Sistema de seguridad en cámaras frigoríficas

Generado por Doxygen 1.12.0

Capítulo 1

Índice jerárquico

1.1. Jerarquía de clases

Este listado de herencia está ordenado de forma general pero no está en orden alfabético estricto:

Capítulo 2

Índice de estructuras de datos

2.1. Estructuras de datos

Lista de estructuras con breves descripciones:

Capítulo 3

Índice de archivos

3.1. Lista de archivos

Lista de todos los archivos documentados y con breves descripciones:

Capítulo 4

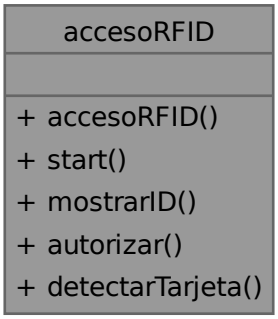
Documentación de estructuras de datos

4.1. Referencia de la clase accesoRFID

Clase para manejar el acceso usando el módulo RFID-RC522.

`#include <accesoRFID.h>`

Diagrama de colaboración de accesoRFID:



Métodos públicos

- accesoRFID (byte ssPin, byte rstPin, String autorizarID)

  Constructor de la clase accesoRFID.
- void **start** ()

  void start(); Inicia la comunicación SPI con el módulo RFID (usando sus métodos).
- void **mostrarID** ()

  void mostrarID(); Imprime por puerto serie el UID de la tarjeta detectada por el RFID
- bool **autorizar** ()

  bool autorizar(); Compara el UID de la tarjeta con el UID autorizado. Devuelve true (acceso permitido) o un false (acceso denegado).
- bool **detectarTarjeta** ()

  bool detectarTarjeta(); Lee el UID de la tarjeta para implementarla en el sistema.

4.1.1. Descripción detallada

Clase para manejar el acceso usando el módulo RFID-RC522.

Esta clase permite inicializar el lector RFID y verificar si una tarjeta tiene autorización para entrar.

4.1.2. Documentación de constructores y destructores

4.1.2.1. accesoRFID()

```
accesoRFID::accesoRFID (
            byte ssPin,
            byte rstPin,
            String autorizarID)
```

Constructor de la clase accesoRFID.

Parámetros

ssPin	El pin SS (Slave Select) para la comunicación SPI.
rstPin	es el pin de Reset para el módulo RFID.
autorizarID	El UID de la tarjeta autorizada para el acceso.

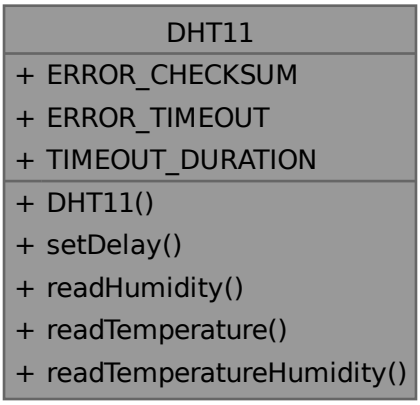
La documentación de esta clase está generada del siguiente archivo:

- main/libraries/RFID-RC522/accesoRFID.h

4.2. Referencia de la clase DHT11

```
#include <DHT11.h>
```

Diagrama de colaboración de DHT11:



Métodos públicos

- DHT11 (int pin)
- void setDelay (unsigned long delay)
- int readHumidity ()
- int readTemperature ()
- int readTemperatureHumidity (int &temperature, int &humidity)

Atributos públicos estáticos

- static const int **ERROR_CHECKSUM** = 254
- static const int **ERROR_TIMEOUT** = 253
- static const int **TIMEOUT_DURATION** = 1000

Amigas

- void setPinDHT11 (DHT11 &sensor, int pin)

4.2.1. Descripción detallada

DHT11.h Header file for the DHT11 library, providing functionalities to interface with the DHT11 temperature & humidity sensor.

Author: Dhruba Saha Version: 2.1.0 License: MIT DHT11 Class Provides methods to read temperature and humidity data from the DHT11 sensor.

4.2.2. Documentación de constructores y destructores

4.2.2.1. DHT11()

```
DHT11::DHT11 (
            int pin)
```

Constructor Initializes the data pin to be used for communication with the DHT11 sensor.

Parámetros

pin	Digital pin number on the Arduino board to which the DHT11 sensor is connected.

4.2.3. Documentación de funciones miembro

4.2.3.1. readHumidity()

```
int DHT11::readHumidity ()
```

Reads and returns the humidity from the DHT11 sensor.

Devuelve

: Humidity value in percentage. Returns DHT11_ERROR_TIMEOUT if reading times out. Returns DHT11_↩ ERROR_CHECKSUM if checksum validation fails.

4.2.3.2. readTemperature()

```
int DHT11::readTemperature ()
```

Reads and returns the temperature from the DHT11 sensor.

Devuelve

: Temperature value in Celsius. Returns DHT11_ERROR_TIMEOUT if reading times out. Returns DHT11_↩ ERROR_CHECKSUM if checksum validation fails.

4.2.3.3. readTemperatureHumidity()

```
int DHT11::readTemperatureHumidity (
            int & temperature,
            int & humidity)
```

Reads and returns the temperature and humidity from the DHT11 sensor.

Parámetros

temperature	Reference to a variable where the temperature value will be stored.
humidity	Reference to a variable where the humidity value will be stored.

Devuelve

: true if the reading is successful, false if it fails due to timeout or checksum error.

4.2.3.4. setDelay()

```
void DHT11::setDelay (
            unsigned long delay)
```

Sets the delay between consecutive sensor readings. If this method is not called, a default delay of 500 milliseconds is used.

Parámetros

delay	Delay duration in milliseconds between sensor readings.

4.2.4. Documentación de símbolos amigos y relacionados

4.2.4.1. setPinDHT11

```
void setPinDHT11 (
            DHT11 & sensor,
            int pin)  [friend]
```

Bloque if-else Controla si el pin ingresado es uno físicamente válido. Imprime por puerto serie UART un mensaje de error en caso de no serlo.

pinMode(sensor._pin, INPUT) "Desactiva" el pin anterior.

sensor._pin = pin Actualiza el pin.

pinMode(pin, OUTPUT) y digitalWrite(pin, HIGH) Inicializa el nuevo pin como lo hace internamente la clase DHT11

delay(250) Tiempo de espera a que el sensor se estabilice.

La documentación de esta clase está generada del siguiente archivo:

- main/libraries/DHT11-HW-481/DHT11.h

4.3. Referencia de la clase MFRC522

Diagrama de herencia de MFRC522

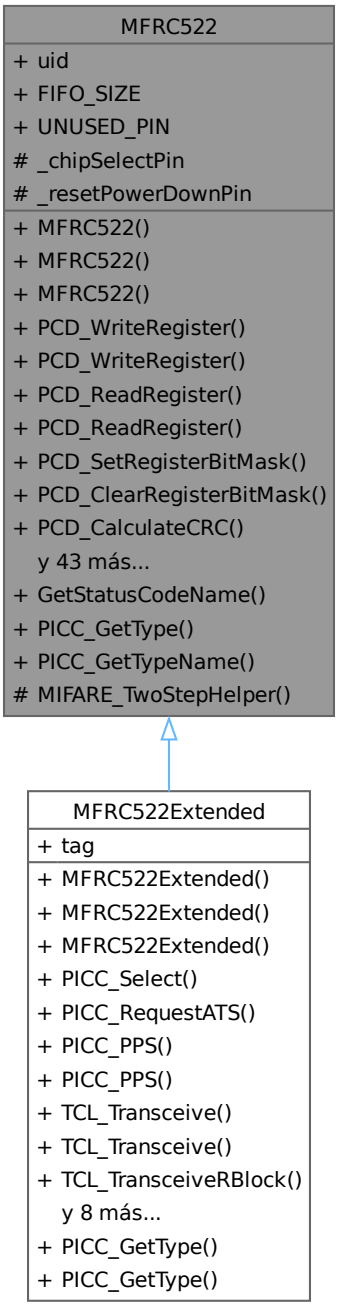


Diagrama de colaboración de MFRC522:

Estructuras de datos

- struct MIFARE_Key
- struct Uid

Tipos públicos

- enum **PCD_Register** : byte {
  CommandReg = 0x01 << 1 , **ComIEnReg** = 0x02 << 1 , **DivIEnReg** = 0x03 << 1 , **ComIrqReg** = 0x04

<< 1 ,
DivIrqReg = 0x05 << 1 , **ErrorReg** = 0x06 << 1 , **Status1Reg** = 0x07 << 1 , **Status2Reg** = 0x08 << 1 ,
FIFODataReg = 0x09 << 1 , **FIFOLevelReg** = 0x0A << 1 , **WaterLevelReg** = 0x0B << 1 , **ControlReg** = 0x0C << 1 ,
BitFramingReg = 0x0D << 1 , **CollReg** = 0x0E << 1 , **ModeReg** = 0x11 << 1 , **TxModeReg** = 0x12 << 1 ,
RxModeReg = 0x13 << 1 , **TxControlReg** = 0x14 << 1 , **TxASKReg** = 0x15 << 1 , **TxSelReg** = 0x16 << 1 ,
RxSelReg = 0x17 << 1 , **RxThresholdReg** = 0x18 << 1 , **DemodReg** = 0x19 << 1 , **MfTxReg** = 0x1C << 1 ,
MfRxReg = 0x1D << 1 , **SerialSpeedReg** = 0x1F << 1 , **CRCResultRegH** = 0x21 << 1 , **CRCResultRegL** = 0x22 << 1 ,
ModWidthReg = 0x24 << 1 , **RFCfgReg** = 0x26 << 1 , **GsNReg** = 0x27 << 1 , **CWGsPReg** = 0x28 << 1 ,
ModGsPReg = 0x29 << 1 , **TModeReg** = 0x2A << 1 , **TPrescalerReg** = 0x2B << 1 , **TReloadRegH** = 0x2C << 1 ,
TReloadRegL = 0x2D << 1 , **TCounterValueRegH** = 0x2E << 1 , **TCounterValueRegL** = 0x2F << 1 ,
TestSel1Reg = 0x31 << 1 ,
TestSel2Reg = 0x32 << 1 , **TestPinEnReg** = 0x33 << 1 , **TestPinValueReg** = 0x34 << 1 , **TestBusReg** = 0x35 << 1 ,
AutoTestReg = 0x36 << 1 , **VersionReg** = 0x37 << 1 , **AnalogTestReg** = 0x38 << 1 , **TestDAC1Reg** = 0x39 << 1 ,
TestDAC2Reg = 0x3A << 1 , **TestADCReg** = 0x3B << 1 }

