**Modbus Stack - API Guide**

**Version: 1.0**

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning SS USA Inc. products described herein. You agree to grant SS USA Inc. a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

All information provided here is subject to change without notice.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

No computer system can be absolutely secure.

\*Other names and brands may be claimed as the property of others.

Copyright © 2021, SS USA Inc. All rights reserved.

**Table of Contents**

[1 Introduction 5](#_Toc73430126)

[1.1 Purpose 5](#_Toc73430127)

[1.2 Abbreviations 5](#_Toc73430128)

[1.3 Document Overview / Structure 5](#_Toc73430129)

[2 Modbus Stack Introduction 6](#_Toc73430130)

[2.1 Features of Modbus Stack 6](#_Toc73430131)

[3 Building Stack 7](#_Toc73430132)

[3.1 System requirements 7](#_Toc73430133)

[3.2 Compiling stack 7](#_Toc73430134)

[3.2.1 Pre-requisite for compilation 7](#_Toc73430135)

[3.3 Exported APIs 7](#_Toc73430136)

[4 Data structures 8](#_Toc73430137)

[4.1 Data type declarations 8](#_Toc73430138)

[4.2 Error codes 8](#_Toc73430139)

[4.3 Modbus function codes 9](#_Toc73430140)

[4.4 Structs definitions 10](#_Toc73430141)

[4.4.1 stException\_t 10](#_Toc73430142)

[4.4.2 stTimeStamps 10](#_Toc73430143)

[4.4.3 stDevConfig\_t 11](#_Toc73430144)

[4.4.4 stMbusAppCallbackParams\_t 11](#_Toc73430145)

[4.4.5 Structures related to read file record 12](#_Toc73430146)

[4.4.6 Structures related to write file record 13](#_Toc73430147)

[4.4.7 Structures related to read device identification 13](#_Toc73430148)

[5 Macro description 15](#_Toc73430149)

[6 API description 16](#_Toc73430150)

[6.1 API – AppMbusMaster\_StackInit 16](#_Toc73430151)

[6.1.1 Description 16](#_Toc73430152)

[6.1.2 Prototype 16](#_Toc73430153)

[6.1.3 Parameter description 16](#_Toc73430154)

[6.1.4 Return type 16](#_Toc73430155)

[6.2 API – AppMbusMaster\_StackDeInit 16](#_Toc73430156)

[6.2.1 Description 16](#_Toc73430157)

[6.2.2 Prototype 16](#_Toc73430158)

[6.2.3 Parameter description 16](#_Toc73430159)

[6.2.4 Return type 16](#_Toc73430160)

[6.3 API – AppMbusMaster\_ removeCtx 16](#_Toc73430161)

[6.3.1 Description 16](#_Toc73430162)

[6.3.2 Prototype 16](#_Toc73430163)

[6.3.3 Parameter description 17](#_Toc73430164)

[6.3.4 Return type 17](#_Toc73430165)

[6.4 API – AppMbusMaster\_ getTCPCtx 17](#_Toc73430166)

[6.4.1 Description 17](#_Toc73430167)

[6.4.2 Prototype 17](#_Toc73430168)

[6.4.3 Parameter description 17](#_Toc73430169)

[6.4.4 Return type 17](#_Toc73430170)

[6.5 API – AppMbusMaster\_ getRTUCtx 17](#_Toc73430171)

[6.5.1 Description 17](#_Toc73430172)

[6.5.2 Prototype 17](#_Toc73430173)

[6.5.3 Parameter description 17](#_Toc73430174)

[6.5.4 Return type 18](#_Toc73430175)

[6.6 API – initSerialPort 18](#_Toc73430176)

[6.6.1 Description 18](#_Toc73430177)

[6.6.2 Prototype 18](#_Toc73430178)

[6.6.3 Parameter description 18](#_Toc73430179)

[6.6.4 Return type 18](#_Toc73430180)

[6.7 API – AppMbusMaster\_SetStackConfigParam 18](#_Toc73430181)

[6.7.1 Description 18](#_Toc73430182)

[6.7.2 Prototype 18](#_Toc73430183)

[6.7.3 Parameter description 19](#_Toc73430184)

[6.7.4 Return type 19](#_Toc73430185)

[6.8 API – AppMbusMaster\_GetStackConfigParam 19](#_Toc73430186)

[6.8.1 Description 19](#_Toc73430187)

[6.8.2 Prototype 19](#_Toc73430188)

[6.8.3 Parameter description 19](#_Toc73430189)

[6.8.4 Return type 19](#_Toc73430190)

[6.9 API – Modbus\_Read\_Coils 19](#_Toc73430191)

[6.9.1 Description 19](#_Toc73430192)

[6.9.2 Prototype 19](#_Toc73430193)

[6.9.3 Parameter description 19](#_Toc73430194)

[6.9.4 Return type 20](#_Toc73430195)

[6.9.5 Callback Function 20](#_Toc73430196)

[6.10 API – Modbus\_Read\_Discrete\_Inputs 20](#_Toc73430197)

[6.10.1 Description 20](#_Toc73430198)

[6.10.2 Prototype 21](#_Toc73430199)

[6.10.3 Parameter description 21](#_Toc73430200)

[6.10.4 Return type 21](#_Toc73430201)

[6.10.5 Callback Function 21](#_Toc73430202)

[6.11 API – Modbus\_Read\_Holding\_Registers 22](#_Toc73430203)

[6.11.1 Description 22](#_Toc73430204)

[6.11.2 Prototype 22](#_Toc73430205)

[6.11.3 Parameter description 22](#_Toc73430206)

[6.11.4 Return type 22](#_Toc73430207)

[6.11.5 Callback Function 22](#_Toc73430208)

[6.12 API – Modbus\_Read\_Input\_Registers 23](#_Toc73430209)

[6.12.1 Description 23](#_Toc73430210)

[6.12.2 Prototype 23](#_Toc73430211)

[6.12.3 Parameter description 23](#_Toc73430212)

[6.12.4 Return type 23](#_Toc73430213)

[6.12.5 Callback Function 23](#_Toc73430214)

[6.13 API – Modbus\_Write\_Single\_Coil 24](#_Toc73430215)

[6.13.1 Description 24](#_Toc73430216)

[6.13.2 Prototype 24](#_Toc73430217)

[6.13.3 Parameter description 24](#_Toc73430218)

[6.13.4 Return type 25](#_Toc73430219)

[6.13.5 Callback Function 25](#_Toc73430220)

[6.14 API – Modbus\_Write\_Single\_Register 25](#_Toc73430221)

[6.14.1 Description 25](#_Toc73430222)

[6.14.2 Prototype 25](#_Toc73430223)

[6.14.3 Parameter description 25](#_Toc73430224)

[6.14.4 Return type 26](#_Toc73430225)

[6.14.5 Callback Function 26](#_Toc73430226)

