13

Region Manager

13.1 Introduction

The **region manager** provides facilities to dynamically allocate memory in variable sized units. The directives provided by the region manager are:

Name	Directive Description
region_create	Create a region
region_ident	Get ID of a region
region_delete	Delete a region
region_extend	Add memory to a region
region_get_segment	Get segment from a region
region_return_segment	Return segment to a region
region_get_segment_size	Obtain size of a segment

13.2 Background

13.2.1 Definitions

A **region** makes up a physically contiguous memory space with user-defined boundaries from which variable-sized segments are dynamically allocated and deallocated. A **segment** is a variable size section of memory which is allocated in multiples of a user-defined page size. This page size is required to be a multiple of four greater than or equal to four. For example, if a request for a 350-byte

segment is made in a region with 256-byte pages, then a 512-byte segment is allocated.

Regions are organized as doubly linked chains of variable sized memory blocks. Memory requests are allocated using a first-fit algorithm. If available, the requester receives the number of bytes requested (rounded up to the next page size). **RTEMS** requires some overhead from the region's memory for each segment that is allocated. Therefore, an application should only modify the memory of a segment that has been obtained from the region. The application should **NOT** modify the memory outside of any obtained segments and within the region's boundaries while the region is currently active in the system.

Upon return to the region, the free block is coalesced with its neighbors (if free) on both sides to produce the largest possible unused block.

13.2.2 Building an Attribute Set

In general, an attribute set is built by a bitwise OR of the desired attribute components. The set of valid region attributes is provided in the following table:

CONSTANT	DESCRIPTION	DEFAULT
FIFO	tasks wait by FIFO	*
PRIORITY	tasks wait by priority	

Attribute values are specifically designed to be mutually exclusive, therefore bitwise OR and addition operations are equivalent as long as each attribute appears exactly once in the component list. An attribute listed as a default is not required to appear in the attribute list, although it is a good programming practice to specify default attributes. If all defaults are desired, the attribute **DEFAULT_ATTRIBUTES** should be specified on this call.

This example demonstrates the **attribute_set** parameter needed to create a region with the task priority waiting queue discipline. The **attribute_set** parameter to the **region_create** directive should be **PRIORITY**.

13.2.3 Building an Option Set

In general, an option is built by a bitwise OR of the desired option components. The set of valid options for the **region_get_segment** directive are listed in the following table:

CONSTANT	DESCRIPTION	DEFAULT
WAIT	wait for segment	*
NO_WAIT	task should not wait	

Option values are specifically designed to be mutually exclusive, therefore bitwise OR and addition operations are equivalent as long as each option appears exactly once in the component list. An option listed as a default is not required to appear in the option list, although it is a good programming practice to specify default options. If all defaults are desired, the option **DEFAULT_OPTIONS** should be specified on this call.

This example demonstrates the **option** parameter needed to poll for a segment. The **option** parameter passed to the **region_get_segment** directive should be **NO_WAIT**.

13.3 Operations

13.3.1 Creating a Region

The **region_create** directive creates a region with the user-defined name. The user may select FIFO or task priority as the method for placing waiting tasks in the task wait queue. **RTEMS** allocates a **Region Control Block** (RNCB) from the RNCB free list to maintain the newly created region. **RTEMS** also generates a unique region ID which is returned to the calling task.

It is not possible to calculate the exact number of bytes available to the user since **RTEMS** requires overhead for each segment allocated. For example, a region with one segment that is the size of the entire region has more available bytes than a region with two segments that collectively are the size of the entire region. This is because the region with one segment requires only the overhead for one segment, while the other region requires the overhead for two segments.

Due to automatic coalescing, the number of segments in the region dynamically changes. Therefore, the total overhead required by **RTEMS** dynamically changes.

13.3.2 Obtaining Region IDs

When a region is created, **RTEMS** generates a unique region ID and assigns it to the created region until it is deleted. The region ID may be obtained by either of two methods. First, as the result of an invocation of the **region_create** directive, the region ID is stored in a user provided location. Second, the region ID may be obtained later using the **region_ident** directive. The region ID is used by other region manager directives to access this region.

13.3.3 Adding Memory to a Region

The **region_extend** directive may be used to add memory to an existing region. The caller specifies the size in bytes and starting address of the memory being added.

NOTE: Please see the release notes or **RTEMS** source code for information regarding restrictions on the location of the memory being added in relation to memory already in the region.

