

Queues

RTOS Queues

There are standard queues, or `<vector>` in C++, but RTOS queues should almost always be used in your application because they are thread-safe (no race conditions with multiple tasks), and they co-operate with your RTOS to schedule the tasks. For instance, your task could optionally sleep while receiving data if the queue is empty, or it can sleep while sending the data if the queue is full.

Queues vs. Semaphore for "Signalling"

Semaphores may be used to "signal" between two contexts (tasks or interrupts), but they do not contain any payload. For example, for an application that captures a keystroke inside of an interrupt, it could "signal" the data processing task to awake upon the semaphore, however, there is no payload associated with it to identify what keystroke was input. With an RTOS queue, the data processing task can wake upon a payload and process a particular keystroke.

The data-gathering tasks can simply send the key-press detected to the queue, and the processing task can receive items from the queue, and perform the corresponding action. Moreover, if there are no items in the queue, the consumer task (the processing one) can sleep until data becomes available. You can see how this scheme lends itself well to having multiple ISRs queue up data for a task (or multiple tasks) to handle.

Example Queue usage for Tasks

After looking through the sample code below, you should then [watch this video](#).

Let's study an example of two tasks communicating to each other over a queue.

```
QueueHandle_t q;  
void producer(void *p)  
{  
    int x = 0;  
  
    while (1) {  
        vTaskDelay(100);  
    }  
}
```

```

    xQueueSend(q, &x, 0); // TODO: Find out the significance of the parameters of xQueueSend()
    ++x;
}
}
void consumer(void *p)
{
    while (1) {
        // We do not need vTaskDelay() because this task will sleep for up to 100ms until there is an item
        if (xQueueReceive(q, &x, 100)) {
            printf("Got %i\n", x);
        }
        else {
            puts("Timeout --> No data received");
        }
    }
}
void main(void)
{
    // Queue handle is not valid until you create it
    q = xQueueCreate(10, sizeof(int));
}

```

Example Queue usage with Interrupts

```

// Queue API is special if you are inside an ISR
void uart_rx_isr(void)
{
    xQueueSendFromISR(q, &x, NULL); // TODO: Find out the significance of the parameters
}
void queue_rx_task(void *p)
{
    int x;
    // Receive is the usual receive because we are not inside an ISR
    while (1) {

```

```
xQueueReceive(q, &x, portMAX_DELAY);  
}  
}
```

Additional Information

[Queue Management \(Amazon Docs\)](#)

[Queue API \(FreeRTOS Docs\)](#)

Revision #1

Created 5 years ago by [Khalil Estell](#)

Updated 5 years ago by [Khalil Estell](#)