[6.15 API – Modbus\_Write\_Multiple\_Coils 26](#_Toc73430227)

[6.15.1 Description 26](#_Toc73430228)

[6.15.2 Prototype 26](#_Toc73430229)

[6.15.3 Parameter description 26](#_Toc73430230)

[6.15.4 Return type 27](#_Toc73430231)

[6.15.5 Callback Function 27](#_Toc73430232)

[6.16 API – Modbus\_Write\_Multiple\_Register 27](#_Toc73430233)

[6.16.1 Description 27](#_Toc73430234)

[6.16.2 Prototype 27](#_Toc73430235)

[6.16.3 Parameter description 28](#_Toc73430236)

[6.16.4 Return type 28](#_Toc73430237)

[6.16.5 Callback Function 28](#_Toc73430238)

[6.17 API – Modbus\_Read\_File\_Record 29](#_Toc73430239)

[6.17.1 Description 29](#_Toc73430240)

[6.17.2 Prototype 29](#_Toc73430241)

[6.17.3 Parameter description 29](#_Toc73430242)

[6.17.4 Return type 29](#_Toc73430243)

[6.17.5 Callback Function 29](#_Toc73430244)

[6.18 API – Modbus\_Write\_File\_Record 30](#_Toc73430245)

[6.18.1 Description 30](#_Toc73430246)

[6.18.2 Prototype 30](#_Toc73430247)

[6.18.3 Parameter description 31](#_Toc73430248)

[6.18.4 Return type 31](#_Toc73430249)

[6.18.5 Callback Function 31](#_Toc73430250)

[6.19 API – Modbus\_Read\_Write\_Registers 32](#_Toc73430251)

[6.19.1 Description 32](#_Toc73430252)

[6.19.2 Prototype 32](#_Toc73430253)

[6.19.3 Parameter description 32](#_Toc73430254)

[6.19.4 Return type 33](#_Toc73430255)

[6.19.5 Callback Function 33](#_Toc73430256)

[6.20 API – Modbus\_Read\_Device\_Identification 33](#_Toc73430257)

[6.20.1 Description 33](#_Toc73430258)

[6.20.2 Prototype 33](#_Toc73430259)

[6.20.3 Parameter description 34](#_Toc73430260)

[6.20.4 Return type 34](#_Toc73430261)

[6.20.5 Callback Function 34](#_Toc73430262)

[7 Using APIs in Application 36](#_Toc73430263)

[7.1 Exported Files 36](#_Toc73430264)

# Introduction

Modconn Modbus Stack is a multi-threaded library that incorporates the functionalities of Modbus protocol. The multi-threaded library comes in form of a library (.so) on Linux platform.

Complete implementation of Modbus stack is in ‘C language’ and as per the standards.

## Purpose

This document provides information on how to use Modconn Modbus stack for developing Modbus applications. This document has information related to usage of Modbus stack’s APIs.

## Abbreviations

| **#** | **Abbreviation** | **Details** |
| --- | --- | --- |
| 1 | API | Application Programming Interface |
| 2 | MEI | Modbus Encapsulated Interface |
| 3 | RTU | Remote Terminal Unit |
| 4 | TCP | Transmission Control Protocol/Internet protocol |

## Document Overview / Structure

This document contains different chapters that explain few details about stack interfaces and its usage by application. This document includes chapters that explain below things:

1. Modbus stacks introduction
2. Building stack
3. Data structures
4. API description
5. Using APIs in Application

Hereafter the word ‘stack’ or ‘Stack’ means Modconn Modbus Stack.

# Modbus Stack Introduction

Modconn Modbus Stack is a **multi-threaded library** that incorporates the functionalities of Modbus protocol. The multi-threaded library comes in form of a library (.so) on Linux platform.

Complete implementation of Modbus stack is in ‘**C** **language**’ and as per the standards.

## Features of Modbus Stack

Following are few of the features that Modconn Modbus stack supports.

1. Supports Modbus TCP Master and RTU Master Functionality.
2. Can be ported on various OS / RTOS.
3. Easy to interface Exported APIs.
4. Configurable stack settings.
5. Modular design and easy customization.
6. Scalable design.

# Building Stack

This section explains how to build the stack.

## System requirements

The design and development of Modbus stack involves usage of tools, libraries, operating system, etc. mentioned below.

* 1. OS: Ubuntu 16.04 or later
  2. GCC: 5.4.0
  3. Safe string library

## Compiling stack

Use make-files to compile stack source-code on Linux platform. The make-files can be found at path mentioned below:

To compile stack in debug mode - **SoftMod\_Stack\Debug\makefile**

To compile stack in release mode - **SoftMod\_Stack\Release\makefile**

Use command “**make clean all**” on the terminal to compile the library.

Post successful compilation, the output binary file (libModbusMasterStack.so) is created in the folder where the makefile is present (i.e. Debug or Release).

### Pre-requisite for compilation

To build the library for Modbus TCP mode, a preprocessor “MODBUS\_STACK\_TCPIP\_ENABLED” shall be added in “**Release\Src\subdir.mk**” (or “**Debug\Src\subdir.mk**”) file in “**gcc**” command as shown below:

gcc -std=c11 **-DMODBUS\_STACK\_TCPIP\_ENABLED** -I ….

To build the library for Modbus RTU mode, the preprocessor “MODBUS\_STACK\_TCPIP\_ENABLED” should be removed from above mentioned location.

## Exported APIs

Exported APIs are the part of Modbus stack library that can be used by application to communicate with Modbus enabled devices. Below are the types, in which these APIs are categorized.

1. Stack Initialization and de-initialization
2. Configuration settings
3. Modbus read values from slave devices
4. Modbus write data to slave devices

# Data structures

This section describes data types, structs, error codes and enumerations defined in Modbus stack library.

## Data type declarations

| **#** | **Custom defined data types** | **Modbus data types** |
| --- | --- | --- |
| 1 | unsigned char | uint8\_t |
| 2 | signed char | int8\_t |
| 3 | unsigned short | uint16\_t |
| 4 | signed short | int16\_t |
| 5 | long | ilong32\_t |
| 6 | unsigned long | ulong32\_t |

