

Unified Collective Communications (UCC) Specification

Version 1.0



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	Starting and Completing the Collectives	7
7	Module Documentation	8
7.1	Library initialization data-structures	8
7.1.1	Detailed Description	9
7.1.2	Data Structure Documentation	10
7.1.2.1	struct ucc_lib_params	10
7.1.2.2	struct ucc_lib_attr	10
7.1.3	Typedef Documentation	10
7.1.3.1	ucc_lib_params_t	10
7.1.3.2	ucc_lib_attr_t	11
7.1.3.3	ucc_lib_h	11
7.1.3.4	ucc_lib_config_h	11
7.1.4	Enumeration Type Documentation	11
7.1.4.1	ucc_reduction_op_t	11
7.1.4.2	ucc_coll_type_t	12
7.1.4.3	ucc_datatype_t	12
7.1.4.4	ucc_thread_mode_t	13
7.1.4.5	ucc_coll_sync_type_t	13
7.1.4.6	ucc_lib_params_field	13
7.1.4.7	ucc_lib_attr_field	14
7.2	Library initialization and finalization routines	15
7.2.1	Detailed Description	15

7.2.2	Function Documentation	15
7.2.2.1	ucc_lib_config_read()	15
7.2.2.2	ucc_lib_config_release()	16
7.2.2.3	ucc_lib_config_print()	16
7.2.2.4	ucc_lib_config_modify()	16
7.2.2.5	ucc_init()	16
7.2.2.6	ucc_finalize()	17
7.2.2.7	ucc_lib_get_attr()	17
7.3	Context abstraction data-structures	19
7.3.1	Detailed Description	19
7.3.2	Data Structure Documentation	19
7.3.2.1	struct ucc_context_params	19
7.3.2.2	struct ucc_context_attr	20
7.3.3	Typedef Documentation	20
7.3.3.1	ucc_context_oob_coll_t	20
7.3.3.2	ucc_context_params_t	20
7.3.3.3	ucc_context_attr_t	20
7.3.3.4	ucc_context_h	20
7.3.3.5	ucc_context_config_h	21
7.3.4	Enumeration Type Documentation	21
7.3.4.1	ucc_context_type_t	21
7.3.4.2	ucc_context_params_field	21
7.3.4.3	ucc_context_attr_field	21
7.4	Context abstraction routines	22
7.4.1	Detailed Description	22
7.4.2	Function Documentation	22
7.4.2.1	ucc_context_config_read()	22
7.4.2.2	ucc_context_config_release()	22
7.4.2.3	ucc_context_create()	23
7.4.2.4	ucc_context_progress()	23
7.4.2.5	ucc_context_destroy()	23
7.4.2.6	ucc_context_get_attr()	24
7.5	Team abstraction data-structures	25
7.5.1	Detailed Description	26
7.5.2	Data Structure Documentation	26
7.5.2.1	struct ucc_mem_map_params	26
7.5.2.2	struct ucc_ep_map_strided	26
7.5.2.3	struct ucc_ep_map_array	26
7.5.2.4	struct ucc_ep_map_t	26
7.5.2.5	union ucc_ep_map_t.__unnamed__	27

7.5.2.6	struct ucc_team_params	27
7.5.2.7	struct ucc_team_attr	27
7.5.3	Typedef Documentation	28
7.5.3.1	ucc_mem_map_params_t	28
7.5.3.2	ucc_team_p2p_conn	28
7.5.3.3	ucc_team_oob_coll_t	28
7.5.3.4	ucc_ep_map_t	28
7.5.3.5	ucc_team_params_t	28
7.5.3.6	ucc_team_attr_t	28
7.5.3.7	ucc_team_h	28
7.5.3.8	ucc_p2p_conn_t	28
7.5.3.9	ucc_context_addr_t	29
7.5.3.10	ucc_context_addr_len_t	29
7.5.4	Enumeration Type Documentation	29
7.5.4.1	ucc_team_params_field	29
7.5.4.2	ucc_team_attr_field	29
7.5.4.3	ucc_mem_constraints_t	29
7.5.4.4	ucc_mem_hints_t	30
7.5.4.5	ucc_post_ordering_t	30
7.5.4.6	ucc_ep_range_type_t	30
7.5.4.7	ucc_ep_map_type_t	30
7.6	Team abstraction routines	31
7.6.1	Detailed Description	31
7.6.2	Function Documentation	31
7.6.2.1	ucc_team_create_post()	31
7.6.2.2	ucc_team_create_test()	32
7.6.2.3	ucc_team_destroy()	32
7.6.2.4	ucc_team_get_attr()	32
7.6.2.5	ucc_team_create_from_parent()	33
7.6.2.6	ucc_team_get_size()	33
7.6.2.7	ucc_team_get_my_ep()	33
7.6.2.8	ucc_team_get_all_eps()	34
7.7	Collective operations data-structures	35
7.7.1	Detailed Description	35
7.7.2	Data Structure Documentation	35
7.7.2.1	struct ucc_coll_buffer_info	35
7.7.3	Typedef Documentation	36
7.7.3.1	ucc_coll_buffer_info_t	36
7.7.3.2	ucc_coll_req_h	36
7.7.3.3	ucc_count_t	36

7.7.3.4	<code>ucc_aint_t</code>	36
7.7.3.5	<code>ucc_coll_id_t</code>	36
7.7.4	Enumeration Type Documentation	36
7.7.4.1	<code>ucc_coll_buffer_flags_t</code>	36
7.7.4.2	<code>ucc_error_type_t</code>	36
7.7.4.3	<code>ucc_coll_op_args_field</code>	37
7.8	Collective Operations	38
7.8.1	Detailed Description	38
7.8.2	Data Structure Documentation	38
7.8.2.1	<code>struct ucc_coll_op_args</code>	38
7.8.3	Typedef Documentation	40
7.8.3.1	<code>ucc_reduction_dtype_mapper_t</code>	40
7.8.3.2	<code>ucc_userdefined_reduction_op_t</code>	40
7.8.3.3	<code>ucc_coll_op_args_t</code>	40
7.8.3.4	<code>ucc_mem_h</code>	41
7.8.4	Function Documentation	41
7.8.4.1	<code>ucc_collective_init()</code>	41
7.8.4.2	<code>ucc_collective_post()</code>	41
7.8.4.3	<code>ucc_collective_init_and_post()</code>	42
7.8.4.4	<code>ucc_collective_test()</code>	42
7.8.4.5	<code>ucc_collective_finalize()</code>	42
7.9	Utility Operations	44
7.9.1	Detailed Description	44
7.9.2	Enumeration Type Documentation	44
7.9.2.1	<code>ucc_config_print_flags_t</code>	44
7.9.2.2	<code>ucc_status_t</code>	44
7.9.3	Function Documentation	45
7.9.3.1	<code>ucc_status_string()</code>	45
8	Data Structure Documentation	46
8.1	<code>ucc_context_oob_coll</code> Struct Reference	46
8.1.1	Field Documentation	46
8.1.1.1	<code>allgather</code>	46
8.1.1.2	<code>req_test</code>	46
8.1.1.3	<code>req_free</code>	46
8.1.1.4	<code>participants</code>	46
8.1.1.5	<code>coll_info</code>	46
8.2	<code>ucc_ep_map_cb</code> Struct Reference	47
8.2.1	Field Documentation	47
8.2.1.1	<code>cb</code>	47

8.2.1.2	cb_ctx	47
8.3	ucc_team_oob_coll Struct Reference	47
8.3.1	Field Documentation	47
8.3.1.1	allgather	47
8.3.1.2	req_test	47
8.3.1.3	req_free	47
8.3.1.4	participants	47
8.3.1.5	coll_info	48
8.4	ucc_team_p2p_conn Struct Reference	48
8.4.1	Field Documentation	48
8.4.1.1	conn_info_lookup	48
8.4.1.2	conn_info_release	48
8.4.1.3	conn_ctx	48
8.4.1.4	req_test	48
8.4.1.5	req_free	48
	Index	49

