-structures, 46
 - ucc_aint_t, 48
 - ucc_coll_args_field, 50
 - UCC_COLL_ARGS_FIELD_CB, 50
 - UCC_COLL_ARGS_FIELD_FLAGS, 50
 - UCC_COLL_ARGS_FIELD_GLOBAL_WORK_BUFFER, 50
 - UCC_COLL_ARGS_FIELD_TAG, 50
 - UCC_COLL_ARGS_FLAG_CONTIG_DST_BUFFER, 49
 - UCC_COLL_ARGS_FLAG_CONTIG_SRC_BUFFER, 49
 - UCC_COLL_ARGS_FLAG_COUNT_64BIT, 49
 - UCC_COLL_ARGS_FLAG_DISPLACEMENTS_64BIT, 49
 - UCC_COLL_ARGS_FLAG_IN_PLACE, 49
 - UCC_COLL_ARGS_FLAG_MEM_MAPPED_BUFFERS, 49
 - UCC_COLL_ARGS_FLAG_PERSISTENT, 49
 - UCC_COLL_ARGS_FLAG_TIMEOUT, 49
 - ucc_coll_args_flags_t, 49
 - ucc_coll_buffer_info_t, 48
 - ucc_coll_buffer_info_v_t, 48
 - ucc_coll_callback_t, 48
 - ucc_coll_id_t, 49
 - ucc_coll_req_h, 48
 - ucc_count_t, 48
 - UCC_ERR_TYPE_GLOBAL, 50
 - UCC_ERR_TYPE_LOCAL, 50
 - ucc_error_type_t, 50
 - ucc_memory_type, 49
 - UCC_MEMORY_TYPE_CUDA, 49
 - UCC_MEMORY_TYPE_CUDA_MANAGED, 49
 - UCC_MEMORY_TYPE_HOST, 49
 - UCC_MEMORY_TYPE_LAST, 49
 - UCC_MEMORY_TYPE_ROCM, 49
 - UCC_MEMORY_TYPE_ROCM_MANAGED, 49
 - ucc_memory_type_t, 48
 - UCC_MEMORY_TYPE_UNKNOWN, 49
- conn_ctx
 - ucc_team_p2p_conn, 65
- conn_info_lookup
 - ucc_team_p2p_conn, 65
- conn_info_release
 - ucc_team_p2p_conn, 65
- Context abstraction data-structures, 28
 - ucc_context_attr_field, 32
 - UCC_CONTEXT_ATTR_FIELD_CTX_ADDR, 32
 - UCC_CONTEXT_ATTR_FIELD_CTX_ADDR_LEN, 32
 - UCC_CONTEXT_ATTR_FIELD_SYNC_TYPE, 32
 - UCC_CONTEXT_ATTR_FIELD_TYPE, 32
 - UCC_CONTEXT_ATTR_FIELD_WORK_BUFFER_SIZE, 32
 - ucc_context_attr_t, 31
 - ucc_context_config_h, 31
 - UCC_CONTEXT_EXCLUSIVE, 31
 - ucc_context_h, 31
 - UCC_CONTEXT_PARAM_FIELD_ID, 32
 - UCC_CONTEXT_PARAM_FIELD_MEM_PARAMS, 32
 - UCC_CONTEXT_PARAM_FIELD_OOB, 32
 - UCC_CONTEXT_PARAM_FIELD_SYNC_TYPE, 32
 - UCC_CONTEXT_PARAM_FIELD_TYPE, 32
 - ucc_context_params_field, 32
 - ucc_context_params_t, 31
 - UCC_CONTEXT_SHARED, 31
 - ucc_context_type_t, 31
 - ucc_mem_map_params_t, 31
 - ucc_mem_map_t, 31
 - ucc_oob_coll_t, 30

- Context abstraction routines, 32
 - ucc_context_config_modify, 34
 - ucc_context_config_print, 33
 - ucc_context_config_read, 33
 - ucc_context_config_release, 33
 - ucc_context_create, 34
 - ucc_context_destroy, 35
 - ucc_context_get_attr, 35
 - ucc_context_progress, 35
- contig_size
 - ucc_generic_dt_ops, 63
- data
 - ucc_coll_callback, 62
- Datatypes data-structures and functions, 19
 - cb, 24
 - cb_ctx, 24
 - finish, 24
 - pack, 23
 - packed_size, 23
 - reduce, 24
 - start_pack, 22
 - start_unpack, 23
 - ucc_datatype_t, 21
 - ucc_dt_create_generic, 22
 - ucc_dt_destroy, 22
 - ucc_generic_dt_ops_field, 21
 - UCC_GENERIC_DT_OPS_FIELD_FLAGS, 21
 - UCC_GENERIC_DT_OPS_FLAG_CONTIG, 22
 - UCC_GENERIC_DT_OPS_FLAG_REDUCE, 22
 - ucc_generic_dt_ops_flags_t, 21
 - ucc_generic_dt_ops_t, 21
 - ucc_reduce_cb_params_t, 21
 - unpack, 24
- Events and Triggered Operations, 56
 - ucc_collective_triggered_post, 58
 - ucc_ee_ack_event, 57
 - ucc_ee_create, 56
 - ucc_ee_destroy, 57
 - ucc_ee_get_event, 57
 - ucc_ee_set_event, 58
 - ucc_ee_wait, 58
- Events and Triggered operations' datastructures
 - UCC_EE_CPU_THREAD, 56
 - UCC_EE_CUDA_STREAM, 56
 - UCC_EE_LAST, 56
 - UCC_EE_UNKNOWN, 56
 - UCC_EVENT_COLLECTIVE_COMPLETE, 55
 - UCC_EVENT_COLLECTIVE_POST, 55
