Notes

- The term GPU is scattered throghout this document. Perhaps this should be replaced with DEVICE so that we're more device-agnostic. However, my concern with that is what if we want to support spaces on multiple device types (i.e., GPUs and SmartNICs, etc.)? The current language can be extended for future devices/spaces but it seems likely that that would result in lots of bloat. Not exactly sure how to rectify this.

Memory Space Management Routines

PEs in an OpenSHMEM program allocate symmetric memory using either the traditional `shmem_malloc` and `shmem_calloc` routines—which allocate from the default symmetric heap—or the space-aware allocation routines such as `shmem_space_malloc` and `shmem_space_calloc` which allow programs to specify which symmetric heap to allocate from. OpenSHMEM memory spaces represent distinct symmetric heaps that may reside in different memory domains, such as the traditional host memory domain or secondary symmetric heaps in GPU memory, high-bandwidth memory, or other specialized memory types.

Default Space and Backward Compatibility: There is exactly one default space in an OpenSHMEM program. The traditional `shmem_malloc` and `shmem_calloc` routines allocate memory from the default symmetric heap, which is accessed through the `SHMEM_SPACE_DEFAULT` handle. The default space aliases to either `SHMEM_SPACE_CPU` or `SHMEM_SPACE_GPU` based on implementation choice or user configuration via the `SHMEM_DEFAULT_SPACE` environment variable. The default space shall be world-accessible: its team equals `SHMEM_TEAM_WORLD`. This requirement preserves the semantics of existing OpenSHMEM applications that use world-team collectives on default-space buffers.

Predefined Memory Spaces

An OpenSHMEM memory space may be predefined (i.e., provided by the OpenSHMEM library) or potentially defined by future extensions. Currently, only predefined spaces are supported.

All predefined memory spaces are valid for the duration of the OpenSHMEM portion of an application. All valid memory spaces contain at least one symmetric memory region.

Space Handles

A memory space handle is an opaque object with type `shmem_space_t` that is used to reference a memory space. Memory space handles are not remotely accessible objects. The predefined memory spaces may be accessed via the space handles listed in the **Spaces Constants** section below.

OpenSHMEM memory allocation routines that do not accept a space handle argument operate on the default symmetric heap, which may be accessed through the `SHMEM_SPACE_DEFAULT` handle. The default space aliases to either `SHMEM_SPACE_CPU` or `SHMEM_SPACE_GPU` based on implementation choice or user configuration, and memory allocated from this space is equivalent to memory allocated via `shmem_malloc` and `shmem_calloc`.

Memory allocation routines can access symmetric heaps in different memory domains through their respective space handles. The SHMEM_SPACE_CPU handle provides access to the traditional host symmetric heap, while the SHMEM_SPACE_GPU handle provides access to a symmetric heap in GPU memory when GPU hardware is available and accessible.

A memory space handle may be initialized to or assigned the value SHMEM_SPACE_INVALID to indicate that handle does not reference a valid memory space. When managed in this way, applications can use an equality comparison to test whether a given space handle references a valid memory space.

Type Definition

```
typedef void *shmem_space_t;
```

Spaces Handles and Constants (Add to "Library Constants" Table in Section 6)

Handle or Constant	Description
C/C++: SHMEM_SPACE_DEFAULT	A memory space handle representing the default symmetric heap. This space aliases to either SHMEM_SPACE_CPU or SHMEM_SPACE_GPU based on implementation choice or user configuration via the SHMEM_DEFAULT_SPACE environment variable. This space is equivalent to memory allocated via shmem_malloc and shmem_calloc.
C/C++: SHMEM_SPACE_CPU	A memory space handle representing a symmetric heap in CPU (host) memory. If the CPU space is not available (e.g., disabled or not supported), this handle is set to SHMEM_SPACE_INVALID.
C/C++: SHMEM_SPACE_GPU	A memory space handle representing a symmetric heap in GPU memory. If the GPU space is not available (e.g., disabled or not supported), this handle is set to SHMEM_SPACE_INVALID.
C/C++: SHMEM_SPACE_INVALID	A value corresponding to an invalid memory space handle.

Note: When a memory space is not available (e.g., disabled by environment variable or not supported by the system), the corresponding space handle (e.g., SHMEM_SPACE_GPU, SHMEM_SPACE_CPU) is set to SHMEM_SPACE_INVALID.

