Telemetry Firmware

Generated by Doxygen 1.8.14

Contents

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

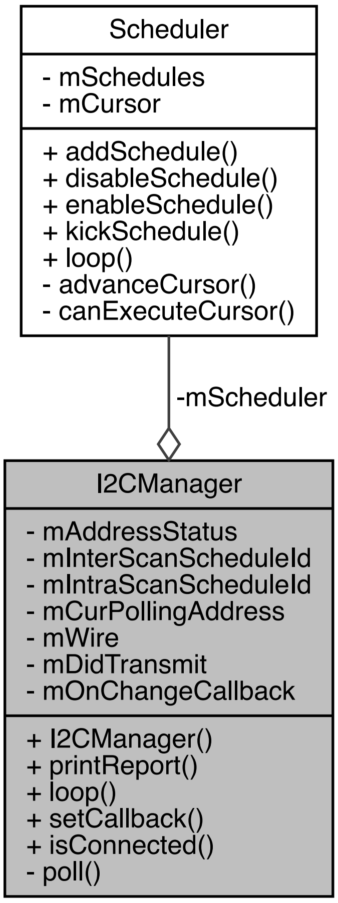
Class Documentation

3.1 I2CManager Class Reference

Responsible for discovering the connectivity status of I2C peripherals.

```
#include <I2CManager.h>
```

Collaboration diagram for I2CManager:



Public Member Functions

- I2CManager (TwoWire ∗wire, Duration interScanPeriod=I2CMANAGER_DEFAULT_INTER_SCAN_PERIOD, Duration intraScanPeriod=I2CMANAGER_DEFAULT_INTRA_SCAN_PERIOD)
- void printReport (Stream ∗stream)
- void loop ()

  Standard Arduino style `loop` function.
- void setCallback (std::shared_ptr< std::function< void(uint8_t)>> f)

  Set a callback for when connectivity status changes.
- bool isConnected (uint8_t busAddr)

Private Member Functions

- void poll ()

Private Attributes

- std::bitset< 128 > mAddressStatus
- Scheduler mScheduler
- ScheduleId mInterScanScheduleId
- ScheduleId mIntraScanScheduleId
- uint8_t mCurPollingAddress
- TwoWire ∗ mWire
- bool mDidTransmit
- std::shared_ptr< std::function< void(uint8_t)> > mOnChangeCallback

3.1.1 Detailed Description

Responsible for discovering the connectivity status of I2C peripherals.

```
*        @@ <- Intra Scan Period
*           @@          &&&&&&&&&&&&&& <- Inter Scan Period
*                                      @@
*       |  |  | ...... | ............ |  | ....
* Start ^  |  |        |              |  |
*   Poll 1 ^  |        |              |  |
*      Poll 2 ^        |              |  |
* Finish address range ^              |  |
*                               Start ^  |
*                                 Poll 1 ^
*                                          continues...
*
```

3.1.2 Constructor & Destructor Documentation

3.1.2.1 I2CManager()

```
I2CManager::I2CManager (
             TwoWire * wire,
             Duration interScanPeriod = I2CMANAGER_DEFAULT_INTER_SCAN_PERIOD,
             Duration intraScanPeriod = I2CMANAGER_DEFAULT_INTRA_SCAN_PERIOD )
```

Parameters

interScanPeriod	Time betweeen the start and end of a full poll across the address range.
intraScanPeriod	Time between scanning single addresses within a full poll.

3.1.3 Member Function Documentation

3.1.3.1 isConnected()

```
bool I2CManager::isConnected (
            uint8_t busAddr )
```

Check whether a peripheral is connected at a particular bus address.

3.1.3.2 loop()

```
void I2CManager::loop ( )
```

Standard Arduino style `loop` function.

Call this from your `loop` function. This function is non-blocking (e.g., avoids the use of `delay`).

Manages auto-discovery of I2C device connection status.

3.1.3.3 poll()

```
void I2CManager::poll ( )  [private]
```

3.1.3.4 printReport()

```
void I2CManager::printReport (
            Stream * stream )
```

3.1.3.5 setCallback()

```
void I2CManager::setCallback (
            std::shared_ptr< std::function< void(uint8_t)>> f )
```

Set a callback for when connectivity status changes.

The callback will receive the bus address that changed.

3.1.4 Member Data Documentation

3.1.4.1 mAddressStatus

```
std::bitset<128> I2CManager::mAddressStatus  [private]
```

3.1.4.2 mCurPollingAddress

```
uint8_t I2CManager::mCurPollingAddress [private]
```

3.1.4.3 mDidTransmit

```
bool I2CManager::mDidTransmit [private]
```

3.1.4.4 mInterScanScheduleId

```
ScheduleId I2CManager::mInterScanScheduleId [private]
```

3.1.4.5 mIntraScanScheduleId

```
ScheduleId I2CManager::mIntraScanScheduleId [private]
```

3.1.4.6 mOnChangeCallback

```
std::shared_ptr<std::function<void(uint8_t)> > I2CManager::mOnChangeCallback [private]
```

3.1.4.7 mScheduler

```
Scheduler I2CManager::mScheduler [private]
```

3.1.4.8 mWire

```
TwoWire* I2CManager::mWire [private]
```

The documentation for this class was generated from the following files:

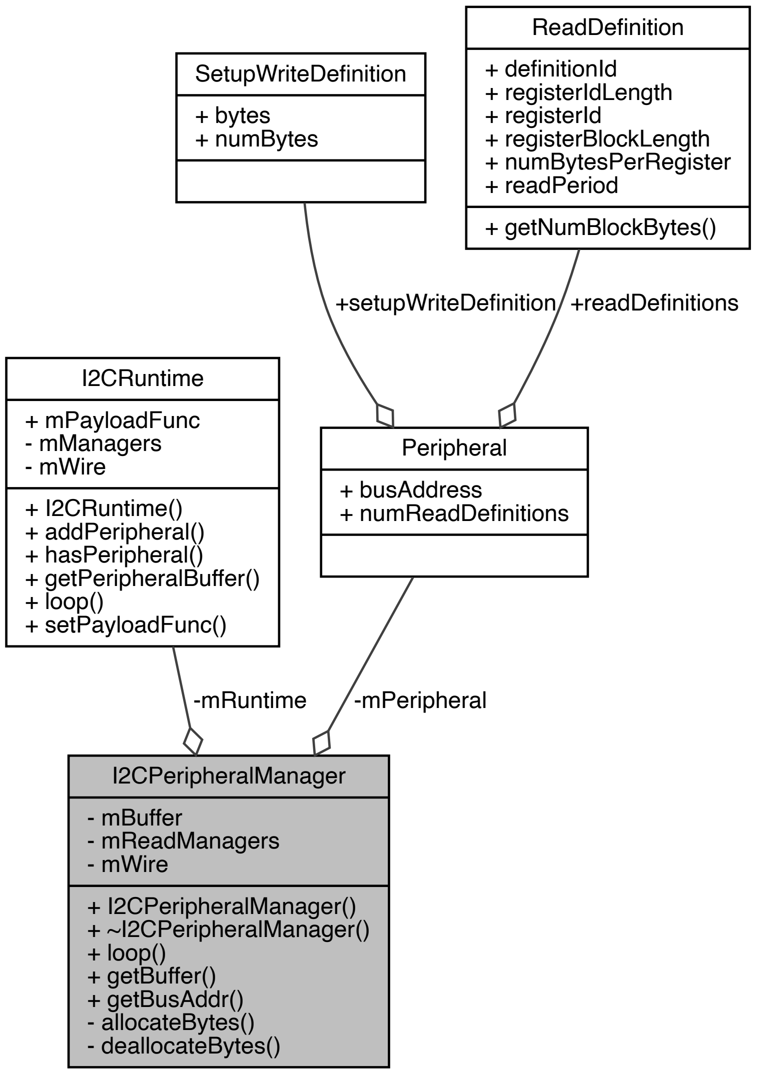
- include/I2CManager.h
- src/I2CManager.cpp

3.2 I2CPeripheralManager Class Reference

Manages all the read managers and memory allocation for a single peripheral.

```
#include <I2CRuntime.h>
```

Collaboration diagram for I2CPeripheralManager:



Public Member Functions

- I2CPeripheralManager (Peripheral ∗peripheral, TwoWire ∗wire, I2CRuntime ∗runtime)
- ∼I2CPeripheralManager ()
- void loop ()
- uint8_t ∗∗ getBuffer ()
- uint8_t getBusAddr ()

Static Private Member Functions

- static uint8_t ∗∗ allocateBytes (Peripheral ∗)
- static void deallocateBytes (Peripheral ∗, uint8_t ∗∗)

Private Attributes

- Peripheral ∗ mPeripheral
- uint8_t ∗∗ mBuffer
- std::vector< I2CReadManager ∗ > mReadManagers
- TwoWire ∗ mWire
- I2CRuntime ∗ mRuntime

3.2.1 Detailed Description

Manages all the read managers and memory allocation for a single peripheral.

Only intended to be used by an I2CRuntime instance.