## Error codes

Modbus stack returns success or error code on execution of an API. These are defined in enum eStackErrorCode

| **#** | **Stack error** | **Description** |
| --- | --- | --- |
| 1 | *STACK\_NO\_ERROR* | no error or successful |
| 2 | *STACK\_TXNID\_OR\_UNITID\_MISSMATCH* | Modbus request transaction id or unit id in response is mismatching |
| 3 | *STACK\_ERROR\_SOCKET\_FAILED* | failed to read data from socket connected to Modbus device |
| 4 | *STACK\_ERROR\_CONNECT\_FAILED* | socket connection with Modbus device failed |
| 5 | *STACK\_ERROR\_SEND\_FAILED* | failed to send request on Modbus via socket |
| 6 | *STACK\_ERROR\_RECV\_FAILED* | failed to receive response from Modbus device on socket |
| 7 | *STACK\_ERROR\_RECV\_TIMEOUT* | failed to receive response within timeout duration |
| 8 | *STACK\_ERROR\_MALLOC\_FAILED* | failed to allocate memory |
| 9 | *STACK\_ERROR\_QUEUE\_SEND* | failed to send data in queue |
| 10 | *STACK\_ERROR\_QUEUE\_RECIVE* | failed to receive data from queue |
| 11 | *STACK\_ERROR\_THREAD\_CREATE* | failed to create thread |
| 12 | *STACK\_ERROR\_INVALID\_INPUT\_PARAMETER* | invalid input provided |
| 13 | *STACK\_ERROR\_PACKET\_LENGTH\_EXCEEDED* | packet length exceeded than limit |
| 14 | *STACK\_ERROR\_SOCKET\_LISTEN\_FAILED* | failed to listen on socket connected to Modbus device |
| 15 | *STACK\_ERROR\_MAX\_REQ\_SENT* | already sent maximum number of requests |
| 16 | *STACK\_ERROR\_FAILED\_Q\_SENT\_REQ* | Unable to add request for timeout tracking. marking it as failed |
| 17 | *STACK\_INIT\_FAILED* | Stack initialization fail error |
| 18 | *STACK\_ERROR\_QUEUE\_CREATE* | Linux queue creation failed error |
| 19 | *STACK\_ERROR\_STACK\_IS\_NOT\_INITIALIZED* | when stack is not initialized |
| 20 | *STACK\_ERROR\_STACK\_IS\_ALREADY\_INITIALIZED* | When stack is already initialized |
| 21 | *STACK\_ERROR\_SERIAL\_PORT\_ERROR* | When serial port not initialized |
| 22 | *STACK\_ERROR\_SERIAL\_PORT\_ALREADY\_IN\_USE* | When port is already used for communication |
| 23 | *STACK\_ERROR\_PORT\_NAME\_LENGTH\_EXCEEDED* | When port name exceed maximum length |
| 24 | *STACK\_ERROR\_MAX* | max error |

## Modbus function codes

Following is a list of supported function codes. These are defined in enum eModbusFuncCode\_enum

| **#** | **Enum** | **Description** |
| --- | --- | --- |
| 1 | READ\_COIL\_STATUS | Function code 1 |
| 2 | READ\_INPUT\_STATUS | Function code 2 |
| 3 | READ\_HOLDING\_REG | Function code 3 |
| 4 | READ\_INPUT\_REG | Function code 4 |
| 5 | WRITE\_SINGLE\_COIL | Function code 5 |
| 6 | WRITE\_SINGLE\_REG | Function code 6 |
| 7 | WRITE\_MULTIPLE\_COILS | Function code 15 |
| 8 | WRITE\_MULTIPLE\_REG | Function code 16 |
| 9 | READ\_FILE\_RECORD | Function code 20 |
| 10 | WRITE\_FILE\_RECORD | Function code 21 |
| 11 | READ\_WRITE\_MUL\_REG | Function code 23 |
| 12 | READ\_DEVICE\_IDENTIFICATION | Function code 43 |

## Structs definitions

### stException\_t

This data structure provides an information about an error if one occurs during operation execution.

| **#** | **Data type** | **Field name** | **Description** |
| --- | --- | --- | --- |
| 1 | uint8\_t | m\_u8ExcStatus | Exception type:  Value 1: It represents response from slave device  Value 2: It represents stack error code. |
| 2 | uint8\_t | m\_u8ExcCode | Actual error code  For “m\_u8ExcStatus = 2”, error code values are as per eStackErrorCode |

### stTimeStamps

This data structure provides timestamps taken at various stages inside the stack while performing a request API.

| **#** | **Data type** | **Field name** | **Description** |
| --- | --- | --- | --- |
| 1 | struct timespec | tsReqRcvd | Timestamp when a request is received in stack |
| 2 | struct timespec | tsReqSent | Timestamp when the request is sent on Modbus network |
| 3 | struct timespec | tsRespRcvd | Timestamp when the response is received from Modbus network |
| 4 | struct timespec | tsRespSent | Timestamp when the response is posted to application from stack |

### stDevConfig\_t

This structure is used to set stack config parameters.

| **#** | **Data type** | **Field name** | **Description** |
| --- | --- | --- | --- |
| 1 | long | m\_lInterframedelay | Used to store interframe delay for TCP and RTU |
| 2 | long | m\_lResponseTimeout | This value is used for response timeout mechanism for every request in stack |

### stMbusAppCallbackParams\_t

This data structure is used in a callback function when a response is received from a slave device for a request of type read/write register/coil/discrete input.

| **#** | **Data type** | **Field name** | **Description** |
| --- | --- | --- | --- |
| 1 | uint16\_t | m\_u16TransactionID | Transaction id, used for request-response matching between application and stack.  This transaction id is further used to send a request to a TCP slave device. |
| 2 | uint8\_t | m\_u8FunctionCode | Function code used for sending corresponding request |
| 3 | uint8\_t | m\_u8MbusRXDataLength | Length of response data |
| 4 | uint8\_t | m\_u8UnitID | Unit ID of slave device. This field is for TCP mode. |
| 5 | uint8\_t | m\_u8IpAddr[4] | IP address of slave device. This field is for TCP mode. |
| 6 | uint16\_t | u16Port | Port of slave device. This field is for TCP mode. |
| 7 | uint8\_t | m\_u8ReceivedDestination | Destination received in response. This field is for RTU mode. |
| 8 | uint8\_t | m\_au8MbusRXDataDataFields[ MODBUS\_DATA\_LENGTH ] | Response data, if any.  Maximum length = 260 |
| 9 | uint16\_t | m\_u16StartAdd | Start address of a coil / register / discrete input on which the request is performed |
| 10 | uint16\_t | m\_u16Quantity | Quantity of coils / registers / discrete inputs on which the request is performed |
| 11 | long | m\_lPriority | Priority with which the request was initiated |
| 12 | stTimeStamps | m\_objTimeStamps | Timestamps associate with this request execution |
| 13 | uint8\_t | m\_u8ExceptionExcStatus | Exception status. Values 1 or 2 |
| 14 | uint8\_t | m\_u8ExceptionExcCode | Error code |

### Structures related to read file record

Following data structures are used to send a request and received a response for read file record (function code 20):

#### stMbusReadFileRecord\_t

This data structure represents a list of sub-requests forming a request. Each node represents a group to be read.

| **#** | **Data type** | **Field name** | **Description** |
| --- | --- | --- | --- |
| 1 | uint8\_t | m\_u8RefType | Reference type (must be specified as 6) |
| 2 | uint16\_t | m\_u16FileNum | File number |
| 3 | uint16\_t | m\_u16RecordNum | Starting record number within file |
| 4 | uint16\_t | m\_u16RecordLength | Length of record |
| 5 | Self pointer | pstNextNode | Pointer to next group in the list |