Memory Space Properties

In OpenSHMEM, a memory space encapsulates resources used to manage symmetric memory allocation within a specific memory domain. Each memory space can have associated properties including size limits, accessibility constraints, and device affinity.

Space Sizing:

- SHMEM_SPACE_DEFAULT is sized according to the SHMEM_SYMMETRIC_SIZE environment variable

- `SHMEM_SPACE_CPU` is sized according to the `SHMEM_CPU_SYMMETRIC_SIZE` environment variable
- `SHMEM_SPACE_GPU` is sized according to the `SHMEM_GPU_SYMMETRIC_SIZE` environment variable

Note: When `SHMEM_SPACE_DEFAULT` aliases to `SHMEM_SPACE_CPU`, the `SHMEM_CPU_SYMMETRIC_SIZE` environment variable takes precedence over `SHMEM_SYMMETRIC_SIZE` if both are set. Similarly, when `SHMEM_SPACE_DEFAULT` aliases to `SHMEM_SPACE_GPU`, the `SHMEM_GPU_SYMMETRIC_SIZE` environment variable takes precedence over `SHMEM_SYMMETRIC_SIZE` if both are set. If no size environment variables are set for a space, the implementation chooses the default size for that space. Numeric pointer values for symmetric allocations in device spaces are not required to be identical across PEs; the symmetric property is defined by existence, size, and layout across member PEs.

Default Space Configuration:

- `SHMEM_DEFAULT_SPACE=CPU|GPU` requests which space `SHMEM_SPACE_DEFAULT` aliases to.
- If not set, the implementation selects the default space.
- The selected default shall be world-accessible. If the requested default is not available to all PEs or would not be world-accessible, OpenSHMEM initialization shall fail.

Pre-enabled Spaces:

Implementations may enable multiple spaces at initialization under environment variable control (e.g., CPU and GPU). Enabling a space makes its handle valid and its symmetric heap available according to sizing variables. Exactly one space is the default alias `SHMEM_SPACE_DEFAULT`; all other enabled spaces are accessed via space-aware routines. Legacy routines (`shmem_malloc`, `shmem_calloc`) operate only on the default alias.

Environment Variables (Add to "Environment Variables" Table in Section 8)

The OpenSHMEM memory spaces specification provides environment variables that allow users to configure space enablement and sizing. The implementations of the specification are free to define additional variables.

All environment variables that control memory space enablement, sizing, and default selection shall have identical values across all PEs in the job. If these variables differ across PEs, OpenSHMEM initialization shall fail.

Variable	Value	Description
`SHMEM_ENABLE_CPU_SPACE`	Any	Enables CPU symmetric heap. When not set or empty, CPU symmetric heap is disabled.
`SHMEM_ENABLE_GPU_SPACE`	Any	Enables GPU symmetric heap. When not set or empty, GPU symmetric heap is disabled.
`SHMEM_DEFAULT_SPACE`	`CPU` or `GPU`	Controls which space `SHMEM_SPACE_DEFAULT` aliases to. If not set, the implementation chooses the default space.

Variable	Value	Description
SHMEM_SYMMETRIC_SIZE	Non-negative integer or floating point value with an optional character suffix	Specifies the size (in bytes) of the default symmetric heap memory per PE. This applies to SHMEM_SPACE_DEFAULT and maintains backward compatibility with existing OpenSHMEM applications. The resulting size is implementation-defined and must be at least as large as the integer ceiling of the product of the numeric prefix and the scaling factor. The allowed character suffixes for the scaling factor are as follows: • k or K multiplies by 2^{10} (kibibytes) • m or M multiplies by 2^{20} (mebibytes) • g or G multiplies by 2^{30} (gibibytes) • t or T multiplies by 2^{40} (tebibytes)
SHMEM_CPU_SYMMETRIC_SIZE	Non-negative integer or floating point value with an optional character suffix	Specifies the size (in bytes) of the CPU symmetric heap memory per PE. The resulting size is implementation-defined and must be at least as large as the integer ceiling of the product of the numeric prefix and the scaling factor. The allowed character suffixes for the scaling factor are as follows: • k or K multiplies by 2^{10} (kibibytes) • m or M multiplies by 2^{20} (mebibytes) • g or G multiplies by 2^{30} (gibibytes) • t or T multiplies by 2^{40} (tebibytes)
SHMEM_GPU_SYMMETRIC_SIZE	Non-negative integer or floating point value with an optional character suffix	Specifies the size (in bytes) of the GPU symmetric heap memory per PE. The resulting size is implementation-defined and must be at least as large as the integer ceiling of the product of the numeric prefix and the scaling factor. The allowed character suffixes for the scaling factor are as follows: • k or K multiplies by 2^{10} (kibibytes) • m or M multiplies by 2^{20} (mebibytes) • g or G multiplies by 2^{30} (gibibytes) • t or T multiplies by 2^{40} (tebibytes)

