Unit RFID / RFID2 Arduino 使用教程

1.准备工作

- 环境配置：参考 Arduino IDE 上手教程 完成 IDE 安装，并根据实际使用的开发板安装对应的板管理与需要的驱动库。
- 使用到的驱动库：
  - M5Unified
  - M5GFX
  - MFRC522_I2C
- 使用到的硬件产品：
  - Basic v2.7
  - Unit RFID2 或 Unit RFID

2.读取UID示例

编译要求

M5Stack 板管理版本 >= 3.2.2

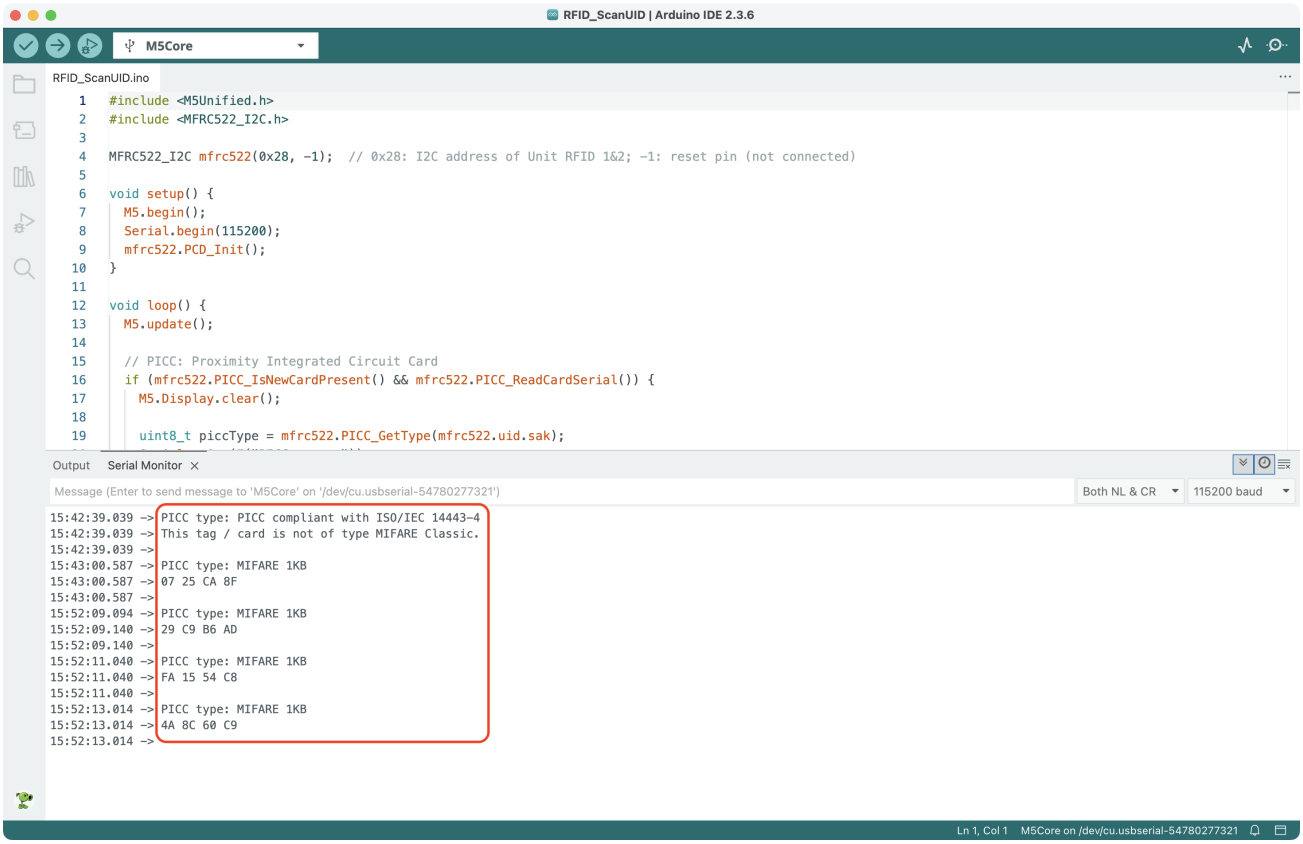
M5Unified 库版本 >= 0.2.8

M5GFX 库版本 >= 0.2.11

MFRC522_I2C 库版本 >= 1.0

cpp

```cpp
#include <M5Unified.h>
#include <MFRC522_I2C.h>

MFRC522_I2C mfrc522(0x28, -1);  // 0x28: I2C address of Unit RFID / RFID2; -1: reset pin (not connec

void setup() {
  M5.begin();
  Serial.begin(115200);
  mfrc522.PCD_Init();
}

void loop() {
  M5.update();

  // PICC: Proximity Integrated Circuit Card
  if (mfrc522.PICC_IsNewCardPresent() && mfrc522.PICC_ReadCardSerial()) {
    M5.Display.clear();

    uint8_t piccType = mfrc522.PICC_GetType(mfrc522.uid.sak);
    Serial.print(F("PICC type: "));
    Serial.println(mfrc522.PICC_GetTypeName(piccType));

    // Check if the tag / card is of type MIFARE Classic
    if (piccType != MFRC522_I2C::PICC_TYPE_MIFARE_MINI
        && piccType != MFRC522_I2C::PICC_TYPE_MIFARE_1K
        && piccType != MFRC522_I2C::PICC_TYPE_MIFARE_4K) {
      Serial.println(F("This tag / card is not of type MIFARE Classic.\n"));
      delay(500);
      return;
    }

    // Output the stored UID data
    for (byte i = 0; i < mfrc522.uid.size; i++) {
      Serial.printf("%02X ", mfrc522.uid.uidByte[i]);
    }
    Serial.println("\n");
    delay(500);
  }
}
```