- enum **PCD_Command** : byte {
  PCD_Idle = 0x00 , **PCD_Mem** = 0x01 , **PCD_GenerateRandomID** = 0x02 , **PCD_CalcCRC** = 0x03 ,
  PCD_Transmit = 0x04 , **PCD_NoCmdChange** = 0x07 , **PCD_Receive** = 0x08 , **PCD_Transceive** = 0x0C ,
  PCD_MFAuthent = 0x0E , **PCD_SoftReset** = 0x0F }
- enum **PCD_RxGain** : byte {
  RxGain_18dB = 0x00 << 4 , **RxGain_23dB** = 0x01 << 4 , **RxGain_18dB_2** = 0x02 << 4 , **RxGain_↩ 23dB_2** = 0x03 << 4 ,
  RxGain_33dB = 0x04 << 4 , **RxGain_38dB** = 0x05 << 4 , **RxGain_43dB** = 0x06 << 4 , **RxGain_48dB** = 0x07 << 4 ,
  RxGain_min = 0x00 << 4 , **RxGain_avg** = 0x04 << 4 , **RxGain_max** = 0x07 << 4 }
- enum **PICC_Command** : byte {
  PICC_CMD_REQA = 0x26 , **PICC_CMD_WUPA** = 0x52 , **PICC_CMD_CT** = 0x88 , **PICC_CMD_SEL_CL1** = 0x93 ,
  PICC_CMD_SEL_CL2 = 0x95 , **PICC_CMD_SEL_CL3** = 0x97 , **PICC_CMD_HLTA** = 0x50 , **PICC_CMD_↩ RATS** = 0xE0 ,
  PICC_CMD_MF_AUTH_KEY_A = 0x60 , **PICC_CMD_MF_AUTH_KEY_B** = 0x61 , **PICC_CMD_MF_READ** = 0x30 , **PICC_CMD_MF_WRITE** = 0xA0 ,
  PICC_CMD_MF_DECREMENT = 0xC0 , **PICC_CMD_MF_INCREMENT** = 0xC1 , **PICC_CMD_MF_↩ RESTORE** = 0xC2 , **PICC_CMD_MF_TRANSFER** = 0xB0 ,
  PICC_CMD_UL_WRITE = 0xA2 }
- enum **MIFARE_Misc** { **MF_ACK** = 0xA , **MF_KEY_SIZE** = 6 }
- enum **PICC_Type** : byte {
  PICC_TYPE_UNKNOWN , **PICC_TYPE_ISO_14443_4** , **PICC_TYPE_ISO_18092** , **PICC_TYPE_↩ MIFARE_MINI** ,
  PICC_TYPE_MIFARE_1K , **PICC_TYPE_MIFARE_4K** , **PICC_TYPE_MIFARE_UL** , **PICC_TYPE_↩ MIFARE_PLUS** ,
  PICC_TYPE_MIFARE_DESFIRE , **PICC_TYPE_TNP3XXX** , **PICC_TYPE_NOT_COMPLETE** = 0xff }
- enum **StatusCode** : byte {
  STATUS_OK , **STATUS_ERROR** , **STATUS_COLLISION** , **STATUS_TIMEOUT** ,
  STATUS_NO_ROOM , **STATUS_INTERNAL_ERROR** , **STATUS_INVALID** , **STATUS_CRC_WRONG** ,
  STATUS_MIFARE_NACK = 0xff }

Métodos públicos

- **MFRC522** (byte resetPowerDownPin)

- **MFRC522** (byte chipSelectPin, byte resetPowerDownPin)
- void **PCD_WriteRegister** (PCD_Register reg, byte value)
- void **PCD_WriteRegister** (PCD_Register reg, byte count, byte *values)
- byte **PCD_ReadRegister** (PCD_Register reg)
- void **PCD_ReadRegister** (PCD_Register reg, byte count, byte *values, byte rxAlign=0)
- void **PCD_SetRegisterBitMask** (PCD_Register reg, byte mask)
- void **PCD_ClearRegisterBitMask** (PCD_Register reg, byte mask)
- StatusCode **PCD_CalculateCRC** (byte *data, byte length, byte *result)
- void **PCD_Init** ()
- void **PCD_Init** (byte resetPowerDownPin)
- void **PCD_Init** (byte chipSelectPin, byte resetPowerDownPin)
- void **PCD_Reset** ()
- void **PCD_AntennaOn** ()
- void **PCD_AntennaOff** ()
- byte **PCD_GetAntennaGain** ()
- void **PCD_SetAntennaGain** (byte mask)
- bool **PCD_PerformSelfTest** ()
- void **PCD_SoftPowerDown** ()
- void **PCD_SoftPowerUp** ()
- StatusCode **PCD_TransceiveData** (byte *sendData, byte sendLen, byte *backData, byte *backLen, byte *validBits=nullptr, byte rxAlign=0, bool checkCRC=false)
- StatusCode **PCD_CommunicateWithPICC** (byte command, byte waitIRq, byte *sendData, byte sendLen, byte *backData=nullptr, byte *backLen=nullptr, byte *validBits=nullptr, byte rxAlign=0, bool checkCRC=false)
- StatusCode **PICC_RequestA** (byte *bufferATQA, byte *bufferSize)
- StatusCode **PICC_WakeupA** (byte *bufferATQA, byte *bufferSize)
- StatusCode **PICC_REQA_or_WUPA** (byte command, byte *bufferATQA, byte *bufferSize)
- virtual StatusCode **PICC_Select** (Uid *uid, byte validBits=0)
- StatusCode **PICC_HaltA** ()
- StatusCode **PCD_Authenticate** (byte command, byte blockAddr, MIFARE_Key *key, Uid *uid)
- void **PCD_StopCrypto1** ()
- StatusCode **MIFARE_Read** (byte blockAddr, byte *buffer, byte *bufferSize)
- StatusCode **MIFARE_Write** (byte blockAddr, byte *buffer, byte bufferSize)
- StatusCode **MIFARE_Ultralight_Write** (byte page, byte *buffer, byte bufferSize)
- StatusCode **MIFARE_Decrement** (byte blockAddr, int32_t delta)
- StatusCode **MIFARE_Increment** (byte blockAddr, int32_t delta)
- StatusCode **MIFARE_Restore** (byte blockAddr)
- StatusCode **MIFARE_Transfer** (byte blockAddr)
- StatusCode **MIFARE_GetValue** (byte blockAddr, int32_t *value)
- StatusCode **MIFARE_SetValue** (byte blockAddr, int32_t value)
- StatusCode **PCD_NTAG216_AUTH** (byte *passWord, byte pACK[])
- StatusCode **PCD_MIFARE_Transceive** (byte *sendData, byte sendLen, bool acceptTimeout=false)
- void **PCD_DumpVersionToSerial** ()
- void **PICC_DumpToSerial** (Uid *uid)
- void **PICC_DumpDetailsToSerial** (Uid *uid)
- void **PICC_DumpMifareClassicToSerial** (Uid *uid, PICC_Type piccType, MIFARE_Key *key)
- void **PICC_DumpMifareClassicSectorToSerial** (Uid *uid, MIFARE_Key *key, byte sector)
- void **PICC_DumpMifareUltralightToSerial** ()
- void **MIFARE_SetAccessBits** (byte *accessBitBuffer, byte g0, byte g1, byte g2, byte g3)
- bool **MIFARE_OpenUidBackdoor** (bool logErrors)
- bool **MIFARE_SetUid** (byte *newUid, byte uidSize, bool logErrors)
- bool **MIFARE_UnbrickUidSector** (bool logErrors)
- virtual bool **PICC_IsNewCardPresent** ()
- virtual bool **PICC_ReadCardSerial** ()

Métodos públicos estáticos

- static const __FlashStringHelper * **GetStatusCodeName** (StatusCode code)
- static PICC_Type **PICC_GetType** (byte sak)
- static const __FlashStringHelper * **PICC_GetTypeName** (PICC_Type type)

Campos de datos

- Uid **uid**

Atributos públicos estáticos

- static constexpr byte **FIFO_SIZE** = 64
- static constexpr uint8_t **UNUSED_PIN** = UINT8_MAX

Métodos protegidos

- StatusCode **MIFARE_TwoStepHelper** (byte command, byte blockAddr, int32_t data)

Atributos protegidos

- byte **_chipSelectPin**
- byte **_resetPowerDownPin**

4.3.1. Documentación de estructuras de datos

4.3.1.1. struct MFRC522::MIFARE_Key

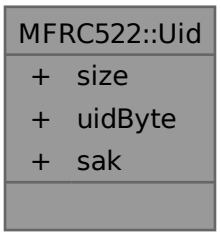
Diagrama de colaboración de MFRC522::MIFARE_Key:

Campos de datos

byte	keyByte[MF_KEY_SIZE]	

4.3.1.2. struct MFRC522::Uid

Diagrama de colaboración de MFRC522::Uid:



Campos de datos

byte	sak	
byte	size	
byte	uidByte[10]	

La documentación de esta clase está generada del siguiente archivo:

- main/libraries/RFID-RC522/MFRC522.h

4.4. Referencia de la clase MFRC522Extended

```
#include <MFRC522Extended.h>
```

Diagrama de herencia de MFRC522Extended

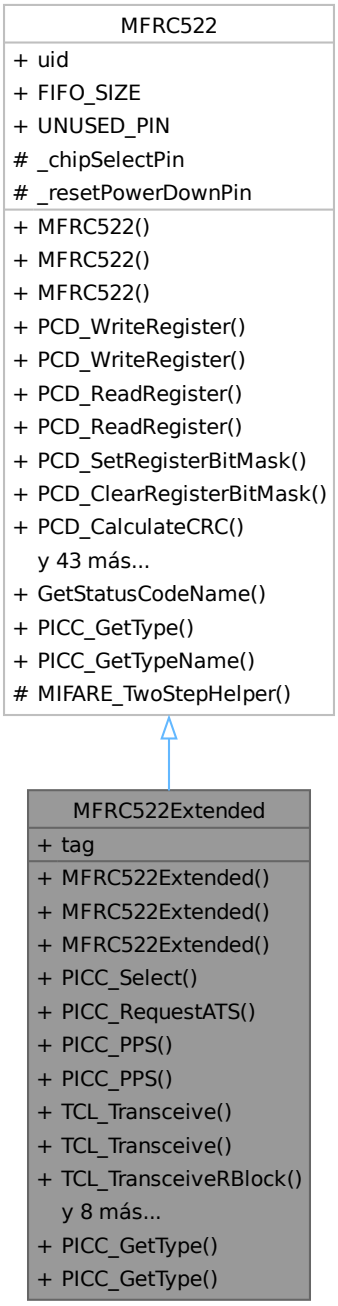
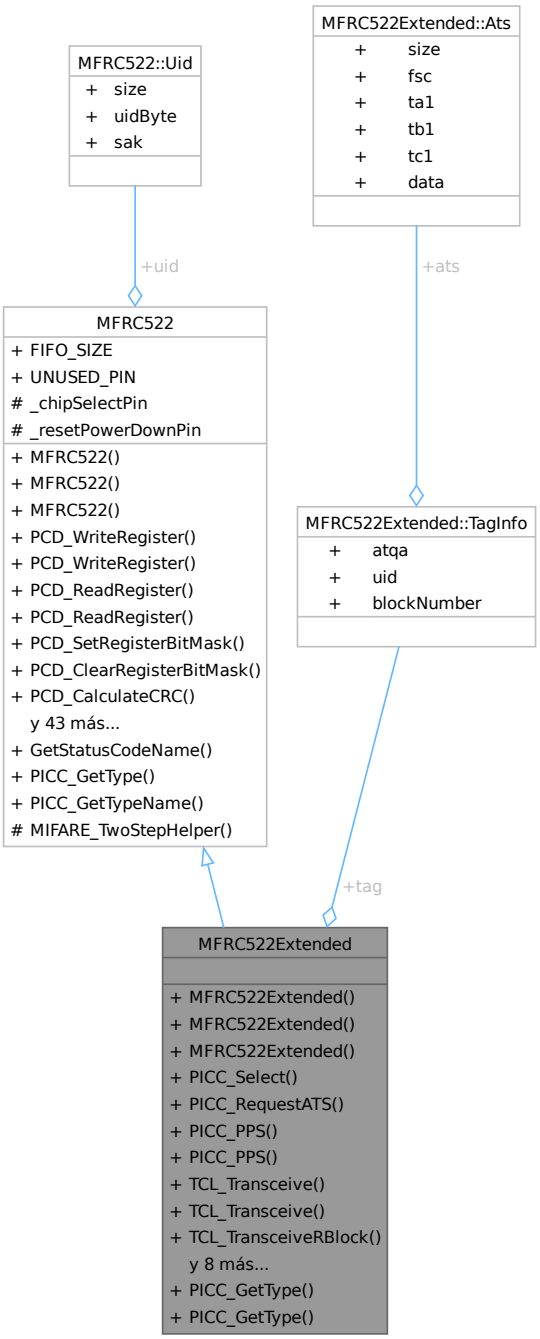


Diagrama de colaboración de MFRC522Extended:



Estructuras de datos

- struct Ats
- struct Ats.ta1
- struct Ats.tb1
- struct Ats.tc1
- struct PcbBlock

- struct PcbBlock.inf
- struct PcbBlock.prologue
- struct TagInfo

Tipos públicos

- enum **TagBitRates** : byte { **BITRATE_106KBITS** = 0x00 , **BITRATE_212KBITS** = 0x01 , **BITRATE_424KBITS** = 0x02 , **BITRATE_848KBITS** = 0x03 }

Tipos públicos heredados de MFRC522

- enum **PCD_Register** : byte {
  CommandReg = 0x01 << 1 , **ComIEnReg** = 0x02 << 1 , **DivIEnReg** = 0x03 << 1 , **ComIrqReg** = 0x04 << 1 ,
  DivIrqReg = 0x05 << 1 , **ErrorReg** = 0x06 << 1 , **Status1Reg** = 0x07 << 1 , **Status2Reg** = 0x08 << 1 , **FIFODataReg** = 0x09 << 1 , **FIFOLevelReg** = 0x0A << 1 , **WaterLevelReg** = 0x0B << 1 , **ControlReg** = 0x0C << 1 ,
  BitFramingReg = 0x0D << 1 , **CollReg** = 0x0E << 1 , **ModeReg** = 0x11 << 1 , **TxModeReg** = 0x12 << 1 ,
  RxModeReg = 0x13 << 1 , **TxControlReg** = 0x14 << 1 , **TxASKReg** = 0x15 << 1 , **TxSelReg** = 0x16 << 1 ,
  RxSelReg = 0x17 << 1 , **RxThresholdReg** = 0x18 << 1 , **DemodReg** = 0x19 << 1 , **MfTxReg** = 0x1C << 1 ,
  MfRxReg = 0x1D << 1 , **SerialSpeedReg** = 0x1F << 1 , **CRCResultRegH** = 0x21 << 1 , **CRCResultRegL** = 0x22 << 1 ,
  ModWidthReg = 0x24 << 1 , **RFCfgReg** = 0x26 << 1 , **GsNReg** = 0x27 << 1 , **CWGsPReg** = 0x28 << 1 ,
  ModGsPReg = 0x29 << 1 , **TModeReg** = 0x2A << 1 , **TPrescalerReg** = 0x2B << 1 , **TReloadRegH** = 0x2C << 1 ,
  TReloadRegL = 0x2D << 1 , **TCounterValueRegH** = 0x2E << 1 , **TCounterValueRegL** = 0x2F << 1 ,
  TestSel1Reg = 0x31 << 1 ,
  TestSel2Reg = 0x32 << 1 , **TestPinEnReg** = 0x33 << 1 , **TestPinValueReg** = 0x34 << 1 , **TestBusReg** = 0x35 << 1 ,
  AutoTestReg = 0x36 << 1 , **VersionReg** = 0x37 << 1 , **AnalogTestReg** = 0x38 << 1 , **TestDAC1Reg** = 0x39 << 1 ,
  TestDAC2Reg = 0x3A << 1 , **TestADCReg** = 0x3B << 1 }
- enum **PCD_Command** : byte {
  PCD_Idle = 0x00 , **PCD_Mem** = 0x01 , **PCD_GenerateRandomID** = 0x02 , **PCD_CalcCRC** = 0x03 ,
  PCD_Transmit = 0x04 , **PCD_NoCmdChange** = 0x07 , **PCD_Receive** = 0x08 , **PCD_Transceive** = 0x0C ,
  PCD_MFAuthent = 0x0E , **PCD_SoftReset** = 0x0F }
- enum **PCD_RxGain** : byte {
  RxGain_18dB = 0x00 << 4 , **RxGain_23dB** = 0x01 << 4 , **RxGain_18dB_2** = 0x02 << 4 , **RxGain_23dB_2** = 0x03 << 4 ,
  RxGain_33dB = 0x04 << 4 , **RxGain_38dB** = 0x05 << 4 , **RxGain_43dB** = 0x06 << 4 , **RxGain_48dB** = 0x07 << 4 ,
  RxGain_min = 0x00 << 4 , **RxGain_avg** = 0x04 << 4 , **RxGain_max** = 0x07 << 4 }
- enum **PICC_Command** : byte {
  PICC_CMD_REQA = 0x26 , **PICC_CMD_WUPA** = 0x52 , **PICC_CMD_CT** = 0x88 , **PICC_CMD_SEL_CL1** = 0x93 ,
  PICC_CMD_SEL_CL2 = 0x95 , **PICC_CMD_SEL_CL3** = 0x97 , **PICC_CMD_HLTA** = 0x50 , **PICC_CMD_RATS** = 0xE0 ,
  PICC_CMD_MF_AUTH_KEY_A = 0x60 , **PICC_CMD_MF_AUTH_KEY_B** = 0x61 , **PICC_CMD_MF_READ** = 0x30 , **PICC_CMD_MF_WRITE** = 0xA0 ,
  PICC_CMD_MF_DECREMENT = 0xC0 , **PICC_CMD_MF_INCREMENT** = 0xC1 , **PICC_CMD_MF_RESTORE** = 0xC2 , **PICC_CMD_MF_TRANSFER** = 0xB0 ,
  PICC_CMD_UL_WRITE = 0xA2 }

- enum **MIFARE_Misc** { **MF_ACK** = 0xA , **MF_KEY_SIZE** = 6 }
- enum **PICC_Type** : byte {
  PICC_TYPE_UNKNOWN , **PICC_TYPE_ISO_14443_4** , **PICC_TYPE_ISO_18092** , **PICC_TYPE_↩ MIFARE_MINI** ,
  PICC_TYPE_MIFARE_1K , **PICC_TYPE_MIFARE_4K** , **PICC_TYPE_MIFARE_UL** , **PICC_TYPE_↩ MIFARE_PLUS** ,
  PICC_TYPE_MIFARE_DESFIRE , **PICC_TYPE_TNP3XXX** , **PICC_TYPE_NOT_COMPLETE** = 0xff }
- enum **StatusCode** : byte {
  STATUS_OK , **STATUS_ERROR** , **STATUS_COLLISION** , **STATUS_TIMEOUT** ,
  STATUS_NO_ROOM , **STATUS_INTERNAL_ERROR** , **STATUS_INVALID** , **STATUS_CRC_WRONG** ,
  STATUS_MIFARE_NACK = 0xff }

Métodos públicos

- **MFRC522Extended** (uint8_t rst)
- **MFRC522Extended** (uint8_t ss, uint8_t rst)
- StatusCode PICC_Select (Uid *uid, byte validBits=0) override
- StatusCode **PICC_RequestATS** (Ats *ats)
- StatusCode **PICC_PPS** ()
- StatusCode **PICC_PPS** (TagBitRates sendBitRate, TagBitRates receiveBitRate)
- StatusCode **TCL_Transceive** (PcbBlock *send, PcbBlock *back)
- StatusCode **TCL_Transceive** (TagInfo *tag, byte *sendData, byte sendLen, byte *backData=NULL, byte *backLen=NULL)
- StatusCode **TCL_TransceiveRBlock** (TagInfo *tag, bool ack, byte *backData=NULL, byte *backLen=NULL)
- StatusCode **TCL_Deselect** (TagInfo *tag)
- void **PICC_DumpToSerial** (TagInfo *tag)
- void **PICC_DumpDetailsToSerial** (TagInfo *tag)
- void **PICC_DumpISO14443_4** (TagInfo *tag)
- bool PICC_IsNewCardPresent () override
- bool PICC_ReadCardSerial () override
- void **PICC_DumpToSerial** (Uid *uid)
- void **PICC_DumpDetailsToSerial** (Uid *uid)