13.3.4 Acquiring a Segment

The **region_get_segment** directive attempts to acquire a segment from a specified region. If the region has enough available free memory, then a segment is returned successfully to the caller. When the segment cannot be allocated, one of the following situations applies:

- *By default, the calling task will wait forever to acquire the segment.*
- *Specifying the **NO_WAIT** option forces an immediate return with an error status code.*
- *Specifying a timeout limits the interval the task will wait before returning with an error status code.*

If the task waits for the segment, then it is placed in the region's task wait queue in either FIFO or task priority order. All tasks waiting on a region are returned an error when the message queue is deleted.

13.3.5 Releasing a Segment

When a segment is returned to a region by the **region_return_segment** directive, it is merged with its unallocated neighbors to form the largest possible segment. The first task on the wait queue is examined to determine if its segment request can now be satisfied. If so, it is given a segment and unblocked. This process is repeated until the first task's segment request cannot be satisfied.

13.3.6 Obtaining the Size of a Segment

The **region_get_segment_size** directive returns the size in bytes of the specified segment. The size returned includes any "extra" memory included in the segment because of rounding up to a page size boundary.

13.3.7 Deleting a Region

A region can be removed from the system and returned to **RTEMS** with the **region_delete** directive. When a region is deleted, its control block is returned to the RNCB free list. A region with segments still allocated is not allowed to be deleted. Any task attempting to do so will be returned an error. As a result of this directive, all tasks blocked waiting to obtain a segment from the region will be readied and returned a status code which indicates that the region was deleted.

13.4 Directives

This section details the region manager's directives. A subsection is dedicated to each of this manager's directives and describes the calling sequence, related constants, usage, and status codes.

13.4.1 REGION_CREATE - Create a region

CALLING SEQUENCE:

```
procedure region_create (
   name             : in     rtems.name;
   starting_address : in     rtems.address;
   length           : in     rtems.unsigned32;
   page_size        : in     rtems.unsigned32;
   attribute_set    : in     rtems.attribute;
   id               :    out rtems.id;
   result           :    out rtems.status_codes
);
```

DIRECTIVE STATUS CODES:

SUCCESSFUL	region created successfully
INVALID_NAME	invalid task name
INVALID_ADDRESS	address not on four byte boundary
TOO_MANY	too many regions created
INVALID_SIZE	invalid page size

DESCRIPTION:

This directive creates a region from a physically contiguous memory space which starts at **starting_address** and is **length** bytes long. Segments allocated from the region will be a multiple of **page_size** bytes in length. The assigned region id is returned in **id**. This region id is used as an argument to other region related directives to access the region.

For control and maintenance of the region, **RTEMS** allocates and initializes an RNCB from the RNCB free pool. Thus memory from the region is not used to store the RNCB. However, some overhead within the region is required by **RTEMS** each time a segment is constructed in the region.

Specifying **PRIORITY** in **attribute_set** causes tasks waiting for a segment to be serviced according to task priority. Specifying **FIFO** in **attribute_set** or selecting **DEFAULT_ATTRIBUTES** will cause waiting tasks to be serviced in First In-First Out order.

The **starting_address** parameter must be aligned on a four byte boundary. The **page_size** parameter must be a multiple of four greater than or equal to four.

NOTES:

This directive will not cause the calling task to be preempted.

The following region attribute constants are defined by **RTEMS**:

CONSTANT	DESCRIPTION	DEFAULT
FIFO	tasks wait by FIFO	*
PRIORITY	tasks wait by priority	

13.4.2 REGION_IDENT - Get ID of a region

CALLING SEQUENCE:

```
procedure region_ident (
   name   : in     rtems.name;
   id     :    out rtems.id;
   result :    out rtems.status_codes
);
```

DIRECTIVE STATUS CODES:

SUCCESSFUL	region identified successfully
INVALID_NAME	region name not found

DESCRIPTION:

This directive obtains the region **id** associated with the region **name** to be acquired. If the region name is not unique, then the region id will match one of the regions with that name. However, this region id is not guaranteed to correspond to the desired region. The region id is used to access this region in other region manager directives.

NOTES:

This directive will not cause the running task to be preempted.

13.4.3 REGION_DELETE - Delete a region

CALLING SEQUENCE:

```
procedure region_delete (
   id     : in     rtems.id;
   result :    out rtems.status_codes
);
```

DIRECTIVE STATUS CODES:

SUCCESSFUL	region deleted successfully
INVALID_ID	invalid region id
RESOURCE_IN_USE	segments still in use

DESCRIPTION:

This directive deletes the region specified by **id**. The region cannot be deleted if any of its segments are still allocated. The RNCB for the deleted region is reclaimed by **RTEMS**.

NOTES:

This directive will not cause the calling task to be preempted.