3.读写卡示例

编译要求

M5Stack 板管理版本 >= 3.2.2

M5Unified 库版本 >= 0.2.8

M5GFX 库版本 >= 0.2.11

MFRC522_I2C 库版本 >= 1.0

cpp

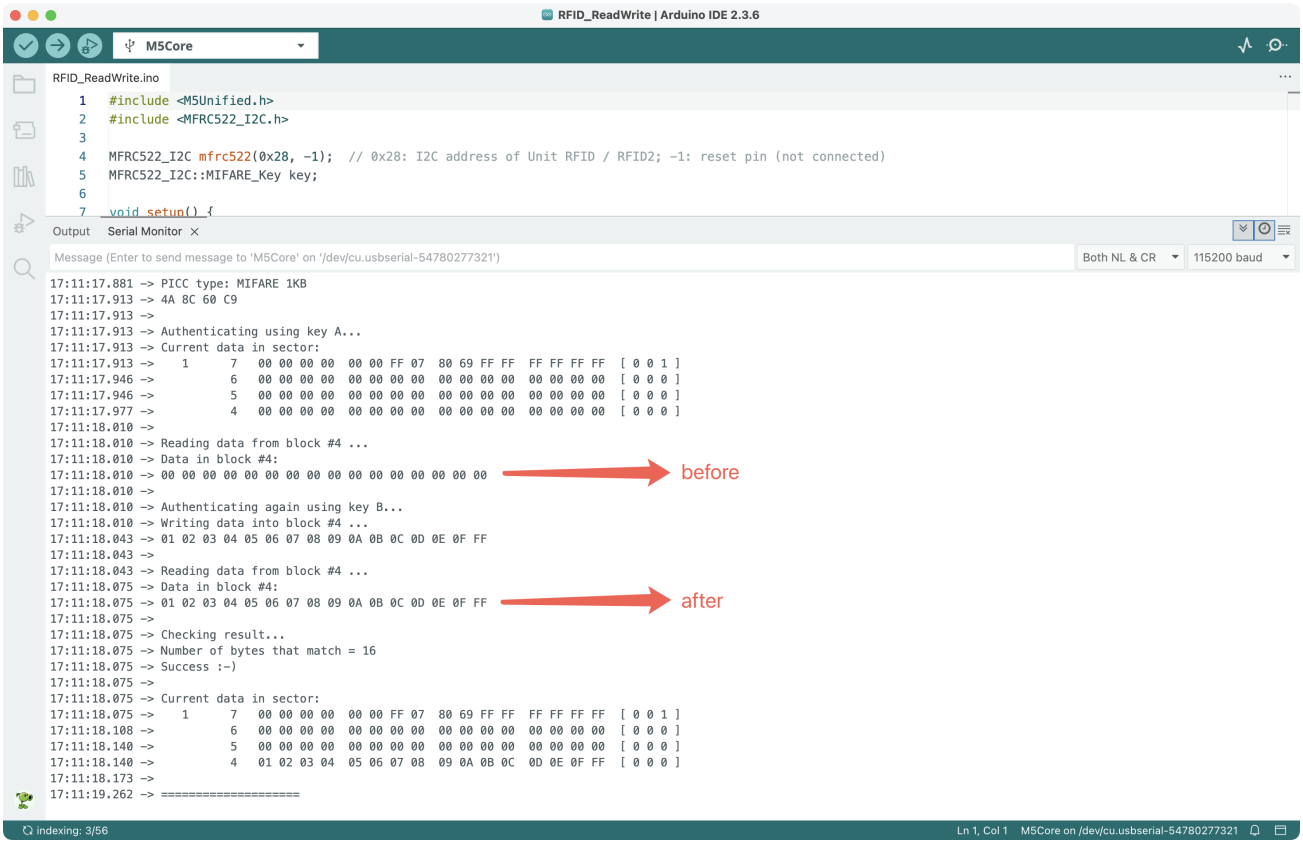
```cpp
#include <M5Unified.h>
#include <MFRC522_I2C.h>

MFRC522_I2C mfrc522(0x28, -1);  // 0x28: I2C address of Unit RFID / RFID2; -1: reset pin (not conne
MFRC522_I2C::MIFARE_Key key;

void setup() {
  M5.begin();
  Serial.begin(115200);
  mfrc522.PCD_Init();

  // Prepare the key (used both as key A and as key B) with FFFFFFFFFFFFh,
  // which is the default at chip delivery from the factory.
  for (byte i = 0; i < 6; i++) {
    key.keyByte[i] = 0xFF;
  }
}

// Helper routine to dump a byte array as hex values to Serial.
void dump_byte_array(byte *buffer, byte bufferSize) {
  for (byte i = 0; i < bufferSize; i++) {
    Serial.printf("%02X ", buffer[i]);
  }
}

void loop() {
  M5.update();

  // PICC: Proximity Integrated Circuit Card
  if (mfrc522.PICC_IsNewCardPresent() && mfrc522.PICC_ReadCardSerial()) {
    M5.Display.clear();

    uint8_t piccType = mfrc522.PICC_GetType(mfrc522.uid.sak);
    Serial.print(F("PICC type: "));
    Serial.println(mfrc522.PICC_GetTypeName(piccType));

    // Check if the tag / card is of type MIFARE Classic
    if (piccType != MFRC522_I2C::PICC_TYPE_MIFARE_MINI
        && piccType != MFRC522_I2C::PICC_TYPE_MIFARE_1K
        && piccType != MFRC522_I2C::PICC_TYPE_MIFARE_4K) {
      Serial.println(F("This tag / card is not of type MIFARE Classic.\n"));
      delay(500);
      return;
    }

    // Output the stored UID data
    String uid = "";
    for (byte i = 0; i < mfrc522.uid.size; i++) {
      Serial.printf("%02X ", mfrc522.uid.uidByte[i]);
      uid += String(mfrc522.uid.uidByte[i], HEX);
```

```
    }
    Serial.println("\n");

    // mfrc522.PICC_DumpToSerial(&(mfrc522.uid));
    // Serial.println("\n");

    // In this example, we use the second sector (sector #1) including blocks #4, #5, #6, #7
    byte sector = 1;
    byte blockAddr = 4;