Métodos públicos heredados de MFRC522

- **MFRC522** (byte resetPowerDownPin)
- **MFRC522** (byte chipSelectPin, byte resetPowerDownPin)
- void **PCD_WriteRegister** (PCD_Register reg, byte value)
- void **PCD_WriteRegister** (PCD_Register reg, byte count, byte *values)
- byte **PCD_ReadRegister** (PCD_Register reg)
- void **PCD_ReadRegister** (PCD_Register reg, byte count, byte *values, byte rxAlign=0)
- void **PCD_SetRegisterBitMask** (PCD_Register reg, byte mask)
- void **PCD_ClearRegisterBitMask** (PCD_Register reg, byte mask)
- StatusCode **PCD_CalculateCRC** (byte *data, byte length, byte *result)
- void **PCD_Init** ()
- void **PCD_Init** (byte resetPowerDownPin)
- void **PCD_Init** (byte chipSelectPin, byte resetPowerDownPin)
- void **PCD_Reset** ()
- void **PCD_AntennaOn** ()
- void **PCD_AntennaOff** ()
- byte **PCD_GetAntennaGain** ()
- void **PCD_SetAntennaGain** (byte mask)
- bool **PCD_PerformSelfTest** ()

- void **PCD_SoftPowerDown** ()
- void **PCD_SoftPowerUp** ()
- StatusCode **PCD_TransceiveData** (byte *sendData, byte sendLen, byte *backData, byte *backLen, byte *validBits=nullptr, byte rxAlign=0, bool checkCRC=false)
- StatusCode **PCD_CommunicateWithPICC** (byte command, byte waitIRq, byte *sendData, byte sendLen, byte *backData=nullptr, byte *backLen=nullptr, byte *validBits=nullptr, byte rxAlign=0, bool checkCRC=false)
- StatusCode **PICC_RequestA** (byte *bufferATQA, byte *bufferSize)
- StatusCode **PICC_WakeupA** (byte *bufferATQA, byte *bufferSize)
- StatusCode **PICC_REQA_or_WUPA** (byte command, byte *bufferATQA, byte *bufferSize)
- StatusCode **PICC_HaltA** ()
- StatusCode **PCD_Authenticate** (byte command, byte blockAddr, MIFARE_Key *key, Uid *uid)
- void **PCD_StopCrypto1** ()
- StatusCode **MIFARE_Read** (byte blockAddr, byte *buffer, byte *bufferSize)
- StatusCode **MIFARE_Write** (byte blockAddr, byte *buffer, byte bufferSize)
- StatusCode **MIFARE_Ultralight_Write** (byte page, byte *buffer, byte bufferSize)
- StatusCode **MIFARE_Decrement** (byte blockAddr, int32_t delta)
- StatusCode **MIFARE_Increment** (byte blockAddr, int32_t delta)
- StatusCode **MIFARE_Restore** (byte blockAddr)
- StatusCode **MIFARE_Transfer** (byte blockAddr)
- StatusCode **MIFARE_GetValue** (byte blockAddr, int32_t *value)
- StatusCode **MIFARE_SetValue** (byte blockAddr, int32_t value)
- StatusCode **PCD_NTAG216_AUTH** (byte *passWord, byte pACK[])
- StatusCode **PCD_MIFARE_Transceive** (byte *sendData, byte sendLen, bool acceptTimeout=false)
- void **PCD_DumpVersionToSerial** ()
- void **PICC_DumpToSerial** (Uid *uid)
- void **PICC_DumpDetailsToSerial** (Uid *uid)
- void **PICC_DumpMifareClassicToSerial** (Uid *uid, PICC_Type piccType, MIFARE_Key *key)
- void **PICC_DumpMifareClassicSectorToSerial** (Uid *uid, MIFARE_Key *key, byte sector)
- void **PICC_DumpMifareUltralightToSerial** ()
- void **MIFARE_SetAccessBits** (byte *accessBitBuffer, byte g0, byte g1, byte g2, byte g3)
- bool **MIFARE_OpenUidBackdoor** (bool logErrors)
- bool **MIFARE_SetUid** (byte *newUid, byte uidSize, bool logErrors)
- bool **MIFARE_UnbrickUidSector** (bool logErrors)

Métodos públicos estáticos

- static PICC_Type **PICC_GetType** (TagInfo *tag)
- static PICC_Type **PICC_GetType** (byte sak)

Métodos públicos estáticos heredados de MFRC522

- static const __FlashStringHelper * **GetStatusCodeName** (StatusCode code)
- static PICC_Type **PICC_GetType** (byte sak)
- static const __FlashStringHelper * **PICC_GetTypeName** (PICC_Type type)

Campos de datos

- TagInfo **tag**

Campos de datos heredados de MFRC522

- Uid **uid**

Otros miembros heredados

Atributos públicos estáticos heredados de MFRC522

- static constexpr byte **FIFO_SIZE** = 64
- static constexpr uint8_t **UNUSED_PIN** = UINT8_MAX

Métodos protegidos heredados de MFRC522

- StatusCode **MIFARE_TwoStepHelper** (byte command, byte blockAddr, int32_t data)

Atributos protegidos heredados de MFRC522

- byte **_chipSelectPin**
- byte **_resetPowerDownPin**

4.4.1. Descripción detallada

Library extends MFRC522.h to support RATS for ISO-14443-4 PICC. RATS - Request for Answer To Select.

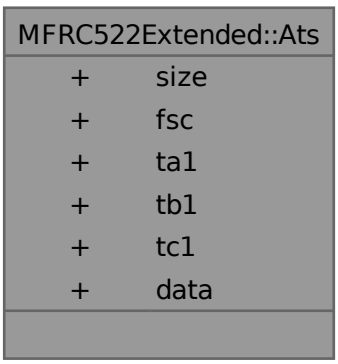
Autor

JPG-Consulting

4.4.2. Documentación de estructuras de datos

4.4.2.1. struct MFRC522Extended::Ats

Diagrama de colaboración de MFRC522Extended::Ats:

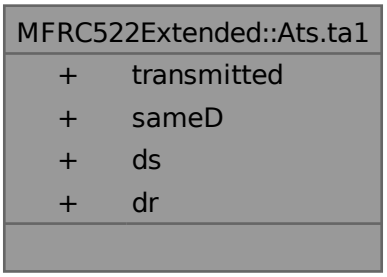


Campos de datos

byte	data[FIFO_SIZE - 2]	
byte	fsc	
byte	size	
struct Ats.ta1	ta1	
struct Ats.tb1	tb1	
struct Ats.tc1	tc1	

4.4.2.2. struct MFRC522Extended::Ats.ta1

Diagrama de colaboración de MFRC522Extended::Ats.ta1:

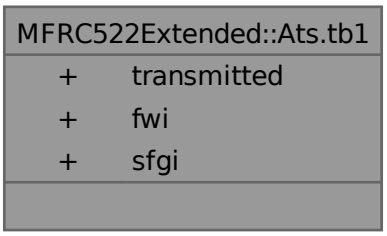


Campos de datos

TagBitRates	dr	
TagBitRates	ds	
bool	sameD	
bool	transmitted	

4.4.2.3. struct MFRC522Extended::Ats.tb1

Diagrama de colaboración de MFRC522Extended::Ats.tb1:

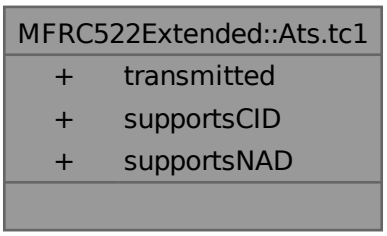


Campos de datos

byte	fwi	
byte	sfgi	
bool	transmitted	

4.4.2.4. struct MFRC522Extended::Ats.tc1

Diagrama de colaboración de MFRC522Extended::Ats.tc1:

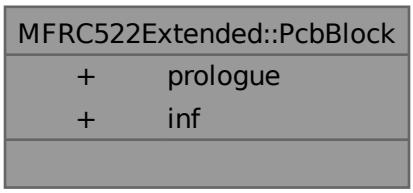


Campos de datos

bool	supportsCID	
bool	supportsNAD	
bool	transmitted	

4.4.2.5. struct MFRC522Extended::PcbBlock

Diagrama de colaboración de MFRC522Extended::PcbBlock:

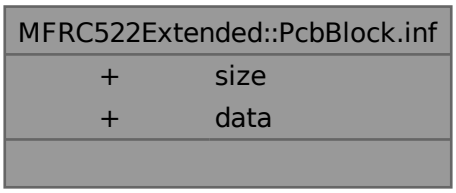


Campos de datos

struct PcbBlock.inf	inf	
struct PcbBlock.prologue	prologue	

4.4.2.6. struct MFRC522Extended::PcbBlock.inf

Diagrama de colaboración de MFRC522Extended::PcbBlock.inf:

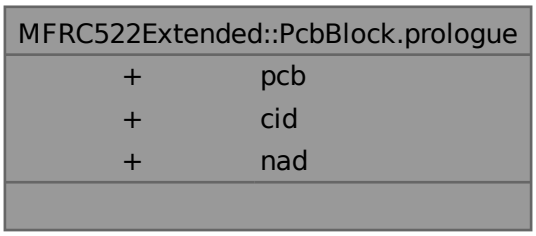


Campos de datos

byte *	data	
byte	size	

4.4.2.7. struct MFRC522Extended::PcbBlock.prologue

Diagrama de colaboración de MFRC522Extended::PcbBlock.prologue:

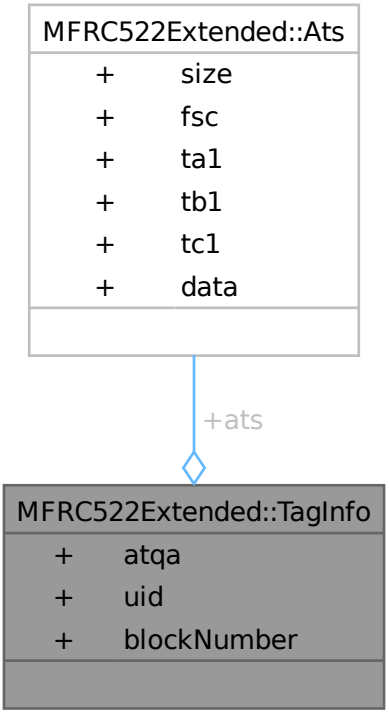


Campos de datos

byte	cid	
byte	nad	
byte	pcb	

4.4.2.8. struct MFRC522Extended::TagInfo

Diagrama de colaboración de MFRC522Extended::TagInfo:



Campos de datos

uint16_t	atqa	
Ats	ats	
bool	blockNumber	
Uid	uid	

4.4.3. Documentación de funciones miembro

4.4.3.1. PICC_IsNewCardPresent()

```
bool MFRC522Extended::PICC_IsNewCardPresent () [override], [virtual]
```

Reimplementado de MFRC522.

