Embedded JSON Serialization Module for Smart Meter Gateway

1. Introduction

Smart meter gateways running on STM32 microcontrollers often need to forward meter readings to backend systems in a structured format. This document presents a lightweight JSON serialization module designed specifically for STM32-based embedded firmware. The focus is on deterministic memory usage and strict adherence to a predefined JSON structure.

2. Design Assumptions

• The firmware runs on an STM32 microcontroller using STM32CubeIDE • The number of connected meters is limited and known at compile time • Dynamic memory allocation is avoided • The JSON structure is fixed and must not be modified

3. Platform and Language

Target Platform: STM32 microcontroller (STM32CubeIDE) Programming Language: C C was selected because it provides full control over memory and execution, which is essential for embedded firmware running on STM32 devices.

4. Internal Data Structures

The following fixed-size data structures are used to represent gateway and meter data:

```
#define MAX_DEVICES 5
#define MAX_DATA_POINTS 5

typedef struct {
    char timestamp[20];
    char meter_datetime[20];
    float total_m3;
    char status[8];
} data_point_t;

typedef struct {
    char media[16];
    char meter[16];
    char deviceId[32];
    char unit[8];
    int data_count;
    data_point_t data[MAX_DATA_POINTS];
} device_reading_t;

typedef struct {
    char gatewayId[32];
    char date[11];
    char deviceType[16];
```

```
    int interval_minutes;
    int total_readings;
    int device_count;
    device_reading_t devices[MAX_DEVICES];
} gateway_data_t;
```

5. Serialization API

The serializer exposes a single transport-independent API:

```
typedef enum {
    JSON_OK = 0,
    JSON_ERROR_BUFFER_TOO_SMALL,
    JSON_ERROR_INVALID_INPUT
} json_status_t;

json_status_t serialize_to_json(
    const gateway_data_t *input,
    char *output,
    size_t output_size
);
```

6. Serialization Logic

JSON generation is implemented manually using snprintf to ensure buffer safety. After each write operation, the remaining buffer size is checked. If the buffer is insufficient, an error code is returned.

7. Example Usage on STM32

```
int main(void) {
    gateway_data_t gw = {
        "gateway_1234",
        "1970-01-01",
        "stromleser",
        15,
        1,
        1
    };

    char json_buffer[512];
    serialize_to_json(&gw, json_buffer, sizeof(json_buffer));

    // JSON can be sent via UART or any STM32 communication interface
}
```

8. Design Decisions

• Fixed-size arrays are used to avoid heap fragmentation • No external JSON libraries are used • Serialization logic is independent of UART, CAN, or Ethernet drivers • Error handling is implemented using return codes

9. Possible Extensions

• Support for multiple gateways • Additional meter types and units • Unit testing within STM32CubeIDE • Integration with STM32 communication peripherals