Space Enablement:

- When a space is disabled, `shmem_space_is_available()` returns nonzero for that space.
- When both spaces are disabled, OpenSHMEM initialization shall fail.

Space Availability:

- SHMEM_SPACE_CPU is available when CPU symmetric heap is enabled and host memory exists
- SHMEM_SPACE_GPU is available when GPU symmetric heap is enabled and GPU hardware is detected and accessible

- Applications can query space availability using `shmem_space_is_available()`
- Space availability is fixed at OpenSHMEM initialization and does not change during the lifetime of the program

---

Spaces, Teams, Collectives

OpenSHMEM teams provide a mechanism for grouping PEs and performing collective operations. Teams are not bound to a single memory space. Collective operations must operate on data from a single memory space.

Team-Space Relationship

Default Teams:

`SHMEM_TEAM_WORLD` includes all PEs in the job. `SHMEM_TEAM_CPU` includes all PEs that can access CPU memory space and is available when `SHMEM_SPACE_CPU` is enabled. `SHMEM_TEAM_GPU` includes all PEs that can access GPU memory space and is available when `SHMEM_SPACE_GPU` is enabled. Teams do not have exclusive ownership of any memory space.

Team Access to Spaces:

Teams may perform RMA and atomic operations on data in any memory space subject to per-space capabilities. An operation on data from space `S` is valid only when every member of the team is a member of the team associated with `S`. Space-specific teams (`SHMEM_TEAM_CPU`, `SHMEM_TEAM_GPU`) are optimized for their respective spaces.

Collective Operations and Memory Spaces

Single-Space Requirement:

Collective operations must operate on data from a single memory space. All PEs participating in a collective must provide pointers to data allocated from the same space, and the collective must be performed on a team whose members are all members of the team associated with that space. The runtime determines the memory space of a buffer via internal range mapping of symmetric allocations.

SHMEM_TEAM_WORLD and Collectives:

`SHMEM_TEAM_WORLD` may be used only when the memory space's associated team equals `SHMEM_TEAM_WORLD`. The most common case is the default space (via `shmem_malloc` or `shmem_calloc`).

Cross-Device Access in Collectives:

PEs participate in collectives on a space only if they are members of the team associated with that space. All buffers must be allocated from that space.

Heterogeneous System Considerations:

In heterogeneous systems, membership in a space's team is implementation-defined. PEs may access a space only when they are members of that space's team.

Collective Space Detection:

When a collective is called, the runtime determines which space each buffer belongs to. If all pointers point to data from the same space, the collective proceeds. If pointers point to data from different spaces, the behavior is undefined.

Inter-Space Communication via RMA and Atomic Operations

OpenSHMEM may support inter-space communication via Remote Memory Access (RMA) and atomic operations. Support is implementation-defined and gated by capabilities. Calls that target a space lacking the required capability have undefined behavior.

Context-Based Inter-Space Communication

Explicit contexts can be created for specific teams/spaces to enable inter-space communication. Contexts provide a mechanism for PEs in different spaces to communicate via RMA and atomic operations. The context determines which space the operations target, regardless of the source PE's space. Context-based communication may be more efficient for repeated operations between specific spaces.

Implicit Routing (no explicit context)

When no explicit context is provided, RMA and atomic operations rely on the destination pointer to determine the target memory space. The runtime detects the memory space of the destination buffer and routes the operation accordingly. This allows inter-space communication without explicit context management and is often simpler for occasional operations.