 - UCC_EVENT_COMPUTE_COMPLETE, 55
 - UCC_EVENT_OVERFLOW, 55
- Events and Triggered operations' datastructures, 54
 - ucc_ee_params_t, 55
 - ucc_ee_type, 56
 - ucc_ee_type_t, 55
 - ucc_ev_t, 55
 - ucc_event_type, 55
 - ucc_event_type_t, 55
- finish
 - Datatypes data-structures and functions, 24
- flags
 - ucc_generic_dt_ops, 63
- Library initialization and finalization routines, 25
 - ucc_finalize, 27
 - ucc_init, 26
 - ucc_lib_config_modify, 26
 - ucc_lib_config_print, 26
 - ucc_lib_config_read, 25
 - ucc_lib_config_release, 25
 - ucc_lib_get_attr, 27
- Library initialization data-structures, 14
 - ucc_coll_sync_type_t, 18
 - UCC_COLL_TYPE_ALLGATHER, 17
 - UCC_COLL_TYPE_ALLGATHERV, 17
 - UCC_COLL_TYPE_ALLREDUCE, 17
 - UCC_COLL_TYPE_ALLTOALL, 17
 - UCC_COLL_TYPE_ALLTOALLV, 17
 - UCC_COLL_TYPE_BARRIER, 17
 - UCC_COLL_TYPE_BCAST, 17
 - UCC_COLL_TYPE_FANIN, 17
 - UCC_COLL_TYPE_FANOUT, 17
 - UCC_COLL_TYPE_GATHER, 17
 - UCC_COLL_TYPE_GATHERV, 17
 - UCC_COLL_TYPE_LAST, 17
 - UCC_COLL_TYPE_REDUCE, 17
 - UCC_COLL_TYPE_REDUCE_SCATTER, 17
 - UCC_COLL_TYPE_REDUCE_SCATTERV, 17
 - UCC_COLL_TYPE_SCATTER, 17
 - UCC_COLL_TYPE_SCATTERV, 17
 - ucc_coll_type_t, 17
 - ucc_lib_attr_field, 19
 - UCC_LIB_ATTR_FIELD_COLL_TYPES, 19
 - UCC_LIB_ATTR_FIELD_REDUCTION_TYPES, 19
 - UCC_LIB_ATTR_FIELD_SYNC_TYPE, 19
 - UCC_LIB_ATTR_FIELD_THREAD_MODE, 19
 - ucc_lib_attr_t, 16
 - ucc_lib_config_h, 17
 - ucc_lib_h, 17
 - UCC_LIB_PARAM_FIELD_COLL_TYPES, 19
 - UCC_LIB_PARAM_FIELD_REDUCTION_TYPES, 19
 - UCC_LIB_PARAM_FIELD_SYNC_TYPE, 19
 - UCC_LIB_PARAM_FIELD_THREAD_MODE, 19
 - ucc_lib_params_field, 19
 - ucc_lib_params_t, 16
 - UCC_NO_SYNC_COLLECTIVES, 19

- UCC_OP_AVG, 18
- UCC_OP_BAND, 18
- UCC_OP_BOR, 18
- UCC_OP_BXOR, 18
- UCC_OP_LAND, 18
- UCC_OP_LAST, 18
- UCC_OP_LOR, 18
- UCC_OP_LXOR, 18
- UCC_OP_MAX, 18
- UCC_OP_MAXLOC, 18
- UCC_OP_MIN, 18
- UCC_OP_MINLOC, 18
- UCC_OP_PROD, 18
- UCC_OP_SUM, 18
- ucc_reduction_op_t, 17
- UCC_SYNC_COLLECTIVES, 19
- UCC_THREAD_FUNNELED, 18
- ucc_thread_mode_t, 18
- UCC_THREAD_MULTIPLE, 18
- UCC_THREAD_SINGLE, 18
- mask
 - ucc_generic_dt_ops, 63
- n_oob_eps
 - ucc_oob_coll, 65
- oob_ep
 - ucc_oob_coll, 65
- pack
 - Datatypes data-structures and functions, 23
- packed_size
 - Datatypes data-structures and functions, 23
- reduce
 - Datatypes data-structures and functions, 24
- req_free
 - ucc_oob_coll, 65
 - ucc_team_p2p_conn, 66
- req_test
 - ucc_oob_coll, 65
 - ucc_team_p2p_conn, 66
- start_pack
 - Datatypes data-structures and functions, 22
- start_unpack
 - Datatypes data-structures and functions, 23
- Team abstraction data-structures, 36
 - UCC_COLLECTIVE_EP_RANGE_CONTIG, 42
 - UCC_COLLECTIVE_EP_RANGE_NONCONTIG, 42
 - UCC_COLLECTIVE_INIT_AND_POST_ORDERED, 42
 - UCC_COLLECTIVE_INIT_AND_POST_UNORDERED, 42
 - UCC_COLLECTIVE_INIT_ORDERED, 42
 - UCC_COLLECTIVE_INIT_UNORDERED, 42
 - UCC_COLLECTIVE_POST_ORDERED, 42
 - UCC_COLLECTIVE_POST_UNORDERED, 42
 - ucc_context_addr_h, 41
 - ucc_context_addr_len_t, 41
 - UCC_EP_MAP_ARRAY, 43
 - UCC_EP_MAP_CB, 43
 - UCC_EP_MAP_FULL, 43
 - UCC_EP_MAP_STRIDED, 43
 - ucc_ep_map_t, 40
 - ucc_ep_map_type_t, 42
 - ucc_ep_range_type_t, 42
 - ucc_p2p_conn_t, 41
 - ucc_post_ordering_t, 42
 - ucc_team_attr_field, 41
