



by Schneider Electric

ModBUS Addons Library User Guide

Publisher: HVAC Solution Center, Alpago (BL) – Italy

Authors: Federico Marcassa, Pierpaolo Armeli

Doc. Version: v0.7





Table of Contents

1. Modbus Commands on Event	6
1.1. Introduction	6
1.2. ModBUS Master Functions in Application	6
1.2.1. Configuration Details	6
1.2.2. ModBUS Master: Application vs Connection	7
1.2.3. ModBUS Master from Application: remarks	7
2. ModBUS Addons Library	8
2.1. Library Overview	8
2.1.1. Library Description	8
2.1.2. Function Blocks and Structures List	9
2.1.3. Library Implementation Overview	9
2.2. Library Contents Details	11
2.2.1. ModbusRTUKeepAlive	11
2.2.2. rtuNodeMgr	12
2.2.3. fbModbusRTU_SendFC01	13
2.2.4. fbModbusRTU_SendFC02	13
2.2.5. fbModbusRTU_SendFC03	13
2.2.6. fbModbusRTU_SendFC04	14
2.2.7. fbModbusRTU_SendFC05	14
2.2.8. fbModbusRTU_SendFC06	15
2.2.9. fbModbusRTU_SendFC15	15
2.2.10. fbModbusRTU_SendFC16	16
2.2.11. strModbusRTUslave	16
2.2.12. enKeepAliveStatus	17
3. Appendix	18
3.1. ModBUS Bridge Functionality	18
3.1.1. ModBUS Addresses	18
3.1.2. Commands supported by the Bridge	19
4. Publisher's Info	20



Safety Information and Good Practices

Before You Begin

General

The products specified in this document have been tested under actual service conditions. Of course, your specific application requirements may be different from those assumed for this and any related examples described herein. In that case, you will have to adapt the information provided in this and other related documents to your particular needs. To do so, you will need to consult the specific product documentation of the hardware and/or software components that you may add or substitute for any examples specified in this documentation. Pay particular attention and conform to any safety information, different electrical requirements and normative standards that would apply to your adaptation.

© 2018 Eliwell Controls Srl. All rights reserved.

WARNING

REGULATORY INCOMPATIBILITY

Be sure that all equipment applied and systems designed comply with all applicable local, regional and national regulations and standards

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Note

Electrical equipment should be installed, operated, serviced, and maintained only by qualified personnel. No responsibility is assumed by Eliwell Controls Srl for any consequences arising out of the use of this material. A qualified person is one who has skills and knowledge related to the construction and operation of electrical equipment and its installation, and has received safety training to recognize and avoid the hazards involved. Failure to observe this information can result in injury or equipment damage.

The use and application of the information contained herein require expertise in the design and programming of automated control systems. Only the user or integrator can be aware of all the conditions and factors present during installation and setup, operation, and maintenance of the machine or process, and can therefore determine the automation and associated equipment and the related safeties and interlocks which can be effectively and properly used. When selecting automation and control equipment, and any other related equipment or software, for a particular application, the user or integrator must also consider any applicable local, regional or national standards and/or regulations.

Some of the major software functions and/or hardware components used in the proposed architectures and examples described in this document cannot be substituted without significantly compromising the performance of your application. Further, any such substitutions or alterations may completely invalidate any proposed architectures, descriptions, examples, instructions, wiring diagrams and/or compatibilities between the various hardware components and software functions specified herein and in related documentation. You must be aware of the consequences of any modifications, additions or substitutions.



A residual risk, as defined by EN/ISO 12100-1, Article 5, will remain if:

- it is necessary to modify the recommended logic and if the added or modified components are not properly integrated in the control circuit;
- you do not follow the required standards applicable to the operation of the machine, or if the adjustments to and the maintenance of the machine are not properly made (it is essential to strictly follow the prescribed machine maintenance schedule);
- the devices connected to any safety outputs do not have mechanically-linked contacts.

CAUTION

EQUIPMENT INCOMPATIBILITY

Read and thoroughly understand all device and software documentation before attempting any component substitutions or other changes related to the application examples provided in the document

Failure to follow these instructions can result in injury, or equipment damage.

Start-Up and Test

Before using electrical control and automation equipment after design and installation, the application and associated functional safety system must be subjected to a start-up test by qualified personnel to verify correct operation of the equipment. It is important that arrangements for such testing be made and that enough time is allowed to perform complete and satisfactory testing.