#### stRdFileSubReq\_t

This data structure represents a list of sub-responses. Each sub-response corresponds to a sub-request in a request.

| **#** | **Data type** | **Field name** | **Description** |
| --- | --- | --- | --- |
| 1 | uint8\_t | m\_u8FileRespLen | Length of data in this sub-response |
| 2 | uint8\_t | m\_u8RefType | Reference type (value is 6) |
| 3 | uint16\_t\* | m\_pu16RecData | Received data |
| 4 | Self pointer | pstNextNode | Pointer to next group in the list |

#### stMbusRdFileRecdResp\_t

This data structure is used to represent response for read file record request:

| **#** | **Data type** | **Field name** | **Description** |
| --- | --- | --- | --- |
| 1 | uint8\_t | m\_u8RespDataLen | Length of response |
| 2 | stRdFileSubReq\_t | m\_stSubReq | Structure containing response groups |

### Structures related to write file record

Following data structures are used to send a request and receive a response for write file record (function code 21):

#### stWrFileSubReq\_t

This data structure represents a list of sub-requests forming a request. Each node represents a group to be written.

| **#** | **Data type** | **Field name** | **Description** |
| --- | --- | --- | --- |
| 1 | uint8\_t | m\_u8RefType | Reference type (must be specified as 6) |
| 2 | uint16\_t | m\_u16FileNum | File number |
| 3 | uint16\_t | m\_u16RecNum | Starting record number within file |
| 4 | uint16\_t | m\_u16RecLen | Length of record |
| 5 | uint16\_t\* | m\_pu16RecData | Pointer to data to be written |
| 6 | Self pointer | pstNextNode | Pointer to next group in the list |

#### stMbusWrFileRecdResp\_t

This data structure is used to represent response for write file record request:

| **#** | **Data type** | **Field name** | **Description** |
| --- | --- | --- | --- |
| 1 | uint8\_t | m\_u8RespDataLen | Length of response |
| 2 | stWrFileSubReq\_t | m\_stSubReq | Structure containing response groups |

### Structures related to read device identification

Following data structures are used to send a request and received a response for reading device id (function code 43):

#### SubObjList\_t

This data structure is used to represent one response object:

| **#** | **Data type** | **Field name** | **Description** |
| --- | --- | --- | --- |
| 1 | unsigned char | m\_u8ObjectID | Length of response |
| 2 | unsigned char | m\_u8ObjectLen | Object length |
| 3 | unsigned char\* | m\_u8ObjectValue | Pointer to data value |
| 4 | stRdFileSubReq\_t | pstNextNode | Pointer to next group in the list |

#### stRdDevIdResp\_t

This data structure is used to represent response data:

| # | Data type | Field name | Description |
| --- | --- | --- | --- |
| 1 | unsigned char | m\_u8MEIType | MEI type |
| 2 | unsigned char | m\_u8RdDevIDCode | Device ID Code (01/02/03/04) |
| 3 | unsigned char | m\_u8ConformityLevel | Conformity level (0x01 or 0x02 or 0x03 or 0x81 or 0x82 or 0x83) |
| 4 | unsigned char | m\_u8MoreFollows | Indicates whether there any other records in slave device (0x00 or 0xFF) |
| 5 | unsigned char | m\_u8NextObjId | Next object ID |
| 6 | unsigned char | m\_u8NumberofObjects | Total number of objects |
| 7 | SubObjList\_t | m\_pstSubObjList | Object containing response chunks |

# Macro description

| # | Macro | Value | Description |
| --- | --- | --- | --- |
| 1 | MAX\_REQUESTS | 5000 | This value decides the maximum length of request array size in stack |
| 2 | THREAD\_PRIORITY | 30 | This value is used to set thread priority for all threads in Stack. (value should be in range of 1 to 99) |
| 3 | THREAD\_SCHEDULER | SCHED\_RR | This value is used to set thread scheduler for all threads in Stack. (value should be either SCHED\_RR (Round Robin) or SCHED\_FIFO (First In First Out)) |
| 4 | DEFAULT\_RESPONSE\_TIMEOUT\_MS | 80 | Default response timeout in milliseconds |
| 6 | DEFAULT\_INTERFRAME\_DELAY\_MS | 0 | Default Interframe delay to be used for TCP and RTU communication if |
| 7 | MAX\_RECV\_PRIORITY | -100000 | Maximum priority used for priority queue |

# API description

This section describes APIs in detail. Unless until specified, following APIs are in general applicable for TCP and RTU mode. If an API or a field is applicable for a certain mode, it is specified in API description.

## API – AppMbusMaster\_StackInit

### Description

Initializes stack library i.e. all threads, semaphores, internal data structures, etc.

### Prototype

uint8\_t AppMbusMaster\_StackInit(void);

### Parameter description

No parameters

### Return type

API status code – success or error code

## API – AppMbusMaster\_StackDeInit

### Description

De-initializes stack library i.e. stops threads, deallocates memory, etc.

### Prototype

void AppMbusMaster\_StackDeInit(void);

### Parameter description

No parameters

### Return type

Void

## API – AppMbusMaster\_ removeCtx

### Description

Remove particular context of TCP/RTU i.e. stops threads, deallocates memory, etc.

### Prototype

MODBUS\_STACK\_EXPORT void removeCtx(int msgQId);

### Parameter description

| **#** | **Argument** | **Type** | **Description** |
| --- | --- | --- | --- |
| 1 | msgQId | In | Message queue id which needs to be removed from list. |

### Return type

Void

## API – AppMbusMaster\_ getTCPCtx

### Description

Used to create TCP context for communication.

### Prototype

MODBUS\_STACK\_EXPORT eStackErrorCode getTCPCtx(int \*tcpCtx, stCtxInfo \*pCtxInfo);

### Parameter description

| **#** | **Argument** | **Type** | **Description** |
| --- | --- | --- | --- |
| 1 | tcpCtx | Out | Stores TCP context after generation. |
| 2 | pCtxInfo | In | Structure containing the Ip-Address and port number. |

### Return type

eStackErrorCode – Returns the error code.

## API – AppMbusMaster\_ getRTUCtx

### Description

Used to create TCP context for communication.

### Prototype

MODBUS\_STACK\_EXPORT eStackErrorCode getRTUCtx(int32\_t \*rtuCtx, stCtxInfo \*pCtxInfo);

### Parameter description

| **#** | **Argument** | **Type** | **Description** |
| --- | --- | --- | --- |
| 1 | rtuCtx | Out | Stores RTU context after generation. |
| 2 | pCtxInfo | In | Structure containing the port name, baud rate and parity. |

### Return type

eStackErrorCode – Returns the error code.

## API – initSerialPort

### Description

This API is applicable for RTU mode. It initializes serial port for RTU communication.