A peripheral manager will manage the read managers for each read definition on a single peripheral.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 I2CPeripheralManager()

```
I2CPeripheralManager::I2CPeripheralManager (
            Peripheral ∗ peripheral,
            TwoWire ∗ wire,
            I2CRuntime ∗ runtime )
```

3.2.2.2 ∼I2CPeripheralManager()

```
I2CPeripheralManager::∼I2CPeripheralManager ( )
```

3.2.3 Member Function Documentation

3.2.3.1 allocateBytes()

```
uint8_t ∗∗ I2CPeripheralManager::allocateBytes (
            Peripheral ∗ peripheral )  [static], [private]
```

3.2.3.2 deallocateBytes()

```
void I2CPeripheralManager::deallocateBytes (
            Peripheral * peripheral,
            uint8_t ** bytes )  [static], [private]
```

3.2.3.3 getBuffer()

```
uint8_t ** I2CPeripheralManager::getBuffer ( )
```

3.2.3.4 getBusAddr()

```
uint8_t I2CPeripheralManager::getBusAddr ( )
```

3.2.3.5 loop()

```
void I2CPeripheralManager::loop ( )
```

3.2.4 Member Data Documentation

3.2.4.1 mBuffer

```
uint8_t** I2CPeripheralManager::mBuffer  [private]
```

3.2.4.2 mPeripheral

```
Peripheral* I2CPeripheralManager::mPeripheral  [private]
```

3.2.4.3 mReadManagers

```
std::vector<I2CReadManager *> I2CPeripheralManager::mReadManagers  [private]
```

3.2.4.4 mRuntime

```
I2CRuntime* I2CPeripheralManager::mRuntime  [private]
```

3.2.4.5 mWire

```
TwoWire* I2CPeripheralManager::mWire  [private]
```

The documentation for this class was generated from the following files:

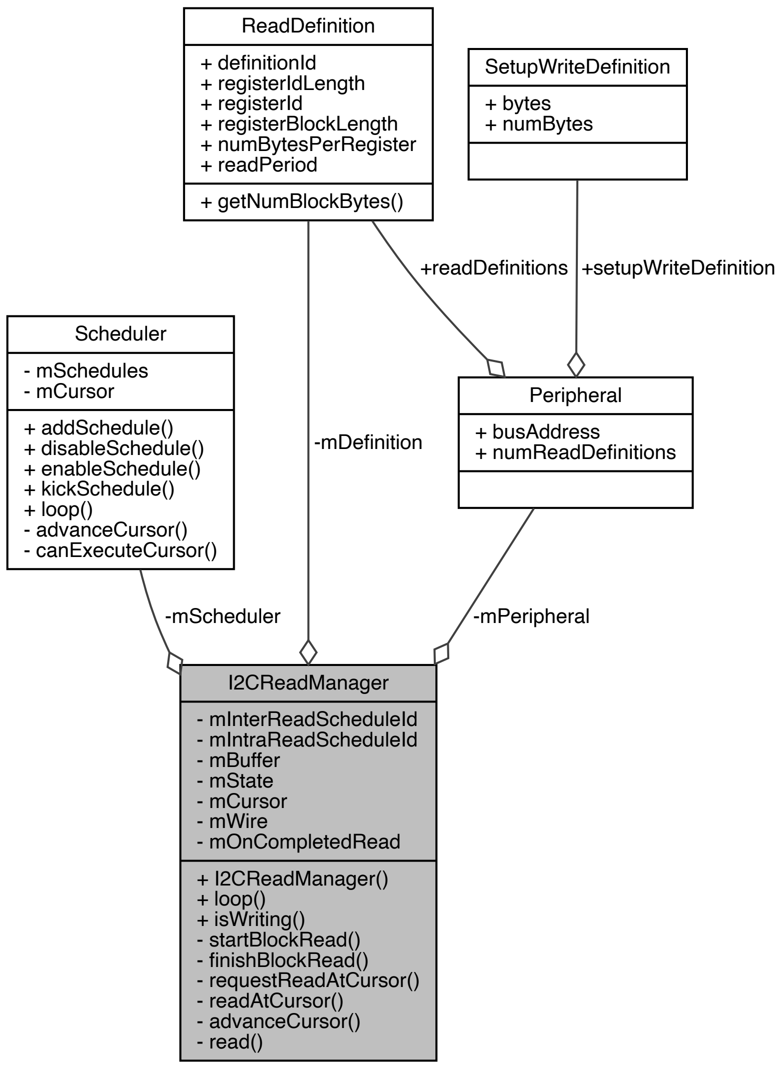
- include/I2CRuntime.h
- src/I2CRuntime.cpp

3.3 I2CReadManager Class Reference

Responsible for a state machine which reads data from the I2C bus in a non-blocking fashion.

```
#include <I2CReadManager.h>
```

Collaboration diagram for I2CReadManager:



Public Member Functions

- I2CReadManager (ReadDefinition ∗, Peripheral ∗, uint8_t ∗, TwoWire ∗, std::shared_ptr< Func >)
- void loop ()
- bool isWriting ()

Private Member Functions

- void startBlockRead ()
- void finishBlockRead ()
- void requestReadAtCursor ()
- void readAtCursor ()
- void advanceCursor ()
- void read ()

Private Attributes

- ReadDefinition ∗ mDefinition
- Peripheral ∗ mPeripheral
- Scheduler mScheduler
- ScheduleId mInterReadScheduleId
- ScheduleId mIntraReadScheduleId
- uint8_t ∗ mBuffer
- ReadManagerState mState
- uint16_t mCursor
- TwoWire ∗ mWire
- std::shared_ptr< Func > mOnCompletedRead

3.3.1 Detailed Description

Responsible for a state machine which reads data from the I2C bus in a non-blocking fashion.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 I2CReadManager()

```
I2CReadManager::I2CReadManager (
            ReadDefinition ∗ readDefinition,
            Peripheral ∗ peripheral,
            uint8_t ∗ buffer,
            TwoWire ∗ wire,
            std::shared_ptr< Func > onCompletedReadCallback )
```

3.3.3 Member Function Documentation

3.3.3.1 advanceCursor()

```
void I2CReadManager::advanceCursor ( ) [private]
```

3.3.3.2 finishBlockRead()

```
void I2CReadManager::finishBlockRead ( ) [private]
```

We've finished reading a from a contiguous block of register IDs and can disable the intra-read schedule and switch back to the inter-read schedule.

3.3.3.3 isWriting()

```
bool I2CReadManager::isWriting ( )
```

3.3.3.4 loop()

```
void I2CReadManager::loop ( )
```

3.3.3.5 read()

```
void I2CReadManager::read ( ) [private]
```

3.3.3.6 readAtCursor()

```
void I2CReadManager::readAtCursor ( ) [private]
```

3.3.3.7 requestReadAtCursor()

```
void I2CReadManager::requestReadAtCursor ( ) [private]
```

3.3.3.8 startBlockRead()

```
void I2CReadManager::startBlockRead ( ) [private]
```

3.3.4 Member Data Documentation

3.3.4.1 mBuffer

```
uint8_t* I2CReadManager::mBuffer [private]
```

The buffer for the whole peripheral is a 2D array that manages bytes per read definition. A single ReadManager instance only manages a single ReaDefinition. This is the buffer for the single instance.

3.3.4.2 mCursor

```
uint16_t I2CReadManager::mCursor [private]
```

3.3.4.3 mDefinition

```
ReadDefinition* I2CReadManager::mDefinition [private]
```

3.3.4.4 mInterReadScheduleId

```
ScheduleId I2CReadManager::mInterReadScheduleId [private]
```

3.3.4.5 mIntraReadScheduleId

```
ScheduleId I2CReadManager::mIntraReadScheduleId [private]
```

3.3.4.6 mOnCompletedRead

```
std::shared_ptr<Func> I2CReadManager::mOnCompletedRead [private]
```

3.3.4.7 mPeripheral

```
Peripheral* I2CReadManager::mPeripheral [private]
```

3.3.4.8 mScheduler

```
Scheduler I2CReadManager::mScheduler [private]
```

3.3.4.9 mState

```
ReadManagerState I2CReadManager::mState [private]
```

3.3.4.10 mWire

```
TwoWire* I2CReadManager::mWire [private]
```

The documentation for this class was generated from the following files:

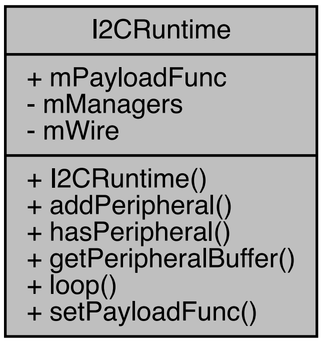
- include/I2CReadManager.h
- src/I2CReadManager.cpp

3.4 I2CRuntime Class Reference

Responsible for the primary event loop for all peripherals on the bus.

```
#include <I2CRuntime.h>
```

Collaboration diagram for I2CRuntime:



Public Member Functions

- I2CRuntime (TwoWire $*$)
- std::size_t addPeripheral (Peripheral $*$peripheral)
- bool hasPeripheral (std::size_t)
- uint8_t $**$ getPeripheralBuffer (std::size_t)
- void loop ()
- void setPayloadFunc (std::shared_ptr$<$ PayloadFunc $>$)

Public Attributes

- std::shared_ptr$<$ PayloadFunc $>$ mPayloadFunc

Private Attributes

- std::vector< I2CPeripheralManager * > mManagers
- TwoWire * mWire

3.4.1 Detailed Description

Responsible for the primary event loop for all peripherals on the bus.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 I2CRuntime()

```
I2CRuntime::I2CRuntime (
            TwoWire * wire )
```

3.4.3 Member Function Documentation

3.4.3.1 addPeripheral()

```
std::size_t I2CRuntime::addPeripheral (
            Peripheral * peripheral )
```

3.4.3.2 getPeripheralBuffer()

```
uint8_t ** I2CRuntime::getPeripheralBuffer (
            std::size_t peripheralId )
```

3.4.3.3 hasPeripheral()

```
bool I2CRuntime::hasPeripheral (
            std::size_t peripheralId )
```

3.4.3.4 loop()

```
void I2CRuntime::loop ( )
```

3.4.3.5 setPayloadFunc()

```
void I2CRuntime::setPayloadFunc (
            std::shared_ptr< PayloadFunc > func )
```

3.4.4 Member Data Documentation

3.4.4.1 mManagers

```
std::vector<I2CPeripheralManager *> I2CRuntime::mManagers  [private]
```

3.4.4.2 mPayloadFunc

```
std::shared_ptr<PayloadFunc> I2CRuntime::mPayloadFunc
```

3.4.4.3 mWire

```
TwoWire* I2CRuntime::mWire  [private]
```

The documentation for this class was generated from the following files:

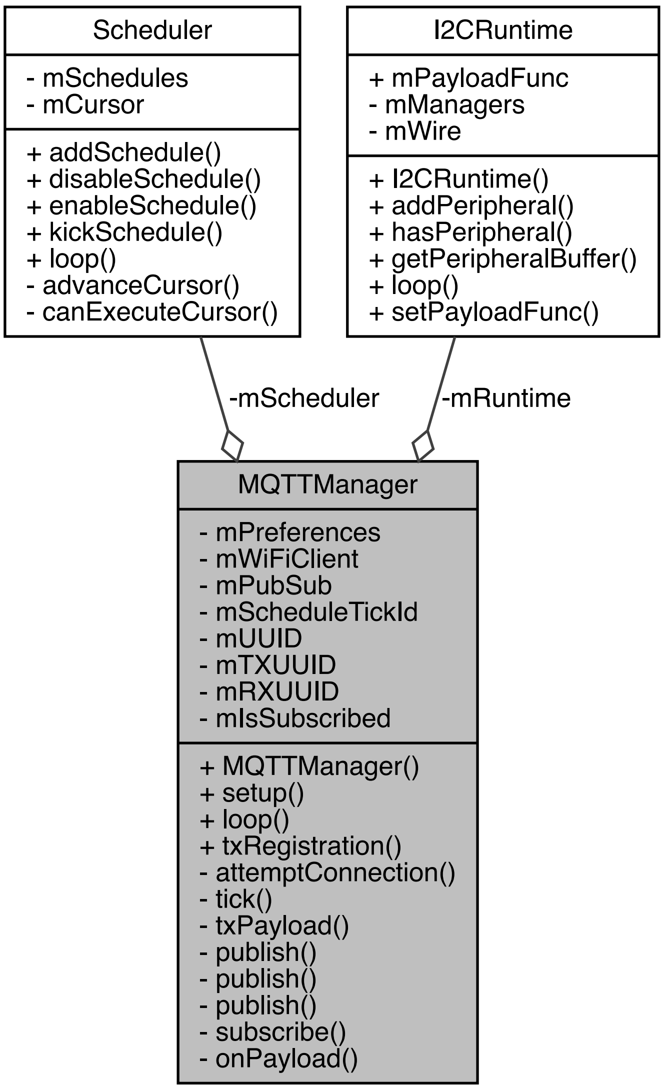
- include/I2CRuntime.h
- src/I2CRuntime.cpp

3.5 MQTTManager Class Reference

Responsible for publishing and subscribing to topics on the MQTT broker.

```
#include <MQTTManager.h>
```

Collaboration diagram for MQTTManager:



Public Member Functions

- MQTTManager (I2CRuntime &runtime)
- void setup ()
- void loop ()
- void txRegistration (std::vector< PeripheralStatus > ∗statuses)

Private Member Functions

- void attemptConnection ()
- void tick ()
- void txPayload (uint32_t busId, uint16_t busAddress, ReadDefinition ∗def, uint8_t ∗payload)
- bool publish (uint8_t ∗payload, unsigned int len)
- bool publish (char ∗payload)
- bool publish (std::string payload)
- void subscribe ()
- void onPayload (char ∗topic, uint8_t ∗payload, unsigned int size)

Private Attributes

- I2CRuntime & mRuntime
- Preferences mPreferences
- WiFiClient mWiFiClient
- PubSubClient mPubSub
- Scheduler mScheduler
- ScheduleId mScheduleTickId
- char mUUID [13]
- char mTXUUID [16]
- char mRXUUID [16]
- bool mIsSubscribed

3.5.1 Detailed Description

Responsible for publishing and subscribing to topics on the MQTT broker.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 MQTTManager()

```
MQTTManager::MQTTManager (
            I2CRuntime & runtime )
```

3.5.3 Member Function Documentation

3.5.3.1 attemptConnection()

```
void MQTTManager::attemptConnection ( )  [private]
```

3.5.3.2 loop()

```
void MQTTManager::loop ( )
```

3.5.3.3 onPayload()

```
void MQTTManager::onPayload (
            char * topic,
            uint8_t * payload,
            unsigned int size )  [private]
```

3.5.3.4 publish() [1/3]

```
bool MQTTManager::publish (
            uint8_t * payload,
            unsigned int len )  [private]
```

3.5.3.5 publish() [2/3]

```
bool MQTTManager::publish (
            char * payload )  [private]
```

3.5.3.6 publish() [3/3]

```
bool MQTTManager::publish (
            std::string payload )  [private]
```

3.5.3.7 setup()

```
void MQTTManager::setup ( )
```

3.5.3.8 subscribe()

```
void MQTTManager::subscribe ( )  [private]
```

3.5.3.9 tick()

```
void MQTTManager::tick ( ) [private]
```

3.5.3.10 txPayload()

```
void MQTTManager::txPayload (
            uint32_t busId,
            uint16_t busAddress,
            ReadDefinition * def,
            uint8_t * payload ) [private]
```

3.5.3.11 txRegistration()

```
void MQTTManager::txRegistration (
            std::vector< PeripheralStatus > * statuses )
```

3.5.4 Member Data Documentation

3.5.4.1 mIsSubscribed

```
bool MQTTManager::mIsSubscribed [private]
```

3.5.4.2 mPreferences

```
Preferences MQTTManager::mPreferences [private]
```

3.5.4.3 mPubSub

```
PubSubClient MQTTManager::mPubSub [private]
```

3.5.4.4 mRuntime

```
I2CRuntime& MQTTManager::mRuntime  [private]
```

3.5.4.5 mRXUUID

```
char MQTTManager::mRXUUID[16]  [private]
```

3.5.4.6 mScheduler

```
Scheduler MQTTManager::mScheduler  [private]
```

3.5.4.7 mScheduleTickId

```
ScheduleId MQTTManager::mScheduleTickId  [private]
```

3.5.4.8 mTXUUID

```
char MQTTManager::mTXUUID[16]  [private]
```

3.5.4.9 mUUID

```
char MQTTManager::mUUID[13]  [private]
```

3.5.4.10 mWiFiClient

```
WiFiClient MQTTManager::mWiFiClient  [private]
```

The documentation for this class was generated from the following files:

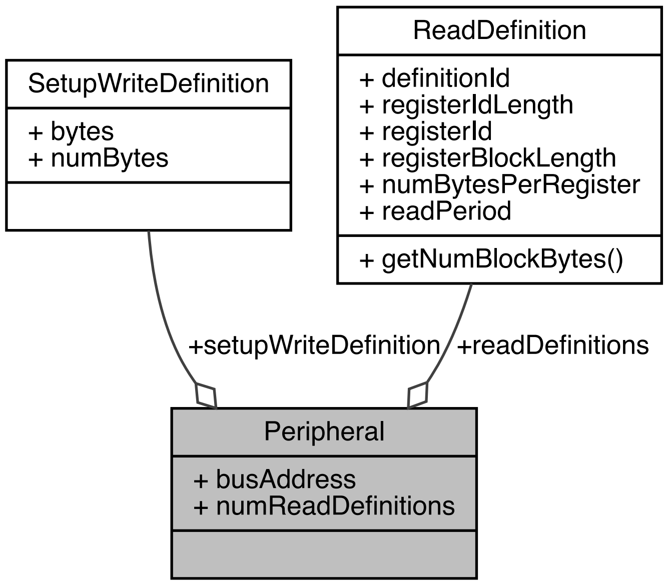
- include/MQTTManager.h
- src/MQTTManager.cpp

3.6 Peripheral Struct Reference

Declarative configuration for the behaviour of a peripheral on the I2C bus.

`#include <I2CPeripheral.h>`

Collaboration diagram for Peripheral:



Public Attributes

- uint16_t busAddress
- SetupWriteDefinition ∗ setupWriteDefinition
- uint8_t numReadDefinitions
- ReadDefinition ∗∗ readDefinitions

3.6.1 Detailed Description

Declarative configuration for the behaviour of a peripheral on the I2C bus.

3.6.2 Member Data Documentation

3.6.2.1 busAddress

`uint16_t Peripheral::busAddress`

The address on the I2C bus. There is only one bus address per peripheral. This is not tied to `ReadDefinition`s.

3.6.2.2 numReadDefinitions

`uint8_t Peripheral::numReadDefinitions`

A Peripheral can define multiple things to read. e.g., a peripheral may provide accelerometer and gyroscope data at non-contiguous register IDs.

3.6.2.3 readDefinitions

`ReadDefinition** Peripheral::readDefinitions`

3.6.2.4 setupWriteDefinition

`SetupWriteDefinition* Peripheral::setupWriteDefinition`

Some peripherals require configuration in the form of bytes sent to the peripheral at startup.

The documentation for this struct was generated from the following file:

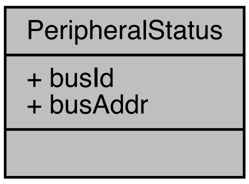
- include/I2CPeripheral.h

3.7 PeripheralStatus Struct Reference

Intermediate object for protobuf encoding.

`#include <TelemetryProtocol.h>`

Collaboration diagram for PeripheralStatus:



Public Attributes

- uint32_t busId
- uint32_t busAddr

3.7.1 Detailed Description

Intermediate object for protobuf encoding.

3.7.2 Member Data Documentation

3.7.2.1 busAddr

```
uint32_t PeripheralStatus::busAddr
```

3.7.2.2 busId

```
uint32_t PeripheralStatus::busId
```

The documentation for this struct was generated from the following file:

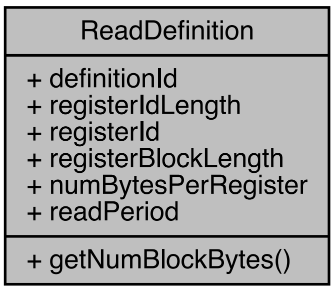
- include/TelemetryProtocol.h

3.8 ReadDefinition Struct Reference

Declarative configuration for reading a contiguous register block from a peripheral on the I2C bus.

```
#include <I2CPeripheral.h>
```

Collaboration diagram for ReadDefinition:



Public Member Functions

- uint16_t getNumBlockBytes ()

Public Attributes

- uint16_t definitionId
- RegisterLength registerIdLength
- uint16_t registerId
- uint8_t registerBlockLength
- uint8_t numBytesPerRegister
- Duration readPeriod

3.8.1 Detailed Description

Declarative configuration for reading a contiguous register block from a peripheral on the I2C bus.