 - UCC_TEAM_ATTR_FIELD_EP, 42
 - UCC_TEAM_ATTR_FIELD_EP_RANGE, 42
 - UCC_TEAM_ATTR_FIELD_MEM_PARAMS, 42
 - UCC_TEAM_ATTR_FIELD_OUTSTANDING_CALLS, 42
 - UCC_TEAM_ATTR_FIELD_POST_ORDERING, 42
 - UCC_TEAM_ATTR_FIELD_SYNC_TYPE, 42
 - ucc_team_attr_t, 40
 - UCC_TEAM_FLAG_COLL_WORK_BUFFER, 42
 - ucc_team_flags, 42
 - ucc_team_h, 41
 - ucc_team_p2p_conn_t, 40
 - UCC_TEAM_PARAM_FIELD_EP, 41
 - UCC_TEAM_PARAM_FIELD_EP_LIST, 41
 - UCC_TEAM_PARAM_FIELD_EP_MAP, 41
 - UCC_TEAM_PARAM_FIELD_EP_RANGE, 41
 - UCC_TEAM_PARAM_FIELD_FLAGS, 41
 - UCC_TEAM_PARAM_FIELD_ID, 41
 - UCC_TEAM_PARAM_FIELD_MEM_PARAMS, 41
 - UCC_TEAM_PARAM_FIELD_OOB, 41
 - UCC_TEAM_PARAM_FIELD_ORDERING, 41
 - UCC_TEAM_PARAM_FIELD_OUTSTANDING_COLL, 41
 - UCC_TEAM_PARAM_FIELD_P2P_CONN, 41
 - UCC_TEAM_PARAM_FIELD_SYNC_TYPE, 41
 - UCC_TEAM_PARAM_FIELD_TEAM_SIZE, 41
 - ucc_team_params_field, 41
 - ucc_team_params_t, 40
 - Team abstraction routines, 43
 - ucc_team_create_from_parent, 45
 - ucc_team_create_post, 43
 - ucc_team_create_test, 44
 - ucc_team_destroy, 44

- ucc_team_get_all_eps, 46
- ucc_team_get_attr, 44
- ucc_team_get_my_ep, 46
- ucc_team_get_size, 45
- ucc_aint_t
 - Collective operations data-structures, 48
- ucc_coll_args, 51
- ucc_coll_args.dst, 52
- ucc_coll_args.src, 51
- ucc_coll_args_field
 - Collective operations data-structures, 50
- UCC_COLL_ARGS_FIELD_CB
 - Collective operations data-structures, 50
- UCC_COLL_ARGS_FIELD_FLAGS
 - Collective operations data-structures, 50
- UCC_COLL_ARGS_FIELD_GLOBAL_WORK_BUFFER
 - Collective operations data-structures, 50
- UCC_COLL_ARGS_FIELD_TAG
 - Collective operations data-structures, 50
- UCC_COLL_ARGS_FLAG_CONTIG_DST_BUFFER
 - Collective operations data-structures, 49
- UCC_COLL_ARGS_FLAG_CONTIG_SRC_BUFFER
 - Collective operations data-structures, 49
- UCC_COLL_ARGS_FLAG_COUNT_64BIT
 - Collective operations data-structures, 49
- UCC_COLL_ARGS_FLAG_DISPLACEMENTS_64BIT
 - Collective operations data-structures, 49
- UCC_COLL_ARGS_FLAG_IN_PLACE
 - Collective operations data-structures, 49
- UCC_COLL_ARGS_FLAG_MEM_MAPPED_BUFFER
 - Collective operations data-structures, 49
- UCC_COLL_ARGS_FLAG_PERSISTENT
 - Collective operations data-structures, 49
- UCC_COLL_ARGS_FLAG_TIMEOUT
 - Collective operations data-structures, 49
- ucc_coll_args_flags_t
 - Collective operations data-structures, 49
- ucc_coll_args_t
 - Collective Operations, 52
- ucc_coll_buffer_info, 48
- ucc_coll_buffer_info_t
 - Collective operations data-structures, 48
- ucc_coll_buffer_info_v, 47
- ucc_coll_buffer_info_v_t
 - Collective operations data-structures, 48
- ucc_coll_callback, 62
 - cb, 62
 - data, 62
- ucc_coll_callback_t
 - Collective operations data-structures, 48
- ucc_coll_id_t
 - Collective operations data-structures, 49
- ucc_coll_req_h
 - Collective operations data-structures, 48
- ucc_coll_sync_type_t
 - Library initialization data-structures, 18
- UCC_COLL_TYPE_ALLGATHER
 - Library initialization data-structures, 17
- UCC_COLL_TYPE_ALLGATHERV
 - Library initialization data-structures, 17
- UCC_COLL_TYPE_ALLREDUCE
 - Library initialization data-structures, 17
- UCC_COLL_TYPE_ALLTOALL
 - Library initialization data-structures, 17