Inter-Space RMA Operations

PEs may perform `shmem_put`, `shmem_get`, and other RMA operations across different memory spaces subject to capabilities. The runtime routes operations based on the destination buffer’s space. If RMA targeting a given space is not supported, behavior is undefined.

Inter-Space Atomic Operations

Atomic operations (e.g., `shmem_atomic_inc`, `shmem_atomic_fetch`) may work across different memory spaces, but availability and guarantees are implementation-defined and advertised via capabilities. If atomic operations targeting a given space are not supported, behavior is undefined. Cross-space atomic operations may have different performance characteristics than within-space operations. Context-based atomic operations provide explicit space targeting when supported. When no explicit context is used, the runtime determines the target space from the destination pointer and the call is valid only when that space advertises atomic support.

Space-Specific Team Availability

Space-specific teams are only available when their corresponding memory space is enabled and accessible:

- `SHMEM_TEAM_CPU` is available when `SHMEM_SPACE_CPU` is enabled and accessible
- `SHMEM_TEAM_GPU` is available when `SHMEM_SPACE_GPU` is enabled and accessible
- When a space-specific team is not available, the corresponding team handle is set to `SHMEM_TEAM_INVALID`

Applications can use `shmem_team_is_valid()` to check team availability before using space-specific teams.

Undefined Behavior in Memory Spaces (Add to Annex C)

The following table summarizes undefined behavior cases specific to OpenSHMEM memory spaces that should be added to Annex C of the OpenSHMEM specification.

Inappropriate Usage	**Undefined Behavior**
Use of invalid space handles	In OpenSHMEM space-aware routines that accept a space handle, if the space handle is invalid (e.g., `SHMEM_SPACE_INVALID`), the behavior is undefined.
Mismatched space allocation and free	Memory allocated from a specific space must be freed using the same space handle. If `shmem_space_free` is called with a different space handle than was used for allocation, the behavior is undefined.
Non-symmetric space allocation	The space-aware memory allocation routines are collectives. All PEs in the team associated with the space must call `shmem_space_malloc` or `shmem_space_calloc` with identical space and size arguments. Program behavior after a mismatched call is undefined.
Allocation to an unavailable space	Attempting to allocate memory from a space whose handle is invalid (space not available) results in undefined behavior.
Invalid default space configuration	OpenSHMEM initialization shall fail.
Both spaces disabled	OpenSHMEM initialization shall fail.
Invalid space handle in availability check	If `shmem_space_is_available` is called with a space handle that is not a valid predefined space handle, the behavior is undefined.
Non-identical environment variable values across PEs	OpenSHMEM initialization shall fail.
Invalid pointer in space query	If `shmem_get_space` is called with a pointer that was not allocated through OpenSHMEM symmetric memory allocation routines, the behavior is undefined.

X.X SHMEM_SPACE_IS_AVAILABLE

Check if a memory space is available.

SYNOPSIS

C/C++:

```
int shmem_space_is_available(shmem_space_t space);
```

DESCRIPTION

Arguments

Parameter	Direction	Description
`space`	IN	An OpenSHMEM memory space handle

API Description

The `shmem_space_is_available` routine checks whether the specified memory space is available for allocation on the calling PE. This routine is not collective and may be called by individual PEs to determine space availability before attempting allocation.

The `SHMEM_SPACE_CPU` space is available when CPU symmetric heap is enabled and host memory exists. The `SHMEM_SPACE_GPU` space is available when GPU symmetric heap is enabled and GPU hardware is detected and accessible. The `SHMEM_SPACE_DEFAULT` space is available when it aliases to an available space.

If `space` compares equal to `SHMEM_SPACE_INVALID`, the routine returns a nonzero value. If `space` is otherwise invalid, the behavior is undefined.

Return Values

Zero if the space is available; otherwise, nonzero.

X.X SHMEM_SPACE_MALLOC

Allocate memory in a specific memory space.

SYNOPSIS

C/C++:

```
void *shmem_space_malloc(shmem_space_t space, size_t size);
```

DESCRIPTION

Arguments

Parameter	Direction	Description
space	IN	An OpenSHMEM memory space handle
size	IN	The size, in bytes, of the memory to allocate

API Description

The shmem_space_malloc routine is a collective operation on the team associated with the specified memory space and returns the symmetric address of a block of at least size bytes from that space, which shall be suitably aligned so that it may be assigned to a pointer to any type of object. This space is allocated from the symmetric memory within the specified space (in contrast to malloc, which allocates from the private heap). Only PEs that are members of the team associated with the space shall call this routine. When size is zero, the shmem_space_malloc routine performs no action and returns a null pointer; otherwise, shmem_space_malloc calls a procedure that is semantically equivalent to a team synchronization on exit for the team associated with the space. The values of the space and size arguments must be identical on all PEs in that team; otherwise, the behavior is undefined.