    byte trailerBlock = 7;
    byte dataBlock[] = {
      0x01, 0x02, 0x03, 0x04,  //  1,  2,  3,   4,
      0x05, 0x06, 0x07, 0x08,  //  5,  6,  7,   8,
      0x09, 0x0a, 0x0b, 0x0c,  //  9, 10, 11,  12,
      0x0d, 0x0e, 0x0f, 0xff   // 13, 14, 15, 255
    };
    MFRC522_I2C::StatusCode status;
    byte buffer[18];
    byte size = sizeof(buffer);

    // Authenticate using key A
    Serial.println(F("Authenticating using key A..."));
    status = (MFRC522_I2C::StatusCode)mfrc522.PCD_Authenticate(MFRC522_I2C::PICC_CMD_MF_AUTH_KEY_A,
    if (status != MFRC522_I2C::STATUS_OK) {
      Serial.print(F("PCD_Authenticate() failed: "));
      Serial.println(mfrc522.GetStatusCodeName(status));
      return;
    }

    // Show the whole sector as it currently is
    Serial.println(F("Current data in sector: "));
    mfrc522.PICC_DumpMifareClassicSectorToSerial(&(mfrc522.uid), &key, sector);
    Serial.println("");

    // Read data from the block
    Serial.print(F("Reading data from block #"));
    Serial.print(blockAddr);
    Serial.println(F(" ..."));
    status = (MFRC522_I2C::StatusCode)mfrc522.MIFARE_Read(blockAddr, buffer, &size);
    if (status != MFRC522_I2C::STATUS_OK) {
      Serial.print(F("MIFARE_Read() failed: "));
      Serial.println(mfrc522.GetStatusCodeName(status));
    }
    Serial.print(F("Data in block #"));
    Serial.print(blockAddr);
    Serial.println(F(": "));
    dump_byte_array(buffer, 16);
    Serial.println("\n");

    // Authenticate using key B
    Serial.println(F("Authenticating again using key B..."));
    status = (MFRC522_I2C::StatusCode)mfrc522.PCD_Authenticate(MFRC522_I2C::PICC_CMD_MF_AUTH_KEY_B,
    if (status != MFRC522_I2C::STATUS_OK) {
```

```
      Serial.print(F("PCD_Authenticate() failed: "));
      Serial.println(mfrc522.GetStatusCodeName(status));
      return;
    }

    // Write data to the block
    Serial.print(F("Writing data into block #"));
    Serial.print(blockAddr);