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

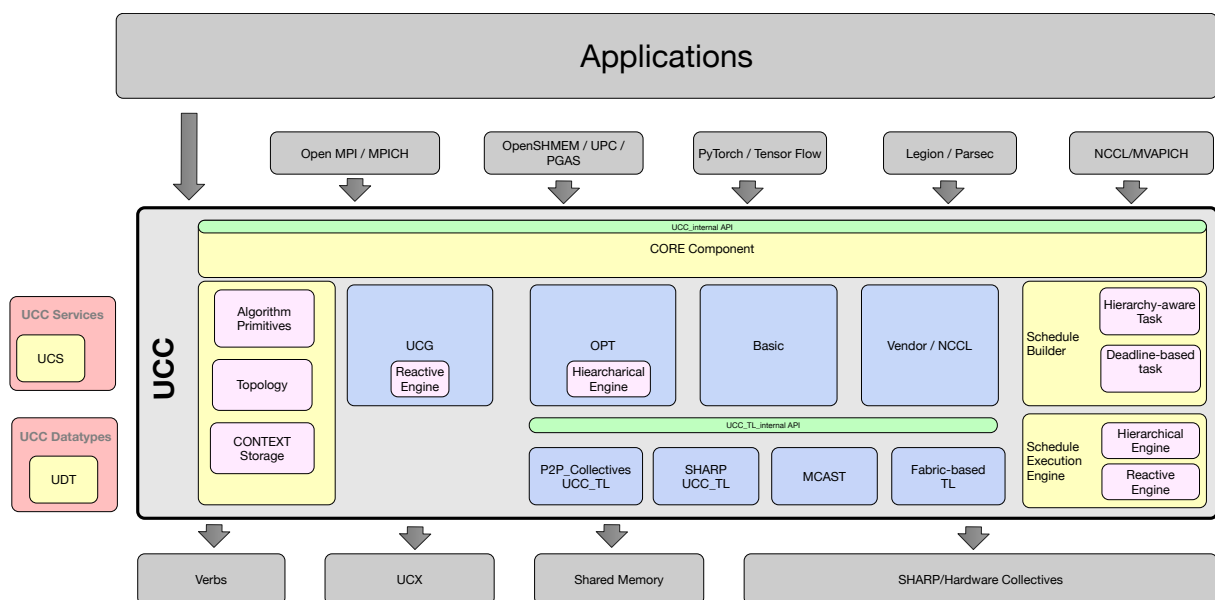


Figure 2.1: UCC Components and Usage

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

Starting and Completing the Collectives

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 routine returns once the participants enter the collective 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`. It supports three types of collective operations: (a) `UCC_{ALLTOALL, ALLGATHER, ALLREDUCE}` operations where all participants contribute to the results and receive the results (b) `UCC_{REDUCE, GATHER, 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, MULTICAST, SCATTER, FANOUT}` where one participant contributes to the result, and all participants receive the result. The participant contributing to the result is designated as root.

Reduction types: The reduction operation supported by `UCC_{ALLREDUCE, REDUCE}` operation is defined by the enumeration `ucc_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 and on return from the routine, the ownership is transferred back to the user. 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.

Chapter 7

Module Documentation

7.1 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_t` * `ucc_lib_h`
UCC library handle.
- typedef struct `ucc_lib_config` * `ucc_lib_config_h`
UCC library configuration handle.

Enumerations

- enum `ucc_reduction_op_t` {
 `UCC_OP_USERDEFINED` = `UCC_BIT(0)`,
 `UCC_OP_SUM` = `UCC_BIT(1)`,
 `UCC_OP_PROD` = `UCC_BIT(2)`,
 `UCC_OP_MAX` = `UCC_BIT(3)`,
 `UCC_OP_MIN` = `UCC_BIT(4)`,
 `UCC_OP_AND` = `UCC_BIT(5)`,
 `UCC_OP_OR` = `UCC_BIT(6)`,
 `UCC_OP_XOR` = `UCC_BIT(7)`,
 `UCC_OP_LAND` = `UCC_BIT(8)`,
 `UCC_OP_LOR` = `UCC_BIT(9)`,
 `UCC_OP_LXOR` = `UCC_BIT(10)`,
 `UCC_OP_BAND` = `UCC_BIT(11)`,
 `UCC_OP_BOR` = `UCC_BIT(12)`,
 `UCC_OP_BXOR` = `UCC_BIT(13)`,
 `UCC_OP_MAXLOC` = `UCC_BIT(14)`,
 `UCC_OP_MINLOC` = `UCC_BIT(15)` }

Enumeration representing the UCC reduction operations.

- enum `ucc_coll_type_t` {
`UCC_COLL_TYPE_BARRIER` = `UCC_BIT(0)`,
`UCC_COLL_TYPE_BCAST` = `UCC_BIT(1)`,
`UCC_COLL_TYPE_ALLREDUCE` = `UCC_BIT(2)`,
`UCC_COLL_TYPE_REDUCE` = `UCC_BIT(3)`,
`UCC_COLL_TYPE_ALLTOALL` = `UCC_BIT(4)`,
`UCC_COLL_TYPE_ALLGATHER` = `UCC_BIT(5)`,
`UCC_COLL_TYPE_GATHER` = `UCC_BIT(6)`,
`UCC_COLL_TYPE_SCATTER` = `UCC_BIT(7)`,
`UCC_COLL_TYPE_FANIN` = `UCC_BIT(8)`,
`UCC_COLL_TYPE_FANOUT` = `UCC_BIT(9)` }

Enumeration representing the collective operations.

- enum `ucc_datatype_t` {
`UCC_DT_INT8` = 0,
`UCC_DT_INT16`,
`UCC_DT_INT32`,
`UCC_DT_INT64`,
`UCC_DT_INT128`,
`UCC_DT_UINT8`,
`UCC_DT_UINT16`,
`UCC_DT_UINT32`,
`UCC_DT_UINT64`,
`UCC_DT_UINT128`,
`UCC_DT_FLOAT16`,
`UCC_DT_FLOAT32`,
`UCC_DT_FLOAT64`,
`UCC_DT_USERDEFINED`,
`UCC_DT_OPAQUE` }

Enumeration representing the UCC library's datatype.

- 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_LIB_PARAM_FIELD_REDUCTION_WRAPPER` = `UCC_BIT(4)` }

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)` }

7.1.1 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

7.1.2 Data Structure Documentation

7.1.2.1 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	
ucc_reduction_dtype_mapper_t	reduction_mapper	

7.1.2.2 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	

7.1.3 Typedef Documentation

7.1.3.1 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.

7.1.3.2 `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.

7.1.3.3 `ucc_lib_h`

```
typedef struct ucc_lib_info_t* 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.

7.1.3.4 `ucc_lib_config_h`

```
typedef struct ucc_lib_config* ucc_lib_config_h
```

7.1.4 Enumeration Type Documentation

7.1.4.1 `ucc_reduction_op_t`

```
enum ucc_reduction_op_t
```

Library initialization and finalize

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. The user-defined reductions are represented by `UCC_OP_USERDEFINED`.

Enumerator

<code>UCC_OP_USERDEFINED</code>	User defined reduction operation
<code>UCC_OP_SUM</code>	Predefined addition operation
<code>UCC_OP_PROD</code>	
<code>UCC_OP_MAX</code>	
<code>UCC_OP_MIN</code>	
<code>UCC_OP_AND</code>	
<code>UCC_OP_OR</code>	
<code>UCC_OP_XOR</code>	
<code>UCC_OP_LAND</code>	

Enumerator

UCC_OP_LOR	
UCC_OP_LXOR	
UCC_OP_BAND	
UCC_OP_BOR	
UCC_OP_BXOR	
UCC_OP_MAXLOC	
UCC_OP_MINLOC	

7.1.4.2 ucc_coll_type_t

enum [ucc_coll_type_t](#)

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_BARRIER	
UCC_COLL_TYPE_BCAST	
UCC_COLL_TYPE_ALLREDUCE	
UCC_COLL_TYPE_REDUCE	
UCC_COLL_TYPE_ALLTOALL	
UCC_COLL_TYPE_ALLGATHER	
UCC_COLL_TYPE_GATHER	
UCC_COLL_TYPE_SCATTER	
UCC_COLL_TYPE_FANIN	
UCC_COLL_TYPE_FANOUT	

7.1.4.3 ucc_datatype_t

enum [ucc_datatype_t](#)

Description

[ucc_datatype_t](#) represents the datatypes supported by the UCC library's collective and reduction operations. The standard operations are signed and unsigned integers of various sizes, float 16, 32, and 64, and user-defined datatypes. The UCC_DT_USERDEFINED represents the user-defined datatype. The UCC_DT_OPAQUE is used to represent the user-defined datatypes for user-defined reductions. When UCC_DT_OPAQUE is used, the library passes the data to the user-defined reductions without any modifications.

Enumerator

UCC_DT_INT8	
UCC_DT_INT16	
UCC_DT_INT32	
UCC_DT_INT64	
UCC_DT_INT128	
UCC_DT_UINT8	
UCC_DT_UINT16	

Enumerator

UCC_DT_UINT32	
UCC_DT_UINT64	
UCC_DT_UINT128	
UCC_DT_FLOAT16	
UCC_DT_FLOAT32	
UCC_DT_FLOAT64	
UCC_DT_USERDEFINED	
UCC_DT_OPAQUE	

7.1.4.4 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

7.1.4.5 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

7.1.4.6 ucc_lib_params_field

enum [ucc_lib_params_field](#)

Enumerator

UCC_LIB_PARAM_FIELD_THREAD_MODE	
---------------------------------	--

Enumerator

UCC_LIB_PARAM_FIELD_COLL_TYPES	
UCC_LIB_PARAM_FIELD_REDUCTION_TYPES	
UCC_LIB_PARAM_FIELD_SYNC_TYPE	
UCC_LIB_PARAM_FIELD_REDUCTION_WRAPPER	

7.1.4.7 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	

7.2 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.
- `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.

7.2.1 Detailed Description

Library initialization and finalization routines

7.2.2 Function Documentation

7.2.2.1 `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`

7.2.2.2 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.

7.2.2.3 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.

7.2.2.4 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

7.2.2.5 ucc_init()

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

