Note: This mailbox pattern does not exactly match the FreeRTOS guide. I modified it for my own needs.

Queue-Mailbox Pattern

Master/Slave Relationship

```
QueueHandle_t xQue_MSTCmdRequestQue = xQueueCreate(1, sizeof(SLV_Response *));
SLV_Request *ptrSLVRequest = new Slave_CmdRequest();
SLV_Response *ptrSLVResponse = new Slave_Response();
```

```
xQueueHandle xQue_SLVCmdRequests = xQueueCreate(1, sizeof(Slave_CmdRequest *));
SLV_CmdRequest *ptrSLVCmdRequest = new Slave_CmdRequest;
SLV_Response *ptrSLVResponse = new Slave_Response;
```

Pointer to Slave's Queue / sending Slave_Command

```
ptrSlaveCmdRequest->QueueToSendResponse = xQue_MSTCmdRequestQue
ptrSlaveCmdRequest->Cmd = SLV_COMMAND::Cmd1;
xQueueSend(xQue_SLVCmdRequests, &ptrSLVCmdRequest, 0);
```

SLV Cmd Request Queue

Pointer to Master's Queue / sending Slave_Response

```
xQueueReceive(xQue_SLVCmdRequests, &ptrSLVCmdRequest, pdMS_TO_TICKS(300)
< Act on ptrSLVCmdRequest->Cmd >
ptrSLVResponse->ResponseCode = SLV_STATUS::Ok;
ptrSLVResponse->JsonResponse = new std::string(Data);  // Populate response data
xQueueSend(ptrSLVCmdRequest->QueueToSendResponse, &ptrSLVResponse, portMAX_Delay);
```

Cmd Request Queue
I may be someone else's Slave

SLV Response Queue

```
enum class SLV_COMMAND : uint8_t
{
    None,
    Cmd1,
    Cmd2
};
```

```
enum class SLV_STATUS : uint8_t
{
    OK,
    ERROR
};
```

```
struct Slave_CmdRequest
{
    QueueHandle_t QueueToSendResponse,
    SLV_COMMAND Cmd
};
```

```
struct Slave_Response
{
    SLV_STATUS ResponseCode,
    std::string *JsonResponse
};
```

Using a Queue as a mailbox significantly reduces the cost of memory, but at the expense of some time as 2 objects must coordinate when a command is sent and the response is generated. Typically the master will need to wait for a response. When the master doesn't need a response AND the command data structure is small, a normal queue can be created at the Slave.

Note: All the data structures are defined in terms of the Slave for consistency. The pattern requires that Master have a separate Response Queues for each and every slave who responds to a command because the returning data is formed to match the Slave's needs.

Notice that there are matching Request/Response structures on both sides. RTOS copies data between the two for you. Care must be taken when working with dynamic data.

Master Calling Object

Slave Responding Object