4.4.3.2. PICC_ReadCardSerial()

```
bool MFRC522Extended::PICC_ReadCardSerial () [override], [virtual]
```

Reimplementado de MFRC522.

4.4.3.3. PICC_Select()

```
StatusCode MFRC522Extended::PICC_Select (
            Uid * uid,
            byte validBits = 0)  [override], [virtual]
```

Reimplementado de MFRC522.

La documentación de esta clase está generada del siguiente archivo:

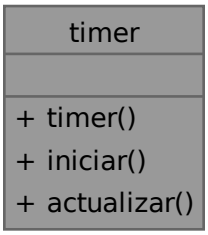
- main/libraries/RFID-RC522/MFRC522Extended.h

4.5. Referencia de la clase timer

Clase para gestionar un temporizador con dos salidas, cada una es activada en intervalos distintos, con la funcionalidad de antirrebote en los pines de ingreso y reset.

```
#include <timer.h>
```

Diagrama de colaboración de timer:



Métodos públicos

- timer (int pIngreso, int pReset, int p1, int p2, unsigned long t1, unsigned long t2, unsigned long rebote)

  Constructor de la clase timer.
- void **iniciar** ()

  void iniciar(): configura los pines del objeto de clase timer como entradas y salidas.
- void **actualizar** ()

  void actualizar(): Actualiza el estado del temporizador y controla las salidas según los tiempos establecidos. Verifica el estado de los pines de ingreso y reset, aplicando antirrebote.

4.5.1. Descripción detallada

Clase para gestionar un temporizador con dos salidas, cada una es activada en intervalos distintos, con la funcionalidad de antirrebote en los pines de ingreso y reset.

4.5.2. Documentación de constructores y destructores

4.5.2.1. timer()

```
timer::timer (
            int pIngreso,
            int pReset,
            int p1,
            int p2,
            unsigned long t1,
            unsigned long t2,
            unsigned long rebote)
```

Constructor de la clase timer.

Parámetros

pIngreso	Pin de ingreso para activar el temporizador.
pReset	Pin para resetear el temporizador.
p1	Pin de la primera salida, activado después de tiempoUno.
p2	Pin de la segunda salida, activado después de tiempoDos.
t1	Tiempo (en milisegundos) después para activar la primera salida.
t2	Tiempo (en milisegundos) después para activar la segunda salida.
rebote	Tiempo de antirrebote para los pines de ingreso y reset.

La documentación de esta clase está generada del siguiente archivo:

- main/libraries/TIMER/timer.h

Capítulo 5

Documentación de archivos

5.1. DHT11.h

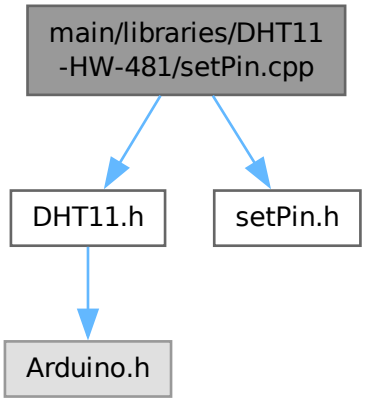
```
00001
00011 #ifndef DHT11_h
00012 #define DHT11_h
00013
00014 #include "Arduino.h"
00015
00021 class DHT11
00022 {
00023
00024 public:
00031   DHT11(int pin);
00032
00039   void setDelay(unsigned long delay);
00040
00047   int readHumidity();
00048
00055   int readTemperature();
00056
00064   int readTemperatureHumidity(int &temperature, int &humidity);
00065
00066   // Constants to represent error codes.
00067   static const int ERROR_CHECKSUM = 254;    // Error code indicating checksum mismatch.
00068   static const int ERROR_TIMEOUT = 253;     // Error code indicating a timeout occurred during
      reading.
00069   static const int TIMEOUT_DURATION = 1000; // Duration (in milliseconds) to wait before timing out.
00070
00077   // static String getErrorString(int errorCode);
00078
00079 private:
00080   int _pin;                     // Pin number used for communication with the DHT11 sensor.
00081   unsigned long _delayMS = 500; // Default delay in milliseconds between sensor readings.
00082
00093   int readRawData(byte data[5]);
00094
00100   byte readByte();
00101
00107   void startSignal();
00108
00109   friend void setPinDHT11(DHT11& sensor, int pin);
00110
00111 };
00112
00113 #endif
```

5.2. Referencia del archivo main/libraries/DHT11-HW-481/setPin.cpp

Función amiga setPinDHT11.

```
#include "DHT11.h"
#include "setPin.h"
```

Gráfico de dependencias incluidas en setPin.cpp:



Funciones

- void setPinDHT11 (DHT11 &sensor, int pin)

5.2.1. Descripción detallada

Función amiga setPinDHT11.

Parámetros

DHT11&	sensor: recibe una referencia al objeto de la clase DHT11 al cual se le cambiará el pin de conexión correspondiente al sensor de temperatura.
int	pin: es el nuevo pin para la entrada de datos del sensor de temperatura.

La función comprueba si el pin es válido, luego "desactiva" el pin anterior para inicializar el nuevo.

5.2.2. Documentación de funciones

5.2.2.1. setPinDHT11()

```
void setPinDHT11 (
            DHT11 & sensor,
            int pin)
```

Bloque if-else Controla si el pin ingresado es uno físicamente válido. Imprime por puerto serie UART un mensaje de error en caso de no serlo.

pinMode(sensor._pin, INPUT) "Desactiva" el pin anterior.

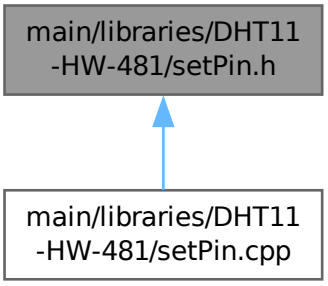
sensor._pin = pin Actualiza el pin.

pinMode(pin, OUTPUT) y digitalWrite(pin, HIGH) Inicializa el nuevo pin como lo hace internamente la clase DHT11

delay(250) Tiempo de espera a que el sensor se estabilice.

5.3. Referencia del archivo main/libraries/DHT11-HW-481/setPin.h

Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Funciones

- void setPinDHT11 (DHT11 &sensor, int pin)

5.3.1. Descripción detallada

Este archivo contiene las declaraciones de la clase DHT11 y su función amiga setPinDHT11.

5.3.2. Documentación de funciones

5.3.2.1. setPinDHT11()

```
void setPinDHT11 (
            DHT11 & sensor,
            int pin)
```

Bloque if-else Controla si el pin ingresado es uno físicamente válido. Imprime por puerto serie UART un mensaje de error en caso de no serlo.

pinMode(sensor._pin, INPUT) "Desactiva" el pin anterior.

sensor._pin = pin Actualiza el pin.

pinMode(pin, OUTPUT) y digitalWrite(pin, HIGH) Inicializa el nuevo pin como lo hace internamente la clase DHT11

delay(250) Tiempo de espera a que el sensor se estabilice.

5.4. setPin.h

Ir a la documentación de este archivo.

```
00001 #ifndef SETPIN_H
00002 #define SETPIN_H
00003
00009 class DHT11;
00010 void setPinDHT11(DHT11& sensor, int pin);
00011
00012 #endif
```

5.5. accesoRFID.h

```
00001 #ifndef ACCESORRFID_H
00002 #define ACCESORRFID_H
00003
00004 #include <SPI.h>
00005 #include "MFRC522.h"
00006
00016 class accesoRFID {
00017     public:
00018
00027       accesoRFID(byte ssPin, byte rstPin, String autorizarID);
00028
00034       void start();
00035
00042       void mostrarID();
00043
00049       bool autorizar();
00050
00056       bool detectarTarjeta();
00057
00058     private:
00059         MFRC522 rdif;
00060         String autorizarID;
00061
00062 };
00063
00064 #endif
```

5.6. deprecated.h

```
00001
00007 #ifndef DEPRECATED_H
00008 #define DEPRECATED_H
00009
00010 #ifdef __has_cpp_attribute
00011 #if __has_cpp_attribute(deprecated)
00012 #define DEPRECATED [[deprecated]]
00013 #define DEPRECATED_MSG(msg) [[deprecated(msg)]]
00014 #endif // __has_cpp_attribute(deprecated)
00015 #else
00016 #define DEPRECATED __attribute__((deprecated))
00017 #define DEPRECATED_MSG(msg) __attribute__((deprecated(msg)))
00018 #endif // __has_cpp_attribute
00019
00020 #endif // DEPRECATED_H
```