```
const ucc_lib_config_h * config,
ucc_lib_h * lib_p )
```

Parameters

in	<i>params</i>	user provided parameters to customize the library functionality
in	<i>config</i>	UCC configuration descriptor allocated through <code>ucc_config_read()</code> 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`

7.2.2.6 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`

7.2.2.7 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`

7.3 Context abstraction data-structures

Data Structures

- struct `ucc_context_oob_coll`
OOB collective operation for creating the context.
- 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_context_oob_coll ucc_context_oob_coll_t`
OOB collective operation for creating the context.
- 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_COLL_SYNC_TYPE` = `UCC_BIT(1)`,
 `UCC_CONTEXT_PARAM_FIELD_COLL_OOB` = `UCC_BIT(2)`,
 `UCC_CONTEXT_PARAM_FIELD_ID` = `UCC_BIT(3)` }
- enum `ucc_context_attr_field` {
 `UCC_CONTEXT_ATTR_FIELD_TYPE` = `UCC_BIT(0)`,
 `UCC_CONTEXT_ATTR_FIELD_COLL_SYNC_TYPE` = `UCC_BIT(1)`,
 `UCC_CONTEXT_ATTR_FIELD_CONTEXT_ADDR` = `UCC_BIT(2)`,
 `UCC_CONTEXT_ATTR_FIELD_CONTEXT_ADDR_LEN` = `UCC_BIT(3)` }

7.3.1 Detailed Description

Data-structures associated with context creation and management routines

7.3.2 Data Structure Documentation

7.3.2.1 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

<code>uint64_t</code>	<code>mask</code>	
-----------------------	-------------------	--

Data Fields

ucc_context_type_t	ctx_type	
ucc_coll_sync_type_t	sync_type	
ucc_context_oob_coll_t	oob	
uint64_t	ctx_id	

7.3.2.2 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	ctx_type	
ucc_coll_sync_type_t	sync_type	
ucc_context_addr_t	ctx_addr	
ucc_context_addr_len_t	ctx_addr_len	

7.3.3 Typedef Documentation**7.3.3.1 ucc_context_oob_coll_t**

```
typedef struct ucc_context_oob_coll ucc_context_oob_coll_t
```

7.3.3.2 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.

7.3.3.3 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.

7.3.3.4 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.

7.3.3.5 ucc_context_config_h

```
typedef struct ucc_context_config* ucc_context_config_h
```

7.3.4 Enumeration Type Documentation

7.3.4.1 ucc_context_type_t

```
enum ucc_context_type_t
```

Enumerator

UCC_CONTEXT_EXCLUSIVE	
UCC_CONTEXT_SHARED	

7.3.4.2 ucc_context_params_field

```
enum ucc_context_params_field
```

Enumerator

UCC_CONTEXT_PARAM_FIELD_TYPE	
UCC_CONTEXT_PARAM_FIELD_COLL_SYNC_TYPE	
UCC_CONTEXT_PARAM_FIELD_COLL_OOB	
UCC_CONTEXT_PARAM_FIELD_ID	

7.3.4.3 ucc_context_attr_field

```
enum ucc_context_attr_field
```

Enumerator

UCC_CONTEXT_ATTR_FIELD_TYPE	
UCC_CONTEXT_ATTR_FIELD_COLL_SYNC_TYPE	
UCC_CONTEXT_ATTR_FIELD_CONTEXT_ADDR	
UCC_CONTEXT_ATTR_FIELD_CONTEXT_ADDR_LEN	

7.4 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.
- `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.

7.4.1 Detailed Description

Context create and management routines

7.4.2 Function Documentation

7.4.2.1 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`

7.4.2.2 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 ucc_context_config_h
----	---------------	---

Description

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

7.4.2.3 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
out	<i>params</i>	Customizations for the communication context
out	<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`

7.4.2.4 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 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`

7.4.2.5 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`

7.4.2.6 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`

7.5 Team abstraction data-structures

Data Structures

- struct `ucc_mem_map_params`
- struct `ucc_team_p2p_conn`
- struct `ucc_team_oob_coll`
- struct `ucc_ep_map_strided`
- struct `ucc_ep_map_array`
- struct `ucc_ep_map_cb`
- struct `ucc_ep_map_t`
- union `ucc_ep_map_t.__unnamed__`
- struct `ucc_team_params`

Structure representing the parameters to customize the team. [More...](#)

- struct `ucc_team_attr`

Structure representing the team attributes. [More...](#)

Typedefs

- typedef struct `ucc_mem_map_params` `ucc_mem_map_params_t`
- typedef struct `ucc_team_p2p_conn` `ucc_team_p2p_conn`
- typedef struct `ucc_team_oob_coll` `ucc_team_oob_coll_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_t`
- typedef void * `ucc_context_addr_len_t`

Enumerations

- enum `ucc_team_params_field` {
`UCC_TEAM_PARAM_FIELD_POST_ORDERING` = `UCC_BIT(0)`,
`UCC_TEAM_PARAM_FIELD_OUTSTANDING_CALLS` = `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_TYPE` = `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)` }
- 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_TYPE` = `UCC_BIT(3)`,
`UCC_TEAM_ATTR_FIELD_SYNC_TYPE` = `UCC_BIT(4)`,
`UCC_TEAM_ATTR_FIELD_MEM_PARAMS` = `UCC_BIT(5)` }

- enum `ucc_mem_constraints_t` {
`UCC_MEM_CONSTRAINT_SYMMETRIC` = `UCC_BIT(0)`,
`UCC_MEM_CONSTRAINT_PERSISTENT` = `UCC_BIT(1)`,
`UCC_MEM_CONSTRAINT_ALIGN32` = `UCC_BIT(2)`,
`UCC_MEM_CONSTRAINT_ALIGN64` = `UCC_BIT(3)`,
`UCC_MEM_CONSTRAINT_ALIGN128` = `UCC_BIT(4)` }
- enum `ucc_mem_hints_t` {
`UCC_MEM_HINT_REMOTE_ATOMICS` = 0,
`UCC_MEM_HINT_REMOTE_COUNTERS` }
- enum `ucc_post_ordering_t` {
`UCC_COLLECTIVE_POST_ORDERED` = 0,
`UCC_COLLECTIVE_POST_UNORDERED` = 1 }
- 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 }

7.5.1 Detailed Description

Data-structures associated with team create and management routines

7.5.2 Data Structure Documentation

7.5.2.1 struct `ucc_mem_map_params`

Data Fields

<code>void *</code>	address	
<code>size_t</code>	len	
<code>ucc_mem_hints_t</code>	hints	
<code>ucc_mem_constraints_t</code>	constraints	

7.5.2.2 struct `ucc_ep_map_strided`

Data Fields

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

7.5.2.3 struct `ucc_ep_map_array`

Data Fields

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

7.5.2.4 struct `ucc_ep_map_t`

Data Fields

<code>ucc_ep_map_type_t</code>	type	
--------------------------------	------	--

Data Fields

uint64_t	ep_num	number of eps mapped to ctx
union ucc_ep_map_t	__unnamed__	

7.5.2.5 union [ucc_ep_map_t](#). __unnamed__

Data Fields

struct ucc_ep_map_strided	strided	
struct ucc_ep_map_array	array	
struct ucc_ep_map_cb	cb	

7.5.2.6 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	
ucc_post_ordering_t	ordering	
uint64_t	outstanding_colls	
uint64_t	ep	
uint64_t *	ep_list	
ucc_ep_range_type_t	ep_range	
uint64_t	team_size	
ucc_coll_sync_type_t	sync_type	
ucc_team_oob_coll_t	oob	
ucc_team_p2p_conn	p2p_conn	
ucc_mem_map_params_t	mem_params	
ucc_ep_map_t	ep_map	

7.5.2.7 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	

Data Fields

ucc_mem_map_params_t	mem_params	
--------------------------------------	------------	--

7.5.3 Typedef Documentation

7.5.3.1 [ucc_mem_map_params_t](#)

```
typedef struct ucc\_mem\_map\_params ucc\_mem\_map\_params\_t
```

7.5.3.2 [ucc_team_p2p_conn](#)

```
typedef struct ucc\_team\_p2p\_conn ucc\_team\_p2p\_conn
```

7.5.3.3 [ucc_team_oob_coll_t](#)

```
typedef struct ucc\_team\_oob\_coll ucc\_team\_oob\_coll\_t
```

7.5.3.4 [ucc_ep_map_t](#)

```
typedef struct ucc\_ep\_map\_t ucc\_ep\_map\_t
```

7.5.3.5 [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.

7.5.3.6 [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.

7.5.3.7 [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.

7.5.3.8 [ucc_p2p_conn_t](#)

```
typedef void* ucc\_p2p\_conn\_t
```

7.5.3.9 ucc_context_addr_t

```
typedef void* ucc_context_addr_t
```

7.5.3.10 ucc_context_addr_len_t

```
typedef void* ucc_context_addr_len_t
```

7.5.4 Enumeration Type Documentation**7.5.4.1 ucc_team_params_field**