    Serial.println(F(" ..."));
    dump_byte_array(dataBlock, 16);
    Serial.println();
    status = (MFRC522_I2C::StatusCode)mfrc522.MIFARE_Write(blockAddr, dataBlock, 16);
    if (status != MFRC522_I2C::STATUS_OK) {
      Serial.print(F("MIFARE_Write() failed: "));
      Serial.println(mfrc522.GetStatusCodeName(status));
    }
    Serial.println("");

    // Read data from the block again, now it should be what we have written
    Serial.print(F("Reading data from block #"));
    Serial.print(blockAddr);
    Serial.println(F(" ..."));
    status = (MFRC522_I2C::StatusCode)mfrc522.MIFARE_Read(blockAddr, buffer, &size);
    if (status != MFRC522_I2C::STATUS_OK) {
      Serial.print(F("MIFARE_Read() failed: "));
      Serial.println(mfrc522.GetStatusCodeName(status));
    }
    Serial.print(F("Data in block #"));
    Serial.print(blockAddr);
    Serial.println(F(": "));
    dump_byte_array(buffer, 16);
    Serial.println("\n");

    // Check if the data in block is what we have written, by counting the number of bytes that are
    Serial.println(F("Checking result..."));
    byte count = 0;
    for (byte i = 0; i < 16; i++) {
      // Compare buffer (what we've read) with dataBlock (what we've written)
      if (buffer[i] == dataBlock[i]) {
        count++;
      }
    }
    Serial.print(F("Number of bytes that match = "));
    Serial.println(count);
    if (count == 16) {
      Serial.println(F("Success :-)"));
    } else {
      Serial.println(F("Failure, no match :-("));
      Serial.println(F("  perhaps the write didn't work properly..."));
    }
    Serial.println();

    // Dump the sector data
```

```
    Serial.println(F("Current data in sector: "));
    mfrc522.PICC_DumpMifareClassicSectorToSerial(&(mfrc522.uid), &key, sector);
    Serial.println("");

    // Halt PICC
    mfrc522.PICC_HaltA();
    // Stop encryption on PCD
    mfrc522.PCD_StopCrypto1();

    Serial.println("======================");
  }
}
```