5.7. MFRC522.h

```
00001
00010 #ifndef MFRC522_h
00011 #define MFRC522_h
00012
00013 #include "require_cpp11.h"
00014 #include "deprecated.h"
00015 // Enable integer limits
00016 #define __STDC_LIMIT_MACROS
00017 #include <stdint.h>
00018 #include <Arduino.h>
00019 #include <SPI.h>
00020
00021 #ifndef MFRC522_SPICLOCK
```

```
00022 #define MFRC522_SPICLOCK (4000000u) // MFRC522 accept upto 10MHz, set to 4MHz.
00023 #endif
00024
00025 // Firmware data for self-test
00026 // Reference values based on firmware version
00027 // Hint: if needed, you can remove unused self-test data to save flash memory
00028 //
00029 // Version 0.0 (0x90)
00030 // Philips Semiconductors; Preliminary Specification Revision 2.0 - 01 August 2005; 16.1 self-test
00031 const byte MFRC522_firmware_referenceV0_0[] PROGMEM = {
00032     0x00, 0x87, 0x98, 0x0f, 0x49, 0xFF, 0x07, 0x19,
00033     0xBF, 0x22, 0x30, 0x49, 0x59, 0x63, 0xAD, 0xCA,
00034     0x7F, 0xE3, 0x4E, 0x03, 0x5C, 0x4E, 0x49, 0x50,
00035     0x47, 0x9A, 0x37, 0x61, 0xE7, 0xE2, 0xC6, 0x2E,
00036     0x75, 0x5A, 0xED, 0x04, 0x3D, 0x02, 0x4B, 0x78,
00037     0x32, 0xFF, 0x58, 0x3B, 0x7C, 0xE9, 0x00, 0x94,
00038     0xB4, 0x4A, 0x59, 0x5B, 0xFD, 0xC9, 0x29, 0xDF,
00039     0x35, 0x96, 0x98, 0x9E, 0x4F, 0x30, 0x32, 0x8D
00040 };
00041 // Version 1.0 (0x91)
00042 // NXP Semiconductors; Rev. 3.8 - 17 September 2014; 16.1.1 self-test
00043 const byte MFRC522_firmware_referenceV1_0[] PROGMEM = {
00044     0x00, 0xC6, 0x37, 0xD5, 0x32, 0xB7, 0x57, 0x5C,
00045     0xC2, 0xD8, 0x7C, 0x4D, 0xD9, 0x70, 0xC7, 0x73,
00046     0x10, 0xE6, 0xD2, 0xAA, 0x5E, 0xA1, 0x3E, 0x5A,
00047     0x14, 0xAF, 0x30, 0x61, 0xC9, 0x70, 0xDB, 0x2E,
00048     0x64, 0x22, 0x72, 0xB5, 0xBD, 0x65, 0xF4, 0xEC,
00049     0x22, 0xBC, 0xD3, 0x72, 0x35, 0xCD, 0xAA, 0x41,
00050     0x1F, 0xA7, 0xF3, 0x53, 0x14, 0xDE, 0x7E, 0x02,
00051     0xD9, 0x0F, 0xB5, 0x5E, 0x25, 0x1D, 0x29, 0x79
00052 };
00053 // Version 2.0 (0x92)
00054 // NXP Semiconductors; Rev. 3.8 - 17 September 2014; 16.1.1 self-test
00055 const byte MFRC522_firmware_referenceV2_0[] PROGMEM = {
00056     0x00, 0xEB, 0x66, 0xBA, 0x57, 0xBF, 0x23, 0x95,
00057     0xD0, 0xE3, 0x0D, 0x3D, 0x27, 0x89, 0x5C, 0xDE,
00058     0x9D, 0x3B, 0xA7, 0x00, 0x21, 0x5B, 0x89, 0x82,
00059     0x51, 0x3A, 0xEB, 0x02, 0x0C, 0xA5, 0x00, 0x49,
00060     0x7C, 0x84, 0x4D, 0xB3, 0xCC, 0xD2, 0x1B, 0x81,
00061     0x5D, 0x48, 0x76, 0xD5, 0x71, 0x61, 0x21, 0xA9,
00062     0x86, 0x96, 0x83, 0x38, 0xCF, 0x9D, 0x5B, 0x6D,
00063     0xDC, 0x15, 0xBA, 0x3E, 0x7D, 0x95, 0x3B, 0x2F
00064 };
00065 // Clone
00066 // Fudan Semiconductor FM17522 (0x88)
00067 const byte FM17522_firmware_reference[] PROGMEM = {
00068     0x00, 0xD6, 0x78, 0x8C, 0xE2, 0xAA, 0x0C, 0x18,
00069     0x2A, 0xB8, 0x7A, 0x7F, 0xD3, 0x6A, 0xCF, 0x0B,
00070     0xB1, 0x37, 0x63, 0x4B, 0x69, 0xAE, 0x91, 0xC7,
00071     0xC3, 0x97, 0xAE, 0x77, 0xF4, 0x37, 0xD7, 0x9B,
00072     0x7C, 0xF5, 0x3C, 0x11, 0x8F, 0x15, 0xC3, 0xD7,
00073     0xC1, 0x5B, 0x00, 0x2A, 0xD0, 0x75, 0xDE, 0x9E,
00074     0x51, 0x64, 0xAB, 0x3E, 0xE9, 0x15, 0xB5, 0xAB,
00075     0x56, 0x9A, 0x98, 0x82, 0x26, 0xEA, 0x2A, 0x62
00076 };
00077
00078 class MFRC522 {
00079 public:
00080     // Size of the MFRC522 FIFO
00081     static constexpr byte FIFO_SIZE = 64;       // The FIFO is 64 bytes.
00082     // Default value for unused pin
00083     static constexpr uint8_t UNUSED_PIN = UINT8_MAX;
00084
00085     // MFRC522 registers. Described in chapter 9 of the datasheet.
00086     // When using SPI all addresses are shifted one bit left in the "SPI address byte" (section
      8.1.2.3)
00087     enum PCD_Register : byte {
00088         // Page 0: Command and status
00089         //                      0x00          // reserved for future use
00090         CommandReg              = 0x01 << 1,    // starts and stops command execution
00091         ComIEnReg               = 0x02 << 1,    // enable and disable interrupt request control bits
00092         DivIEnReg               = 0x03 << 1,    // enable and disable interrupt request control bits
00093         ComIrqReg               = 0x04 << 1,    // interrupt request bits
00094         DivIrqReg               = 0x05 << 1,    // interrupt request bits
00095         ErrorReg                = 0x06 << 1,    // error bits showing the error status of the last
      command executed
00096         Status1Reg              = 0x07 << 1,    // communication status bits
00097         Status2Reg              = 0x08 << 1,    // receiver and transmitter status bits
00098         FIFODataReg             = 0x09 << 1,    // input and output of 64 byte FIFO buffer
00099         FIFOLevelReg            = 0x0A << 1,    // number of bytes stored in the FIFO buffer
00100         WaterLevelReg           = 0x0B << 1,    // level for FIFO underflow and overflow warning
00101         ControlReg              = 0x0C << 1,    // miscellaneous control registers
00102         BitFramingReg           = 0x0D << 1,    // adjustments for bit-oriented frames
00103         CollReg                 = 0x0E << 1,    // bit position of the first bit-collision detected on
      the RF interface
00104         //                      0x0F          // reserved for future use
00105
```

```
00106         // Page 1: Command
00107         //                       0x10          // reserved for future use
00108         ModeReg                 = 0x11 << 1,    // defines general modes for transmitting and receiving

00109         TxModeReg               = 0x12 << 1,    // defines transmission data rate and framing
00110         RxModeReg               = 0x13 << 1,    // defines reception data rate and framing
00111         TxControlReg            = 0x14 << 1,    // controls the logical behavior of the antenna driver
      pins TX1 and TX2
00112         TxASKReg                = 0x15 << 1,    // controls the setting of the transmission modulation
00113         TxSelReg                = 0x16 << 1,    // selects the internal sources for the antenna driver
00114         RxSelReg                = 0x17 << 1,    // selects internal receiver settings
00115         RxThresholdReg          = 0x18 << 1,    // selects thresholds for the bit decoder
00116         DemodReg                = 0x19 << 1,    // defines demodulator settings
00117         //                       0x1A          // reserved for future use
00118         //                       0x1B          // reserved for future use
00119         MfTxReg                 = 0x1C << 1,    // controls some MIFARE communication transmit
      parameters
00120         MfRxReg                 = 0x1D << 1,    // controls some MIFARE communication receive
      parameters
00121         //                       0x1E          // reserved for future use
00122         SerialSpeedReg          = 0x1F << 1,    // selects the speed of the serial UART interface
00123
00124         // Page 2: Configuration
00125         //                       0x20          // reserved for future use
00126         CRCResultRegH           = 0x21 << 1,    // shows the MSB and LSB values of the CRC calculation
00127         CRCResultRegL           = 0x22 << 1,
00128         //                       0x23          // reserved for future use
00129         ModWidthReg             = 0x24 << 1,    // controls the ModWidth setting?
00130         //                       0x25          // reserved for future use
00131         RFCfgReg                = 0x26 << 1,    // configures the receiver gain
00132         GsNReg                  = 0x27 << 1,    // selects the conductance of the antenna driver pins
      TX1 and TX2 for modulation
00133         CWGsPReg                = 0x28 << 1,    // defines the conductance of the p-driver output
      during periods of no modulation
00134         ModGsPReg               = 0x29 << 1,    // defines the conductance of the p-driver output
      during periods of modulation
00135         TModeReg                = 0x2A << 1,    // defines settings for the internal timer
00136         TPrescalerReg           = 0x2B << 1,    // the lower 8 bits of the TPrescaler value. The 4 high
      bits are in TModeReg.
00137         TReloadRegH             = 0x2C << 1,    // defines the 16-bit timer reload value
00138         TReloadRegL             = 0x2D << 1,
00139         TCounterValueRegH       = 0x2E << 1,    // shows the 16-bit timer value
00140         TCounterValueRegL       = 0x2F << 1,
00141
00142         // Page 3: Test Registers
00143         //                       0x30          // reserved for future use
00144         TestSel1Reg             = 0x31 << 1,    // general test signal configuration
00145         TestSel2Reg             = 0x32 << 1,    // general test signal configuration
00146         TestPinEnReg            = 0x33 << 1,    // enables pin output driver on pins D1 to D7
00147         TestPinValueReg         = 0x34 << 1,    // defines the values for D1 to D7 when it is used as
      an I/O bus
00148         TestBusReg              = 0x35 << 1,    // shows the status of the internal test bus
00149         AutoTestReg             = 0x36 << 1,    // controls the digital self-test
00150         VersionReg              = 0x37 << 1,    // shows the software version
00151         AnalogTestReg           = 0x38 << 1,    // controls the pins AUX1 and AUX2
00152         TestDAC1Reg             = 0x39 << 1,    // defines the test value for TestDAC1
00153         TestDAC2Reg             = 0x3A << 1,    // defines the test value for TestDAC2
00154         TestADCReg              = 0x3B << 1     // shows the value of ADC I and Q channels
00155         //                       0x3C          // reserved for production tests
00156         //                       0x3D          // reserved for production tests
00157         //                       0x3E          // reserved for production tests
00158         //                       0x3F          // reserved for production tests
00159     };
00160
00161     // MFRC522 commands. Described in chapter 10 of the datasheet.
00162     enum PCD_Command : byte {
00163         PCD_Idle                = 0x00,     // no action, cancels current command execution
00164         PCD_Mem                 = 0x01,     // stores 25 bytes into the internal buffer
00165         PCD_GenerateRandomID    = 0x02,     // generates a 10-byte random ID number
00166         PCD_CalcCRC             = 0x03,     // activates the CRC coprocessor or performs a self-test
00167         PCD_Transmit            = 0x04,     // transmits data from the FIFO buffer
00168         PCD_NoCmdChange         = 0x07,     // no command change, can be used to modify the CommandReg
      register bits without affecting the command, for example, the PowerDown bit
00169         PCD_Receive             = 0x08,     // activates the receiver circuits
00170         PCD_Transceive          = 0x0C,     // transmits data from FIFO buffer to antenna and
      automatically activates the receiver after transmission
00171         PCD_MFAuthent           = 0x0E,     // performs the MIFARE standard authentication as a reader
00172         PCD_SoftReset           = 0x0F      // resets the MFRC522
00173     };
00174
00175     // MFRC522 RxGain[2:0] masks, defines the receiver's signal voltage gain factor (on the PCD).
00176     // Described in 9.3.3.6 / table 98 of the datasheet at
      http://www.nxp.com/documents/data_sheet/MFRC522.pdf
00177     enum PCD_RxGain : byte {
00178         RxGain_18dB             = 0x00 << 4,    // 000b - 18 dB, minimum
00179         RxGain_23dB             = 0x01 << 4,    // 001b - 23 dB
00180         RxGain_18dB_2           = 0x02 << 4,    // 010b - 18 dB, it seems 010b is a duplicate for 000b
```

```
00181         RxGain_23dB_2           = 0x03 << 4,    // 011b - 23 dB, it seems 011b is a duplicate for 001b
00182         RxGain_33dB             = 0x04 << 4,    // 100b - 33 dB, average, and typical default
00183         RxGain_38dB             = 0x05 << 4,    // 101b - 38 dB
00184         RxGain_43dB             = 0x06 << 4,    // 110b - 43 dB
00185         RxGain_48dB             = 0x07 << 4,    // 111b - 48 dB, maximum
00186         RxGain_min              = 0x00 << 4,    // 000b - 18 dB, minimum, convenience for RxGain_18dB
00187         RxGain_avg              = 0x04 << 4,    // 100b - 33 dB, average, convenience for RxGain_33dB
00188         RxGain_max              = 0x07 << 4     // 111b - 48 dB, maximum, convenience for RxGain_48dB
00189     };
00190
00191     // Commands sent to the PICC.
00192     enum PICC_Command : byte {
00193         // The commands used by the PCD to manage communication with several PICCs (ISO 14443-3, Type
      A, section 6.4)
00194         PICC_CMD_REQA           = 0x26,     // REQuest command, Type A. Invites PICCs in state IDLE to
      go to READY and prepare for anticollision or selection. 7 bit frame.
00195         PICC_CMD_WUPA           = 0x52,     // Wake-UP command, Type A. Invites PICCs in state IDLE
      and HALT to go to READY(*) and prepare for anticollision or selection. 7 bit frame.
00196         PICC_CMD_CT             = 0x88,     // Cascade Tag. Not really a command, but used during anti
      collision.
00197         PICC_CMD_SEL_CL1        = 0x93,     // Anti collision/Select, Cascade Level 1
00198         PICC_CMD_SEL_CL2        = 0x95,     // Anti collision/Select, Cascade Level 2
00199         PICC_CMD_SEL_CL3        = 0x97,     // Anti collision/Select, Cascade Level 3
00200         PICC_CMD_HLTA           = 0x50,     // HaLT command, Type A. Instructs an ACTIVE PICC to go to
      state HALT.
00201         PICC_CMD_RATS           = 0xE0,     // Request command for Answer To Reset.
00202         // The commands used for MIFARE Classic (from
      http://www.mouser.com/ds/2/302/MF1S503x-89574.pdf, Section 9)
00203         // Use PCD_MFAuthent to authenticate access to a sector, then use these commands to
      read/write/modify the blocks on the sector.
00204         // The read/write commands can also be used for MIFARE Ultralight.
00205         PICC_CMD_MF_AUTH_KEY_A  = 0x60,     // Perform authentication with Key A
00206         PICC_CMD_MF_AUTH_KEY_B  = 0x61,     // Perform authentication with Key B
00207         PICC_CMD_MF_READ        = 0x30,     // Reads one 16 byte block from the authenticated sector
      of the PICC. Also used for MIFARE Ultralight.
00208         PICC_CMD_MF_WRITE       = 0xA0,     // Writes one 16 byte block to the authenticated sector of
      the PICC. Called "COMPATIBILITY WRITE" for MIFARE Ultralight.
00209         PICC_CMD_MF_DECREMENT   = 0xC0,     // Decrements the contents of a block and stores the
      result in the internal data register.
00210         PICC_CMD_MF_INCREMENT   = 0xC1,     // Increments the contents of a block and stores the
      result in the internal data register.
00211         PICC_CMD_MF_RESTORE     = 0xC2,     // Reads the contents of a block into the internal data
      register.
00212         PICC_CMD_MF_TRANSFER    = 0xB0,     // Writes the contents of the internal data register to a
      block.
00213         // The commands used for MIFARE Ultralight (from
      http://www.nxp.com/documents/data_sheet/MF0ICU1.pdf, Section 8.6)
00214         // The PICC_CMD_MF_READ and PICC_CMD_MF_WRITE can also be used for MIFARE Ultralight.
00215         PICC_CMD_UL_WRITE       = 0xA2      // Writes one 4 byte page to the PICC.
00216     };
00217
00218     // MIFARE constants that does not fit anywhere else
00219     enum MIFARE_Misc {
00220         MF_ACK                  = 0xA,      // The MIFARE Classic uses a 4 bit ACK/NAK. Any other
      value than 0xA is NAK.
00221         MF_KEY_SIZE             = 6         // A Mifare Crypto1 key is 6 bytes.
00222     };
00223
00224     // PICC types we can detect. Remember to update PICC_GetTypeName() if you add more.
00225     // last value set to 0xff, then compiler uses less ram, it seems some optimisations are triggered
00226     enum PICC_Type : byte {
00227         PICC_TYPE_UNKNOWN       ,
00228         PICC_TYPE_ISO_14443_4   ,   // PICC compliant with ISO/IEC 14443-4
00229         PICC_TYPE_ISO_18092     ,   // PICC compliant with ISO/IEC 18092 (NFC)
00230         PICC_TYPE_MIFARE_MINI   ,   // MIFARE Classic protocol, 320 bytes
00231         PICC_TYPE_MIFARE_1K     ,   // MIFARE Classic protocol, 1KB
00232         PICC_TYPE_MIFARE_4K     ,   // MIFARE Classic protocol, 4KB
00233         PICC_TYPE_MIFARE_UL     ,   // MIFARE Ultralight or Ultralight C
00234         PICC_TYPE_MIFARE_PLUS   ,   // MIFARE Plus
00235         PICC_TYPE_MIFARE_DESFIRE,   // MIFARE DESFire
00236         PICC_TYPE_TNP3XXX       ,   // Only mentioned in NXP AN 10833 MIFARE Type Identification
      Procedure
00237         PICC_TYPE_NOT_COMPLETE  = 0xff  // SAK indicates UID is not complete.
00238     };
00239
00240     // Return codes from the functions in this class. Remember to update GetStatusCodeName() if you
      add more.
00241     // last value set to 0xff, then compiler uses less ram, it seems some optimisations are triggered
00242     enum StatusCode : byte {
00243         STATUS_OK               ,   // Success
00244         STATUS_ERROR            ,   // Error in communication
00245         STATUS_COLLISION        ,   // Collission detected
00246         STATUS_TIMEOUT          ,   // Timeout in communication.
00247         STATUS_NO_ROOM          ,   // A buffer is not big enough.
00248         STATUS_INTERNAL_ERROR   ,   // Internal error in the code. Should not happen ;-)
00249         STATUS_INVALID          ,   // Invalid argument.
00250         STATUS_CRC_WRONG        ,   // The CRC_A does not match
```

```
00251         STATUS_MIFARE_NACK      = 0xff  // A MIFARE PICC responded with NAK.
00252     };
00253
00254     // A struct used for passing the UID of a PICC.
00255     typedef struct {
00256         byte        size;           // Number of bytes in the UID. 4, 7 or 10.
00257         byte        uidByte[10];
00258         byte        sak;            // The SAK (Select acknowledge) byte returned from the PICC after
      successful selection.
00259     } Uid;
00260
00261     // A struct used for passing a MIFARE Crypto1 key
00262     typedef struct {
00263         byte        keyByte[MF_KEY_SIZE];
00264     } MIFARE_Key;
00265
00266     // Member variables
00267     Uid uid;                            // Used by PICC_ReadCardSerial().
00268
00270     // Functions for setting up the Arduino
00272     MFRC522();
00273     MFRC522(byte resetPowerDownPin);
00274     MFRC522(byte chipSelectPin, byte resetPowerDownPin);
00275
00277     // Basic interface functions for communicating with the MFRC522
00279     void PCD_WriteRegister(PCD_Register reg, byte value);
00280     void PCD_WriteRegister(PCD_Register reg, byte count, byte *values);
00281     byte PCD_ReadRegister(PCD_Register reg);
00282     void PCD_ReadRegister(PCD_Register reg, byte count, byte *values, byte rxAlign = 0);
00283     void PCD_SetRegisterBitMask(PCD_Register reg, byte mask);
00284     void PCD_ClearRegisterBitMask(PCD_Register reg, byte mask);
00285     StatusCode PCD_CalculateCRC(byte *data, byte length, byte *result);
00286
00288     // Functions for manipulating the MFRC522
00290     void PCD_Init();
00291     void PCD_Init(byte resetPowerDownPin);
00292     void PCD_Init(byte chipSelectPin, byte resetPowerDownPin);
00293     void PCD_Reset();
00294     void PCD_AntennaOn();
00295     void PCD_AntennaOff();
00296     byte PCD_GetAntennaGain();
00297     void PCD_SetAntennaGain(byte mask);
00298     bool PCD_PerformSelfTest();
00299
00301     // Power control functions
00303     void PCD_SoftPowerDown();
00304     void PCD_SoftPowerUp();
00305
00307     // Functions for communicating with PICCs
00309     StatusCode PCD_TransceiveData(byte *sendData, byte sendLen, byte *backData, byte *backLen, byte
      *validBits = nullptr, byte rxAlign = 0, bool checkCRC = false);
00310     StatusCode PCD_CommunicateWithPICC(byte command, byte waitIRq, byte *sendData, byte sendLen, byte
      *backData = nullptr, byte *backLen = nullptr, byte *validBits = nullptr, byte rxAlign = 0, bool
      checkCRC = false);
00311     StatusCode PICC_RequestA(byte *bufferATQA, byte *bufferSize);
00312     StatusCode PICC_WakeupA(byte *bufferATQA, byte *bufferSize);
00313     StatusCode PICC_REQA_or_WUPA(byte command, byte *bufferATQA, byte *bufferSize);
00314     virtual StatusCode PICC_Select(Uid *uid, byte validBits = 0);
00315     StatusCode PICC_HaltA();
00316
00318     // Functions for communicating with MIFARE PICCs
00320     StatusCode PCD_Authenticate(byte command, byte blockAddr, MIFARE_Key *key, Uid *uid);
00321     void PCD_StopCrypto1();
00322     StatusCode MIFARE_Read(byte blockAddr, byte *buffer, byte *bufferSize);
00323     StatusCode MIFARE_Write(byte blockAddr, byte *buffer, byte bufferSize);
00324     StatusCode MIFARE_Ultralight_Write(byte page, byte *buffer, byte bufferSize);
00325     StatusCode MIFARE_Decrement(byte blockAddr, int32_t delta);
00326     StatusCode MIFARE_Increment(byte blockAddr, int32_t delta);
00327     StatusCode MIFARE_Restore(byte blockAddr);
00328     StatusCode MIFARE_Transfer(byte blockAddr);
00329     StatusCode MIFARE_GetValue(byte blockAddr, int32_t *value);
00330     StatusCode MIFARE_SetValue(byte blockAddr, int32_t value);
00331     StatusCode PCD_NTAG216_AUTH(byte *passWord, byte pACK[]);
00332
00334     // Support functions
00336     StatusCode PCD_MIFARE_Transceive(byte *sendData, byte sendLen, bool acceptTimeout = false);
00337     // old function used too much memory, now name moved to flash; if you need char, copy from flash
      to memory
00338     //const char *GetStatusCodeName(byte code);
00339     static const __FlashStringHelper *GetStatusCodeName(StatusCode code);
00340     static PICC_Type PICC_GetType(byte sak);
00341     // old function used too much memory, now name moved to flash; if you need char, copy from flash
      to memory
00342     //const char *PICC_GetTypeName(byte type);
00343     static const __FlashStringHelper *PICC_GetTypeName(PICC_Type type);
00344
00345     // Support functions for debuging
```

```
00346     void PCD_DumpVersionToSerial();
00347     void PICC_DumpToSerial(Uid *uid);
00348     void PICC_DumpDetailsToSerial(Uid *uid);
00349     void PICC_DumpMifareClassicToSerial(Uid *uid, PICC_Type piccType, MIFARE_Key *key);
00350     void PICC_DumpMifareClassicSectorToSerial(Uid *uid, MIFARE_Key *key, byte sector);
00351     void PICC_DumpMifareUltralightToSerial();
00352
00353     // Advanced functions for MIFARE
00354     void MIFARE_SetAccessBits(byte *accessBitBuffer, byte g0, byte g1, byte g2, byte g3);
00355     bool MIFARE_OpenUidBackdoor(bool logErrors);
00356     bool MIFARE_SetUid(byte *newUid, byte uidSize, bool logErrors);
00357     bool MIFARE_UnbrickUidSector(bool logErrors);
00358
00360     // Convenience functions - does not add extra functionality
00362     virtual bool PICC_IsNewCardPresent();
00363     virtual bool PICC_ReadCardSerial();
00364
00365 protected:
00366     byte _chipSelectPin;        // Arduino pin connected to MFRC522's SPI slave select input (Pin 24,
      NSS, active low)
00367     byte _resetPowerDownPin;    // Arduino pin connected to MFRC522's reset and power down input (Pin
      6, NRSTPD, active low)
00368     StatusCode MIFARE_TwoStepHelper(byte command, byte blockAddr, int32_t data);
00369 };
00370
00371 #endif
```