```
enum ucc_team_params_field
```

Enumerator

UCC_TEAM_PARAM_FIELD_POST_ORDERING	
UCC_TEAM_PARAM_FIELD_OUTSTANDING_CALLS	
UCC_TEAM_PARAM_FIELD_EP	
UCC_TEAM_PARAM_FIELD_EP_LIST	
UCC_TEAM_PARAM_FIELD_EP_TYPE	
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	

7.5.4.2 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_TYPE	
UCC_TEAM_ATTR_FIELD_SYNC_TYPE	
UCC_TEAM_ATTR_FIELD_MEM_PARAMS	

7.5.4.3 ucc_mem_constraints_t

```
enum ucc_mem_constraints_t
```

Enumerator

UCC_MEM_CONSTRAINT_SYMMETRIC	
UCC_MEM_CONSTRAINT_PERSISTENT	

Enumerator

UCC_MEM_CONSTRAINT_ALIGN32	
UCC_MEM_CONSTRAINT_ALIGN64	
UCC_MEM_CONSTRAINT_ALIGN128	

7.5.4.4 ucc_mem_hints_t

enum `ucc_mem_hints_t`

Enumerator

UCC_MEM_HINT_REMOTE_ATOMICS	
UCC_MEM_HINT_REMOTE_COUNTERS	

7.5.4.5 ucc_post_ordering_t

enum `ucc_post_ordering_t`

Enumerator

UCC_COLLECTIVE_POST_ORDERED	
UCC_COLLECTIVE_POST_UNORDERED	

7.5.4.6 ucc_ep_range_type_t

enum `ucc_ep_range_type_t`

Enumerator

UCC_COLLECTIVE_EP_RANGE_CONTIG	
UCC_COLLECTIVE_EP_RANGE_NONCONTIG	

7.5.4.7 ucc_ep_map_type_t

enum `ucc_ep_map_type_t`

Enumerator

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

7.6 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.

7.6.1 Detailed Description

Team create and management routines

7.6.2 Function Documentation

7.6.2.1 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	<i>contexts</i>	Communication contexts abstracting the resources
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`

7.6.2.2 `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`

7.6.2.3 `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 blocking 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.

Returns

Error code as defined by `ucc_status_t`

7.6.2.4 `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`

7.6.2.5 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`

7.6.2.6 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`

7.6.2.7 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`

7.6.2.8 `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`

7.7 Collective operations data-structures

Data Structures

- struct `ucc_coll_buffer_info`

Typedefs

- 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 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_coll_buffer_flags_t` {
 `UCC_COLL_BUFF_FLAG_IN_PLACE` = `UCC_BIT(0)`,
 `UCC_COLL_BUFF_FLAG_PERSISTENT` = `UCC_BIT(1)`,
 `UCC_COLL_BUFF_FLAG_COUNT_64BIT` = `UCC_BIT(2)`,
 `UCC_COLL_BUFF_FLAG_DISPLACEMENTS_64BIT` = `UCC_BIT(3)` }
- enum `ucc_error_type_t` {
 `UCC_ERR_TYPE_LOCAL` = 0,
 `UCC_ERR_TYPE_GLOBAL` = 1 }
- enum `ucc_coll_op_args_field` {
 `UCC_COLL_ARG_FIELD_COLL_TYPE` = `UCC_BIT(0)`,
 `UCC_COLL_ARG_FIELD_BUFFER_INFO` = `UCC_BIT(1)`,
 `UCC_COLL_ARG_FIELD_PREDEFINED_REDUCTIONS` = `UCC_BIT(2)`,
 `UCC_COLL_ARG_FIELD_USERDEFINED_REDUCTIONS` = `UCC_BIT(3)`,
 `UCC_COLL_ARG_FIELD_ERROR_TYPE` = `UCC_BIT(4)`,
 `UCC_COLL_ARG_FIELD_TAG` = `UCC_BIT(5)`,
 `UCC_COLL_ARG_FIELD_ROOT` = `UCC_BIT(6)` }

7.7.1 Detailed Description

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

7.7.2 Data Structure Documentation

7.7.2.1 struct `ucc_coll_buffer_info`

Data Fields

<code>void *</code>	<code>src_buffer</code>	
<code>ucc_count_t *</code>	<code>src_counts</code>	
<code>ucc_aint_t *</code>	<code>src_displacements</code>	
<code>void *</code>	<code>dst_buffer</code>	
<code>ucc_count_t *</code>	<code>dst_counts</code>	
<code>ucc_aint_t *</code>	<code>dst_displacements</code>	
<code>ucc_datatype_t</code>	<code>src_datatype</code>	
<code>ucc_datatype_t</code>	<code>dst_datatype</code>	
<code>uint64_t</code>	<code>flags</code>	

7.7.3 Typedef Documentation

7.7.3.1 ucc_coll_buffer_info_t

```
typedef struct ucc_coll_buffer_info ucc_coll_buffer_info_t
```

7.7.3.2 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.

7.7.3.3 ucc_count_t

```
typedef uint64_t ucc_count_t
```

7.7.3.4 ucc_aint_t

```
typedef uint64_t ucc_aint_t
```

7.7.3.5 ucc_coll_id_t

```
typedef uint16_t ucc_coll_id_t
```

7.7.4 Enumeration Type Documentation

7.7.4.1 ucc_coll_buffer_flags_t

```
enum ucc_coll_buffer_flags_t
```

Enumerator

UCC_COLL_BUFF_FLAG_IN_PLACE	
UCC_COLL_BUFF_FLAG_PERSISTENT	
UCC_COLL_BUFF_FLAG_COUNT_64BIT	
UCC_COLL_BUFF_FLAG_DISPLACEMENTS_64BIT	

7.7.4.2 ucc_error_type_t

```
enum ucc_error_type_t
```

Enumerator

UCC_ERR_TYPE_LOCAL	
UCC_ERR_TYPE_GLOBAL	

7.7.4.3 ucc_coll_op_args_field

enum `ucc_coll_op_args_field`

Enumerator

UCC_COLL_ARG_FIELD_COLL_TYPE	
UCC_COLL_ARG_FIELD_BUFFER_INFO	
UCC_COLL_ARG_FIELD_PREDEFINED_REDUCTIONS	
UCC_COLL_ARG_FIELD_USERDEFINED_REDUCTIONS	
UCC_COLL_ARG_FIELD_ERROR_TYPE	
UCC_COLL_ARG_FIELD_TAG	
UCC_COLL_ARG_FIELD_ROOT	

7.8 Collective Operations

Data Structures

- struct `ucc_coll_op_args`

Structure representing arguments for the collective operations. [More...](#)

Typedefs

- typedef void(* `ucc_reduction_dtype_mapper_t`) (void *invec, void *inoutvec, `ucc_count_t` *count, `ucc_datatype_t` dtype)
The reduction wrapper provides a method to map custom user types to higher level programming model datatypes.
- typedef void(* `ucc_userdefined_reduction_op_t`) (void *invec, void *inoutvec, `ucc_count_t` *count, `ucc_datatype_t` dtype)
The user-defined reduction function signature.
- typedef struct `ucc_coll_op_args` `ucc_coll_op_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_op_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_op_args_t` *coll_args, `ucc_coll_req_h` *request, `ucc_team_h` team)
The routine to initialize and post a collective operation.
- `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.

7.8.1 Detailed Description

Collective operations invocation and progress

7.8.2 Data Structure Documentation

7.8.2.1 struct `ucc_coll_op_args`

Description

`ucc_coll_op_args_t` defines the parameters that can be used to customize the collective operation. The “mask” bit array fields are defined by `ucc_coll_op_args_field`. The bits in “mask” bit array is defined by `ucc_coll_op_args_field`, which correspond to fields in structure `ucc_coll_op_args_t`. The valid fields of the structure are specified by setting the corresponding bit to “1” in the bit-array “mask”. When bits corresponding to the fields are not set, the fields are not defined.

The collective operation is selected by field “coll_type”. 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 provides the participant endpoint value. The user can request either “local” or “global” error information using the “error_type” field.

Data Fields

<code>uint64_t</code>	<code>mask</code>	
<code>ucc_coll_type_t</code>	<code>coll_type</code>	Type of collective operation
<code>ucc_coll_buffer_info_t</code>	<code>buffer_info</code>	Buffer info for the collective
<code>ucc_reduction_op_t</code>	<code>predefined_reduction_op</code>	Reduction operation, if reduce or all-reduce operation selected
<code>ucc_userdefined_reduction_op_t</code>	<code>custom_reduction_op</code>	User defined reduction operation
<code>ucc_error_type_t</code>	<code>error_type</code>	Error type
<code>ucc_coll_id_t</code>	<code>tag</code>	Used for ordering collectives
<code>uint64_t</code>	<code>root</code>	Root endpoint for rooted collectives

7.8.3 Typedef Documentation

7.8.3.1 `ucc_reduction_dtype_mapper_t`