### Prototype

int initSerialPort(stRTUConnectionData\_t\* pstRTUConnectionData,

uint8\_t \*portName, uint32\_t baudrate, uint8\_t parity);

### Parameter description

| **#** | **Argument** | **Type** | **Description** |
| --- | --- | --- | --- |
| 1 | pstRTUConnectionData | out | Structure containing fd and interframe delay. These values will be created and assigned here based on port name and baudrate. |
| 2 | portName | In | Name of port to be used for communication (e.g. “/dev/ttyUSB0”) |
| 3 | baudrate | In | Baud rate to be used. Standard baud rates are 9600, 19200, 38400, 57600, 115200 |
| 4 | parity | In | Parity to be used (0 = No parity, 1 = Even parity, 2 = Odd parity) |

### Return type

API status code – Returns negative value in case of an error

## API – AppMbusMaster\_SetStackConfigParam

### Description

Sets the stack configurations parameters (like interframe delay, response timeout) for end device communication.

### Prototype

uint8\_t AppMbusMaster\_SetStackConfigParam(stDevConfig\_t \*a\_pstDevConf

);

### Parameter description

| **#** | **Argument** | **Type** | **Description** |
| --- | --- | --- | --- |
| 1 | a\_pstDevConf | In | Pointer to structure for setting response timeout (in usec) and Inter-frame delay in stack from application. |

### Return type

API status code – success or error code

## API – AppMbusMaster\_GetStackConfigParam

### Description

Gets response timeout and Interframe delay set in stack. These parameters are used while connecting a TCP slave device.

### Prototype

stDevConfig\_t\* AppMbusMaster\_GetStackConfigParam(void

);

### Parameter description

No parameters

### Return type

Returns response timeout and interframe delay set in stack as a pointer.

## API – Modbus\_Read\_Coils

### Description

API to read coils from a Modbus slave device. This API corresponds to Modbus function code 1 (i.e. 0x01).

### Prototype

uint8\_t Modbus\_Read\_Coils(

uint16\_t u16StartCoil, uint16\_t u16NumOfcoils, uint16\_t u16TransacID, uint8\_t u8UnitId, long lPriority, int32\_t i32Ctx, void\* pFunCallBack)

### Parameter description

| **#** | **Argument** | **Type** | **Description** |
| --- | --- | --- | --- |
| 1 | u16StartCoil | In | Address of a coil to be read |
| 2 | u16NumOfcoils | In | Number of coils to be read |
| 3 | u16TransacID | In | Transaction id used for request-response matching between application and stack.  This transaction id is further used to send a request to a TCP slave device. |
| 4 | u8UnitId | In | Slave ID or Unit ID |
| 5 | lPriority | In | Priority to be assigned to this request. Lower the number, higher the priority. |
| 6 | i32Ctx | In | TCP/RTU context |
| 7 | pFunCallBack | In | Callback function from application to be used on receipt of a response from slave device |

### Return type

API status code – success or error code

### Callback Function

Callback function from application to be used on receipt of a response from slave device.

#### Callback Prototype

void (\*ModbusMaster\_ApplicationCallback)(

stMbusAppCallbackParams\_t \*pstMbusAppCallbackParams);

#### Callback Parameters

| **#** | **Argument** | **Type** | **Description** |
| --- | --- | --- | --- |
| 1 | pstMbusAppCallbackParams | In | Pointer to a data structure “stMbusAppCallbackParams\_t” containing response data |

#### Callback Return type

None

## API – Modbus\_Read\_Discrete\_Inputs

### Description

API to read discrete inputs from a Modbus slave device. This API corresponds to Modbus function code 2 (i.e. 0x02).

### Prototype

uint8\_t Modbus\_Read\_Discrete\_Inputs(

uint16\_t u16StartDI, uint16\_t u16NumOfDI, uint16\_t u16TransacID, uint8\_t u8UnitId, long lPriority, int32\_t i32Ctx, void\* pFunCallBack)

### Parameter description

| **#** | **Argument** | **Type** | **Description** |
| --- | --- | --- | --- |
| 1 | u16StartDI | In | Address of a discrete input to be read |
| 2 | u16NumOfDI | In | Number of discrete inputs to be read |
| 3 | u16TransacID | In | Transaction id used for request-response matching between application and stack.  This transaction id is further used to send a request to a TCP slave device. |
| 4 | u8UnitId | In | Slave ID or Unit ID |
| 5 | lPriority | In | Priority to be assigned to this request. Lower the number, higher the priority. |
| 6 | i32Ctx | In | TCP/RTU context |
| 7 | pFunCallBack | In | Callback function from application to be used on receipt of a response from slave device |

### Return type

API status code – success or error code

### Callback Function

Callback function from application to be used on receipt of a response from slave device.

#### Callback Prototype

void (\*ModbusMaster\_ApplicationCallback)(

stMbusAppCallbackParams\_t \*pstMbusAppCallbackParams);

#### Callback Parameters

| **#** | **Argument** | **Type** | **Description** |
| --- | --- | --- | --- |
| 1 | pstMbusAppCallbackParams | In | Pointer to a data structure “stMbusAppCallbackParams\_t” containing response data |

#### Callback Return type

None

## API – Modbus\_Read\_Holding\_Registers

### Description

API to read holding registers from a Modbus slave device. This API corresponds to Modbus function code 3 (i.e. 0x03).

### Prototype

Same for TCP and RTU

uint8\_t Modbus\_Read\_Holding\_Registers(

uint16\_t u16StartReg, uint16\_t u16NumberOfRegisters, uint16\_t u16TransacID, uint8\_t u8UnitId, long lPriority, int32\_t i32Ctx, void\* pFunCallBack)

### Parameter description

| **#** | **Argument** | **Type** | **Description** |
| --- | --- | --- | --- |
| 1 | u16StartReg | In | Address of a holding register to be read |
| 2 | u16NumberOfRegisters | In | Number of holding registers to be read |
| 3 | u16TransacID | In | Transaction id used for request-response matching between application and stack.  This transaction id is further used to send a request to a TCP slave device. |
| 4 | u8UnitId | In | Slave ID or Unit ID |
| 5 | lPriority | In | Priority to be assigned to this request. Lower the number, higher the priority. |
| 6 | i32Ctx | In | TCP/RTU context |
| 7 | pFunCallBack | In | Callback function from application to be used on receipt of a response from slave device |

### Return type

API status code – success or error code

### Callback Function

Callback function from application to be used on receipt of a response from slave device.

#### Callback Prototype

void (\*ModbusMaster\_ApplicationCallback)(

stMbusAppCallbackParams\_t \*pstMbusAppCallbackParams);

#### Callback Parameters

| **#** | **Argument** | **Type** | **Description** |
| --- | --- | --- | --- |
| 1 | pstMbusAppCallbackParams | In | Pointer to a data structure “stMbusAppCallbackParams\_t” containing response data |

#### Callback Return type

None

## API – Modbus\_Read\_Input\_Registers

### Description

API to read input registers from a Modbus slave device. This API corresponds to Modbus function code 4 (i.e. 0x04).