CAUTION

EQUIPMENT OPERATION HAZARD

- Verify that all installation and set up procedures have been completed.
- Before operational tests are performed, remove all blocks or other temporary holding means used for shipment from all component devices
- Remove tools, meters, and debris from equipment.

Failure to follow these instructions can result in injury, or equipment damage.

Verify that the completed system, including the functional safety system, is free from all short circuits and grounds, except those grounds installed according to local regulations. If high-potential voltage testing is necessary, follow the recommendations in equipment documentation to help prevent injury or equipment damage.



Operations and Adjustments

Regardless of the care exercised in the design and manufacture of equipment or in the selection and ratings of components, there are hazards that can be encountered if such equipment is improperly installed and operated.

In some applications, such as packaging machinery, additional operator protection such as point-of-operation guarding must be provided. This is necessary if the hands and other parts of the body are free to enter the pinch points or other hazardous areas where serious injury can occur. Software products alone cannot protect an operator from injury. For this reason, the software cannot be substituted for or take the place of point-of-operation protection.

WARNING

UNGUARDED MACHINERY CAN CAUSE SERIOUS INJURY

- Do not use this software and related automation equipment on equipment which does not have point-of -operation protection.
- Do not reach into machinery during operation.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Ensure that appropriate safeties and mechanical/electrical interlocks related to point-of-operation protection have been installed and are operational before placing the equipment into service. All interlocks and safeties related to point-of-operation protection must be coordinated with the related automation equipment and software programming.

NOTE: Coordination of safeties and mechanical/electrical interlocks for point-of-operation protection is outside the scope of the examples and implementations suggested herein. It is sometimes possible to adjust the equipment incorrectly and this produce unsatisfactory or unsafe operation. Always use the manufacturer instructions as a guide to functional adjustments. Personnel who have access to these adjustments must be familiar with the equipment manufacturer instructions and the machinery used with the electrical equipment.

Only those operational adjustments actually required by the machine operator should be accessible to the operator. Access to other controls should be restricted to help prevent unauthorized changes in operating characteristics.



ModBUS Commands on Event

1.1. Introduction









It is possible to send ModBUS commands on event from FreeStudio Application starting from the following targets on:

- **ADVANCE** with BIOS 596.6 and greater;
- **ADVANCE 4DIN** with BIOS 668.6 and greater.

The first release that supports this functionality is **FreeStudio 3.9.1**. Until then, it was possible to send ModBUS commands only by using Connection, on those devices.

1.2. ModBUS Master Functions in Application

From FreeStudio 3.9.1, the followings ModBUS Master functions have been added in Application:

Operators and blocks			
Name	Type	Group	Description
 sysMbMRTU_FC01	Function		Send 0x01 command.
 sysMbMRTU_FC02	Function		Send 0x02 command.
 sysMbMRTU_FC03	Function		Send 0x03 command.
 sysMbMRTU_FC04	Function		Send 0x04 command.
 sysMbMRTU_FC05	Function		Send 0x05 command.
 sysMbMRTU_FC06	Function		Send 0x06 command.
 sysMbMRTU_FC15	Function		Send 0x0F command.
 sysMbMRTU_FC16	Function		Send 0x10 command.
Operator and standard blocks Target variables Target blocks			

These functions allow to send **ModBUS commands on Event** through the master RS485 port by calling them in Application.

The messages can be send **only** from programs assigned to the **Background** task.

1.2.1. Configuration Details

In order to set the RS485 port as ModBUS Master, it is required to set it from Connection.

Therefore, **the Connection project is always required so that the ModBUS Master functions available in Application can be used.**

Mode
☐ Modbus Slave - BACnet MS/TP
☒ Modbus Master (for field)



The message that has to be sent will be read from the FC function by the given *object pointer*.

The pointer has a @INT data type, therefore **shorter/longer messages** (e.g. USINT, UDINT, REAL) to be sent **have to be first stored/split in an INT variable**. Then, the pointer to this last one has to be passed to the function.

Function: sysMbMRTU_FC06 (ver.1.0.0, EMBEDDED)		
Return Value: UINT		
Input:		
Name	Type	Description
addr	USINT	Physical address of the target slave
base	UINT	Address of the Register to write
object	@INT	Register pointer
timeout	UINT	Timeout [ms]

1.2.2. ModBUS Master: Application vs Connection

The ModBUS Master commands available in Application are **NOT a substitution** for the ModBUS functionalities available in **Connection**!