```
typedef void(* ucc_reduction_dtype_mapper_t) (void *invec, void *inoutvec, ucc_count_t *count,
ucc_datatype_t dtype)
```

Parameters

in	<i>invec</i>	The input elements to be reduced by the user function
in	<i>inoutvec</i>	The input elements to be reduced and output of the reduction
in	<i>count</i>	The number of elements of type "dtype" to be reduced
in	<i>dtype</i>	Datatype passed to the reduction operation

Description

This function is called by the UCC library before calling the user-defined reduction. Hence, the signature of this function is same as `ucc_userdefined_reductions_op_t`. It maps the custom user types to higher level programming model datatypes (such as MPI datatypes)

7.8.3.2 `ucc_userdefined_reduction_op_t`

```
typedef void(* ucc_userdefined_reduction_op_t) (void *invec, void *inoutvec, ucc_count_t *count,
ucc_datatype_t dtype)
```

Parameters

in	<i>invec</i>	The input elements to be reduced by the user function
in	<i>inoutvec</i>	The input elements to be reduced and output of the reduction
in	<i>count</i>	The number of elements of type "dtype" to be reduced
in	<i>dtype</i>	Datatype passed to the reduction operation

Description

`ucc_userdefined_reduction_op_t` is a reduction operation signature for user-defined reductions. The signature closely follows the MPI signature.

7.8.3.3 `ucc_coll_op_args_t`

```
typedef struct ucc_coll_op_args ucc_coll_op_args_t
```

Description

`ucc_coll_op_args_t` defines the parameters that can be used to customize the collective operation. The “mask” bit array fields are defined by `ucc_coll_op_args_field`. The bits in “mask” bit array is defined by `ucc_coll_op_args_field`, which correspond to fields in structure `ucc_coll_op_args_t`. The valid fields of the structure are specified by setting the corresponding bit to “1” in the bit-array “mask”. When bits corresponding to the fields are not set, the fields are not defined.

The collective operation is selected by field “coll_type”. 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 provides the participant endpoint value. The user can request either “local” or “global” error information using the “error_type” field.

7.8.3.4 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.

7.8.4 Function Documentation

7.8.4.1 ucc_collective_init()

```
ucc_status_t ucc_collective_init (
    ucc_coll_op_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`

7.8.4.2 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`

7.8.4.3 `ucc_collective_init_and_post()`

```
ucc_status_t ucc_collective_init_and_post (
    ucc_coll_op_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`

7.8.4.4 `ucc_collective_test()`

```
ucc_status_t ucc_collective_test (
    ucc_coll_req_h request )
```

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`

7.8.4.5 `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`

7.9 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_OP_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_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.

7.9.1 Detailed Description

Helper functions to be used across the library

7.9.2 Enumeration Type Documentation

7.9.2.1 `ucc_config_print_flags_t`

enum `ucc_config_print_flags_t`

Enumerator

<code>UCC_CONFIG_PRINT_CONFIG</code>	
<code>UCC_CONFIG_PRINT_HEADER</code>	
<code>UCC_CONFIG_PRINT_DOC</code>	
<code>UCC_CONFIG_PRINT_HIDDEN</code>	

7.9.2.2 `ucc_status_t`

enum `ucc_status_t`

Enumerator

<code>UCC_OK</code>	
<code>UCC_INPROGRESS</code>	
<code>UCC_OPERATION_INITIALIZED</code>	
<code>UCC_ERR_OP_NOT_SUPPORTED</code>	

Enumerator

UCC_ERR_NOT_IMPLEMENTED	
UCC_ERR_INVALID_PARAM	
UCC_ERR_NO_MEMORY	
UCC_ERR_NO_RESOURCE	
UCC_ERR_LAST	

7.9.3 Function Documentation

7.9.3.1 ucc_status_string()

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

Chapter 8

Data Structure Documentation

8.1 ucc_context_oob_coll Struct Reference

OOB collective operation for creating the context.

Data Fields

- `int(* 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)`
- `uint32_t participants`
- `void * coll_info`

8.1.1 Field Documentation

8.1.1.1 allgather

```
int(* ucc_context_oob_coll::allgather) (void *src_buf, void *recv_buf, size_t size, void *allgather←  
_info, void **request)
```

8.1.1.2 req_test

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

8.1.1.3 req_free

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

8.1.1.4 participants

```
uint32_t ucc_context_oob_coll::participants
```

8.1.1.5 coll_info

```
void* ucc_context_oob_coll::coll_info
```

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

- `ucc.h`

8.2 ucc_ep_map_cb Struct Reference

Data Fields

- `uint64_t(* cb)(uint64_t ep, void *cb_ctx)`
- `void *cb_ctx`

8.2.1 Field Documentation

8.2.1.1 cb

```
uint64_t(* ucc_ep_map_cb::cb)(uint64_t ep, void *cb_ctx)
```

8.2.1.2 cb_ctx

```
void* ucc_ep_map_cb::cb_ctx
```

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

- `ucc.h`

8.3 ucc_team_oob_coll Struct Reference

Data Fields

- `int(* 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)`
- `uint32_t participants`
- `void *coll_info`

8.3.1 Field Documentation

8.3.1.1 allgather

```
int(* ucc_team_oob_coll::allgather)(void *src_buf, void *recv_buf, size_t size, void *allgather_info, void **request)
```

8.3.1.2 req_test

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

8.3.1.3 req_free

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

8.3.1.4 participants