3.8.2 Member Function Documentation

3.8.2.1 getNumBlockBytes()

```
uint16_t ReadDefinition::getNumBlockBytes ( )
```

3.8.3 Member Data Documentation

3.8.3.1 definitionId

```
uint16_t ReadDefinition::definitionId
```

An arbitrary ID for external reference. This does not relate to the hardware or any application logic in the I2CRuntime. This is used externally to track read definitions.

3.8.3.2 numBytesPerRegister

```
uint8_t ReadDefinition::numBytesPerRegister
```

The number of bytes that will be read at each register ID in the contiguous block. This means that the total number of bytes retrieved from one ReadDefinition instance is:

```
numBytesPerRegister * registerBlockLength
```

3.8.3.3 readPeriod

```
Duration ReadDefinition::readPeriod
```

How many milliseconds between reading all bytes from the block of registers?

3.8.3.4 registerBlockLength

```
uint8_t ReadDefinition::registerBlockLength
```

Data is read on this output from a contiguous block of register IDs. e.g., 0x80 to 0xFF. This defines the number of register IDs in the block. This is essentially how many times the loop will read bytes at a register and then advance to the next register. For many peripherals this value is simple 1 (i.e., there is no need to advance).

3.8.3.5 registerId

```
uint16_t ReadDefinition::registerId
```

Data is read on this output from a contiguous block of register IDs. e.g., 0x80 to 0xFF. This is the first register ID of the block.

3.8.3.6 registerIdLength

```
RegisterLength ReadDefinition::registerIdLength
```

Some peripherals have 16-bit register IDs and some have 8-bit register IDs.

The documentation for this struct was generated from the following files:

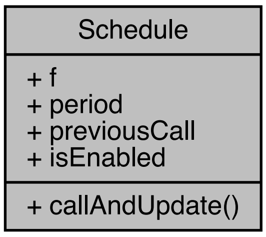
- include/I2CPeripheral.h
- src/I2CRuntime.cpp

3.9 Schedule Struct Reference

Declarative configuration for a managed schedule.

```
#include <Scheduler.h>
```

Collaboration diagram for Schedule:



Public Member Functions

- void callAndUpdate ()

Public Attributes

- std::shared_ptr< Func > f
- Duration period
- Timestamp previousCall
- bool isEnabled

3.9.1 Detailed Description

Declarative configuration for a managed schedule.

3.9.2 Member Function Documentation

3.9.2.1 callAndUpdate()

```
void Schedule::callAndUpdate ( )
```

3.9.3 Member Data Documentation

3.9.3.1 f

```
std::shared_ptr<Func> Schedule::f
```

3.9.3.2 isEnabled

```
bool Schedule::isEnabled
```

3.9.3.3 period

```
Duration Schedule::period
```

How many milliseconds between executions of `f`?

3.9.3.4 previousCall

`Timestamp Schedule::previousCall`

When was the previous call of `f`? If `f` has never been called then this will be `HAS_NEVER_RAN_TIMESTAMP`.

The documentation for this struct was generated from the following files:

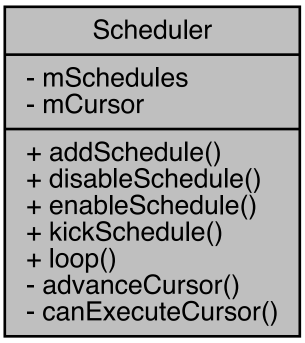
- include/Scheduler.h
- src/Scheduler.cpp

3.10 Scheduler Class Reference

Generic scheduler for non-blocking tasks given schedules for callbacks.

`#include <Scheduler.h>`

Collaboration diagram for Scheduler:



Public Member Functions

- ScheduleId addSchedule (std::shared_ptr< Func >, Duration, bool isEnabled=true)
- void disableSchedule (ScheduleId)
- void enableSchedule (ScheduleId)
- void kickSchedule (ScheduleId)
- void loop ()

Private Member Functions

- void advanceCursor ()
- bool canExecuteCursor ()

Private Attributes

- std::vector< Schedule > mSchedules
- std::vector< Schedule >::iterator mCursor

3.10.1 Detailed Description

Generic scheduler for non-blocking tasks given schedules for callbacks.

3.10.2 Member Function Documentation

3.10.2.1 addSchedule()

```
ScheduleId Scheduler::addSchedule (
             std::shared_ptr< Func > f,
             Duration period,
             bool isEnabled = true )
```

3.10.2.2 advanceCursor()

```
void Scheduler::advanceCursor ( )  [private]
```

3.10.2.3 canExecuteCursor()

```
bool Scheduler::canExecuteCursor ( )  [private]
```

3.10.2.4 disableSchedule()

```
void Scheduler::disableSchedule (
             ScheduleId id )
```

3.10.2.5 enableSchedule()

```
void Scheduler::enableSchedule (
             ScheduleId id )
```

3.10.2.6 kickSchedule()

```
void Scheduler::kickSchedule (
            ScheduleId id )
```

Update the schedule such that it will run after its period relative to the call of `kickSchedule`. e.g., if `kick←Schedule` is called at t = 40 with a period of 500 then the schedule will be executed at t = 540.

3.10.2.7 loop()

```
void Scheduler::loop ( )
```

This call should be non-blocking and consume as little CPU time as possible per execution. `mCursor` is used to cycle through the schedules and check one schedule for execution per call.

3.10.3 Member Data Documentation

3.10.3.1 mCursor

```
std::vector<Schedule>::iterator Scheduler::mCursor  [private]
```

3.10.3.2 mSchedules

```
std::vector<Schedule> Scheduler::mSchedules  [private]
```

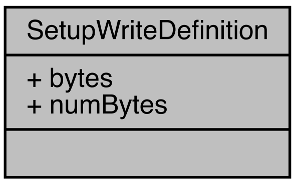
The documentation for this class was generated from the following files:

- include/Scheduler.h
- src/Scheduler.cpp

3.11 SetupWriteDefinition Struct Reference

`#include <I2CPeripheral.h>`

Collaboration diagram for SetupWriteDefinition:



Public Attributes

- uint8_t ∗ bytes
- uint8_t numBytes

3.11.1 Member Data Documentation

3.11.1.1 bytes

```
uint8_t* SetupWriteDefinition::bytes
```

3.11.1.2 numBytes

```
uint8_t SetupWriteDefinition::numBytes
```

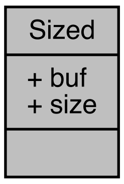
The documentation for this struct was generated from the following file:

- include/I2CPeripheral.h

3.12 Sized Struct Reference

Buffer/size pair.

Collaboration diagram for Sized:



Public Attributes

- const uint8_t ∗ buf
- size_t size

3.12.1 Detailed Description

Buffer/size pair.

3.12.2 Member Data Documentation

3.12.2.1 buf

```
const uint8_t* Sized::buf
```

3.12.2.2 size

```
size_t Sized::size
```

The documentation for this struct was generated from the following file:

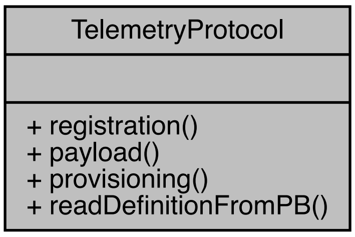
- src/TelemetryProtocol.cc

3.13 TelemetryProtocol Class Reference

Responsible for encoding and decoding protobuf/nanopb payloads.

```
#include <TelemetryProtocol.h>
```

Collaboration diagram for TelemetryProtocol:



Static Public Member Functions

- static size_t registration (std::vector< PeripheralStatus > ∗statuses, uint8_t ∗buffer)
- static size_t payload (uint32_t busId, uint16_t busAddress, ReadDefinition ∗def, uint8_t ∗payload, uint8_t ∗buffer)
- static Peripheral ∗ provisioning (uint8_t ∗buffer, unsigned int size)
- static ReadDefinition ∗ readDefinitionFromPB (Provisioning_ReadDef &msg)

3.13.1 Detailed Description

Responsible for encoding and decoding protobuf/nanopb payloads.