- UCC_COLL_TYPE_ALLTOALLV
 - Library initialization data-structures, 17
- UCC_COLL_TYPE_BARRIER
 - Library initialization data-structures, 17
- UCC_COLL_TYPE_BCAST
 - Library initialization data-structures, 17
- UCC_COLL_TYPE_FANIN
 - Library initialization data-structures, 17
- UCC_COLL_TYPE_FANOUT
 - Library initialization data-structures, 17
- UCC_COLL_TYPE_GATHER
 - Library initialization data-structures, 17
- UCC_COLL_TYPE_GATHERV
 - Library initialization data-structures, 17
- UCC_COLL_TYPE_LAST
 - Library initialization data-structures, 17
- UCC_COLL_TYPE_REDUCE
 - Library initialization data-structures, 17
- UCC_COLL_TYPE_REDUCE_SCATTER
 - Library initialization data-structures, 17
- UCC_COLL_TYPE_REDUCE_SCATTERV
 - Library initialization data-structures, 17
- UCC_COLL_TYPE_SCATTER
 - Library initialization data-structures, 17
- UCC_COLL_TYPE_SCATTERV
 - Library initialization data-structures, 17
- ucc_coll_type_t
 - Library initialization data-structures, 17
- UCC_COLLECTIVE_EP_RANGE_CONTIG
 - Team abstraction data-structures, 42
- UCC_COLLECTIVE_EP_RANGE_NONCONTIG
 - Team abstraction data-structures, 42
- ucc_collective_finalize
 - Collective Operations, 54
- ucc_collective_init
 - Collective Operations, 52
- ucc_collective_init_and_post
 - Collective Operations, 53
- UCC_COLLECTIVE_INIT_AND_POST_ORDERED
 - Team abstraction data-structures, 42
- UCC_COLLECTIVE_INIT_AND_POST_UNORDERED
 - Team abstraction data-structures, 42
- UCC_COLLECTIVE_INIT_ORDERED
 - Team abstraction data-structures, 42
- UCC_COLLECTIVE_INIT_UNORDERED
 - Team abstraction data-structures, 42
- ucc_collective_post
 - Collective Operations, 53
- UCC_COLLECTIVE_POST_ORDERED
 - Team abstraction data-structures, 42
- UCC_COLLECTIVE_POST_UNORDERED
 - Team abstraction data-structures, 42

- ucc_collective_test
 - Collective Operations, [53](#)
- ucc_collective_triggered_post
 - Events and Triggered Operations, [58](#)
- UCC_CONFIG_PRINT_CONFIG
 - Utility Operations, [61](#)
- UCC_CONFIG_PRINT_DOC
 - Utility Operations, [61](#)
- ucc_config_print_flags_t
 - Utility Operations, [59](#)
- UCC_CONFIG_PRINT_HEADER
 - Utility Operations, [61](#)
- UCC_CONFIG_PRINT_HIDDEN
 - Utility Operations, [61](#)
- ucc_context_addr_h
 - Team abstraction data-structures, [41](#)
- ucc_context_addr_len_t
 - Team abstraction data-structures, [41](#)
- ucc_context_attr, [30](#)
- ucc_context_attr_field
 - Context abstraction data-structures, [32](#)
- UCC_CONTEXT_ATTR_FIELD_CTX_ADDR
 - Context abstraction data-structures, [32](#)
- UCC_CONTEXT_ATTR_FIELD_CTX_ADDR_LEN
 - Context abstraction data-structures, [32](#)
- UCC_CONTEXT_ATTR_FIELD_SYNC_TYPE
 - Context abstraction data-structures, [32](#)
- UCC_CONTEXT_ATTR_FIELD_TYPE
 - Context abstraction data-structures, [32](#)
- UCC_CONTEXT_ATTR_FIELD_WORK_BUFFER_SIZE
 - Context abstraction data-structures, [32](#)
- ucc_context_attr_t
 - Context abstraction data-structures, [31](#)
- ucc_context_config_h
 - Context abstraction data-structures, [31](#)
- ucc_context_config_modify
 - Context abstraction routines, [34](#)
- ucc_context_config_print
 - Context abstraction routines, [33](#)
- ucc_context_config_read