这段程序会在完成 Key A 认证后读取 Sector 1 中的 Block 4 的数据，然后在完成 Key B 认证后修改 Sector 1 中的 Block 4 的数据。在这个过程中标签 / 卡片需要保持靠近 Unit RFID2。程序输出如下：



4.Init API

以下为常用 API 的使用说明。一般读写 MIFARE 卡的流程为：

1. 初始化 RFID
2. 检测新卡存在，获取卡 UID
3. 通过 UID 选中卡片，进入 active 状态
4. 通过 Key A、Key B 解锁对应的 Block
5. 读 / 写数据
6. 控制卡进入休眠状态

操作返回值状态码

cpp

```cpp
enum StatusCode {
    STATUS_OK             = 1,  // Success
    STATUS_ERROR          = 2,  // Error in communication
    STATUS_COLLISION      = 3,  // Collision detected
    STATUS_TIMEOUT        = 4,  // Timeout in communication
    STATUS_NO_ROOM        = 5,  // The buffer is not big enough
    STATUS_INTERNAL_ERROR = 6,  // Internal error in the code. Should not happen ;-)
    STATUS_INVALID        = 7,  // Invalid argument
    STATUS_CRC_WRONG      = 8,  // The CRC_A does not match
    STATUS_MIFARE_NACK    = 9   // The MIFARE PICC responded with NAK
};
```

实例化

函数原型：

```cpp
MFRC522_I2C(byte chipAddress, byte resetPowerDownPin);
```

功能说明：

- 创建一个 RFID 实例

传入参数：

- byte chipAddress
  - 模块的 I2C 地址（0x28）
- byte resetPowerDownPin
  - 主控连接到模块的复位引脚，由于未连接所以设置为 -1

返回值：

- MFRC522_I2C 类的对象

PCD_Init

函数原型：

```cpp
void PCD_Init();
```

功能说明：

- 初始化读写器

传入参数：

- null

返回值：

- null

PICC_IsNewCardPresent

函数原型:

```
bool PICC_IsNewCardPresent();
```

功能说明:

- 扫描是否存在未检测过且处于IDLE状态的卡片，处于HALT状态的卡片将被忽略。

传入参数:

- null

返回值:

- bool
  - true: 扫描到了新卡
  - false: 未扫描到新卡

PICC_ReadCardSerial

函数原型:

```
bool PICC_ReadCardSerial();
```

功能说明:

- 读取卡 UID，读取成功后的 UID 可以在类成员Uid uid;中读取。读取操作前需执行PICC_IsNewCardPresent()或PICC_RequestA()或PICC_WakeupA()确保读取到卡片。

cpp

```cpp
for (byte i = 0; i < mfrc522.uid.size; i++) {
  Serial.printf("%02X ", mfrc522.uid.uidByte[i]);
}
```

传入参数:

- null

返回值:

- bool
  - true: 读取成功
  - false: 读取失败

PICC_RequestA

函数原型:

```
uint8_t PICC_RequestA(uint8_t *bufferATQA, uint8_t *bufferSize);
```

功能说明:

- 扫描检测读取范围内的 Type A 标准的卡片

传入参数:

- uint8_t *bufferATQA
  - 存储请求响应 ATQA (Answer to request) 的 buffer
- uint8_t *bufferSize
  - buffer 长度 (>2 byte)

返回值:

