10

Event Manager

10.1 Introduction

The **event manager** provides a high performance method of intertask communication and synchronization. The directives provided by the event manager are:

Name	Directive Description
event_send	Send event set to a task
event_receive	Receive event condition

10.2 Background

10.2.1 Event Sets

An **event flag** is used by a task (or ISR) to inform another task of the occurrence of a significant situation. Thirty-two event flags are associated with each task. A collection of one or more event flags is referred to as an **event set**. The application developer should remember the following key characteristics of event operations when utilizing the event manager:

- *Events provide a simple synchronization facility.*
- *Events are aimed at tasks.*
- *Tasks can wait on more than one event simultaneously.*

- *Events are independent of one another.*
- *Events do not hold or transport data.*
- *Events are not queued. In other words, if an event is sent more than once before being received, the second and subsequent send operations have no effect.*

An event set is **posted** when it is directed (or sent) to a task. A **pending event** is an event that has been posted but not received. An **event condition** is used to specify the events which the task desires to receive and the algorithm which will be used to determine when the request is satisfied. An event condition is **satisfied** based upon one of two algorithms which are selected by the user. The **EVENT_ANY** algorithm states that an event condition is satisfied when at least a single requested event is posted. The **EVENT_ALL** algorithm states that an event condition is satisfied when every requested event is posted.

10.2.2 Building an Event Set or Condition

An event set or condition is built by a bitwise OR of the desired events. The set of valid events is **EVENT_0** through **EVENT_31**. If an event is not explicitly specified in the set or condition, then it is not present. Events are specifically designed to be mutually exclusive, therefore bitwise OR and addition operations are equivalent as long as each event appears exactly once in the event set list.

For example, when sending the event set consisting of **EVENT_6**, **EVENT_15**, and **EVENT_31**, the **event** parameter to the **event_send** directive should be **EVENT_6 | EVENT_15 | EVENT_31**.

10.2.3 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 **event_receive** directive are listed in the following table:

CONSTANT	DESCRIPTION	DEFAULT
WAIT	task will wait for event	*
NO_WAIT	task should not wait	
EVENT_ALL	return after all events	*
EVENT_ANY	return after any events	

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 all events in a particular event condition to arrive. The **option** parameter passed to the **event_receive** directive should be either **EVENT_ALL | NO_WAIT** or **NO_WAIT**. The **option** parameter can be set to **NO_WAIT** because **EVENT_ALL** is the default condition for **event_receive**.

10.3 Operations

10.3.1 Sending an Event Set

The **event_send** directive allows a task (or an ISR) to direct an event set to a target task. Based upon the state of the target task, one of the following situations applies:

- *Target Task is Blocked Waiting for Events*
  - If the waiting task's input event condition is satisfied, then the task is made ready for execution.
  - If the waiting task's input event condition is not satisfied, then the event set is posted but left pending and the task remains blocked.
- *Target Task is Not Waiting for Events*
  - The event set is posted and left pending.

10.3.2 Receiving an Event Set

The **event_receive** directive is used by tasks to accept a specific input event condition. The task also specifies whether the request is satisfied when all requested events are available or any single requested event is available. If the requested event condition is satisfied by pending events, then a successful return code and the satisfying event set are returned immediately. If the condition is not satisfied, then one of the following situations applies:

- *By default, the calling task will wait forever for the event condition to be satisfied.*
- *Specifying the **NO_WAIT** option forces an immediate return with an error status code.*

- *Specifying a timeout limits the period the task will wait before returning with an error status code.*

10.3.3 Determining the Pending Event Set

A task can determine the pending event set by calling the **event_receive** directive with a value of **PENDING_EVENTS** for the input event condition. The pending events are returned to the calling task but the event set is left unaltered.

10.3.4 Receiving all Pending Events

A task can receive all of the currently pending events by calling the **event_receive** directive with a value of **ALL_EVENTS** for the input event condition and **NO_WAIT | EVENT_ANY** for the option set. The pending events are returned to the calling task and the event set is cleared. If no events are pending then the **UNSATISFIED** status code will be returned.

10.4 Directives

This section details the event 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.

10.4.1 EVENT_SEND - Send event set to a task

CALLING SEQUENCE:

```
rtems_status_code rtems_event_send (
  rtems_id         id,
  rtems_event_set  event_in
);
```

DIRECTIVE STATUS CODES:

SUCCESSFUL	event set sent successfully
INVALID_ID	invalid task id

DESCRIPTION:

This directive sends an event set, **event_in**, to the task specified by **id**. If a blocked task's input event condition is satisfied by this directive, then it will be made ready. If its input event condition is not satisfied, then the events satisfied are updated and the events not satisfied are left pending. If the task specified by **id** is not blocked waiting for events, then the events sent are left pending.

NOTES:

Specifying **SELF** for **id** results in the event set being sent to the calling task.

Identical events sent to a task are not queued. In other words, the second, and subsequent, posting of an event to a task before it can perform an **event_receive** has no effect.

The calling task will be preempted if it has preemption enabled and a higher priority task is unblocked as the result of this directive.

Sending an event set to a global task which does not reside on the local node will generate a request telling the remote node to send the event set to the appropriate task.

10.4.2 EVENT_RECEIVE - Receive event condition

CALLING SEQUENCE:

```
rtems_status_code rtems_event_receive (
  rtems_event_set  event_in,
  rtems_option     option_set,
  rtems_interval   ticks,
  rtems_event_set *event_out
);
```

DIRECTIVE STATUS CODES:

SUCCESSFUL	event received successfully
UNSATISFIED	input event not satisfied (**NO_WAIT**)
TIMEOUT	timed out waiting for event

DESCRIPTION:

This directive attempts to receive the event condition specified in **event_in**. If **event_in** is set to **PENDING_EVENTS**, then the current pending events are returned in **event_out** and left pending. The **WAIT** and **NO_WAIT** options in the **option_set** parameter are used to specify whether or not the task is willing to wait for the event condition to be satisfied. **EVENT_ANY** and **EVENT_ALL** are used in the **option_set** parameter are used to specify whether a single event or the complete event set is necessary to satisfy the event condition. The **event_out** parameter is returned to the calling task with the value that corresponds to the events in **event_in** that were satisfied.

If pending events satisfy the event condition, then **event_out** is set to the satisfied events and the pending events in the event condition are cleared. If the event condition is not satisfied and **NO_WAIT** is specified, then **event_out** is set to the currently satisfied events. If the calling task chooses to wait, then it will block waiting for the event condition.

If the calling task must wait for the event condition to be satisfied, then the **timeout** parameter is used to specify the maximum interval to wait. If it is set to **NO_TIMEOUT**, then the calling task will wait forever.

NOTES:

This directive only affects the events specified in **event_in**. Any pending events that do not correspond to any of the events specified in **event_in** will be left pending.

The following event receive option constants are defined by **RTEMS**:

CONSTANT	DESCRIPTION	DEFAULT
WAIT	**task will wait for event**	*
NO_WAIT	**task should not wait**	
EVENT_ALL	**return after all events**	*
EVENT_ANY	**return after any events**	