The ModBUS commands sent from Application will be executed in the Background task and therefore **will block the Background task** during their execution. The ModBUS commands on Event will be sent only after that the RS485 is free from other commands.

For example, if the RS485 is busy because of commands sent from Connection (e.g. a Slave is not available), *a on Event command sent from Application will block the background task execution until the RS458 will be free* and able to execute the on-Event command.

This means that the **commands on Event have to used only when required** and not instead of using Connection (to be used when possible).



1.2.3. ModBUS Master from Application: remarks

It is also important to **evaluate the command timeout** used when calling the FC functions, as it will block the Background task for the whole *timeout* period in case of any communication delay/error.

```
sysMbMRTU_FC16(0,ADR_sysClockSet_seconds,8,ADR(i_sysClock),1000);
```

In addition, it has to be checked if any **slave is unreachable**: the **commands** sent to the unreachable slaves should be **skipped**, otherwise the Background task will be blocked for the whole *timeout* period of each command.

It is a good practice **not to send too many ModBUS messages** per each background task cycle execution.



ModBUS Addons Library

2.1. Library Overview

The current documentation (v0.7) is related to the **ModBUS Addons library v.0.9.2**. The library is tested/supported on **FreeStudio 3.9.1 and greater versions**.

2.1.1. Library Description

The ModBUS Addons library makes full use of the ModBUS Master on Event functionalities available in FreeStudio Application beginning from the controller targets included with the version 3.9.1.

The **key functionalities** of the library are:

- The **Keep Alive** functionality, which allows to keep alive the slaves which require to receive at least one ModBUS message in a specifically defined timeframe, in order to keep the ModBUS communication active.
This goes around the limitation arisen by the timeouts for the messages defined in SoM Connection: in case some slaves are not connected, their timeouts may slow down the communication and therefore not allow to send messages to a certain (connected) slave with the desired frequency, leading to the loss of communication with that slave.
- Ability to **automatically manage** the node presence state (**sysMbRtuNodePresence**) based on the slave status, therefore enabling or disabling the SoM Connection nodes.
- **Optimized ModBUS on event messaging** operation, by sending messages on event only to the slaves which are effectively present.
- Ability to **monitor and inspect the status of the slaves**, keeping also a history of the ModBUS messages on event related to each slave.

The **targets** supported by this library version are:

- **ADVANCE** with BIOS 596.6 and greater;
- **ADVANCE 4DIN** with BIOS 668.6 and greater.

Note: the **rtuNodeMgr** function block of this library cannot be used for those slaves that use the Communication library. This has no impact on the Keep Alive functionality, which is independent.



2.1.2. Function Blocks and Structures List

The function blocks (FB), functions (F), structures (str) present in the library are the following:

Type	Name	Description
FB	ModbusRTUKeepAlive	It executes the Keep Alive functionality and monitor the slave connection status.
FB	rtuNodeMgr	It enables/disables the nodes defined in Connection, based on their presence.
FB	fbModbusRTU_SendFC03	FBs that use the ModBUS on event FC03, FC06 and FC16 functions, but send the messages only when the slave is effectively present.
FB	fbModbusRTU_SendFC06	
FB	fbModbusRTU_SendFC16	
str	strModbusRTUslave	Structure that contains all the information regarding the on event ModBUS messaging operation originating from a certain slave.

2.1.3. Library Implementation Overview

The below reported schematic synthetizes how the ModBUS Addons library has been designed to work and how the various function blocks and structures are interconnected.