Memory allocated from a space can only be freed using shmem_space_free with the same space handle.

If space compares equal to SHMEM_SPACE_INVALID, then a null pointer is returned. If space is otherwise invalid, the behavior is undefined.

Return Values

The shmem_space_malloc routine returns the symmetric address of the allocated space; otherwise, it returns a null pointer.

X.X SHMEM_SPACE_CALLOC

Allocate and zero-initialize memory in a specific memory space.

SYNOPSIS

C/C++:

```
void *shmem_space_calloc(shmem_space_t space, size_t count, size_t size);
```

DESCRIPTION

Arguments

Parameter	Direction	Description
space	IN	An OpenSHMEM memory space handle
count	IN	The number of elements to allocate
size	IN	The size, in bytes, of each element

API Description

The `shmem_space_calloc` routine is a collective operation on the team associated with the specified memory space that allocates a region of remotely accessible memory for an array of `count` objects of `size` bytes each from that space and returns a pointer to the lowest byte address of the allocated symmetric memory. Only PEs that are members of the team associated with the space shall call this routine. The space is initialized to all bits zero.

If the allocation succeeds, the pointer returned shall be suitably aligned so that it may be assigned to a pointer to any type of object. If the allocation does not succeed, or either `count` or `size` is 0, the return value is a null pointer.

The values for `space`, `count`, and `size` shall each be equal across all PEs in the team associated with the space calling `shmem_space_calloc`; otherwise, the behavior is undefined.

Memory allocated from a space can only be freed using `shmem_space_free` with the same space handle.

When `count` or `size` is 0, the `shmem_space_calloc` routine returns without performing a synchronization. Otherwise, this routine calls a procedure that is semantically equivalent to a team synchronization on exit for the team associated with the space.

If `space` compares equal to `SHMEM_SPACE_INVALID`, then a null pointer is returned. If `space` is otherwise invalid, the behavior is undefined.

Return Values

The `shmem_space_calloc` routine returns a pointer to the lowest byte address of the allocated space; otherwise, it returns a null pointer.

X.X SHMEM_SPACE_FREE

Free memory allocated in a specific memory space.

SYNOPSIS

C/C++:

```
void shmem_space_free(shmem_space_t space, void *ptr);
```

DESCRIPTION

Arguments

Parameter	Direction	Description
`space`	IN	An OpenSHMEM memory space handle
`ptr`	IN	A pointer to memory to free

API Description

The `shmem_space_free` routine is a collective operation on the team associated with the specified memory space that causes the block to which `ptr` points to be deallocated from that space, that is, made available for further allocation within that space. Only PEs that are members of the team associated with the space shall call this routine. If `ptr` is a null pointer, no action is performed; otherwise, `shmem_space_free` calls a synchronization on entry for the team associated with the space. It is the user's responsibility to ensure that no communication operations involving the given memory block are pending on other communication contexts prior to calling `shmem_space_free`. The `ptr` argument must be a pointer that was returned by `shmem_space_malloc` or `shmem_space_calloc` using the same space handle.

The values of the `space` and `ptr` arguments must be identical on all PEs; otherwise, the behavior is undefined.

If `space` compares equal to `SHMEM_SPACE_INVALID`, no operation is performed. If `space` is otherwise invalid, or if `ptr` was not allocated from the specified space, the behavior is undefined.

Return Values

None.

EXAMPLE

```
#include <shmem.h>

int main(void) {
  shmem_init();

  shmem_space_t space = SHMEM_SPACE_DEFAULT;

  size_t nelems = 16;
  int *a = (int *) shmem_space_malloc(space, nelems * sizeof(int));
  int *b = (int *) shmem_space_calloc(space, nelems, sizeof(int));

  shmem_space_free(space, a);
  shmem_space_free(space, b);

  shmem_finalize();
  return 0;
}
```