### Prototype

Same for both TCP and RTU

uint8\_t Modbus\_Read\_Input\_Registers(

uint16\_t u16StartReg, uint16\_t u16NumberOfRegisters, uint16\_t u16TransacID, uint8\_t u8UnitId, long lPriority, int32\_t i32Ctx, void\* pFunCallBack)

### Parameter description

| **#** | **Argument** | **Type** | **Description** |
| --- | --- | --- | --- |
| 1 | u16StartReg | In | Address of an input register to be read |
| 2 | u16NumberOfRegisters | In | Number of input registers to be read |
| 3 | u16TransacID | In | Transaction id used for request-response matching between application and stack.  This transaction id is further used to send a request to a TCP slave device. |
| 4 | u8UnitId | In | Slave ID or Unit ID |
| 5 | lPriority | In | Priority to be assigned to this request. Lower the number, higher the priority. |
| 6 | i32Ctx | In | TCP/RTU context |
| 7 | pFunCallBack | In | Callback function from application to be used on receipt of a response from slave device |

### Return type

API status code – success or error code

### Callback Function

Callback function from application to be used on receipt of a response from slave device.

#### Callback Prototype

void (\*ModbusMaster\_ApplicationCallback)(

stMbusAppCallbackParams\_t \*pstMbusAppCallbackParams);

#### Callback Parameters

| **#** | **Argument** | **Type** | **Description** |
| --- | --- | --- | --- |
| 1 | pstMbusAppCallbackParams | In | Pointer to a data structure “stMbusAppCallbackParams\_t” containing response data |

#### Callback Return type

None

## API – Modbus\_Write\_Single\_Coil

### Description

API to write a coil present in a Modbus slave device. This API corresponds to Modbus function code 5 (i.e. 0x05).

### Prototype

uint8\_t Modbus\_Write\_Single\_Coil(

uint16\_t u16StartCoil, uint16\_t u16OutputVal, uint16\_t u16TransacID, uint8\_t u8UnitId, long lPriority, int32\_t i32Ctx, void\* pFunCallBack)

### Parameter description

| **#** | **Argument** | **Type** | **Description** |
| --- | --- | --- | --- |
| 1 | u16StartCoil | In | Address of a coil to write |
| 2 | u16OutputVal | In | Value to be written in the coil |
| 3 | u16TransacID | In | Transaction id used for request-response matching between application and stack.  This transaction id is further used to send a request to a TCP slave device. |
| 4 | u8UnitId | In | Slave ID or Unit ID |
| 5 | lPriority | In | Priority to be assigned to this request. Lower the number, higher the priority. |
| 6 | i32Ctx | In | TCP/RTU context |
| 7 | pFunCallBack | In | Callback function from application to be used on receipt of a response from slave device |

### Return type

API status code – success or error code

### Callback Function

Callback function from application to be used on receipt of a response from slave device.

#### Callback Prototype

void (\*ModbusMaster\_ApplicationCallback)(

stMbusAppCallbackParams\_t \*pstMbusAppCallbackParams);

#### Callback Parameters

| **#** | **Argument** | **Type** | **Description** |
| --- | --- | --- | --- |
| 1 | pstMbusAppCallbackParams | In | Pointer to a data structure “stMbusAppCallbackParams\_t” containing response data |

#### Callback Return type

None

## API – Modbus\_Write\_Single\_Register

### Description

API to write a register present in a Modbus slave device. This API corresponds to Modbus function code 6 (i.e. 0x06).

### Prototype

uint8\_t Modbus\_Write\_Single\_Register(

uint16\_t u16StartReg, uint16\_t u16RegOutputVal, uint16\_t u16TransacID, uint8\_t u8UnitId, long lPriority, int32\_t i32Ctx, void\* pFunCallBack)

### Parameter description

| **#** | **Argument** | **Type** | **Description** |
| --- | --- | --- | --- |
| 1 | u16StartReg | In | Address of a register to write |
| 2 | u16RegOutputVal | In | Value to be written in the register |
| 3 | u16TransacID | In | Transaction id used for request-response matching between application and stack.  This transaction id is further used to send a request to a TCP slave device. |
| 4 | u8UnitId | In | Slave ID or Unit ID |
| 5 | lPriority | In | Priority to be assigned to this request. Lower the number, higher the priority. |
| 6 | i32Ctx | In | TCP/RTU context |
| 7 | pFunCallBack | In | Callback function from application to be used on receipt of a response from slave device |

### Return type

API status code – success or error code

### Callback Function

Callback function from application to be used on receipt of a response from slave device.

#### Callback Prototype

void (\*ModbusMaster\_ApplicationCallback)(

stMbusAppCallbackParams\_t \*pstMbusAppCallbackParams);

#### Callback Parameters

| **#** | **Argument** | **Type** | **Description** |
| --- | --- | --- | --- |
| 1 | pstMbusAppCallbackParams | In | Pointer to a data structure “stMbusAppCallbackParams\_t” containing response data |

#### Callback Return type

None

## API – Modbus\_Write\_Multiple\_Coils

### Description

API to write multiple coils present in a Modbus slave device. This API corresponds to Modbus function code 15 (i.e. 0x0F).

### Prototype

uint8\_t Modbus\_Write\_Multiple\_Coils(

uint16\_t u16Startcoil, uint16\_t u16NumOfCoil, uint16\_t u16TransacID, uint8\_t \*pu8OutputVal, uint8\_t u8UnitId, long lPriority, int32\_t i32Ctx, void\* pFunCallBack)

### Parameter description

| **#** | **Argument** | **Type** | **Description** |
| --- | --- | --- | --- |
| 1 | u16StartCoil | In | Address of a coil to write |
| 2 | u16NumOfCoil | In | Number of coils to write |
| 3 | u16OutputVal | In | Value to be written in coils |
| 4 | u16TransacID | In | Transaction id used for request-response matching between application and stack.  This transaction id is further used to send a request to a TCP slave device. |
| 5 | u8UnitId | In | Slave ID or Unit ID |
| 6 | lPriority | In | Priority to be assigned to this request. Lower the number, higher the priority. |
| 7 | i32Ctx | In | TCP/RTU context |
| 8 | pFunCallBack | In | Callback function from application to be used on receipt of a response from slave device |

### Return type

API status code – success or error code

### Callback Function

Callback function from application to be used on receipt of a response from slave device.

#### Callback Prototype

void (\*ModbusMaster\_ApplicationCallback)(

stMbusAppCallbackParams\_t \*pstMbusAppCallbackParams);

#### Callback Parameters

| **#** | **Argument** | **Type** | **Description** |
| --- | --- | --- | --- |
| 1 | pstMbusAppCallbackParams | In | Pointer to a data structure “stMbusAppCallbackParams\_t” containing response data |

#### Callback Return type

None

## API – Modbus\_Write\_Multiple\_Register

### Description

API to write multiple registers present in a Modbus slave device. This API corresponds to Modbus function code 16 (i.e. 0x10).