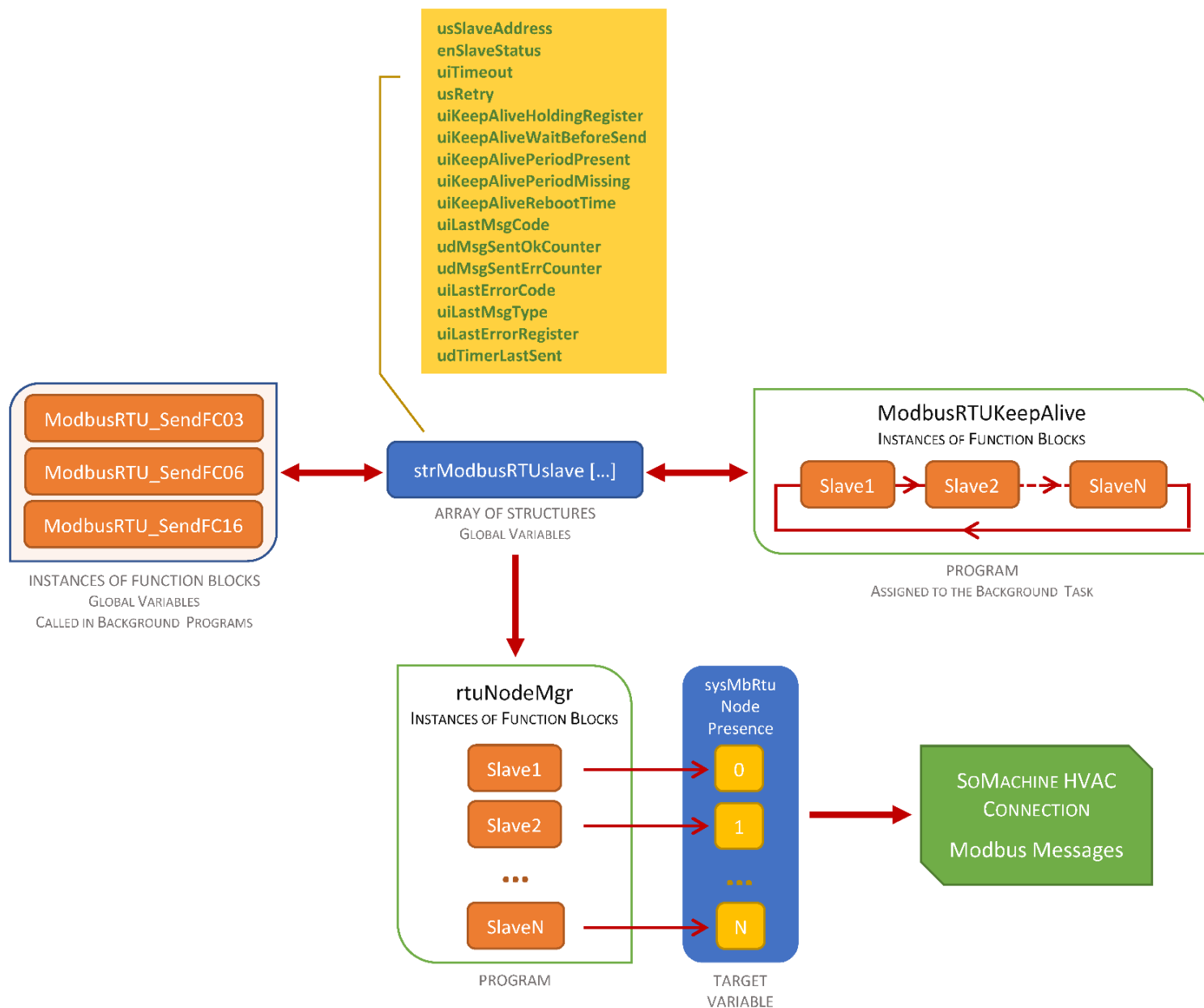
There is a **central structure**, the **strModbusRTUslave**, which will be defined as a global variable and there will be one per slave. This **structure** contains all the relevant information about the slave (address, status, timeout for the retry, number of retries, Keep Alive settings, communication statistics) and is used in all the function blocks that make use of the ModBUS on event functionalities.

This allows to keep the **structure** always aligned every time a message is sent and, on the other hand, it optimizes the communication, by not sending messages to the slaves which are not present.

The **ModbusRTUKeepAlive** is a function block that sends a message at each time interval defined by the *uiKeepAlivePeriodPresent* value, when the node status is PRESENT, and at each *uiKeepAlivePeriodMissing*, in all the other cases. This allows to keep alive the slaves that require to receive at least one ModBUS message at each specific timeframe, else those slaves disable the ModBUS communication. This could not be guaranteed if the messages are sent from Connection, e.g. in case there are not connected slaves that lead to long waits on the RS485, because of the timeout associated to each message.

Based on the outcome of each message sent by the ModbusRTUKeepAlive, the status of the related slave is updated. Considering that the slave information **structure** is passed whenever a ModBUS command is sent by using the fbModbusRTU_SendFC** FBs (e.g. a on event command defined in background and used by the specific application), it may happen that some or all of the messages that are supposed to be sent by the ModbusRTUKeepAlive are skipped, whenever in the defined timeframe another message has already been sent. This optimizes the functionality of the Keep Alive FB and reduces the traffic on the RS485.

