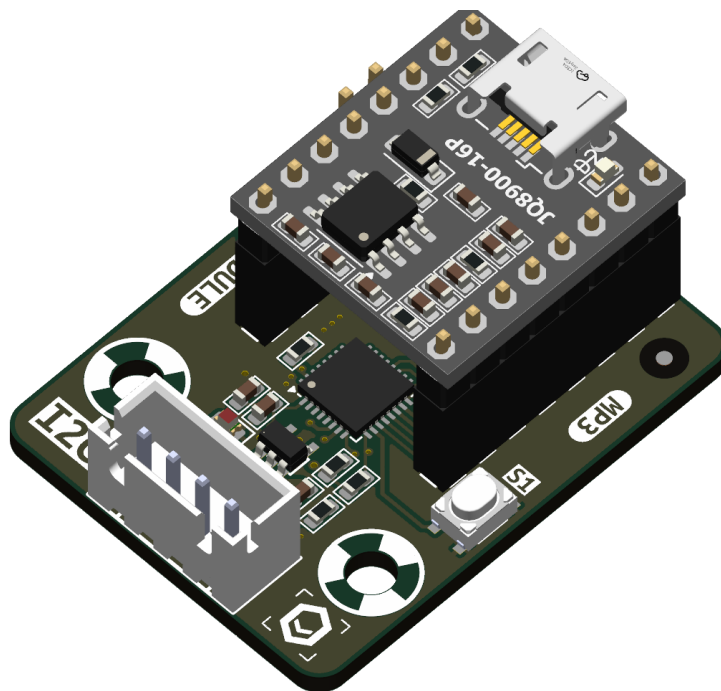
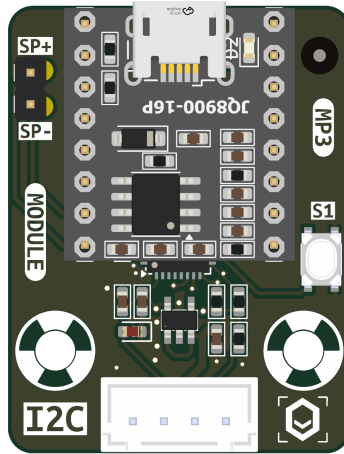


VNEHC I2C MP3 JQ8900 Module User Manual



This document provides instructions on how to use the JQ8900 MP3 Module. It is divided into 2 parts:

- **Part 1:** For students, teachers, and beginners (built-in Flash setup and button operations).
- **Part 2:** For developers (I2C protocol guide for writing Master code).

Part 1: Basic Setup and Usage

1.1. Built-in Flash Layout and MP3 File Naming

This JQ8900 module stores audio files in its built-in 4 MB Flash. Load files using the Flash programming tool/method supplied for the module, and name them correctly so the module can automatically read numbers and play system voice prompts.

Folder Structure:

[!WARNING] **The MF folder is a system area reserved exclusively for the manufacturer.** Normal users must absolutely **NOT DELETE OR MODIFY** files in this area. Store user audio in the Root area or custom numbered folders (01, 02, etc.) when programming Flash.

- **MF** folder: Mandatory system area. Contains system voice prompts and automatic number-reading files.
- **01, 02, ... 99** folders: User audio folders in Flash.
- Root area: MP3 files **must have a 5-digit prefix**. You can add text after these 5 digits (Example: 00001_BackgroundMusic.mp3, 00002_Song.mp3).

System File Naming Rules in the MF folder (For Manufacturer): These files are used by the module to announce statuses during button setup. The file names **must start with 3 digits**. You can add a short description after these 3 digits (Example: 002_Enter_SetupAddress.mp3).

List of mandatory 3-digit prefixes for the system:

- 001_...mp3 - "Factory reset completed. Default address is 50."
- 002_...mp3 - "Entering address setup mode."
- 003_...mp3 - "Address saved!"
- 004_...mp3 - "Canceled."
- 005_...mp3 - "Current address is:"
- 010_...mp3 - Digit "Zero"
- 011_...mp3 - Digit "One"
- ...
- 019_...mp3 - Digit "Nine"

1.2. Button Operation Guide (Setup Mode)

The module features a built-in button that allows you to check information and change the I2C address without writing any code:

1. Check Current Address:

- **Action:** Press once (Click).
- **Response:** The module will announce "Current address is: Five Zero" (if the address is 50).

2. Change I2C Address:

- **Step 1 (Enter Setup):** Press and hold the button for about 3 seconds. As soon as you hear "Entering address setup mode", release the button.
- **Step 2 (Change Address):** Double-click quickly. Each double-click increases the address by 1 (from 50 up to a maximum of 55, then loops back to 50).

- **Step 3 (Save or Cancel):**

- To **SAVE**: Press and hold for another 3 seconds until you hear "Address saved!".
- To **CANCEL**: Press once (Click). The module will announce "Canceled" and revert to the previous address.

3. Factory Reset:

- **Action**: Press and hold the button continuously for about 6 seconds (past the 3-second mark).
- **Response**: The module announces "Factory reset completed" and forces the I2C address back to the default of 50.