3.13.2 Member Function Documentation

3.13.2.1 payload()

```
size_t TelemetryProtocol::payload (
            uint32_t busId,
            uint16_t busAddress,
            ReadDefinition ∗ def,
            uint8_t ∗ payload,
            uint8_t ∗ buffer )  [static]
```

3.13.2.2 provisioning()

```
Peripheral ∗ TelemetryProtocol::provisioning (
            uint8_t ∗ buffer,
            unsigned int size )  [static]
```

3.13.2.3 readDefinitionFromPB()

```
ReadDefinition ∗ TelemetryProtocol::readDefinitionFromPB (
            Provisioning_ReadDef & msg )  [static]
```

3.13.2.4 registration()

```
size_t TelemetryProtocol::registration (
            std::vector< PeripheralStatus > ∗ statuses,
            uint8_t ∗ buffer )  [static]
```

The documentation for this class was generated from the following files:

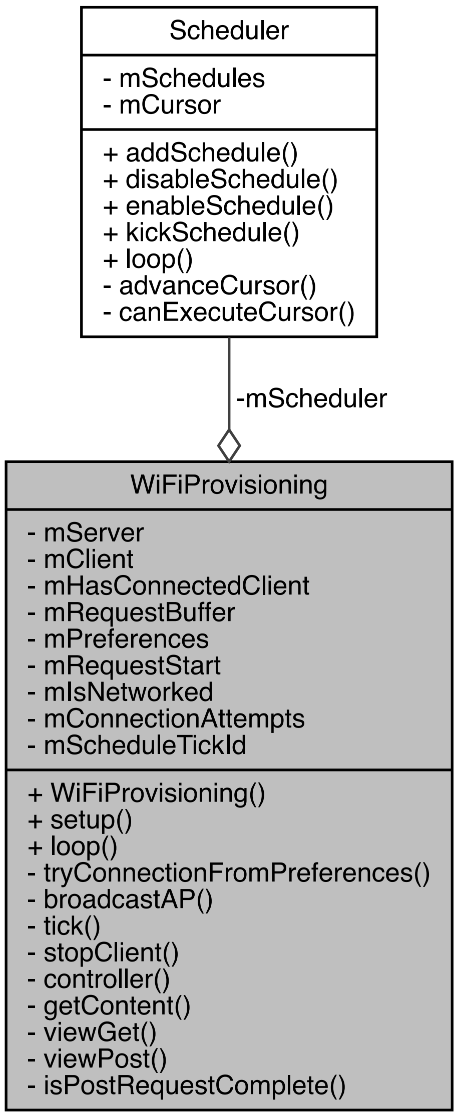
- include/TelemetryProtocol.h
- src/TelemetryProtocol.cc

3.14 WiFiProvisioning Class Reference

Manages the web application for provisioning connectivity information.

```
#include <WiFiProvisioning.h>
```

Collaboration diagram for WiFiProvisioning:



Public Member Functions

- WiFiProvisioning ()
- void setup ()
- void loop ()

Private Member Functions

- bool tryConnectionFromPreferences ()
- void broadcastAP ()
- void tick ()
- void stopClient ()
- void controller ()
- std::string getContent ()
- void viewGet ()
- void viewPost ()
- bool isPostRequestComplete ()

Private Attributes

- WiFiServer mServer
- WiFiClient mClient
- bool mHasConnectedClient
- std::string mRequestBuffer
- Preferences mPreferences
- size_t mRequestStart
- bool mIsNetworked
- uint8_t mConnectionAttempts
- Scheduler mScheduler
- ScheduleId mScheduleTickId

3.14.1 Detailed Description

Manages the web application for provisioning connectivity information.

Responsiblities:

```
- Connecting to an SSID if one has been stored in Flash memory
  (Preferences).
- Broadcasting an AP if a network cannot be successfully connected to.
- Serving a webpage that allows configuration of WiFi and MQTT
  parameters.
- Writing Preferences to Flash memory.
```

3.14.2 Constructor & Destructor Documentation

3.14.2.1 WiFiProvisioning()

```
WiFiProvisioning::WiFiProvisioning ( )
```

3.14.3 Member Function Documentation

3.14.3.1 broadcastAP()

```
void WiFiProvisioning::broadcastAP ( ) [private]
```

3.14.3.2 controller()

```
void WiFiProvisioning::controller ( ) [private]
```

[Web Server] Parse an incoming request and dispatch to either the GET or POST view if we've received a complete request.

3.14.3.3 getContent()

```
std::string WiFiProvisioning::getContent ( ) [private]
```

[Web Server] Get the form page and populate default values from Flash memory.

3.14.3.4 isPostRequestComplete()

```
bool WiFiProvisioning::isPostRequestComplete ( ) [private]
```

[Web Server] Determine if we've received Content-Length amount of payload.

3.14.3.5 loop()

```
void WiFiProvisioning::loop ( )
```

3.14.3.6 setup()

```
void WiFiProvisioning::setup ( )
```

3.14.3.7 stopClient()

```
void WiFiProvisioning::stopClient ( ) [private]
```

[Web Server]

3.14.3.8 tick()

```
void WiFiProvisioning::tick ( )  [private]
```

Check to see if we're connected yet. If we exceed the connection attempts then we should bail and just broadcast an AP.

3.14.3.9 tryConnectionFromPreferences()

```
bool WiFiProvisioning::tryConnectionFromPreferences ( )  [private]
```

Returns false if SSID and password are not set.

3.14.3.10 viewGet()

```
void WiFiProvisioning::viewGet ( )  [private]
```

[Web Server]

3.14.3.11 viewPost()

```
void WiFiProvisioning::viewPost ( )  [private]
```

[Web Server]

3.14.4 Member Data Documentation

3.14.4.1 mClient

```
WiFiClient WiFiProvisioning::mClient  [private]
```

3.14.4.2 mConnectionAttempts

```
uint8_t WiFiProvisioning::mConnectionAttempts  [private]
```

3.14.4.3 mHasConnectedClient

```
bool WiFiProvisioning::mHasConnectedClient  [private]
```

3.14.4.4 mIsNetworked

```
bool WiFiProvisioning::mIsNetworked  [private]
```

Are we connected to a network or broadcasting an AP?

3.14.4.5 mPreferences

```
Preferences WiFiProvisioning::mPreferences  [private]
```

3.14.4.6 mRequestBuffer

```
std::string WiFiProvisioning::mRequestBuffer  [private]
```

3.14.4.7 mRequestStart

```
size_t WiFiProvisioning::mRequestStart  [private]
```

Allow us to timeout a HTTP request.

3.14.4.8 mScheduler

```
Scheduler WiFiProvisioning::mScheduler  [private]
```

3.14.4.9 mScheduleTickId

```
ScheduleId WiFiProvisioning::mScheduleTickId  [private]
```

3.14.4.10 mServer

```
WiFiServer WiFiProvisioning::mServer  [private]
```

The documentation for this class was generated from the following files:

- include/WiFiProvisioning.h
- src/WiFiProvisioning.cpp

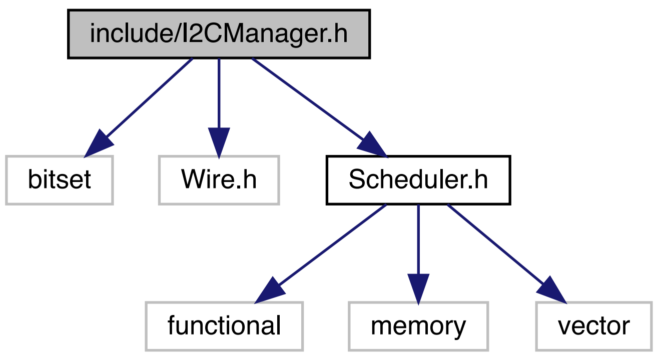
Chapter 4

File Documentation

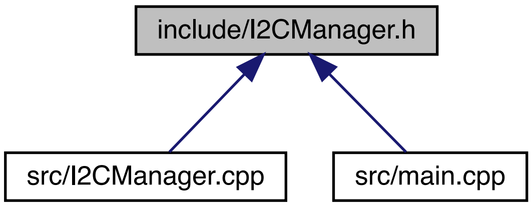
4.1 include/I2CManager.h File Reference

```
#include <bitset>
#include <Wire.h>
#include "Scheduler.h"
```

Include dependency graph for I2CManager.h:



This graph shows which files directly or indirectly include this file:



Classes

- class I2CManager

  Responsible for discovering the connectivity status of I2C peripherals.

Macros

- #define I2CMANAGER_DEFAULT_INTER_SCAN_PERIOD 1000
- #define I2CMANAGER_DEFAULT_INTRA_SCAN_PERIOD 10

4.1.1 Macro Definition Documentation

4.1.1.1 I2CMANAGER_DEFAULT_INTER_SCAN_PERIOD

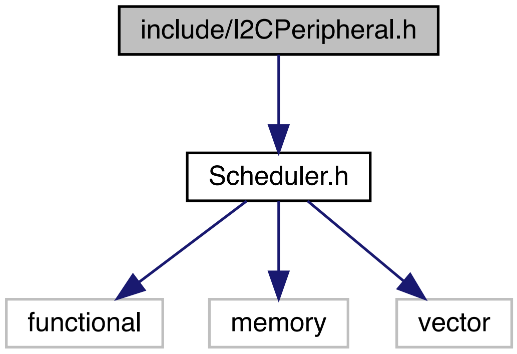
```
#define I2CMANAGER_DEFAULT_INTER_SCAN_PERIOD 1000
```

4.1.1.2 I2CMANAGER_DEFAULT_INTRA_SCAN_PERIOD

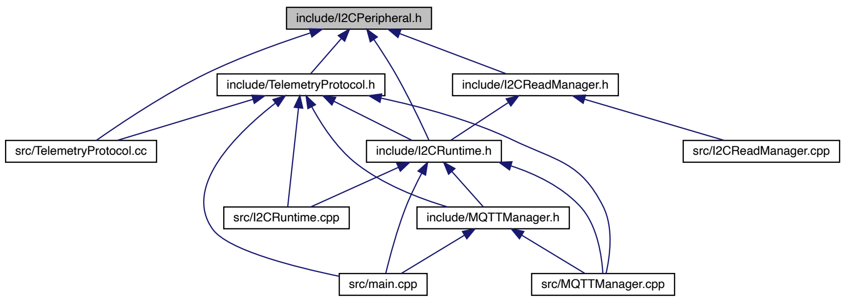
```
#define I2CMANAGER_DEFAULT_INTRA_SCAN_PERIOD 10
```

4.2 include/I2CPeripheral.h File Reference

`#include "Scheduler.h"`
Include dependency graph for I2CPeripheral.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct ReadDefinition

    Declarative configuration for reading a contiguous register block from a peripheral on the I2C bus.
- struct SetupWriteDefinition
- struct Peripheral

    Declarative configuration for the behaviour of a peripheral on the I2C bus.

Macros

- #define REGISTER_REQ_DELAY_MILLI 15

Typedefs

- typedef struct ReadDefinition ReadDefinition
- typedef struct SetupWriteDefinition SetupWriteDefinition
- typedef struct Peripheral Peripheral

Enumerations

- enum RegisterLength { RL16, RL8 }

4.2.1 Macro Definition Documentation

4.2.1.1 REGISTER_REQ_DELAY_MILLI

```
#define REGISTER_REQ_DELAY_MILLI 15
```

This value can be very important.