It is required to instance one ModbusRTUKeepAlive per slave and assign the related program to the background task.



The **fbModbusRTU_SendFC03**, **fbModbusRTU_SendFC06** and **fbModbusRTU_SendFC16** function blocks use the ModBUS on event FC03, FC06 and FC16 functions, but keep also into account the content of the slave **structure**. This has the following benefits:

- a message is sent only if the target slave is effectively present;
- in case of a communication error, a send retry takes place;
- the **structure** is updated at each message sent; in details, the information about the status is updated and also the time of the last sent message, which allows to skip a Keep Alive message, if not required: in this case, it acts, in reality, in place of the Keep Alive.

In the ModBUS Addons library, the **fbModbusRTU_SendFC03**, **fbModbusRTU_SendFC06** and **fbModbusRTU_SendFC16** function blocks are already instanced in the **ModbusRTU_SendFC03**, **ModbusRTU_SendFC06** and **ModbusRTU_SendFC16** instances respectively. This allows to simplify the call of the FBs and reduces the RAM memory usage. These function blocks have to be called in Background.



The **rtuNodeMgr** function block automatically enables and disables the node presence related to the nodes defined in Connection, based on the slave status defined in the slave **structure**. In this way, when it is found that a slave is not present, the related node in Connection is disabled, in order not to delay the communication with the other present nodes.

This is done by writing on the `sysMbRtuNodePresence` target variable. In addition, the FB allows to manually force the presence of a node and to forcefully disable the writes for a node. One function block instance per slave has to be created.

For any additional detail about how to implement the library, the key reference is the sample project attached to the library itself.

2.2. Library Contents Details

This chapter goes through the details of each component of the ModBUS Addons library.

2.2.1. ModbusRTUKeepAlive

This function block allows to get the following functionalities:

- Monitor the slave communication status.
- Keep alive the slaves that require to receive at least one ModBUS message at each specific timeframe, else those slaves disable the ModBUS communication.

These goals are achieved by sending a ModBUS message to the targeted slave at each specific period. Based on the outcome of each message, the status of the related slave is updated. The slave is considered to be present only if it gets a correct answer.

It is required to create one instance of the FB per slave and assign the related program to the background task.

Inputs Description

All the settings, which are not defined as inputs, are included in the `strSlaveMonitored` **structure**.

Before sending a message, the FB waits for the `xMsgSentByOther` input to be TRUE. This allows to avoid the various instances of the FB sending messages all at the same time. The message is sent based on the `udTimerLastSent` information which is present in the slave **structure**; if a message has already been sent to the slave in the specific timeframe by any other call in the application, it is not sent again by this FB, in order to reduce the traffic on the RS485.

The `xWaitingReboot` input can be used to wait for the slave until its reboot, right after a reboot command has been sent (no ModBUS messages will be sent in the meanwhile). This will set the Slave in the REBOOTWAIT status and back to PRESENT after the reboot time, if the Slave will be reachable again.

Note that, if the Slave status is manually set to REBOOTWAIT and the Keep Alive functionality is not used, the status has to be restored to PRESENT manually.

After getting no reply from the Slave, its status will be set to MISSING. In this case, the Keep Alive will try to reach the slave with a different timeout, which is normally longer in order to reduce the traffic on the RS485.

The `xEN` input, when FALSE, will set the Slave status to DISABLED. In this case no message will be sent using the FCxx function blocks. Setting the `xEN` to TRUE will re-enable the slave and check whether it is present.

**Input:**

Name	Type	Description
xEN	BOOL	Slave Enabled
strSlaveMonitored	@strModbusRTUslave	Monitored Slave
xMsgSentByOther	BOOL	Message Sent by another instance
xWaitingReboot	BOOL	The slave has been rebooted externally, it works on rising edge

Output:

Name	Type	Description
iLastValueRead	INT	Last value read from the selected register
xMsgSent	BOOL	Message Sent by one of the instances

2.2.2. rtuNodeMgr

This function block automatically enables and disables the node presence related to the nodes defined in Connection, based on the slave status defined in the `strSlaveMonitored` **structure**.

In this way, when it is found that a slave is not present, the related node in Connection is disabled, in order not to delay the communication with the other present nodes. This is done by writing on the `sysMbRtuNodePresence` target variable.