```
uint32_t ucc_team_oob_coll::participants
```

8.3.1.5 coll_info

`void* ucc_team_oob_coll::coll_info`

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

- ucc.h

8.4 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)`

8.4.1 Field Documentation

8.4.1.1 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)`

8.4.1.2 conn_info_release

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

8.4.1.3 conn_ctx

`void* ucc_team_p2p_conn::conn_ctx`

8.4.1.4 req_test

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

8.4.1.5 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_context_oob_coll, [46](#)
 - ucc_team_oob_coll, [47](#)
- cb
 - ucc_ep_map_cb, [47](#)
- cb_ctx
 - ucc_ep_map_cb, [47](#)
- coll_info
 - ucc_context_oob_coll, [46](#)
 - ucc_team_oob_coll, [47](#)
- Collective Operations, [38](#)
 - ucc_coll_op_args_t, [40](#)
 - ucc_collective_finalize, [42](#)
 - ucc_collective_init, [41](#)
 - ucc_collective_init_and_post, [42](#)
 - ucc_collective_post, [41](#)
 - ucc_collective_test, [42](#)
 - ucc_mem_h, [41](#)
 - ucc_reduction_dtype_mapper_t, [40](#)
 - ucc_userdefined_reduction_op_t, [40](#)
- Collective operations data-structures, [35](#)
 - ucc_aint_t, [36](#)
 - UCC_COLL_ARG_FIELD_BUFFER_INFO, [37](#)
 - UCC_COLL_ARG_FIELD_COLL_TYPE, [37](#)
 - UCC_COLL_ARG_FIELD_ERROR_TYPE, [37](#)
 - UCC_COLL_ARG_FIELD_PREDEFINED_REDUCTIONS, [37](#)
 - UCC_COLL_ARG_FIELD_ROOT, [37](#)
 - UCC_COLL_ARG_FIELD_TAG, [37](#)
 - UCC_COLL_ARG_FIELD_USERDEFINED_REDUCTIONS, [37](#)
 - UCC_COLL_BUFF_FLAG_COUNT_64BIT, [36](#)
 - UCC_COLL_BUFF_FLAG_DISPLACEMENTS_64BIT, [36](#)
 - UCC_COLL_BUFF_FLAG_IN_PLACE, [36](#)
 - UCC_COLL_BUFF_FLAG_PERSISTENT, [36](#)
 - ucc_coll_buffer_flags_t, [36](#)
 - ucc_coll_buffer_info_t, [36](#)
 - ucc_coll_id_t, [36](#)
 - ucc_coll_op_args_field, [36](#)
 - ucc_coll_req_h, [36](#)
 - ucc_count_t, [36](#)
 - UCC_ERR_TYPE_GLOBAL, [36](#)
 - UCC_ERR_TYPE_LOCAL, [36](#)
 - ucc_error_type_t, [36](#)
- conn_ctx
 - ucc_team_p2p_conn, [48](#)
- conn_info_lookup
 - ucc_team_p2p_conn, [48](#)
- conn_info_release
 - ucc_team_p2p_conn, [48](#)
- Context abstraction data-structures, [19](#)
 - ucc_context_attr_field, [21](#)
 - UCC_CONTEXT_ATTR_FIELD_COLL_SYNC_TYPE, [21](#)
 - UCC_CONTEXT_ATTR_FIELD_CONTEXT_ADDR, [21](#)
 - UCC_CONTEXT_ATTR_FIELD_CONTEXT_ADDR_LEN, [21](#)
 - UCC_CONTEXT_ATTR_FIELD_TYPE, [21](#)
 - ucc_context_attr_t, [20](#)
 - ucc_context_config_h, [21](#)
 - UCC_CONTEXT_EXCLUSIVE, [21](#)
 - ucc_context_h, [20](#)
 - ucc_context_oob_coll_t, [20](#)
 - UCC_CONTEXT_PARAM_FIELD_COLL_OOB, [21](#)
 - UCC_CONTEXT_PARAM_FIELD_COLL_SYNC_TYPE, [21](#)
 - UCC_CONTEXT_PARAM_FIELD_ID, [21](#)
 - UCC_CONTEXT_PARAM_FIELD_TYPE, [21](#)
 - ucc_context_params_field, [21](#)
 - ucc_context_params_t, [20](#)
 - UCC_CONTEXT_SHARED, [21](#)
 - ucc_context_type_t, [21](#)
- Context abstraction routines, [22](#)
 - ucc_context_config_read, [22](#)
 - ucc_context_config_release, [22](#)
 - ucc_context_create, [23](#)
 - ucc_context_destroy, [23](#)
 - ucc_context_get_attr, [24](#)
 - ucc_context_progress, [23](#)
- Library initialization and finalization routines, [15](#)
 - ucc_finalize, [17](#)
 - ucc_init, [16](#)
 - ucc_lib_config_modify, [16](#)
 - ucc_lib_config_print, [16](#)
 - ucc_lib_config_read, [15](#)
 - ucc_lib_config_release, [15](#)
 - ucc_lib_get_attr, [17](#)
- Library initialization data-structures, [8](#)
 - ucc_coll_sync_type_t, [13](#)
 - UCC_COLL_TYPE_ALLGATHER, [12](#)
 - UCC_COLL_TYPE_ALLREDUCE, [12](#)
 - UCC_COLL_TYPE_ALLTOALL, [12](#)

- UCC_COLL_TYPE_BARRIER, 12
- UCC_COLL_TYPE_BCAST, 12
- UCC_COLL_TYPE_FANIN, 12
- UCC_COLL_TYPE_FANOUT, 12
- UCC_COLL_TYPE_GATHER, 12
- UCC_COLL_TYPE_REDUCE, 12
- UCC_COLL_TYPE_SCATTER, 12
- ucc_coll_type_t, 12
- ucc_datatype_t, 12
- UCC_DT_FLOAT16, 13
- UCC_DT_FLOAT32, 13
- UCC_DT_FLOAT64, 13
- UCC_DT_INT128, 12
- UCC_DT_INT16, 12
- UCC_DT_INT32, 12
- UCC_DT_INT64, 12
- UCC_DT_INT8, 12
- UCC_DT_OPAQUE, 13
- UCC_DT_UINT128, 13
- UCC_DT_UINT16, 12
- UCC_DT_UINT32, 13
- UCC_DT_UINT64, 13
- UCC_DT_UINT8, 12
- UCC_DT_USERDEFINED, 13
- ucc_lib_attr_field, 14
- UCC_LIB_ATTR_FIELD_COLL_TYPES, 14
- UCC_LIB_ATTR_FIELD_REDUCTION_TYPES, 14
- UCC_LIB_ATTR_FIELD_SYNC_TYPE, 14
- UCC_LIB_ATTR_FIELD_THREAD_MODE, 14
- ucc_lib_attr_t, 11
- ucc_lib_config_h, 11
- ucc_lib_h, 11
- UCC_LIB_PARAM_FIELD_COLL_TYPES, 14
- UCC_LIB_PARAM_FIELD_REDUCTION_TYPES, 14
- UCC_LIB_PARAM_FIELD_REDUCTION_WRAPPER, 14
- UCC_LIB_PARAM_FIELD_SYNC_TYPE, 14
- UCC_LIB_PARAM_FIELD_THREAD_MODE, 13
- ucc_lib_params_field, 13
- ucc_lib_params_t, 10
- UCC_NO_SYNC_COLLECTIVES, 13
- UCC_OP_AND, 11
- UCC_OP_BAND, 12
- UCC_OP_BOR, 12
- UCC_OP_BXOR, 12
- UCC_OP_LAND, 11
- UCC_OP_LOR, 12
- UCC_OP_LXOR, 12
- UCC_OP_MAX, 11
- UCC_OP_MAXLOC, 12
- UCC_OP_MIN, 11
- UCC_OP_MINLOC, 12
- UCC_OP_OR, 11
- UCC_OP_PROD, 11
- UCC_OP_SUM, 11
- UCC_OP_USERDEFINED, 11
- UCC_OP_XOR, 11
- ucc_reduction_op_t, 11
- UCC_SYNC_COLLECTIVES, 13
- UCC_THREAD_FUNNELED, 13
- ucc_thread_mode_t, 13
- UCC_THREAD_MULTIPLE, 13
- UCC_THREAD_SINGLE, 13
- participants
 - ucc_context_oob_coll, 46
 - ucc_team_oob_coll, 47
- req_free
 - ucc_context_oob_coll, 46
 - ucc_team_oob_coll, 47
 - ucc_team_p2p_conn, 48
- req_test
 - ucc_context_oob_coll, 46
 - ucc_team_oob_coll, 47
 - ucc_team_p2p_conn, 48
- Team abstraction data-structures, 25
 - UCC_COLLECTIVE_EP_RANGE_CONTIG, 30
 - UCC_COLLECTIVE_EP_RANGE_NONCONTIG, 30
 - UCC_COLLECTIVE_POST_ORDERED, 30
 - UCC_COLLECTIVE_POST_UNORDERED, 30
 - ucc_context_addr_len_t, 29
 - ucc_context_addr_t, 28
 - UCC_EP_MAP_ARRAY, 30
 - UCC_EP_MAP_CB, 30
 - UCC_EP_MAP_FULL, 30
 - UCC_EP_MAP_STRIDED, 30
 - ucc_ep_map_t, 28
 - ucc_ep_map_type_t, 30
 - ucc_ep_range_type_t, 30
 - UCC_MEM_CONSTRAINT_ALIGN128, 30
 - UCC_MEM_CONSTRAINT_ALIGN32, 30
 - UCC_MEM_CONSTRAINT_ALIGN64, 30
 - UCC_MEM_CONSTRAINT_PERSISTENT, 29
 - UCC_MEM_CONSTRAINT_SYMMETRIC, 29
 - ucc_mem_constraints_t, 29
 - UCC_MEM_HINT_REMOTE_ATOMICS, 30
 - UCC_MEM_HINT_REMOTE_COUNTERS, 30
 - ucc_mem_hints_t, 30
 - ucc_mem_map_params_t, 28
 - ucc_p2p_conn_t, 28
 - ucc_post_ordering_t, 30
 - ucc_team_attr_field, 29
 - UCC_TEAM_ATTR_FIELD_EP, 29
 - UCC_TEAM_ATTR_FIELD_EP_TYPE, 29

- UCC_TEAM_ATTR_FIELD_MEM_PARAMS, UCC_COLL_BUFF_FLAG_DISPLACEMENTS_64BIT
29 Collective operations data-structures, 36
- UCC_TEAM_ATTR_FIELD_OUTSTANDING_64BIT, UCC_COLL_BUFF_FLAG_IN_PLACE
29 Collective operations data-structures, 36
- UCC_TEAM_ATTR_FIELD_POST_ORDERING, UCC_COLL_BUFF_FLAG_PERSISTENT
29 Collective operations data-structures, 36
- UCC_TEAM_ATTR_FIELD_SYNC_TYPE, ucc_coll_buffer_flags_t
29 Collective operations data-structures, 36
- ucc_team_attr_t, 28
- ucc_team_h, 28
- ucc_team_oob_coll_t, 28
- ucc_team_p2p_conn, 28
- UCC_TEAM_PARAM_FIELD_EP, 29
- UCC_TEAM_PARAM_FIELD_EP_LIST, 29
- UCC_TEAM_PARAM_FIELD_EP_MAP, 29
- UCC_TEAM_PARAM_FIELD_EP_TYPE, 29
- UCC_TEAM_PARAM_FIELD_MEM_PARAMS, ucc_coll_buffer_info, 35
- 29 ucc_coll_buffer_info_t
Collective operations data-structures, 36
- UCC_TEAM_PARAM_FIELD_OOB, 29
- UCC_TEAM_PARAM_FIELD_OUTSTANDING_64BIT, ucc_coll_id_t
29 Collective operations data-structures, 36
- UCC_TEAM_PARAM_FIELD_P2P_CONN, ucc_coll_op_args, 38
- 29 ucc_coll_op_args_field
Collective operations data-structures, 36
- UCC_TEAM_PARAM_FIELD_POST_ORDERING, ucc_coll_op_args_t
29 Collective Operations, 40
- UCC_TEAM_PARAM_FIELD_SYNC_TYPE, ucc_coll_req_h