Part 2: I2C Protocol Guide (For Developers)

The module communicates with the main microcontroller (Master) via I2C. The default address is **50** (0x32).

2.1. Packet Structure

The I2C protocol uses a standard Header structure: `[AddressId(1 byte)] [ModeId(1 byte)] [Value32(4 bytes)]`

- Master Write command: Send the `ModeId` followed by the `Value` via I2C.
- Master Read data: Send the `ModeId` you want to read, then use the `Wire.requestFrom` command to receive 4 bytes of data.

2.2. Core System Commands

These commands are common across the MakerEdu ecosystem.

Mode ID	Command Name	Function	Value (4 bytes)
1	<code>SetAddress</code>	Change I2C Address (Requires Admin Mode)	New Address (0 - 127)
2	<code>Get_ID_Module</code>	Read module identifier	Returns 5 (MP3 ID)
10	<code>GetAddress</code>	Read current I2C address	Returns current address
200	<code>Set_Admin_Mode</code>	Enable/Disable Admin Mode	Send <code>0xA5A5A5A5</code> to unlock

2.3. MP3 Control Commands

Specific commands from 50 - 63 used for playing and controlling audio.

Mode ID	Command Name	Description	Value (To send)
50	MP3_Play	Resume Playback	0
51	MP3_Pause	Pause Playback	0
52	MP3_Stop	Stop Playback	0
53	MP3_Next	Next Track	0
54	MP3_Prev	Previous Track	0
55	MP3_Set_Volume	Adjust Volume (0 - 30)	Volume level (e.g., 20)
56	MP3_Play_Track	Play track by Index (Root)	File number (e.g., 1, 2)
57	MP3_Get_Status	Read Play/Pause/Stop status	Returns 1 (Play), 2 (Pause), 0 (Stop)
58	MP3_Get_Volume	Read current volume	Returns 0 - 30
59	MP3_Play_Track_in_Folder_Number	Play file from Numbered Folder	Folder * 1000 + File (e.g., Folder 2, File 5 -> 2005)
60	MP3_Play_Track_in_Folder_MF	Play file from "MF" Folder	File number
61	MP3_Play_Insert_Track	Insert/Interrupt track (Ad mode)	File number
62	MP3_Play_Insert_Track_in_Folder_Number	Insert track from Numbered Folder	Folder * 1000 + File
63	MP3_Play_Insert_Track_in_Folder_MF	Insert track from "MF" Folder	File number

Programming Tip: For Get commands (like Status 57, Volume 58), the Master must first send the Mode ID using a Write operation, and then call the read function (RequestFrom 4 bytes).

2.4. Arduino Master Sample Code

Below is a visual example of how the main microcontroller (e.g., Arduino Uno, ESP32) can send commands and read data from the MP3 Module.

```

#include <Wire.h>

#define MP3_I2C_ADDRESS 50 // 0x32 (Default address)

// =====
// I2C COMMUNICATION UTILITY FUNCTIONS
// =====

// Write Function: Send I2C Command (AddressId + ModeId + 32-bit Value)
void sendI2CCommand(uint8_t address, uint8_t modeId, uint32_t value) {
    Wire.beginTransaction(address);

    Wire.write(address); // AddressId (usually matches the I2C address)
    Wire.write(modeId);  // ModeId

    // Send 4 bytes of Value32 (from highest to lowest byte - Big Endian)
    Wire.write((uint8_t)(value >> 24));
    Wire.write((uint8_t)(value >> 16));
    Wire.write((uint8_t)(value >> 8));
    Wire.write((uint8_t)(value & 0xFF));

    Wire.endTransmission();
}

// Read Function: Fetch 32-bit data from the Module
uint32_t readI2CValue(uint8_t address, uint8_t modeId) {
    // Step 1: Send the read request command (ModeId)
    sendI2CCommand(address, modeId, 0);
    delay(10); // Wait for the Slave to process and buffer the data

    // Step 2: Request 4 bytes of data
    Wire.requestFrom(address, (uint8_t)4);
    uint32_t result = 0;
    if (Wire.available() >= 4) {
        result |= ((uint32_t)Wire.read() << 24);
        result |= ((uint32_t)Wire.read() << 16);
        result |= ((uint32_t)Wire.read() << 8);
        result |= ((uint32_t)Wire.read());
    }
    return result;
}

// =====
// MAIN PROGRAM
// =====

void setup() {
    Serial.begin(115200);
    Wire.begin();
    delay(1000); // Wait for the module to boot

    // Example 1: Play track number 2 in the Root folder

```

```
// ModeID = 56 (MP3_Play_Track), Value = 2
Serial.println("Playing track number 2...");
sendI2CCommand(MP3_I2C_ADDRESS, 56, 2);

delay(3000); // Wait 3 seconds

// Example 2: Read current volume level
// ModeID = 58 (MP3_Get_Volume)
uint32_t vol = readI2CValue(MP3_I2C_ADDRESS, 58);
Serial.print("Current Volume is: ");
Serial.println(vol);
}

void loop() {
    // Main loop
}
```