The two additional inputs allow to:

- `xForceManualPresence`: manually force the presence of a node;
- `xForceWriteDisable`: forcefully disable the writes for a node.

Note: since the `rtuNodeMgr` function block of the ModBUS Addons library and the function blocks of the Communication library act both actively on the `sysMbMRtuNodePresence` array, the `rtuNodeMgr` cannot be used for those slaves that use the Communication library. This has no impact on the Keep Alive functionality, that can be used without encountering issues as it works independently.

Input:

Name	Type	Description
xEN	BOOL	
strSlaveMonitored	@strModbusRTUslave	Monitored Slave
iNode	INT	Slave Node Index
xForceManualPresence	BOOL	If true, <code>sysMbMRtuNodePresence</code> is forced to TRUE
xForceWriteDisable	BOOL	If true, <code>sysMbMRtuNodeDisableWrites</code> is forced to TRUE

Output:

Name	Type	Description
usStatus	USINT	0=Disabled 1=Running 2=Cfg Error



2.2.3. *fbModbusRTU_SendFC01*

This function block executes the `sysMbMRTU_FC01` command (Read Coils), by also taking into account the following:

- status of the slave (if not present, the message is not sent);
- number of retries in case of error;
- the `strSlaveMonitored` **structure** is updated with the info about the slave status, the time of the last sent message, the messages/errors counters, the last message code.

The function block returns a `UINT` which could have the following meanings:

- 0 = No error occurred
- 8 = Communication channel configuration error
- 14 = Invalid response message
- 16 = Timeout reached
- 17 = Function not executed due to try to call it in Timed
- 18 = Number of object out of range
- 19 = Function not executed because broadcast not allowed
- 255 = Message not sent, slave not present

Other values correspond to the Modbus Exception codes

2.2.4. *fbModbusRTU_SendFC02*

This function block executes the `sysMbMRTU_FC02` command (Read Discrete Input), by also taking into account the following:

- status of the slave (if not present, the message is not sent);
- number of retries in case of error;
- the `strSlaveMonitored` **structure** is updated with the info about the slave status, the time of the last sent message, the messages/errors counters, the last message code.

The function block returns a `UINT` which could have the following meanings:

- 0 = No error occurred
- 8 = Communication channel configuration error
- 14 = Invalid response message
- 16 = Timeout reached
- 17 = Function not executed due to try to call it in Timed
- 18 = Number of object out of range
- 19 = Function not executed because broadcast not allowed
- 255 = Message not sent, slave not present

Other values correspond to the Modbus Exception codes

2.2.5. *fbModbusRTU_SendFC03*

This function block executes the `sysMbMRTU_FC03` command (Read Holding Register), by also taking into account the following:

- status of the slave (if not present, the message is not sent);
- number of retries in case of error;



- the `strSlaveMonitored` **structure** is updated with the info about the slave status, the time of the last sent message, the messages/errors counters, the last message code.

The function block returns a UINT which could have the following meanings:

- 0 = No error occurred
- 8 = Communication channel configuration error
- 14 = Invalid response message
- 16 = Timeout reached
- 17 = Function not executed due to try to call it in Timed
- 18 = Number of object out of range
- 19 = Function not executed because broadcast not allowed
- 255 = Message not sent, slave not present

Other values correspond to the Modbus Exception codes

2.2.6. *fbModbusRTU_SendFC04*

This function block executes the `sysMbMRTU_FC04` command (Read Input Registers), by also taking into account the following:

- status of the slave (if not present, the message is not sent);
- number of retries in case of error;
- the `strSlaveMonitored` **structure** is updated with the info about the slave status, the time of the last sent message, the messages/errors counters, the last message code.

The function block returns a UINT which could have the following meanings:

- 0 = No error occurred
- 8 = Communication channel configuration error
- 14 = Invalid response message
- 16 = Timeout reached
- 17 = Function not executed due to try to call it in Timed
- 18 = Number of object out of range
- 19 = Function not executed because broadcast not allowed
- 255 = Message not sent, slave not present

Other values correspond to the Modbus Exception codes

2.2.7. *fbModbusRTU_SendFC05*

This function block executes the `sysMbMRTU_FC05` command (Write Single Coil), by also taking into account the following:

- status of the slave (if not present, the message is not sent);
- number of retries in case of error;
- the `strSlaveMonitored` **structure** is updated with the info about the slave status, the time of the last sent message, the messages/errors counters, the last message code.