29 Collective operations data-structures, 36
- UCC_TEAM_PARAM_FIELD_TEAM_SIZE, ucc_coll_sync_type_t
29 Library initialization data-structures, 13
- ucc_team_params_field, 29
- ucc_team_params_t, 28
- Team abstraction routines, 31
- ucc_team_create_from_parent, 32
- ucc_team_create_post, 31
- ucc_team_create_test, 32
- ucc_team_destroy, 32
- ucc_team_get_all_eps, 34
- ucc_team_get_attr, 32
- ucc_team_get_my_ep, 33
- ucc_team_get_size, 33
- ucc_aint_t
Collective operations data-structures, 36
- UCC_COLL_ARG_FIELD_BUFFER_INFO
Collective operations data-structures, 37
- UCC_COLL_ARG_FIELD_COLL_TYPE
Collective operations data-structures, 37
- UCC_COLL_ARG_FIELD_ERROR_TYPE
Collective operations data-structures, 37
- UCC_COLL_ARG_FIELD_PREDEFINED_REDUCTIONS
Collective operations data-structures, 37
- UCC_COLL_ARG_FIELD_ROOT
Collective operations data-structures, 37
- UCC_COLL_ARG_FIELD_TAG
Collective operations data-structures, 37
- UCC_COLL_ARG_FIELD_USERDEFINED_REDUCTIONS
Collective operations data-structures, 37
- UCC_COLL_BUFF_FLAG_COUNT_64BIT
Collective operations data-structures, 36
- UCC_COLL_BUFF_FLAG_DISPLACEMENTS_64BIT
Collective operations data-structures, 36
- UCC_COLL_BUFF_FLAG_IN_PLACE
Collective operations data-structures, 36
- UCC_COLL_BUFF_FLAG_PERSISTENT
Collective operations data-structures, 36
- ucc_coll_buffer_info, 35
- ucc_coll_buffer_info_t
Collective operations data-structures, 36
- ucc_coll_id_t
Collective operations data-structures, 36
- ucc_coll_op_args, 38
- ucc_coll_op_args_field
Collective operations data-structures, 36
- ucc_coll_op_args_t
Collective Operations, 40
- ucc_coll_req_h
Collective operations data-structures, 36
- ucc_coll_sync_type_t
Library initialization data-structures, 13
- UCC_COLL_TYPE_ALLGATHER
Library initialization data-structures, 12
- UCC_COLL_TYPE_ALLREDUCE
Library initialization data-structures, 12
- UCC_COLL_TYPE_ALLTOALL
Library initialization data-structures, 12
- UCC_COLL_TYPE_BARRIER
Library initialization data-structures, 12
- UCC_COLL_TYPE_BCAST
Library initialization data-structures, 12
- UCC_COLL_TYPE_FANIN
Library initialization data-structures, 12
- UCC_COLL_TYPE_FANOUT
Library initialization data-structures, 12
- UCC_COLL_TYPE_GATHER
Library initialization data-structures, 12
- UCC_COLL_TYPE_REDUCE
Library initialization data-structures, 12
- UCC_COLL_TYPE_SCATTER
Library initialization data-structures, 12
- ucc_coll_type_t
Library initialization data-structures, 12
- UCC_COLLECTIVE_EP_RANGE_CONTIG
Team abstraction data-structures, 30
- UCC_COLLECTIVE_EP_RANGE_NONCONTIG
Team abstraction data-structures, 30
- ucc_collective_finalize
Collective Operations, 42
- ucc_collective_init
Collective Operations, 41
- ucc_collective_init_and_post
Collective Operations, 42
- ucc_collective_post
Collective Operations, 41
- UCC_COLLECTIVE_POST_ORDERED
Team abstraction data-structures, 30

UCC_COLLECTIVE_POST_UNORDERED
 Team abstraction data-structures, 30
 ucc_collective_test
 Collective Operations, 42
 UCC_CONFIG_PRINT_CONFIG
 Utility Operations, 44
 UCC_CONFIG_PRINT_DOC
 Utility Operations, 44
 ucc_config_print_flags_t
 Utility Operations, 44
 UCC_CONFIG_PRINT_HEADER
 Utility Operations, 44
 UCC_CONFIG_PRINT_HIDDEN
 Utility Operations, 44
 ucc_context_addr_len_t
 Team abstraction data-structures, 29
 ucc_context_addr_t
 Team abstraction data-structures, 28
 ucc_context_attr, 20
 ucc_context_attr_field
 Context abstraction data-structures, 21
 UCC_CONTEXT_ATTR_FIELD_COLL_SYNC_TYPE
 Context abstraction data-structures, 21
 UCC_CONTEXT_ATTR_FIELD_CONTEXT_ADDR
 Context abstraction data-structures, 21
 UCC_CONTEXT_ATTR_FIELD_CONTEXT_ADDR_LEN
 Context abstraction data-structures, 21
 UCC_CONTEXT_ATTR_FIELD_TYPE
 Context abstraction data-structures, 21
 ucc_context_attr_t
 Context abstraction data-structures, 20
 ucc_context_config_h
 Context abstraction data-structures, 21
 ucc_context_config_read
 Context abstraction routines, 22
 ucc_context_config_release
 Context abstraction routines, 22
 ucc_context_create
 Context abstraction routines, 23
 ucc_context_destroy
 Context abstraction routines, 23
 UCC_CONTEXT_EXCLUSIVE
 Context abstraction data-structures, 21
 ucc_context_get_attr
 Context abstraction routines, 24
 ucc_context_h
 Context abstraction data-structures, 20
 ucc_context_oob_coll, 46
 allgather, 46
 coll_info, 46
 participants, 46
 req_free, 46
 req_test, 46
 ucc_context_oob_coll_t
 Context abstraction data-structures, 20
 UCC_CONTEXT_PARAM_FIELD_COLL_OOB
 Context abstraction data-structures, 21
 UCC_CONTEXT_PARAM_FIELD_COLL_SYNC_TYPE
 Context abstraction data-structures, 21
 Context abstraction data-structures, 21
 UCC_CONTEXT_PARAM_FIELD_ID
 Context abstraction data-structures, 21
 UCC_CONTEXT_PARAM_FIELD_TYPE
 Context abstraction data-structures, 21
 ucc_context_params, 19
 ucc_context_params_field
 Context abstraction data-structures, 21
 ucc_context_params_t
 Context abstraction data-structures, 20
 ucc_context_progress
 Context abstraction routines, 23
 UCC_CONTEXT_SHARED
 Context abstraction data-structures, 21
 ucc_context_type_t
 Context abstraction data-structures, 21
 ucc_count_t
 Collective operations data-structures, 36
 ucc_datatype_t
 Library initialization data-structures, 12
 UCC_DT_FLOAT16
 Library initialization data-structures, 13
 UCC_DT_FLOAT32
 Library initialization data-structures, 13
 UCC_DT_FLOAT64
 Library initialization data-structures, 13
 UCC_DT_INT128
 Library initialization data-structures, 12
 UCC_DT_INT16
 Library initialization data-structures, 12
 UCC_DT_INT32
 Library initialization data-structures, 12
 UCC_DT_INT64
 Library initialization data-structures, 12
 UCC_DT_INT8
 Library initialization data-structures, 12
 UCC_DT_OPAQUE
 Library initialization data-structures, 13
 UCC_DT_UINT128
 Library initialization data-structures, 13
 UCC_DT_UINT16
 Library initialization data-structures, 12
 UCC_DT_UINT32
 Library initialization data-structures, 13
 UCC_DT_UINT64
 Library initialization data-structures, 13
 UCC_DT_UINT8
 Library initialization data-structures, 12
 UCC_DT_USERDEFINED
 Library initialization data-structures, 13
 UCC_EP_MAP_ARRAY
 Team abstraction data-structures, 30
 ucc_ep_map_array, 26
 UCC_EP_MAP_CB
 Team abstraction data-structures, 30
 ucc_ep_map_cb, 47
 cb, 47
 cb_ctx, 47

- UCC_EP_MAP_FULL
 - Team abstraction data-structures, 30
- UCC_EP_MAP_STRIDED
 - Team abstraction data-structures, 30
- ucc_ep_map_strided, 26
- ucc_ep_map_t, 26
 - Team abstraction data-structures, 28
- ucc_ep_map_t. __unnamed __, 27
- ucc_ep_map_type_t
 - Team abstraction data-structures, 30
- ucc_ep_range_type_t
 - Team abstraction data-structures, 30
- UCC_ERR_INVALID_PARAM
 - Utility Operations, 45
- UCC_ERR_LAST
 - Utility Operations, 45
- UCC_ERR_NO_MEMORY
 - Utility Operations, 45
- UCC_ERR_NO_RESOURCE
 - Utility Operations, 45
- UCC_ERR_NOT_IMPLEMENTED
 - Utility Operations, 45
- UCC_ERR_OP_NOT_SUPPORTED
 - Utility Operations, 44
- UCC_ERR_TYPE_GLOBAL
 - Collective operations data-structures, 36
- UCC_ERR_TYPE_LOCAL
 - Collective operations data-structures, 36
- ucc_error_type_t
 - Collective operations data-structures, 36
- ucc_finalize
 - Library initialization and finalization routines, 17
- ucc_init
 - Library initialization and finalization routines, 16
- UCC_INPROGRESS
 - Utility Operations, 44
- ucc_lib_attr, 10