 - Context abstraction routines, [33](#)
- ucc_context_config_release
 - Context abstraction routines, [33](#)
- ucc_context_create
 - Context abstraction routines, [34](#)
- ucc_context_destroy
 - Context abstraction routines, [35](#)
- UCC_CONTEXT_EXCLUSIVE
 - Context abstraction data-structures, [31](#)
- ucc_context_get_attr
 - Context abstraction routines, [35](#)
- ucc_context_h
 - Context abstraction data-structures, [31](#)
- UCC_CONTEXT_PARAM_FIELD_ID
 - Context abstraction data-structures, [32](#)
- UCC_CONTEXT_PARAM_FIELD_MEM_PARAMS
 - Context abstraction data-structures, [32](#)
- UCC_CONTEXT_PARAM_FIELD_OOB
 - Context abstraction data-structures, [32](#)
- UCC_CONTEXT_PARAM_FIELD_SYNC_TYPE
 - Context abstraction data-structures, [32](#)
- UCC_CONTEXT_PARAM_FIELD_TYPE
 - Context abstraction data-structures, [32](#)
- ucc_context_params, [30](#)
- ucc_context_params_field
 - Context abstraction data-structures, [32](#)
- ucc_context_params_t
 - Context abstraction data-structures, [31](#)
- ucc_context_progress
 - Context abstraction routines, [35](#)
- UCC_CONTEXT_SHARED
 - Context abstraction data-structures, [31](#)
- ucc_context_type_t
 - Context abstraction data-structures, [31](#)
- ucc_count_t
 - Collective operations data-structures, [48](#)
- ucc_datatype_t
 - Datatypes data-structures and functions, [21](#)
- ucc_dt_create_generic
 - Datatypes data-structures and functions, [22](#)
- ucc_dt_destroy
 - Datatypes data-structures and functions, [22](#)
- ucc_ee_ack_event
 - Events and Triggered Operations, [57](#)
- UCC_EE_CPU_THREAD
 - Events and Triggered operations' datastructures, [56](#)
- ucc_ee_create
 - Events and Triggered Operations, [56](#)
- UCC_EE_CUDA_STREAM
 - Events and Triggered operations' datastructures, [56](#)
- ucc_ee_destroy
 - Events and Triggered Operations, [57](#)
- ucc_ee_get_event
 - Events and Triggered Operations, [57](#)
- UCC_EE_LAST
 - Events and Triggered operations' datastructures, [56](#)
- ucc_ee_params, [55](#)
- ucc_ee_params_t
 - Events and Triggered operations' datastructures, [55](#)
- ucc_ee_set_event
 - Events and Triggered Operations, [58](#)
- ucc_ee_type
 - Events and Triggered operations' datastructures, [56](#)
- ucc_ee_type_t
 - Events and Triggered operations' datastructures, [55](#)
- UCC_EE_UNKNOWN
 - Events and Triggered operations' datastructures, [56](#)
- ucc_ee_wait
 - Events and Triggered Operations, [58](#)

- UCC_EP_MAP_ARRAY
 - Team abstraction data-structures, [43](#)
- ucc_ep_map_array, [37](#)
- UCC_EP_MAP_CB
 - Team abstraction data-structures, [43](#)
- ucc_ep_map_cb, [62](#)
 - cb, [62](#)
 - cb_ctx, [62](#)
- UCC_EP_MAP_FULL
 - Team abstraction data-structures, [43](#)
- UCC_EP_MAP_STRIDED
 - Team abstraction data-structures, [43](#)
- ucc_ep_map_strided, [37](#)
- ucc_ep_map_t, [37](#)
 - Team abstraction data-structures, [40](#)
- ucc_ep_map_t.__unnamed2__, [40](#)
- ucc_ep_map_type_t
 - Team abstraction data-structures, [42](#)
- ucc_ep_range_type_t
 - Team abstraction data-structures, [42](#)
- UCC_ERR_INVALID_PARAM
 - Utility Operations, [61](#)
- UCC_ERR_LAST
 - Utility Operations, [61](#)
- UCC_ERR_NO_MEMORY
 - Utility Operations, [61](#)