Too low: Insufficient time between the register ID transmission and a request to read bytes. No data will be read.

Too high: Performance will suffer for read definitions that have a long contiguous register block length. (e.g., something like the thermal camera that reads from 128 different registers.)

4.2.2 Typedef Documentation

4.2.2.1 Peripheral

```
typedef struct Peripheral Peripheral
```

4.2.2.2 ReadDefinition

```
typedef struct ReadDefinition ReadDefinition
```

4.2.2.3 SetupWriteDefinition

```
typedef struct SetupWriteDefinition SetupWriteDefinition
```

4.2.3 Enumeration Type Documentation

4.2.3.1 RegisterLength

```
enum RegisterLength
```

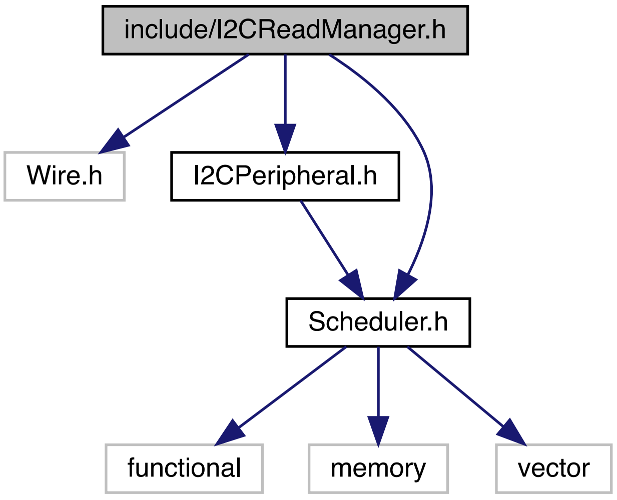
Enumerator

RL16	
RL8	

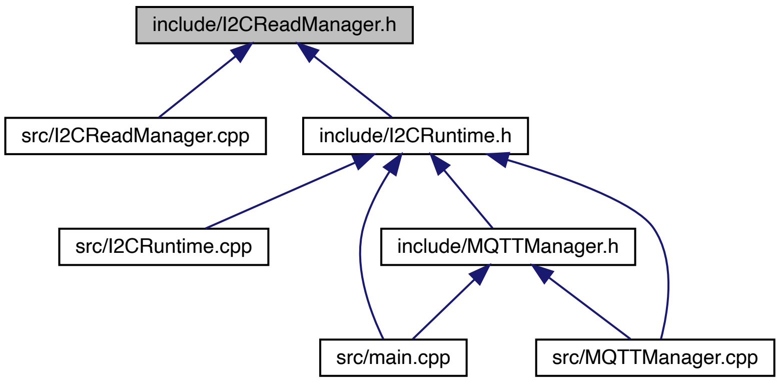
4.3 include/I2CReadManager.h File Reference

```
#include <Wire.h>
#include "I2CPeripheral.h"
#include "Scheduler.h"
```

Include dependency graph for I2CReadManager.h:



This graph shows which files directly or indirectly include this file:



Classes

- class I2CReadManager

  Responsible for a state machine which reads data from the I2C bus in a non-blocking fashion.

Enumerations

- enum ReadManagerState { NOT_READING_BLOCK, REQUESTING_SINGLE_READ, REQUESTED_SINGLE_READ }

4.3.1 Enumeration Type Documentation

4.3.1.1 ReadManagerState

```
enum ReadManagerState
```

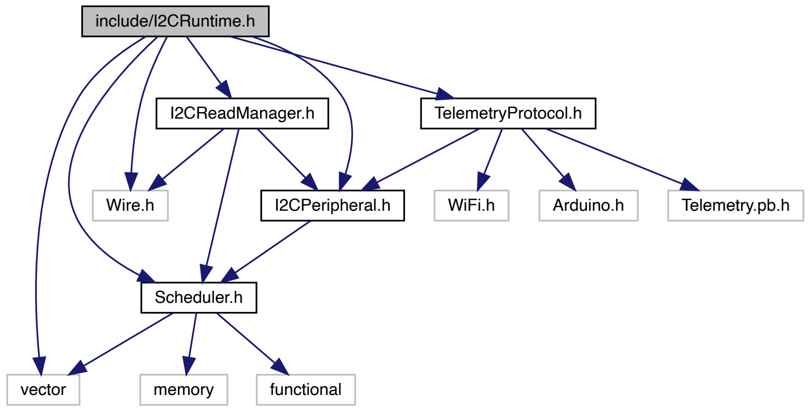
Enumerator

NOT_READING_BLOCK	We haven't started reading a block of register IDs from the peripheral.
REQUESTING_SINGLE_READ	We're going to inform the peripheral which register we want to perform a read on.
REQUESTED_SINGLE_READ	We're going to request the bytes and read them and then either advance the cursor and go to REQUESTING_SINGLE_READ if the block isn't finished or go to NOT_READING_BLOCK and reset the schedules if the block is finished.

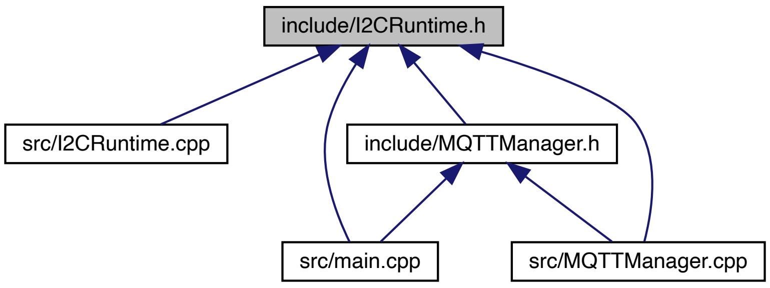
4.4 include/I2CRuntime.h File Reference

```
#include <vector>
#include <Wire.h>
#include "I2CPeripheral.h"
#include "I2CReadManager.h"
#include "Scheduler.h"
#include "TelemetryProtocol.h"
```

Include dependency graph for I2CRuntime.h:



This graph shows which files directly or indirectly include this file:



Classes

- class I2CPeripheralManager

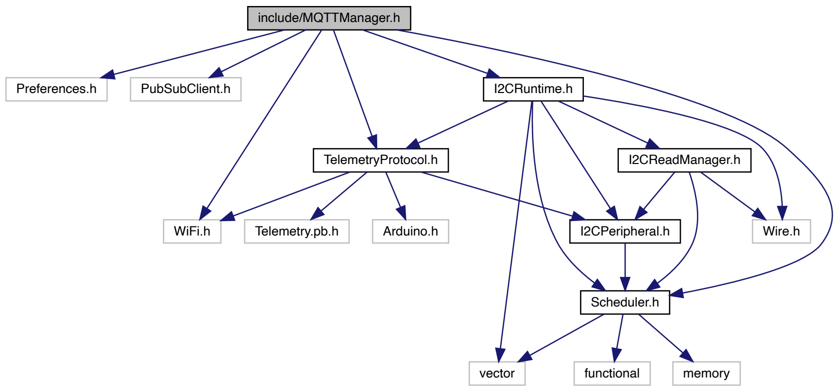
    Manages all the read managers and memory allocation for a single peripheral.
- class I2CRuntime

    Responsible for the primary event loop for all peripherals on the bus.

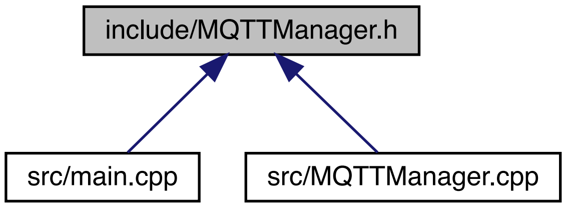
4.5 include/MQTTManager.h File Reference

```
#include <Preferences.h>
#include <PubSubClient.h>
#include <WiFi.h>
#include "I2CRuntime.h"
#include "Scheduler.h"
```

```
#include "TelemetryProtocol.h"
```
Include dependency graph for MQTTManager.h:



This graph shows which files directly or indirectly include this file:



Classes

- class MQTTManager

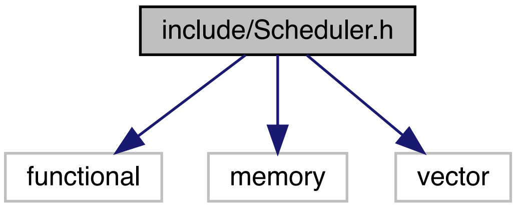
  Responsible for publishing and subscribing to topics on the MQTT broker.

4.6 include/Scheduler.h File Reference

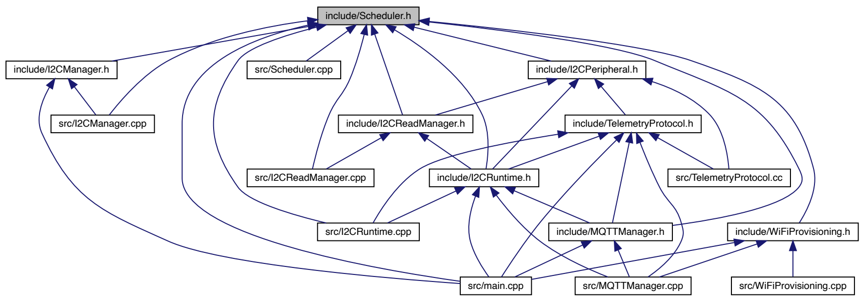
```
#include <functional>
#include <memory>
```

```
#include <vector>
```

Include dependency graph for Scheduler.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct Schedule

    Declarative configuration for a managed schedule.
- class Scheduler

    Generic scheduler for non-blocking tasks given schedules for callbacks.

