4.3. REGION_IDENT

Obtain the identifier of a region with a given name.

Synopsis

```
region_ident( name, rid )
```

Input Parameters

name	: string	user defined region name

Output Parameters

rid	: region_id	kernel defined region identifier

Completion Status

OK	region_ident operation successful
ILLEGAL_USE	operation not callable from XSR or ISR
INVALID_PARAMETER	a parameter refers to an illegal address
NAME_NOT_FOUND	name does not exist on node

Description

This operation searches the kernel data structure in the local node for a region with the given name, and returns its identifier if found. If there is more than one region with the same name, the kernel will return the identifier of one of them, the choice being implementation dependent.

4.4. REGION_GET_SEG

Get a segment from a region.

Synopsis

```
   region_get_seg( rid, seg_size, seg_addr )
```

Input Parameters

rid	: region_id	kernel defined region id
seg_size	: integer	requested segment size in bytes

Output Parameters

seg_addr	: address	address of obtained segment

Completion Status

OK	region_get_seg operation successful
ILLEGAL_USE	operation not callable from ISR
INVALID_PARAMETER	a parameter refers to an illegal address
INVALID_ID	region does not exist
OBJECT_DELETED	region specified has been deleted
NO_MORE_MEMORY	not enough contiguous memory in the region to allocate segment of requested size

Description

The region_get_seg operation is a request for a given sized segment from a given region's free memory pool. If the kernel cannot fulfil the request immediately, it returns the error completion status NO_MORE_MEMORY, otherwise the address of the allocated segment is returned. The allocation algorithm is implementation dependent.

Note that the actual size of the segment returned will be more than the size requested, if the latter is not a multiple of the region's granularity.

4.5. REGION_RET_SEG

Return a segment to its region.

Synopsis

```
region_ret_seg( rid, seg_addr )
```

Input Parameters

rid	: region_id	kernel defined region id
seg_addr	: address	address of segment to be returned

Output Parameters

<none>

Completion Status

OK	region_ret_seg operation successful
ILLEGAL_USE	operation not callable from ISR
INVALID_PARAMETER	a parameter refers to an illegal address
INVALID_ID	region does not exist
OBJECT_DELETED	region specified has been deleted
INVALID_SEGMENT	no segment allocated from this region at seg_addr

Description

This operation returns the given segment to the given region's free memory pool. The kernel checks that this segment was previously allocated from this region, and returns INVALID_SEGMENT if it wasn't.

4.6. REGION_INFO

Obtain information on a region.

Synopsis

```
   region_info( rid, size, max_segment, granularity )
```

Input Parameters

rid	: region_id	kernel defined region id

Output Parameters

size	: integer	length in bytes of overall area in region available for segment allocation
max_segment:	integer	length in bytes of maximum segment allocatable at time of call
granularity:	integer	allocation granularity in bytes

Completion Status

OK	region_info operation successful
ILLEGAL_USE	operation not callable from ISR
INVALID_PARAMETER	a parameter refers to an illegal address
INVALID_ID	region does not exist
OBJECT_DELETED	region specified has been deleted

Description

This operation provides information on the specified region. It returns the size of the region's area for segment allocation, which may be smaller than the region length given in region_create due to a possible formatting overhead. It returns also the size of the biggest segment allocatable from the region. This value should be used with care as it is just a snap-shot of the region's usage at the time of executing the operation. Finally it returns the region's allocatable granularity.

5. PARTITIONS

Partitions are areas of memory organized by the kernel as a pool of fixed size blocks. As for regions, the creating task supplies the area of memory to be used by the partition. The task also supplies the size of the blocks to be allocated from the partition. Any restrictions imposed on the block size are implementation dependent.

Partitions are simpler structures than regions, and are intended for use where speed of allocation is essential. Partitions may also be declared global, and be operated on from more than one node. However, this makes sense only if the nodes accessing the partition are all in the same shared memory system, and the partition is in shared memory.

Once the partition created, tasks may request blocks one at a time from it, and can return them in any order. Because the blocks are all the same size, there is no fragmentation problem in partitions. The exact allocation algorithms are implementation dependent.