- UCC_ERR_NO_MESSAGE
 - Utility Operations, [61](#)
- UCC_ERR_NO_RESOURCE
 - Utility Operations, [61](#)
- UCC_ERR_NOT_FOUND
 - Utility Operations, [61](#)
- UCC_ERR_NOT_IMPLEMENTED
 - Utility Operations, [61](#)
- UCC_ERR_NOT_SUPPORTED
 - Utility Operations, [61](#)
- UCC_ERR_TIMED_OUT
 - Utility Operations, [61](#)
- UCC_ERR_TYPE_GLOBAL
 - Collective operations data-structures, [50](#)
- UCC_ERR_TYPE_LOCAL
 - Collective operations data-structures, [50](#)
- ucc_error_type_t
 - Collective operations data-structures, [50](#)
- ucc_ev_t
 - Events and Triggered operations' datastructures, [55](#)
- ucc_event, [55](#)
- UCC_EVENT_COLLECTIVE_COMPLETE
 - Events and Triggered operations' datastructures, [55](#)
- UCC_EVENT_COLLECTIVE_POST
 - Events and Triggered operations' datastructures, [55](#)
- UCC_EVENT_COMPUTE_COMPLETE
 - Events and Triggered operations' datastructures, [55](#)
- UCC_EVENT_OVERFLOW
 - Events and Triggered operations' datastructures, [55](#)
- ucc_event_type
 - Events and Triggered operations' datastructures, [55](#)
- ucc_event_type_t
 - Events and Triggered operations' datastructures, [55](#)
- ucc_finalize
 - Library initialization and finalization routines, [27](#)
- ucc_generic_dt_ops, [63](#)
 - contig_size, [63](#)
 - flags, [63](#)
 - mask, [63](#)
- ucc_generic_dt_ops.reduce, [64](#)
 - cb, [64](#)
 - cb_ctx, [64](#)
- ucc_generic_dt_ops_field
 - Datatypes data-structures and functions, [21](#)
- UCC_GENERIC_DT_OPS_FIELD_FLAGS
 - Datatypes data-structures and functions, [21](#)
- UCC_GENERIC_DT_OPS_FLAG_CONTIG
 - Datatypes data-structures and functions, [22](#)
- UCC_GENERIC_DT_OPS_FLAG_REDUCE
 - Datatypes data-structures and functions, [22](#)
- ucc_generic_dt_ops_flags_t
 - Datatypes data-structures and functions, [21](#)
- ucc_generic_dt_ops_t
 - Datatypes data-structures and functions, [21](#)
- ucc_init
 - Library initialization and finalization routines, [26](#)
- UCC_INPROGRESS
 - Utility Operations, [61](#)
- ucc_lib_attr, [16](#)
- ucc_lib_attr_field
 - Library initialization data-structures, [19](#)
- UCC_LIB_ATTR_FIELD_COLL_TYPES
 - Library initialization data-structures, [19](#)
- UCC_LIB_ATTR_FIELD_REDUCTION_TYPES
 - Library initialization data-structures, [19](#)
- UCC_LIB_ATTR_FIELD_SYNC_TYPE
 - Library initialization data-structures, [19](#)
- UCC_LIB_ATTR_FIELD_THREAD_MODE
 - Library initialization data-structures, [19](#)
- ucc_lib_attr_t
 - Library initialization data-structures, [16](#)
- ucc_lib_config_h
 - Library initialization data-structures, [17](#)
- ucc_lib_config_modify
 - Library initialization and finalization routines, [26](#)
- ucc_lib_config_print
 - Library initialization and finalization routines, [26](#)
- ucc_lib_config_read
 - Library initialization and finalization routines, [25](#)
- ucc_lib_config_release
 - Library initialization and finalization routines, [25](#)
- ucc_lib_get_attr
 - Library initialization and finalization routines, [27](#)

- ucc_lib_h
 - Library initialization data-structures, 17
- UCC_LIB_PARAM_FIELD_COLL_TYPES
 - Library initialization data-structures, 19
- UCC_LIB_PARAM_FIELD_REDUCTION_TYPES
 - Library initialization data-structures, 19