- ucc_lib_attr_field
 - Library initialization data-structures, 14
- UCC_LIB_ATTR_FIELD_COLL_TYPES
 - Library initialization data-structures, 14
- UCC_LIB_ATTR_FIELD_REDUCTION_TYPES
 - Library initialization data-structures, 14
- UCC_LIB_ATTR_FIELD_SYNC_TYPE
 - Library initialization data-structures, 14
- UCC_LIB_ATTR_FIELD_THREAD_MODE
 - Library initialization data-structures, 14
- ucc_lib_attr_t
 - Library initialization data-structures, 11
- ucc_lib_config_h
 - Library initialization data-structures, 11
- ucc_lib_config_modify
 - Library initialization and finalization routines, 16
- ucc_lib_config_print
 - Library initialization and finalization routines, 16
- ucc_lib_config_read
 - Library initialization and finalization routines, 15
- ucc_lib_config_release
 - Library initialization and finalization routines, 15
- ucc_lib_get_attr
 - Library initialization and finalization routines, 17
- ucc_lib_h
 - Library initialization data-structures, 11
- UCC_LIB_PARAM_FIELD_COLL_TYPES
 - Library initialization data-structures, 14
- UCC_LIB_PARAM_FIELD_REDUCTION_TYPES
 - Library initialization data-structures, 14
- UCC_LIB_PARAM_FIELD_REDUCTION_WRAPPER
 - Library initialization data-structures, 14
- UCC_LIB_PARAM_FIELD_SYNC_TYPE
 - Library initialization data-structures, 14
- UCC_LIB_PARAM_FIELD_THREAD_MODE
 - Library initialization data-structures, 13
- ucc_lib_params, 10
- ucc_lib_params_field
 - Library initialization data-structures, 13
- ucc_lib_params_t
 - Library initialization data-structures, 10
- UCC_MEM_CONSTRAINT_ALIGN128
 - Team abstraction data-structures, 30
- UCC_MEM_CONSTRAINT_ALIGN32
 - Team abstraction data-structures, 30
- UCC_MEM_CONSTRAINT_ALIGN64
 - Team abstraction data-structures, 30
- UCC_MEM_CONSTRAINT_PERSISTENT
 - Team abstraction data-structures, 29
- UCC_MEM_CONSTRAINT_SYMMETRIC
 - Team abstraction data-structures, 29
- ucc_mem_constraints_t
 - Team abstraction data-structures, 29
- ucc_mem_h
 - Collective Operations, 41
- UCC_MEM_HINT_REMOTE_ATOMICS
 - Team abstraction data-structures, 30
- UCC_MEM_HINT_REMOTE_COUNTERS
 - Team abstraction data-structures, 30
- ucc_mem_hints_t
 - Team abstraction data-structures, 30
- ucc_mem_map_params, 26
- ucc_mem_map_params_t
 - Team abstraction data-structures, 28
- UCC_NO_SYNC_COLLECTIVES
 - Library initialization data-structures, 13
- UCC_OK
 - Utility Operations, 44
- UCC_OP_AND
 - Library initialization data-structures, 11
- UCC_OP_BAND
 - Library initialization data-structures, 12
- UCC_OP_BOR
 - Library initialization data-structures, 12
- UCC_OP_BXOR
 - Library initialization data-structures, 12
- UCC_OP_LAND
 - Library initialization data-structures, 11
- UCC_OP_LOR

- Library initialization data-structures, [12](#)
- UCC_OP_LXOR
 - Library initialization data-structures, [12](#)
- UCC_OP_MAX
 - Library initialization data-structures, [11](#)
- UCC_OP_MAXLOC
 - Library initialization data-structures, [12](#)
- UCC_OP_MIN
 - Library initialization data-structures, [11](#)
- UCC_OP_MINLOC
 - Library initialization data-structures, [12](#)
- UCC_OP_OR
 - Library initialization data-structures, [11](#)
- UCC_OP_PROD
 - Library initialization data-structures, [11](#)
- UCC_OP_SUM
 - Library initialization data-structures, [11](#)
- UCC_OP_USERDEFINED
 - Library initialization data-structures, [11](#)
- UCC_OP_XOR
 - Library initialization data-structures, [11](#)
- UCC_OPERATION_INITIALIZED
 - Utility Operations, [44](#)
- ucc_p2p_conn_t
 - Team abstraction data-structures, [28](#)
- ucc_post_ordering_t
 - Team abstraction data-structures, [30](#)
- ucc_reduction_dtype_mapper_t
 - Collective Operations, [40](#)
- ucc_reduction_op_t
 - Library initialization data-structures, [11](#)
- ucc_status_string
 - Utility Operations, [45](#)
- ucc_status_t
 - Utility Operations, [44](#)
- UCC_SYNC_COLLECTIVES
 - Library initialization data-structures, [13](#)
- ucc_team_attr, [27](#)
- ucc_team_attr_field
 - Team abstraction data-structures, [29](#)
- UCC_TEAM_ATTR_FIELD_EP
 - Team abstraction data-structures, [29](#)
- UCC_TEAM_ATTR_FIELD_EP_TYPE
 - Team abstraction data-structures, [29](#)
- UCC_TEAM_ATTR_FIELD_MEM_PARAMS
 - Team abstraction data-structures, [29](#)
- UCC_TEAM_ATTR_FIELD_OUTSTANDING_CALLS
 - Team abstraction data-structures, [29](#)
- UCC_TEAM_ATTR_FIELD_POST_ORDERING
 - Team abstraction data-structures, [29](#)
- UCC_TEAM_ATTR_FIELD_SYNC_TYPE
 - Team abstraction data-structures, [29](#)
- ucc_team_attr_t
 - Team abstraction data-structures, [28](#)
- ucc_team_create_from_parent
 - Team abstraction routines, [32](#)
- ucc_team_create_post
 - Team abstraction routines, [31](#)
- ucc_team_create_test
 - Team abstraction routines, [32](#)
- ucc_team_destroy
 - Team abstraction routines, [32](#)
- ucc_team_get_all_eps
 - Team abstraction routines, [34](#)
- ucc_team_get_attr
 - Team abstraction routines, [32](#)
- ucc_team_get_my_ep
 - Team abstraction routines, [33](#)
- ucc_team_get_size
 - Team abstraction routines, [33](#)
- ucc_team_h
 - Team abstraction data-structures, [28](#)
- ucc_team_oob_coll, [47](#)
 - allgather, [47](#)
 - coll_info, [47](#)
 - participants, [47](#)
 - req_free, [47](#)
 - req_test, [47](#)
- ucc_team_oob_coll_t
 - Team abstraction data-structures, [28](#)
- ucc_team_p2p_conn, [48](#)
 - conn_ctx, [48](#)
 - conn_info_lookup, [48](#)
 - conn_info_release, [48](#)
 - req_free, [48](#)
 - req_test, [48](#)
- Team abstraction data-structures, [28](#)
- UCC_TEAM_PARAM_FIELD_EP
 - Team abstraction data-structures, [29](#)
- UCC_TEAM_PARAM_FIELD_EP_LIST
 - Team abstraction data-structures, [29](#)
- UCC_TEAM_PARAM_FIELD_EP_MAP
 - Team abstraction data-structures, [29](#)
- UCC_TEAM_PARAM_FIELD_EP_TYPE
 - Team abstraction data-structures, [29](#)
- UCC_TEAM_PARAM_FIELD_MEM_PARAMS
 - Team abstraction data-structures, [29](#)
- UCC_TEAM_PARAM_FIELD_OOB
 - Team abstraction data-structures, [29](#)
- UCC_TEAM_PARAM_FIELD_OUTSTANDING_CALLS
 - Team abstraction data-structures, [29](#)
- UCC_TEAM_PARAM_FIELD_P2P_CONN
 - Team abstraction data-structures, [29](#)
- UCC_TEAM_PARAM_FIELD_POST_ORDERING
 - Team abstraction data-structures, [29](#)
- UCC_TEAM_PARAM_FIELD_SYNC_TYPE
 - Team abstraction data-structures, [29](#)
- UCC_TEAM_PARAM_FIELD_TEAM_SIZE
 - Team abstraction data-structures, [29](#)
- ucc_team_params, [27](#)
- ucc_team_params_field
 - Team abstraction data-structures, [29](#)
- ucc_team_params_t
 - Team abstraction data-structures, [28](#)
- UCC_THREAD_FUNNELED
 - Library initialization data-structures, [13](#)

- ucc_thread_mode_t
 - Library initialization data-structures, [13](#)
- UCC_THREAD_MULTIPLE
 - Library initialization data-structures, [13](#)
- UCC_THREAD_SINGLE
 - Library initialization data-structures, [13](#)
- ucc_userdefined_reduction_op_t
 - Collective Operations, [40](#)
- Utility Operations, [44](#)
 - UCC_CONFIG_PRINT_CONFIG, [44](#)
 - UCC_CONFIG_PRINT_DOC, [44](#)
 - ucc_config_print_flags_t, [44](#)
 - UCC_CONFIG_PRINT_HEADER, [44](#)
 - UCC_CONFIG_PRINT_HIDDEN, [44](#)
 - UCC_ERR_INVALID_PARAM, [45](#)
 - UCC_ERR_LAST, [45](#)
 - UCC_ERR_NO_MEMORY, [45](#)
 - UCC_ERR_NO_RESOURCE, [45](#)
 - UCC_ERR_NOT_IMPLEMENTED, [45](#)
 - UCC_ERR_OP_NOT_SUPPORTED, [44](#)
 - UCC_INPROGRESS, [44](#)
 - UCC_OK, [44](#)
 - UCC_OPERATION_INITIALIZED, [44](#)
 - ucc_status_string, [45](#)
 - ucc_status_t, [44](#)