

# Queues

In this assignment, we will learn about RTOS queues. We will setup three tasks:

1. Light sensor task
2. Temperature sensor task
3. File writer task

This is a classic "Many-to-One" design pattern in RTOS. In this scenario, the **File Writer** acts as a centralized consumer, while the sensors act as producers. This prevents "resource contention" where two tasks try to write to the same file or serial port at the same time.

## The Assignment Objective

1. **Create a Queue** capable of holding a custom `struct`.
  2. **Task 1 & 2 (Producers)**: Periodically read a (mock) sensor and "Send" the data to the queue.
  3. **Task 3 (Consumer)**: Block (sleep) until data arrives in the queue, then "Receive" and print it.
- 

## Template

### Step 1: Define the Data Structure

Since the File Writer needs to know *which* sensor sent the data, we use a structure. You could be a bit more fancy and use a "union" for the data structure, but we leave this as an optional exercise for you.

```
typedef enum {
    SOURCE_LIGHT,
    SOURCE_TEMP
} SensorSource_t;

typedef struct {
```

```
    SensorSource_t source;    int32_t
value;    uint32_t
timestamp;
} SensorData_t;
// Global handle for the queue
QueueHandle_t xSensorQueue;
```

---

## Step 2: The Task Templates

### Producer Template (Light & Temp Tasks)

Both sensor tasks will look very similar. You should implement logic to "mock" a sensor reading (e.g., a random number or a counter).

```

void vSensorTask(void *pvParameters)
{
    SensorSource_t mySource = (SensorSource_t)pvParameters;
    SensorData_t xMessage;
    for
(;;) {          // 1. MOCK: Generate a fake sensor value
        xMessage.source = mySource;          xMessage.value = (mySource == SOURCE_LIGHT) ? 450
: 22
;

        xMessage.timestamp = xTaskGetTickCount();
        // 2. SEND: Post to the queue
        // Use a 0ms block time if you don't want to wait if the queue is full
        if (xQueueSend(xSensorQueue, &xMessage, 0
) != pdPASS) {          // Handle queue full error here
            }
            vTaskDelay(pdMS_TO_TICKS(1000)); // Poll every 1 second
        }
}

```

## Consumer Template (File Writer Task)

This task should be the most efficient. It stays in a **Blocked** state until the queue has data, meaning it consumes 0% CPU while waiting.

```

void vFileWriterTask(void *pvParameters)
{
    SensorData_t xReceivedData;
    for
(;;) {          // 3. RECEIVE: Wait indefinitely (portMAX_DELAY) for data
        if
(xQueueReceive(xSensorQueue, &xReceivedData, portMAX_DELAY) == pdPASS) {
            // 4. LOG: Check source and "write" to console/file
            const char* label = (xReceivedData.source == SOURCE_LIGHT) ? "LIGHT" : "TEMP"

```

```

;         printf("[%u] %s Reading: %d\n"
, xReceivedData.timestamp, label, xReceivedData.value);
    }
}
}

```

## Step 3: Main Initialization

The most important part of the assignment is ensuring the queue is created **before** the tasks start.

```

int main(void)
{
    // Initialize the queue: (Length of 10 items, size of our struct)
    xSensorQueue = xQueueCreate(10, sizeof
(SensorData_t));
    if (xSensorQueue != NULL
) {
        // Create Producers
        xTaskCreate(vSensorTask, "Light", 1000, (void*)SOURCE_LIGHT, 1, NULL
);
        xTaskCreate(vSensorTask, "Temp", 1000, (void*)SOURCE_TEMP, 1, NULL
);

        // Create Consumer (Give it a slightly higher priority if data is high-volume)
        xTaskCreate(vFileWriterTask, "Writer", 1000, NULL, 2, NULL
);

        vTaskStartScheduler();
    }
    for (;;) // Should never reach here
}

```

## Assignment Challenges:

- **Buffer Overflow:** What happens if the `vSensorTask` delays for 100ms but the `vFileWriterTask` takes 500ms to process? (Students should observe the queue filling up).
- **Priority Inversion:** Change the priorities so the producers are higher than the consumer. Observe how the

queue behaves.

---

Revision #1

Created 2 months ago by [Preet Kang](#)

Updated 2 months ago by [Preet Kang](#)