Macros

- #define HAS_NEVER_RAN_TIMESTAMP 0

Typedefs

- typedef uint32_t Milli
- typedef Milli Duration
- typedef Milli Timestamp
- typedef std::function< void()> Func
- typedef std::size_t ScheduleId

4.6.1 Macro Definition Documentation

4.6.1.1 HAS_NEVER_RAN_TIMESTAMP

```
#define HAS_NEVER_RAN_TIMESTAMP 0
```

4.6.2 Typedef Documentation

4.6.2.1 Duration

```
typedef Milli Duration
```

4.6.2.2 Func

```
typedef std::function<void ()> Func
```

4.6.2.3 Milli

```
typedef uint32_t Milli
```

4.6.2.4 ScheduleId

```
typedef std::size_t ScheduleId
```

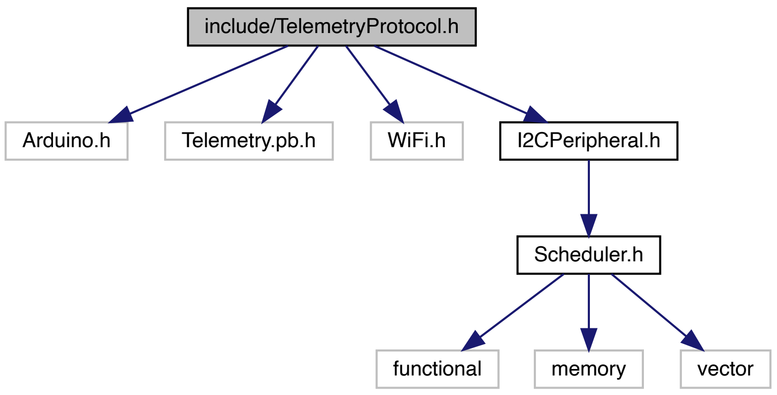
4.6.2.5 Timestamp

```
typedef Milli Timestamp
```

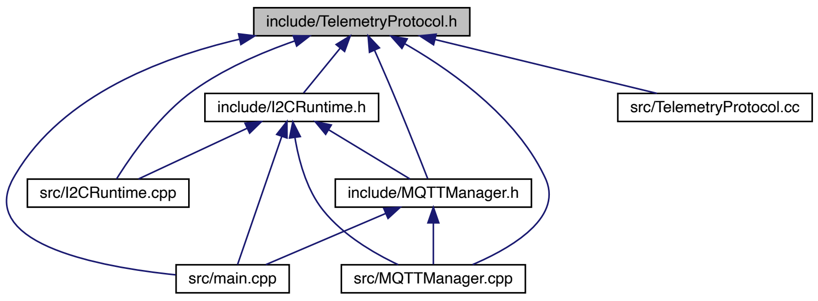
4.7 include/TelemetryProtocol.h File Reference

```
#include <Arduino.h>
#include <Telemetry.pb.h>
#include <WiFi.h>
#include "I2CPeripheral.h"
```

Include dependency graph for TelemetryProtocol.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct PeripheralStatus

  Intermediate object for protobuf encoding.
- class TelemetryProtocol

  Responsible for encoding and decoding protobuf/nanopb payloads.

Typedefs

- typedef std::function< void(uint32_t, uint16_t, ReadDefinition ∗, uint8_t ∗)> PayloadFunc
- typedef struct PeripheralStatus PeripheralStatus

4.7.1 Typedef Documentation

4.7.1.1 PayloadFunc

```
typedef std::function<void (uint32_t, uint16_t, ReadDefinition *, uint8_t *)> PayloadFunc
```

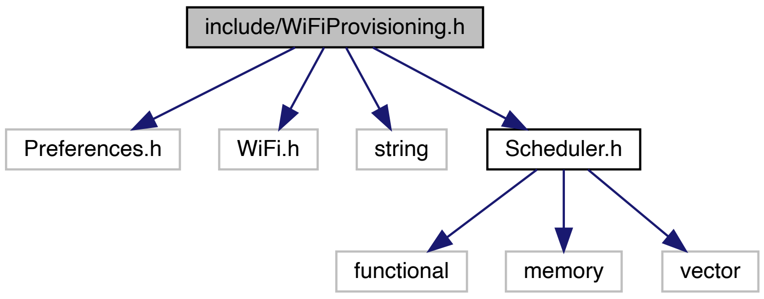
4.7.1.2 PeripheralStatus

```
typedef struct PeripheralStatus PeripheralStatus
```

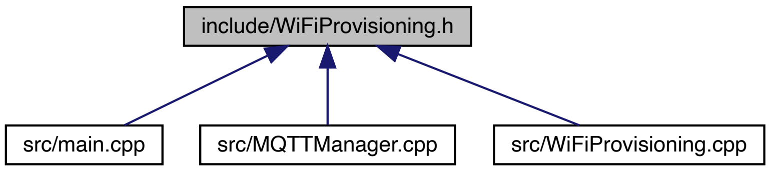
4.8 include/WiFiProvisioning.h File Reference

```
#include <Preferences.h>
#include <WiFi.h>
#include <string>
#include "Scheduler.h"
```

Include dependency graph for WiFiProvisioning.h:



This graph shows which files directly or indirectly include this file:



Classes

- class WiFiProvisioning

  Manages the web application for provisioning connectivity information.

Macros

- #define PROVISIONING_AP_SSID "TelemetryMicrocontroller"
- #define PROVISIONING_AP_PASS "4zp6capstone"
- #define PROVISIONING_CONNECTION_ATTEMPT_LIMIT 10
- #define PREFERENCES_NAMESPACE "telemetry"
- #define WEB_SERVER_TIMEOUT 1000

Functions

- void replace (std::string &haystack, std::string needle, const char ∗replacement)

4.8.1 Macro Definition Documentation

4.8.1.1 PREFERENCES_NAMESPACE

```
#define PREFERENCES_NAMESPACE "telemetry"
```

4.8.1.2 PROVISIONING_AP_PASS

```
#define PROVISIONING_AP_PASS "4zp6capstone"
```

4.8.1.3 PROVISIONING_AP_SSID

```
#define PROVISIONING_AP_SSID "TelemetryMicrocontroller"
```

4.8.1.4 PROVISIONING_CONNECTION_ATTEMPT_LIMIT

```
#define PROVISIONING_CONNECTION_ATTEMPT_LIMIT 10
```

4.8.1.5 WEB_SERVER_TIMEOUT

```
#define WEB_SERVER_TIMEOUT 1000
```

4.8.2 Function Documentation

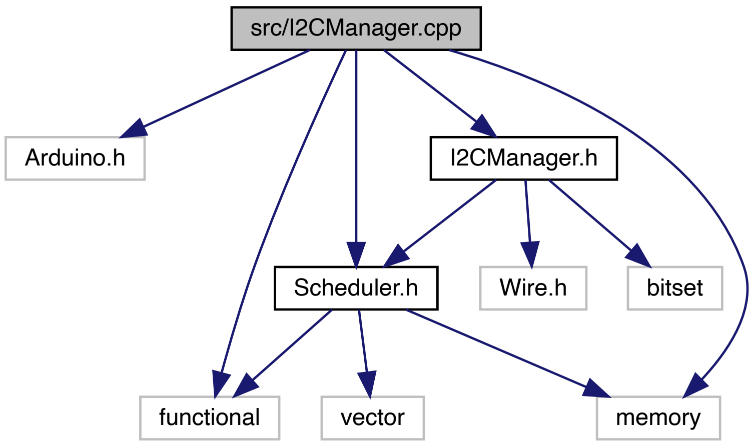
4.8.2.1 replace()

```
void replace (
            std::string & haystack,
            std::string needle,
            const char * replacement )
```

4.9 src/I2CManager.cpp File Reference

```
#include <Arduino.h>
#include <functional>
#include <memory>
#include "I2CManager.h"
#include "Scheduler.h"
```

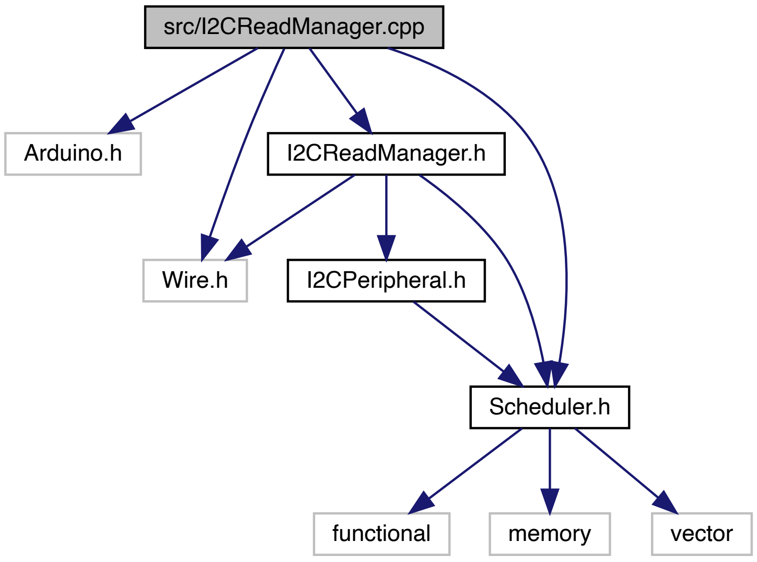
Include dependency graph for I2CManager.cpp:



4.10 src/I2CReadManager.cpp File Reference

```
#include <Arduino.h>
#include <Wire.h>
#include "I2CReadManager.h"
#include "Scheduler.h"
```