5.8. MFRC522Extended.h

```
00001
00006 #ifndef MFRC522Extended_h
00007 #define MFRC522Extended_h
00008
00009 #include <Arduino.h>
00010 #include "MFRC522.h"
00011
00012 class MFRC522Extended : public MFRC522 {
00013
00014 public:
00015     // ISO/IEC 14443-4 bit rates
00016     enum TagBitRates : byte {
00017         BITRATE_106KBITS = 0x00,
00018         BITRATE_212KBITS = 0x01,
00019         BITRATE_424KBITS = 0x02,
00020         BITRATE_848KBITS = 0x03
00021     };
00022
00023     // Structure to store ISO/IEC 14443-4 ATS
00024     typedef struct {
00025         byte size;
00026         byte fsc;                 // Frame size for proximity card
00027
00028         struct {
00029             bool transmitted;
00030             bool        sameD;  // Only the same D for both directions supported
00031             TagBitRates ds;     // Send D
00032             TagBitRates dr;     // Receive D
00033         } ta1;
00034
00035         struct {
00036             bool transmitted;
00037             byte fwi;           // Frame waiting time integer
00038             byte sfgi;          // Start-up frame guard time integer
00039         } tb1;
00040
00041         struct {
00042             bool transmitted;
00043             bool supportsCID;
00044             bool supportsNAD;
00045         } tc1;
00046
00047         // Raw data from ATS
00048         byte data[FIFO_SIZE - 2]; // ATS cannot be bigger than FSD - 2 bytes (CRC), according to ISO
      14443-4 5.2.2
00049     } Ats;
00050
00051     // A struct used for passing the PICC information
00052     typedef struct {
00053         uint16_t    atqa;
00054         Uid          uid;
00055         Ats          ats;
00056
00057         // For Block PCB
```

```
00058         bool blockNumber;
00059     } TagInfo;
00060
00061     // A struct used for passing PCB Block
00062     typedef struct {
00063         struct {
00064             byte pcb;
00065             byte cid;
00066             byte nad;
00067         } prologue;
00068         struct {
00069             byte size;
00070             byte *data;
00071         } inf;
00072     } PcbBlock;
00073
00074     // Member variables
00075     TagInfo tag;
00076
00078     // Contructors
00080     MFRC522Extended() : MFRC522() {};
00081     MFRC522Extended(uint8_t rst) : MFRC522(rst) {};
00082     MFRC522Extended(uint8_t ss, uint8_t rst) : MFRC522(ss, rst) {};
00083
00085     // Functions for communicating with PICCs
00087     StatusCode PICC_Select(Uid *uid, byte validBits = 0) override; // overrride
00088     StatusCode PICC_RequestATS(Ats *ats);
00089     StatusCode PICC_PPS();                                                    // PPS command without
      bitrate parameter
00090     StatusCode PICC_PPS(TagBitRates sendBitRate, TagBitRates receiveBitRate); // Different D values
00091
00093     // Functions for communicating with ISO/IEC 14433-4 cards
00095     StatusCode TCL_Transceive(PcbBlock *send, PcbBlock *back);
00096     StatusCode TCL_Transceive(TagInfo * tag, byte *sendData, byte sendLen, byte *backData = NULL, byte
      *backLen = NULL);
00097     StatusCode TCL_TransceiveRBlock(TagInfo *tag, bool ack, byte *backData = NULL, byte *backLen =
      NULL);
00098     StatusCode TCL_Deselect(TagInfo *tag);
00099
00101     // Support functions
00103     static PICC_Type PICC_GetType(TagInfo *tag);
00104     using MFRC522::PICC_GetType;// // make old PICC_GetType(byte sak) available, otherwise would be
      hidden by PICC_GetType(TagInfo *tag)
00105
00106     // Support functions for debuging
00107     void PICC_DumpToSerial(TagInfo *tag);
00108     using MFRC522::PICC_DumpToSerial; // make old PICC_DumpToSerial(Uid *uid) available, otherwise
      would be hidden by PICC_DumpToSerial(TagInfo *tag)
00109     void PICC_DumpDetailsToSerial(TagInfo *tag);
00110     using MFRC522::PICC_DumpDetailsToSerial; // make old PICC_DumpDetailsToSerial(Uid *uid) available,
      otherwise would be hidden by PICC_DumpDetailsToSerial(TagInfo *tag)
00111     void PICC_DumpISO14443_4(TagInfo *tag);
00112
00114     // Convenience functions - does not add extra functionality
00116     bool PICC_IsNewCardPresent() override; // overrride
00117     bool PICC_ReadCardSerial() override; // overrride
00118 };
00119
00120 #endif
```