- UCC_LIB_PARAM_FIELD_SYNC_TYPE
 - Library initialization data-structures, 19
- UCC_LIB_PARAM_FIELD_THREAD_MODE
 - Library initialization data-structures, 19
- ucc_lib_params, 15
- ucc_lib_params_field
 - Library initialization data-structures, 19
- ucc_lib_params_t
 - Library initialization data-structures, 16
- ucc_mem_h
 - Collective Operations, 52
- ucc_mem_map, 29
- ucc_mem_map_params, 30
- ucc_mem_map_params_t
 - Context abstraction data-structures, 31
- ucc_mem_map_t
 - Context abstraction data-structures, 31
- ucc_memory_type
 - Collective operations data-structures, 49
- UCC_MEMORY_TYPE_CUDA
 - Collective operations data-structures, 49
- UCC_MEMORY_TYPE_CUDA_MANAGED
 - Collective operations data-structures, 49
- UCC_MEMORY_TYPE_HOST
 - Collective operations data-structures, 49
- UCC_MEMORY_TYPE_LAST
 - Collective operations data-structures, 49
- UCC_MEMORY_TYPE_ROCM
 - Collective operations data-structures, 49
- UCC_MEMORY_TYPE_ROCM_MANAGED
 - Collective operations data-structures, 49
- ucc_memory_type_t
 - Collective operations data-structures, 48
- UCC_MEMORY_TYPE_UNKNOWN
 - Collective operations data-structures, 49
- UCC_NO_SYNC_COLLECTIVES
 - Library initialization data-structures, 19
- UCC_OK
 - Utility Operations, 61
- ucc_oob_coll, 64
 - allgather, 64
 - coll_info, 65
 - n_oob_eps, 65
 - oob_ep, 65
 - req_free, 65
 - req_test, 65
- ucc_oob_coll_t
 - Context abstraction data-structures, 30
- UCC_OP_AVG
 - Library initialization data-structures, 18
- UCC_OP_BAND
 - Library initialization data-structures, 18
- UCC_OP_BOR
 - Library initialization data-structures, 18
- UCC_OP_BXOR
 - Library initialization data-structures, 18
- UCC_OP_LAND
 - Library initialization data-structures, 18
- UCC_OP_LAST
 - Library initialization data-structures, 18
- UCC_OP_LOR
 - Library initialization data-structures, 18
- UCC_OP_LXOR
 - Library initialization data-structures, 18
- UCC_OP_MAX
 - Library initialization data-structures, 18
- UCC_OP_MAXLOC
 - Library initialization data-structures, 18
- UCC_OP_MIN
 - Library initialization data-structures, 18
- UCC_OP_MINLOC
 - Library initialization data-structures, 18
- UCC_OP_PROD
 - Library initialization data-structures, 18
- UCC_OP_SUM
 - Library initialization data-structures, 18
- UCC_OPERATION_INITIALIZED
 - Utility Operations, 61
- ucc_p2p_conn_t
 - Team abstraction data-structures, 41
- ucc_post_ordering_t
 - Team abstraction data-structures, 42
- ucc_reduce_cb_params, 20
- ucc_reduce_cb_params_t
 - Datatypes data-structures and functions, 21
- ucc_reduction_op_t
 - Library initialization data-structures, 17
- ucc_status_string
 - Utility Operations, 61
- ucc_status_t
 - Utility Operations, 61
- UCC_SYNC_COLLECTIVES
 - Library initialization data-structures, 19
- ucc_team_attr, 40
- ucc_team_attr_field
 - Team abstraction data-structures, 41
- UCC_TEAM_ATTR_FIELD_EP
 - Team abstraction data-structures, 42
- UCC_TEAM_ATTR_FIELD_EP_RANGE
 - Team abstraction data-structures, 42
- UCC_TEAM_ATTR_FIELD_MEM_PARAMS
 - Team abstraction data-structures, 42
- UCC_TEAM_ATTR_FIELD_OUTSTANDING_CALLS
 - Team abstraction data-structures, 42
- UCC_TEAM_ATTR_FIELD_POST_ORDERING