The function block returns a UINT which could have the following meanings:

- 0 = No error occurred
- 8 = Communication channel configuration error
- 14 = Invalid response message
- 16 = Timeout reached
- 17 = Function not executed due to try to call it in Timed
- 255 = Message not sent, slave not present

Other values correspond to the Modbus Exception codes

2.2.8. *fbModbusRTU_SendFC06*

This function block executes the `sysMbMRTU_FC06` command (Write Single Register), by also taking into account the following:

- status of the slave (if not present, the message is not sent);
- number of retries in case of error;
- the `strSlaveMonitored` structure is updated with the info about the slave status, the time of the last sent message, the messages/errors counters, the last message code.

The function block returns a UINT which could have the following meanings:

- 0 = No error occurred
- 8 = Communication channel configuration error
- 14 = Invalid response message
- 16 = Timeout reached
- 17 = Function not executed due to try to call it in Timed
- 255 = Message not sent, slave not present

Other values correspond to the Modbus Exception codes

2.2.9. *fbModbusRTU_SendFC15*

This function block executes the `sysMbMRTU_FC15` command (Write Multiple Coils), by also taking into account the following:

- status of the slave (if not present, the message is not sent);
- number of retries in case of error;
- the `strSlaveMonitored` structure is updated with the info about the slave status, the time of the last sent message, the messages/errors counters, the last message code.

The function block returns a UINT which could have the following meanings:

- 0 = No error occurred
- 8 = Communication channel configuration error
- 14 = Invalid response message
- 16 = Timeout reached
- 17 = Function not executed due to try to call it in Timed
- 18 = Number of object out of range
- 255 = Message not sent, slave not present

Other values correspond to the Modbus Exception codes



2.2.10. *fbModbusRTU_SendFC16*

This function block executes the `sysMbMRTU_FC16` command (Write Multiple Registers), by also taking into account the following:

- status of the slave (if not present, the message is not sent);
- number of retries in case of error;
- the `strSlaveMonitored` **structure** is updated with the info about the slave status, the time of the last sent message, the messages/errors counters, the last message code.

The function block returns a `UINT` which could have the following meanings:

- 0 = No error occurred
- 8 = Communication channel configuration error
- 14 = Invalid response message
- 16 = Timeout reached
- 17 = Function not executed due to try to call it in Timed
- 18 = Number of object out of range
- 255 = Message not sent, slave not present

Other values correspond to the Modbus Exception codes

2.2.11. *strModbusRTUslave*

This structure contains all the relevant information about the slave (address, status, timeout for the retry, number of retries, Keep Alive settings, communication statistics) and is used in all the function blocks that make use of the ModBUS on event functionalities.

This allows to keep the structure always aligned every time a message is sent and, on the other hand, it optimizes the communication, by not sending messages to the slaves which are not present.

When the Slave Address is equal to 0, which corresponds to a ModBUS Broadcast message, the following settings are applied automatically when calling the FC function blocks that support the broadcast messages (05, 06, 15, 16), by changing the related slave structure:

- The status is set to `PRESENT`, as this property defines whether the message will be sent or not, but is undefined for the 0 address. Note: the Keep Alive cannot be used for the 0 address.
- The Timeout is set to 1 ms, as the slaves do not reply, therefore the waiting time will not bring any benefit, but only extend the background execution time.
- The Retry is set to 0, as the need to retry the send operation cannot be based on any feedback coming from the slaves, because the slaves do not reply to broadcast messages.

Name	Type	Description
usSlaveAddress	USINT	Address of the slave to be monitored
enSlaveStatus	enKeepAliveStatus	0=Missing 1=Present 2=Disabled 4=CFG ERROR 8=Reboot waiting
uiTimeout	UINT	Message Timeout [ms] - Default 400
usRetry	USINT	Number of retries - Default 1
uiKeepAliveHoldingRegister	UINT	Holding Register to be read for KeepAlive



uiKeepAliveWaitBeforeSend	UINT	Min Time to wait before sending a new message for KeepAlive [ms] - Default 600
uiKeepAlivePeriodPresent	UINT	Message period when slave is present [ms] - Default 1000
uiKeepAlivePeriodMissing	UINT	Message period when slave is not present [ms] - Default 30000
uiKeepAliveRebootTime	UINT	Time to be waited after the slave reboot
uiLastMsgType	USINT	Last Valid Message Type
udMsgSentOkCounter	UDINT	Number of messages sent correctly
udMsgSentErrCounter	UDINT	Number of error messages
uiLastErrorCode	UINT	Last Error Code
usLastErrorMsgType	USINT	Last Error Message Type
uiLastErrorRegister	UINT	Last Error Message First Register
udTimerLastSent	UDINT	sysTimer value of last message sent