The calling task does not have to be the task that created the region. Any local task that knows the region id can delete the region.

13.4.4 REGION_EXTEND - Add memory to a region

CALLING SEQUENCE:

```
procedure region_extend (
   id               : in     rtems.id;
   starting_address : in     rtems.address;
   length           : in     rtems.unsigned32;
   result           :    out rtems.status_codes
);
```

DIRECTIVE STATUS CODES:

SUCCESSFUL	region extended successfully
INVALID_ID	invalid region id
INVALID_ADDRESS	invalid address of area to add

DESCRIPTION:

This directive adds the memory which starts at **starting_address** for **length** bytes to the region specified by **id**.

NOTES:

This directive will not cause the calling task to be preempted.

The calling task does not have to be the task that created the region. Any local task that knows the region id can extend the region.

13.4.5 REGION_GET_SEGMENT - Get segment from a region

CALLING SEQUENCE:

```
procedure region_get_segment (
   id         : in     rtems.id;
   size       : in     rtems.unsigned32;
   option_set : in     rtems.option;
   timeout    : in     rtems.interval;
   segment    :    out rtems.address;
   result     :    out rtems.status_codes
);
```

DIRECTIVE STATUS CODES:

SUCCESSFUL	segment obtained successfully
INVALID_ID	invalid region id
INVALID_SIZE	request is for zero bytes or exceeds the size of maximum segment which is possible for this region
UNSATISFIED	segment of requested size not available
TIMEOUT	timed out waiting for segment
OBJECT_WAS_DELETED	semaphore deleted while waiting

DESCRIPTION:

This directive obtains a variable size segment from the region specified by **id**. The address of the allocated segment is returned in **segment**. The **WAIT** and **NO_WAIT** components of the **options** parameter are used to specify whether the calling tasks wish to wait for a segment to become available or return immediately if no segment is available. For either option, if a sufficiently sized segment is available, then the segment is successfully acquired by returning immediately with the **SUCCESSFUL** status code.

If the calling task chooses to return immediately and a segment large enough is not available, then an error code indicating this fact is returned. If the calling task chooses to wait for the segment and a segment large enough is not available, then the calling task is placed on the region's segment wait queue and blocked. If the region was created with the **PRIORITY** option, then the calling task is inserted into the wait queue according to its priority. However, if the region was created with the **FIFO** option, then the calling task is placed at the rear of the wait queue.

The **timeout** parameter specifies the maximum interval that a task is willing to wait to obtain a segment. If **timeout** is set to **NO_TIMEOUT**, then the calling task will wait forever.

NOTES:

The actual length of the allocated segment may be larger than the requested size because a segment size is always a multiple of the region's page size.

The following segment acquisition option constants are defined by **RTEMS**:

CONSTANT	DESCRIPTION	DEFAULT
WAIT	**wait for segment**	*
NO_WAIT	**task should not wait**	

13.4.6 REGION_RETURN_SEGMENT - Return segment to a region

CALLING SEQUENCE:

```
procedure region_return_segment (
   id      : in     rtems.id;
   segment : in     rtems.address;
   result  :    out rtems.status_codes
);
```

DIRECTIVE STATUS CODES:

SUCCESSFUL	segment returned successfully
INVALID_ID	invalid region id
INVALID_ADDRESS	segment address not in region

DESCRIPTION:

This directive returns the segment specified by **segment** to the region specified by **id**. The returned segment is merged with its neighbors to form the largest possible segment. The first task on the wait queue is examined to determine if its segment request can now be satisfied. If so, it is given a segment and unblocked. This process is repeated until the first task's segment request cannot be satisfied.

NOTES:

This directive will cause the calling task to be preempted if one or more local tasks are waiting for a segment and the following conditions exist:

a waiting task has a higher priority than the calling task

the size of the segment required by the waiting task is less than or equal to the size of the segment returned.

13.4.7 REGION_GET_SEGMENT_SIZE - Obtain size of a segment

CALLING SEQUENCE:

```
procedure region_get_segment_size (
   id          : in     rtems.id;
   segment     : in     rtems.address;
   size        :    out rtems.unsigned32;
   result      :    out rtems.status_codes
);
```

DIRECTIVE STATUS CODES:

SUCCESSFUL	segment obtained successfully
INVALID_ID	invalid region id
INVALID_ADDRESS	segment address not in region

DESCRIPTION:

This directive obtains the **size** in bytes of the specified **segment**.

NOTES:

The actual lengh of the allocated segment may be larger than the requested size because a segment size is always a multiple of the region's page size.