 - Team abstraction data-structures, 42
- UCC_TEAM_ATTR_FIELD_SYNC_TYPE
 - Team abstraction data-structures, 42
- ucc_team_attr_t
 - Team abstraction data-structures, 40

- ucc_team_create_from_parent
 - Team abstraction routines, [45](#)
- ucc_team_create_post
 - Team abstraction routines, [43](#)
- ucc_team_create_test
 - Team abstraction routines, [44](#)
- ucc_team_destroy
 - Team abstraction routines, [44](#)
- UCC_TEAM_FLAG_COLL_WORK_BUFFER
 - Team abstraction data-structures, [42](#)
- ucc_team_flags
 - Team abstraction data-structures, [42](#)
- ucc_team_get_all_eps
 - Team abstraction routines, [46](#)
- ucc_team_get_attr
 - Team abstraction routines, [44](#)
- ucc_team_get_my_ep
 - Team abstraction routines, [46](#)
- ucc_team_get_size
 - Team abstraction routines, [45](#)
- ucc_team_h
 - Team abstraction data-structures, [41](#)
- ucc_team_p2p_conn, [65](#)
 - conn_ctx, [65](#)
 - conn_info_lookup, [65](#)
 - conn_info_release, [65](#)
 - req_free, [66](#)
 - req_test, [66](#)
- ucc_team_p2p_conn_t
 - Team abstraction data-structures, [40](#)
- UCC_TEAM_PARAM_FIELD_EP
 - Team abstraction data-structures, [41](#)
- UCC_TEAM_PARAM_FIELD_EP_LIST
 - Team abstraction data-structures, [41](#)
- UCC_TEAM_PARAM_FIELD_EP_MAP
 - Team abstraction data-structures, [41](#)
- UCC_TEAM_PARAM_FIELD_EP_RANGE
 - Team abstraction data-structures, [41](#)
- UCC_TEAM_PARAM_FIELD_FLAGS
 - Team abstraction data-structures, [41](#)
- UCC_TEAM_PARAM_FIELD_ID
 - Team abstraction data-structures, [41](#)
- UCC_TEAM_PARAM_FIELD_MEM_PARAMS
 - Team abstraction data-structures, [41](#)
- UCC_TEAM_PARAM_FIELD_OOB
 - Team abstraction data-structures, [41](#)
- UCC_TEAM_PARAM_FIELD_ORDERING
 - Team abstraction data-structures, [41](#)
- UCC_TEAM_PARAM_FIELD_OUTSTANDING_COLLs
 - Team abstraction data-structures, [41](#)
- UCC_TEAM_PARAM_FIELD_P2P_CONN
 - Team abstraction data-structures, [41](#)
- UCC_TEAM_PARAM_FIELD_SYNC_TYPE
 - Team abstraction data-structures, [41](#)
- UCC_TEAM_PARAM_FIELD_TEAM_SIZE
 - Team abstraction data-structures, [41](#)
- ucc_team_params, [37](#)
- ucc_team_params_field
 - Team abstraction data-structures, [41](#)
- ucc_team_params_t
 - Team abstraction data-structures, [40](#)
- UCC_THREAD_FUNNELED
 - Library initialization data-structures, [18](#)
- ucc_thread_mode_t
 - Library initialization data-structures, [18](#)
- UCC_THREAD_MULTIPLE
 - Library initialization data-structures, [18](#)
- UCC_THREAD_SINGLE
 - Library initialization data-structures, [18](#)
- unpack
 - Datatypes data-structures and functions, [24](#)
- Utility Operations, [59](#)
 - UCC_CONFIG_PRINT_CONFIG, [61](#)
 - UCC_CONFIG_PRINT_DOC, [61](#)
 - ucc_config_print_flags_t, [59](#)
 - UCC_CONFIG_PRINT_HEADER, [61](#)
 - UCC_CONFIG_PRINT_HIDDEN, [61](#)
 - UCC_ERR_INVALID_PARAM, [61](#)
 - UCC_ERR_LAST, [61](#)
 - UCC_ERR_NO_MEMORY, [61](#)
 - UCC_ERR_NO_MESSAGE, [61](#)
 - UCC_ERR_NO_RESOURCE, [61](#)
 - UCC_ERR_NOT_FOUND, [61](#)
 - UCC_ERR_NOT_IMPLEMENTED, [61](#)
 - UCC_ERR_NOT_SUPPORTED, [61](#)
 - UCC_ERR_TIMED_OUT, [61](#)
 - UCC_INPROGRESS, [61](#)
 - UCC_OK, [61](#)
 - UCC_OPERATION_INITIALIZED, [61](#)
 - ucc_status_string, [61](#)
 - ucc_status_t, [61](#)