2.2.12. enKeepAliveStatus

This enumerative is about the Slave Status and can have the following entries:

Name	Value	Description
MISSING	0	Slave Communication Error
PRESENT	1	Slave Online
DISABLED	2	Slave Disabled
CFGERROR	4	Configuration Error
REBOOTWAIT	8	Reboot Time Waiting

The FC function blocks included in this library make use of the Keep Alive status of the slave. The key points are:

- Only if the slave status is PRESENT, the FC function blocks send ModBUS messages.
- When the status is MISSING, the Keep Alive will still send messages, but with a different period (normally longer). In case the Keep Alive is not used, the status has to be manually reset to PRESENT.
- When the slave is DISABLED, no message is sent, not even from the Keep Alive. This status is set by the Keep Alive when its xEN is set to FALSE. It is restored when it is set back to TRUE.
- The REBOOTWAIT status allows not to send messages to a slave for a specific period, during which the slave is supposed to be rebooting. After that, the slave status can be set again back to PRESENT. This takes place automatically when the Keep Alive is used, otherwise it has to be reset manually.



Appendix

3.1. ModBUS Bridge Functionality

The **ADVANCE** can act as a **bridge** through which the Slaves (e.g. AVP) are accessible.

To enable the Bridge functionality, it is required to call the *sysBridge* function from the Application running on the ADVANCE.

- The first input of the *sysBridge* function is the address of the Slave for which you would like the Bridge function to be enabled.
- Passing 255 as address (as in the picture), all the Slaves will be enabled.

The *sysBridge_Priority* function allows to assign to the Bridge the same priority which the ModBUS messages managed by the ADVANCE have.

```
// Enable the bridge functionality
xRet:=sysBridge(255,TRUE,1000,1000,1000);

/* Used to assign the same priority of the ModBUS messages
   managed by the M172P to the bridge functionality */
xRet:=sysBridge_Priority(TRUE);
```

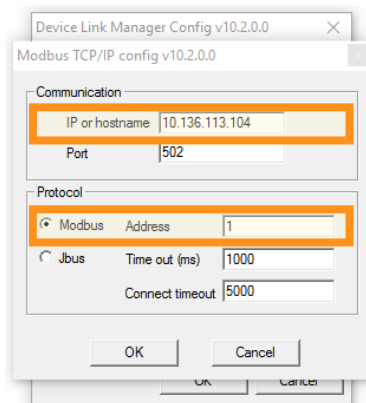
Bridge (ADVANCE) Application code in a Background task program

3.1.1. ModBUS Addresses

- When the bridge is **disabled**, the ADVANCE replies to all the ModBUS addresses [1-255] under the TCP/IP protocol.
- When the bridge is **enabled**, the ADVANCE uses only the 255 ModBUS address. The other addresses are left free for the Slaves. The messages sent to an address other than 255 are routed to the related Slave.

In order to **access a Slave** (e.g. AVP) by using the bridge function, the communication of the Slave has to be set as follows:

- The IP address is the bridge address (ADVANCE).
- The ModBUS address is the one of the Slave you would like to access (AVP address).





3.1.2. *Commands supported by the Bridge*

The Bridge **supports** the ModBUS Commands **0x03** (Read Holding Register) and **0x16** (Write Multiple Register) and the **application/BIOS update** from Device and Application.

In order to execute application/BIOS updates through ModBUS using the Bridge functionality, it is highly recommended to **temporarily disable all the ModBUS communications** on the network that normally take place from the master.

The aim is to have the highest available bandwidth for the specific ModBUS action.

This is equivalent to temporarily enable a **Service mode** that will need to:

- Set to FALSE all the *sysMbMRtuNodePresence* entries.
- Disable all the *on Event* ModBUS commands.



Publisher's Info

The **publisher** of this library is the **HVAC Solution Center** based in Alpago (BL), Italy.
Its main goal is to work on machine architecture solutions, software libraries and application notes.
Its **members** are:

- **Federico Marcassa**
federico.marcassa@schneider-electric.com
- **Pierpaolo Armeli**
pierpaolo.armeli@schneider-electric.com