### Prototype

uint8\_t Modbus\_Write\_Multiple\_Register(

uint16\_t u16StartReg, uint16\_t u16NumOfReg, uint16\_t u16TransacID, uint8\_t \*pu8OutputVal, uint8\_t u8UnitId, long lPriority, int32\_t i32Ctx, void\* pFunCallBack)

### Parameter description

| **#** | **Argument** | **Type** | **Description** |
| --- | --- | --- | --- |
| 1 | u16StartReg | In | Address of a register to write |
| 2 | u16NumOfReg | In | Number of registers to write |
| 3 | u16OutputVal | In | Value to be written in registers |
| 4 | u16TransacID | In | Transaction id used for request-response matching between application and stack.  This transaction id is further used to send a request to a TCP slave device. |
| 5 | u8UnitId | In | Slave ID or Unit ID |
| 6 | lPriority | In | Priority to be assigned to this request. Lower the number, higher the priority. |
| 7 | i32Ctx | In | TCP/RTU context |
| 8 | pFunCallBack | In | Callback function from application to be used on receipt of a response from slave device |

### Return type

API status code – success or error code

### Callback Function

Callback function from application to be used on receipt of a response from slave device.

#### Callback Prototype

void (\*ModbusMaster\_ApplicationCallback)(

stMbusAppCallbackParams\_t \*pstMbusAppCallbackParams);

#### Callback Parameters

| **#** | **Argument** | **Type** | **Description** |
| --- | --- | --- | --- |
| 1 | pstMbusAppCallbackParams | In | Pointer to a data structure “stMbusAppCallbackParams\_t” containing response data |

#### Callback Return type

None

## API – Modbus\_Read\_File\_Record

### Description

API to read file record from a Modbus slave device. This API corresponds to Modbus function code 20 (i.e. 0x14).

### Prototype

uint8\_t Modbus\_Read\_File\_Record(

uint8\_t u8byteCount, uint8\_t u8FunCode, stMbusReadFileRecord\_t \*pstFileRecord, uint16\_t u16TransacID, uint8\_t u8UnitId, long lPriority, int32\_t i32Ctx, void\* pFunCallBack)

### Parameter description

| **#** | **Argument** | **Type** | **Description** |
| --- | --- | --- | --- |
| 1 | u8byteCount | In | Number of bytes to read |
| 2 | u8FunCode | In | Function code i.e. 0x14 |
| 3 | pstFileRecord | In | File record data pointing to a data struct stMbusReadFileRecord\_t |
| 4 | u16TransacID | In | Transaction id used for request-response matching between application and stack.  This transaction id is further used to send a request to a TCP slave device. |
| 5 | u8UnitId | In | Slave ID or Unit ID |
| 6 | lPriority | In | Priority to be assigned to this request. Lower the number, higher the priority. |
| 7 | i32Ctx | In | TCP/RTU context |
| 8 | pFunCallBack | In | Callback function from application to be used on receipt of a response from slave device |

### Return type

API status code – success or error code

### Callback Function

Callback function from application to be used on receipt of a response from slave device.

#### Callback Prototype

For TCP Mode:

void (\*ReadFileRecord\_CallbackFunction)

(uint8\_t u8UnitID, uint8\_t\* pu8IpAddr, uint16\_t u16Port, uint16\_t u16TransactionID, uint8\_t u8FunctionCode, stException\_t \*pstException, stMbusRdFileRecdResp\_t\* pstData);

For RTU Mode:

void (\*ReadFileRecord\_CallbackFunction)

(uint8\_t u8UnitID, uint8\_t\* pu8RcvdDest, uint16\_t u16TransactionID, uint8\_t u8FunctionCode, stException\_t \*pstException, stMbusRdFileRecdResp\_t\* pstData);

#### Callback Parameters

| **#** | **Argument** | **Type** | **Description** |
| --- | --- | --- | --- |
| 1 | u8UnitID | In | Unit ID or slave ID |
| 2 | **Modbus slave details** |  |  |
|  | For TCP slave: |  |  |
| 2a | pu8IpAddr | In | IP address of TCP slave device |
| 2b | u16Port | In | Port number of TCP slave device used for communication |
|  | For RTU slave: |  |  |
| 2a | pu8RcvdDest | In | Received destination |
| 3 | u16TransactionID | In | Transaction id used for request-response matching between application and stack.  This transaction id is one which is sent by the TCP slave device. |
| 4 | u8FunctionCode | In | Function code |
| 5 | pstException | In | Pointer to data structure “stException\_t” which contains details of error, if any |
| 6 | pstData | In | Pointer to data structure “stMbusRdFileRecdResp\_t” which contains data |

#### Callback Return type

None

## API – Modbus\_Write\_File\_Record

### Description

API to rite file records to a Modbus slave device. This API corresponds to Modbus function code 21 (i.e. 0x15).

### Prototype

uint8\_t Modbus\_Write\_File\_Record(

uint8\_t u8ReqDataLen, uint8\_t u8FunCode, stWrFileSubReq\_t \*pstFileRecord, uint16\_t u16TransacID, uint8\_t u8UnitId, long lPriority, int32\_t i32Ctx, void\* pFunCallBack)

### Parameter description

| **#** | **Argument** | **Type** | **Description** |
| --- | --- | --- | --- |
| 1 | u8ReqDataLen | In | Number of bytes to write |
| 2 | u8FunCode | In | Function code i.e. 0x15 |
| 3 | pstFileRecord | In | File record data pointing to a data struct stWrFileSubReq\_t |
| 4 | u16TransacID | In | Transaction id used for request-response matching between application and stack.  This transaction id is further used to send a request to a TCP slave device. |
| 5 | u8UnitId | In | Slave ID or Unit ID |
| 6 | lPriority | In | Priority to be assigned to this request. Lower the number, higher the priority. |
| 7 | i32Ctx | In | TCP/RTU context |
| 8 | pFunCallBack | In | Callback function from application to be used on receipt of a response from slave device |

### Return type

API status code – success or error code

### Callback Function

Callback function from application to be used on receipt of a response from slave device.