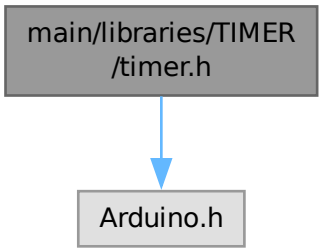
5.9. require_cpp11.h

```
00001
00005 #ifndef REQUIRE_CPP11_H
00006 #define REQUIRE_CPP11_H
00007
00008 #if __cplusplus < 201103L
00009 #error "This library needs at least a C++11 compliant compiler, maybe compiler argument for C++11
      support is missing or if you use Arduino IDE upgrade to version >=1.6.6"
00010 #endif
00011
00012 #endif // REQUIRE_CPP11_H
```

5.10. Referencia del archivo main/libraries/TIMER/timer.h

Declaración de la clase timer para manejar dos fases de un temporizador, con antirrebote y un reset.

`#include <Arduino.h>`
Gráfico de dependencias incluidas en timer.h:



Estructuras de datos

- class timer

  Clase para gestionar un temporizador con dos salidas, cada una es activada en intervalos distintos, con la funcionalidad de antirrebote en los pines de ingreso y reset.

5.10.1. Descripción detallada

Declaración de la clase timer para manejar dos fases de un temporizador, con antirrebote y un reset.

5.11. timer.h

Ir a la documentación de este archivo.

```
00001
00006 #ifndef TIMER_H
00007 #define TIMER_H
00008
00009 #include <Arduino.h>
00010
00015 class timer {
00016     public:
00028        timer(int pIngreso, int pReset, int p1, int p2, unsigned long t1, unsigned long t2, unsigned
      long rebote);
00029
00034        void iniciar();
00039        void actualizar();
00040
00041     private:
00042         const unsigned long tiempoUno;
00043         const unsigned long tiempoDos;
00044         unsigned long tiempoAntirrebote;
00045         unsigned long ultimoTiempoIngreso;
00046         unsigned long ultimoTiempoReset;
00047         unsigned long cont1;
00048         int pinIngreso;
00049         int pinSalidaUno;
00050         int pinSalidaDos;
00051         int pinReset;
00052         bool contON;
00060         bool antirrebote(int, unsigned long&);
00061 };
00062
00063 #endif
00064
```

Índice alfabético