- uint8_t
  - StatusCode

PICC_WakeupA

函数原型:

```
uint8_t PICC_WakeupA(uint8_t *bufferATQA, uint8_t *bufferSize);
```

功能说明:

- 唤醒范围内的 Type A 标准的卡片

传入参数:

- uint8_t *bufferATQA
  - 存储请求响应 ATQA (Answer to request) 的 buffer
- uint8_t *bufferSize
  - buffer 长度 (>2 byte)

返回值:

- uint8_t
  - StatusCode

PICC_Select

函数原型:

```
uint8_t PICC_Select(Uid *uid, uint8_t validBits = 0);
```

功能说明:

- 通过 uid 选中一张卡片进入 active 状态

传入参数:

- Uid *uid
  - 通过扫描获取到的卡片 uid 结构体指针
- uint8_t validBits
  - 最后一个 uint8_t 中的有效位，0 表示 8 个有效位

返回值:

- uint8_t
  - StatusCode

PICC_HaltA

函数原型:

```
uint8_t PICC_HaltA();
```

功能说明:

- 选中当前卡片进入休眠状态

传入参数:

- null

返回值:

- uint8_t
  - StatusCode

5.MIFARE API

PCD_Authenticate

函数原型:

```
uint8_t PCD_Authenticate(uint8_t command, uint8_t blockAddr, MIFARE_Key *key, Uid *uid);
```

功能说明:

- 进行 MIFARE 身份验证。在调用此函数之前，需要对卡片进行 select 操作使其处于 active 状态。经过身份验证的 PICC 通信结束后需调用 `PCD_StopCrypto1()`，否则无法开始新的通信。

传入参数:

- uint8_t command
  - PICC_CMD_MF_AUTH_KEY_A
  - PICC_CMD_MF_AUTH_KEY_B
- uint8_t blockAddr
  - Block 地址
- MIFARE_Key *key
  - 默认状态下，Key A、Key B 均为`FFFFFFFFFFFF`
- Uid *uid

返回值:

- uint8_t
  - StatusCode

PCD_StopCrypto1

函数原型:

```
void PCD_StopCrypto1();
```

功能说明:

- 退出 PCD 的认证状态。经过身份验证的 PICC 通信结束后需调用`PCD_StopCrypto1()`，否则无法开始新的通信。

传入参数:

- null

返回值:

- null

MIFARE_Read

函数原型:

```
uint8_t MIFARE_Read(uint8_t blockAddr, uint8_t *buffer, uint8_t *bufferSize);
```

功能说明:

- 从指定 blockAddr 读取数据

传入参数:

- uint8_t blockAddr
  - 实际卡片扇区中的 block 地址
- uint8_t *buffer
  - 接收数据的 buffer 指针
- uint8_t *bufferSize
  - 接收数据的 buffer 长度 (>=18 byte)

返回值:

- uint8_t
  - StatusCode

MIFARE_Write

函数原型:

```
uint8_t MIFARE_Write(uint8_t blockAddr, uint8_t *buffer, uint8_t bufferSize);
```

功能说明:

- 向指定 blockAddr 写入数据

传入参数:

- uint8_t blockAddr
  - 实际卡片扇区中的 block 地址
- uint8_t *buffer
  - 写入数据的 buffer 指针
- uint8_t *bufferSize
  - 写入数据的 buffer 长度 (16 byte)

返回值:

- uint8_t
  - StatusCode

6.参考链接

- 更多相关的 API 可以参考 MFRC522_I2C 库
- ISO/IEC 14443 - Wikipedia
- MFRC522 - NXP Docs (Standard performance MIFARE and NTAG frontend)
- MFRC523 - NXP Docs (Standard performance ISO/IEC 14443 A/B frontend)
- MIFARE Classic EV1 1K - NXP Docs （中文版）
- MIFARE Classic EV1 4K - NXP Docs
- AN1305 - NXP Docs (MIFARE Classic as NFC Type MIFARE Classic Tag)
- AN10833 - NXP Docs (MIFARE type identification procedure)
- AN10834 - NXP Docs (MIFARE ISO/IEC 14443 PICC selection)