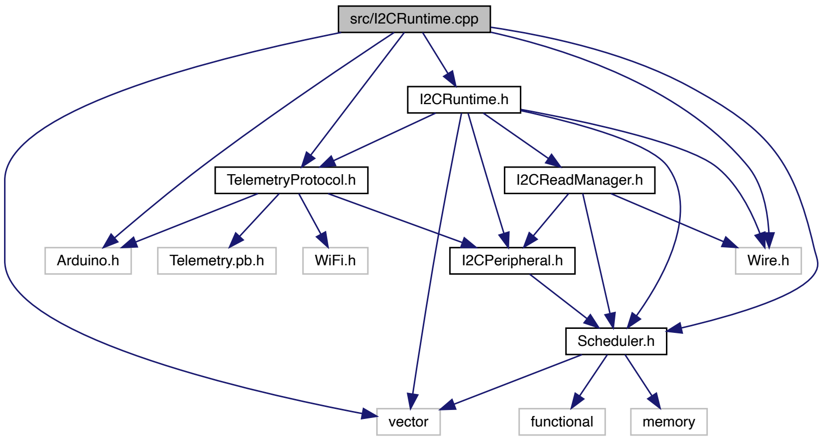
Include dependency graph for I2CReadManager.cpp:



4.11 src/I2CRuntime.cpp File Reference

```
#include <Arduino.h>
#include <vector>
#include <Wire.h>
#include "I2CRuntime.h"
#include "Scheduler.h"
#include "TelemetryProtocol.h"
```

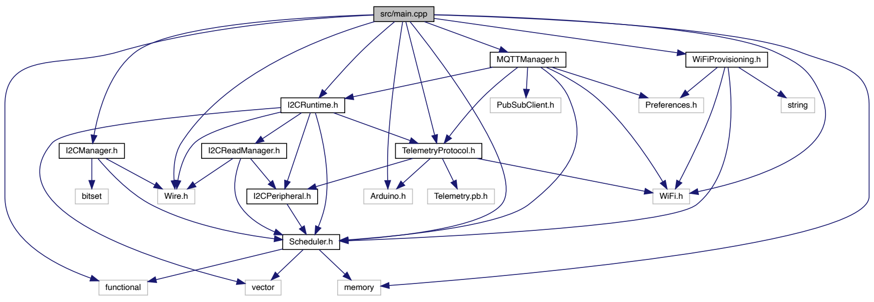
Include dependency graph for I2CRuntime.cpp:



4.12 src/main.cpp File Reference

```
#include <Arduino.h>
#include <functional>
#include <memory>
#include <WiFi.h>
#include <Wire.h>
#include "I2CManager.h"
#include "I2CRuntime.h"
#include "MQTTManager.h"
#include "Scheduler.h"
#include "WiFiProvisioning.h"
#include "TelemetryProtocol.h"
```

Include dependency graph for main.cpp:



Macros

- #define ENABLE_PROVISIONING
- #define ENABLE_PERIPHERALS
- #define ENABLE_MQTT

Functions

- void setup ()
- void loop ()

Variables

- TwoWire ∗ wire = &Wire
- Scheduler scheduler
- I2CRuntime runtime (wire)
- I2CManager manager (wire)
- uint8_t ∗∗ shtBuffer = NULL
- WiFiProvisioning provisioning
- MQTTManager mqttManager (runtime)

4.12.1 Macro Definition Documentation

4.12.1.1 ENABLE_MQTT

```
#define ENABLE_MQTT
```

4.12.1.2 ENABLE_PERIPHERALS

```
#define ENABLE_PERIPHERALS
```

4.12.1.3 ENABLE_PROVISIONING

```
#define ENABLE_PROVISIONING
```

4.12.2 Function Documentation

4.12.2.1 loop()

```
void loop ( )
```

4.12.2.2 setup()

```
void setup ( )
```

4.12.3 Variable Documentation

4.12.3.1 manager

```
I2CManager manager(wire)
```

4.12.3.2 mqttManager

```
MQTTManager mqttManager(runtime)
```

4.12.3.3 provisioning

```
WiFiProvisioning provisioning
```

4.12.3.4 runtime

```
I2CRuntime runtime(wire)
```

4.12.3.5 scheduler

```
Scheduler scheduler
```

4.12.3.6 shtBuffer

```
uint8_t** shtBuffer = NULL
```

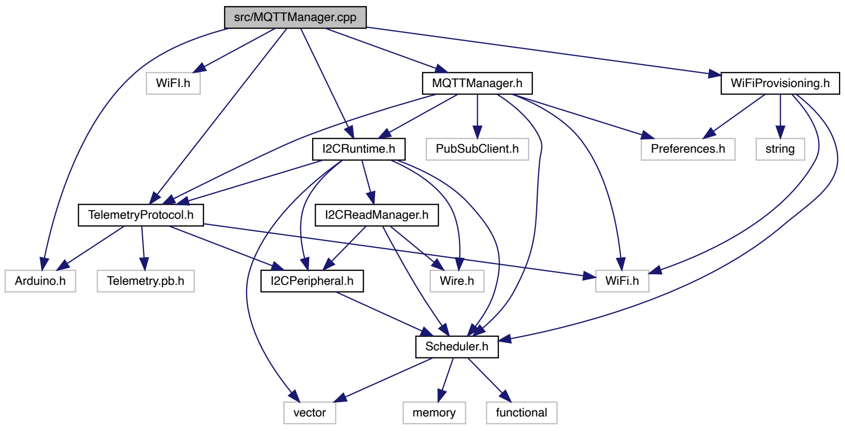
4.12.3.7 wire

```
TwoWire* wire = &Wire
```

4.13 src/MQTTManager.cpp File Reference

```
#include <Arduino.h>
#include <WiFI.h>
#include "I2CRuntime.h"
#include "MQTTManager.h"
#include "TelemetryProtocol.h"
#include "WiFiProvisioning.h"
```

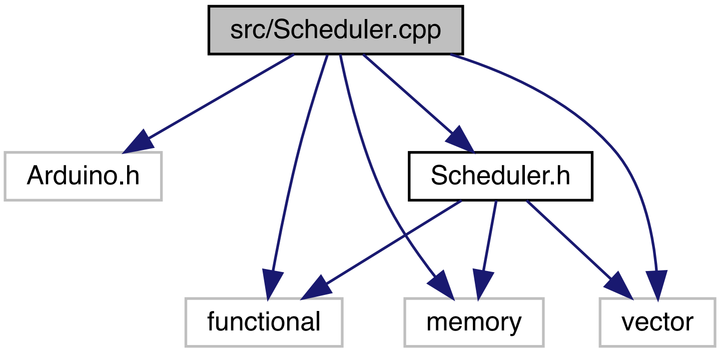
Include dependency graph for MQTTManager.cpp:



4.14 src/Scheduler.cpp File Reference

```
#include <Arduino.h>
#include <functional>
#include <memory>
#include <vector>
#include "Scheduler.h"
```

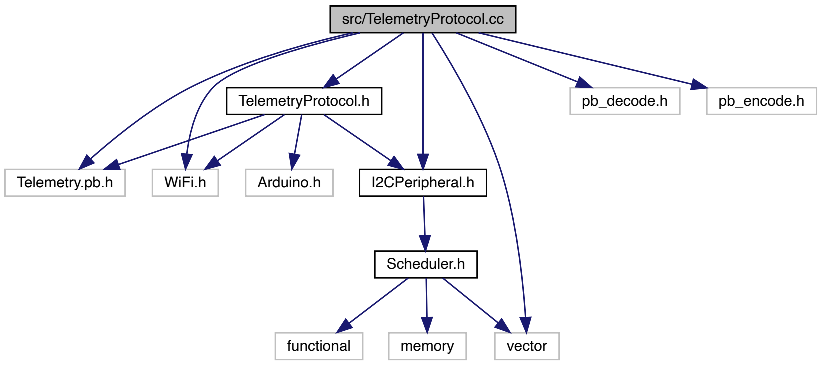
Include dependency graph for Scheduler.cpp:



4.15 src/TelemetryProtocol.cc File Reference

```
#include <Telemetry.pb.h>
#include <WiFi.h>
#include <pb_decode.h>
#include <pb_encode.h>
#include <vector>
#include "I2CPeripheral.h"
#include "TelemetryProtocol.h"
```

Include dependency graph for TelemetryProtocol.cc:



Classes

- struct Sized

  Buffer/size pair.

Typedefs

- typedef struct Sized Sized

Functions

- bool encode_string (pb_ostream_t ∗stream, const pb_field_t ∗field, void ∗const ∗arg)
- bool encode_statuses (pb_ostream_t ∗stream, const pb_field_t ∗field, void ∗const ∗arg)

4.15.1 Typedef Documentation

4.15.1.1 Sized

```
typedef struct Sized Sized
```

4.15.2 Function Documentation

4.15.2.1 encode_statuses()

```
bool encode_statuses (
            pb_ostream_t ∗ stream,
            const pb_field_t ∗ field,
            void ∗const ∗ arg )
```

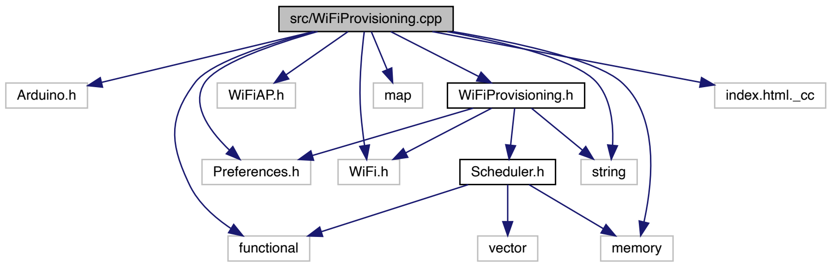
4.15.2.2 encode_string()

```
bool encode_string (
            pb_ostream_t ∗ stream,
            const pb_field_t ∗ field,
            void ∗const ∗ arg )
```

4.16 src/WiFiProvisioning.cpp File Reference

```
#include <Arduino.h>
#include <Preferences.h>
#include <WiFi.h>
#include <WiFiAP.h>
#include <functional>
#include <map>
#include <memory>
#include <string>
#include "WiFiProvisioning.h"
#include "index.html._cc"
```

Include dependency graph for WiFiProvisioning.cpp:



Functions

- void replace (std::string &haystack, std::string needle, const char *replacement)

4.16.1 Function Documentation

4.16.1.1 replace()

```
void replace (
            std::string & haystack,
            std::string needle,
            const char * replacement )
```

Index