#### Callback Prototype

For TCP Mode:

void (\*WriteFileRecord\_CallbackFunction)

(uint8\_t u8UnitID, uint8\_t\* pu8IpAddr, uint16\_t u16Port, uint16\_t u16TransactionID, uint8\_t u8FunctionCode, stException\_t \*pstException, stMbusWrFileRecdResp\_t\* pstData);

For RTU Mode:

void (\*WriteFileRecord\_CallbackFunction)

(uint8\_t u8UnitID, uint8\_t\* pu8RcvdDest, uint16\_t u16TransactionID, uint8\_t u8FunctionCode, stException\_t \*pstException, stMbusWrFileRecdResp\_t\* pstData);

#### Callback Parameters

| **#** | **Argument** | **Type** | **Description** |
| --- | --- | --- | --- |
| 1 | u8UnitID | In | Unit ID or slave ID |
| 2 | **Modbus slave details** |  |  |
|  | For TCP slave: |  |  |
| 2a | pu8IpAddr | In | IP address of TCP slave device |
| 2b | u16Port | In | Port number of TCP slave device used for communication |
|  | For RTU slave: |  |  |
| 2a | pu8RcvdDest | In | Received destination |
| 3 | u16TransactionID | In | Transaction id used for request-response matching between application and stack.  This transaction id is one which is sent by the TCP slave device. |
| 4 | u8FunctionCode | In | Function code |
| 5 | pstException | In | Pointer to data structure “stException\_t” which contains details of error, if any |
| 6 | pstData | In | Pointer to data structure “stMbusWrFileRecdResp\_t” which contains data |

#### Callback Return type

None

## API – Modbus\_Read\_Write\_Registers

### Description

API to read or write registers present in a Modbus slave device. This API corresponds to Modbus function code 23 (i.e. 0x17).

### Prototype

uint8\_t Modbus\_Read\_Write\_Registers(

uint16\_t u16ReadRegAddress, uint8\_t u8FunCode, uint16\_t u16NoOfReadReg, uint16\_t u16WriteRegAddress, uint16\_t u16NoOfWriteReg, uint16\_t u16TransacID, uint8\_t \*pu8OutputVal, uint8\_t u8UnitId, long lPriority, int32\_t i32Ctx, void\* pFunCallBack)

### Parameter description

| **#** | **Argument** | **Type** | **Description** |
| --- | --- | --- | --- |
| 1 | u16ReadRegAddress | In | Start address of a register to read |
| 2 | u8FunCode | In | Function code to used |
| 3 | u16NoOfReadReg | In | Number of registers to read |
| 4 | u16WriteRegAddress | In | Start address of a register to write |
| 5 | u16NoOfWriteReg | In | Number of registers to write |
| 6 | u16TransacID | In | Transaction id used for request-response matching between application and stack.  This transaction id is further used to send a request to a TCP slave device. |
| 7 | pu8OutputVal | In | Value to be written in registers |
| 8 | u8UnitId | In | Slave ID or Unit ID |
| 9 | lPriority | In | Priority to be assigned to this request. Lower the number, higher the priority. |
| 10 | i32Ctx | In | TCP/RTU context |
| 11 | pFunCallBack | In | Callback function from application to be used on receipt of a response from slave device |

### Return type

API status code – success or error code

### Callback Function

Callback function from application to be used on receipt of a response from slave device.

#### Callback Prototype

void (\*ModbusMaster\_ApplicationCallback)(

stMbusAppCallbackParams\_t \*pstMbusAppCallbackParams);

#### Callback Parameters

| **#** | **Argument** | **Type** | **Description** |
| --- | --- | --- | --- |
| 1 | pstMbusAppCallbackParams | In | Pointer to a data structure “stMbusAppCallbackParams\_t” containing response data |

#### Callback Return type

None

## API – Modbus\_Read\_Device\_Identification

### Description

API to read device identification number from a Modbus slave device. This API corresponds to Modbus function code 43 (i.e. 0x2B).

### Prototype

uint8\_t Modbus\_Read\_Device\_Identification(

uint8\_t u8MEIType, uint8\_t u8FunCode, uint8\_t u8ReadDevIdCode, uint8\_t u8ObjectId, uint16\_t u16TransacID, uint8\_t u8UnitId, long lPriority, int32\_t i32Ctx, void\* pFunCallBack)

### Parameter description

| **#** | **Argument** | **Type** | **Description** |
| --- | --- | --- | --- |
| 1 | u8MEIType | In | MEI type i.e. 0x0E |
| 2 | u8FunCode | In | Function code i.e. 0x2B |
| 3 | u8ReadDevIdCode | In | Device ID Code (01/02/03/04) |
| 4 | u8ObjectId | In | Object ID (0x00 to 0xFF) |
| 5 | u16TransacID | In | Transaction id used for request-response matching between application and stack.  This transaction id is further used to send a request to a TCP slave device. |
| 6 | u8UnitId | In | Slave ID or Unit ID |
| 7 | lPriority | In | Priority to be assigned to this request. Lower the number, higher the priority. |
| 8 | i32Ctx | In | TCP/RTU context |
| 9 | pFunCallBack | In | Callback function from application to be used on receipt of a response from slave device |

### Return type

API status code – success or error code

### Callback Function

Callback function from application to be used on receipt of a response from slave device.

#### Callback Prototype

For TCP Mode:

void (\*ReadDeviceIdentification\_CallbackFunction)

(uint8\_t u8UnitID, uint8\_t\* pu8IpAddr, uint16\_t u16Port, uint16\_t u16TransactionID, uint8\_t u8FunctionCode, stException\_t \*pstException, stRdDevIdResp\_t\* pstData);

For RTU Mode:

void (\*ReadDeviceIdentification\_CallbackFunction)

(uint8\_t u8UnitID, uint8\_t\* pu8RcvdDest, uint16\_t u16TransactionID, uint8\_t u8FunctionCode, stException\_t \*pstException, stRdDevIdResp\_t\* pstData);

#### Callback Parameters

| **#** | **Argument** | **Type** | **Description** |
| --- | --- | --- | --- |
| 1 | u8UnitID | In | Unit ID or slave ID |
| 2 | **Modbus slave details** |  |  |
|  | For TCP slave: |  |  |
| 2a | pu8IpAddr | In | IP address of TCP slave device |
| 2b | u16Port | In | Port number of TCP slave device used for communication |
|  | For RTU slave: |  |  |
| 2a | pu8RcvdDest | In | Received destination |
| 3 | u16TransactionID | In | Transaction id used for request-response matching between application and stack.  This transaction id is one which is sent by the TCP slave device. |
| 4 | u8FunctionCode | In | Function code |
| 5 | pstException | In | Pointer to data structure “stException\_t” which contains details of error, if any |
| 6 | pstData | In | Pointer to data structure “stRdDevIdResp\_t” which contains data |

#### Callback Return type

None

# Using APIs in Application

This section explains how application can use Modconn Modbus stack to create Modbus Application.

Block diagram below explains steps on using Modconn Modbus Stack.

Modbus Application

Load Modbus stack library (libModbusMasterStack.so) in application project

Include stack API and datatype definition file (API.h) in application project

Modbus stack library

API.h

## Exported Files

Table below explains exported header files by the stack.

| **Sr. No.** | **File Name** | **Description** |
| --- | --- | --- |
| 1 | API.h | It contains following:  Declaration of exported APIs  Data structure definition |