Smart solutions for comfort and safety

saia-burgess
Smart solutions for comfort and safety

Modbus User Interface Document

FBs (xx7) Interface Description

Version:	V3.2
Date:	June 2009
Status:	Release
Classification:	Public

Contributors:

Names	Company
Luc Decornet	SBC

Revision History:

Version	Primary Authors	Description of Version	Date Approved
V1.0	L. Decornet	Initial version	26.08.2008
V1.1	L. Decornet	- Server Error codes modified - Client FBs added - Swap words added (Map option)	13.10.2008
V2.0	L. Decornet	- Call of FBs witth DBs modified - Baudrate parameter is a DINT	18.11.2008
V3.0	L. Decornet	- Error codes: descriptions modified - UID range authorized: 0-247 - Exception codes: description updated - FB CloseSRPort added	3.12.2008
V3.1	L.Decornet	- Saia Modbus principle - Modbus OFF - Broadcast UID (UID 0) modified	09.01.2009
V3.2	L. Decornet	- Parameter “StartAddr” in SFBs 492 and 493 (Send Read/Write Request) changed from WORD to DINT.	03.06.2009

Smart solutions for comfort and safety

Table of Contents

1 List of abbreviations

UID	Unit Identifier
HR	Holding Registers
IR	Input Registers
DI	Digital Inputs
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
LSB	Least Significant Byte
MSB	Most Significant Byte
MB	Merker Byte
MW	Merker Word
IW	Input Word
OW	Output Word
DB	Data Block
I/O	Inputs/Outputs
SR	Serial

2 Introduction

2.1 Purpose of this document

The Modbus xx7 FBs allow using the Modbus protocol on a PCD.

The library includes the following functionalities:

- Server creation
- Defining mappings (Server)
- Defining UIDs (Server)

2.2 The Modbus Protocol

Modbus is a request/response protocol which can be used through this API. It offers services specified by function codes.

On Client side, the essential communication steps are:

1. Create client instance
2. Establish connection with server (TCP)
3. Send request
4. Process response

Additionally it is possible to close the connection after having received the response or after an idle time.

On Server side, the essential communication steps are:

1. Create server instance
2. Process received request
3. Send response

2.3 Saia® Modbus

2.3.1 Saia® Modbus Server

The Saia® Modbus Server can be accessed over TCP/UDP or over serial line (RTU/ASCII). Many server instances can be defined on a Saia® PCD, allowing multiple accesses over serial or IP.

The Saia® Modbus server is also characterized by one or many Unit Identifiers (UID). A UID is refered by all the server instances of the station. A UID is also associated with a specific media mapping (see 2.4) and a specific data processing. It is possible to configure different UIDs on a Saia® PCD. A UID can be seen as a station number. The Modbus server processes all the requests addressed to one of its UIDs.

In serial mode, the server answers only to its defined UIDs. In case of broadcast UID (UID 0), the server processes the request for all defined UIDs.

In TCP/UDP, the server answers to all the requests, either wit an exception code or with a successful response. There is no broadcast UID.

Through the multiple UIDs, it is possible to use different media mapping configurations with different processing types.

2.3.2 Saia® Modbus Client

The Saia® Modbus Client enables the user to send Modbus Read and Write requests over TCP/UDP or serial line (RTU/ASCII). Many channels can be defined on a Saia® PCD, allowing communication over serial or IP to different Modbus servers.

A Modbus request is addressed to a specific UID (and an IP address for TCP/UDP). It is also possible to set the data processing that has to be applied to sent or received data.

A response from the server is expected within a specified time slice. If no answer comes within this time slice, the server does not exist (or the UID is not defined in serial mode). In case of broadcast UID (UID 0) over serial line, no answer is expected from the client. It has just to be waited for the defined time before sending the next request.

REMARK: It is not allowed to use 2 Modbus masters on the same serial network. The correct behaviour of the network in case 2 clients communicate simultaneously on the bus cannot be guaranteed.

2.4 Modbus Media and xx7 Media

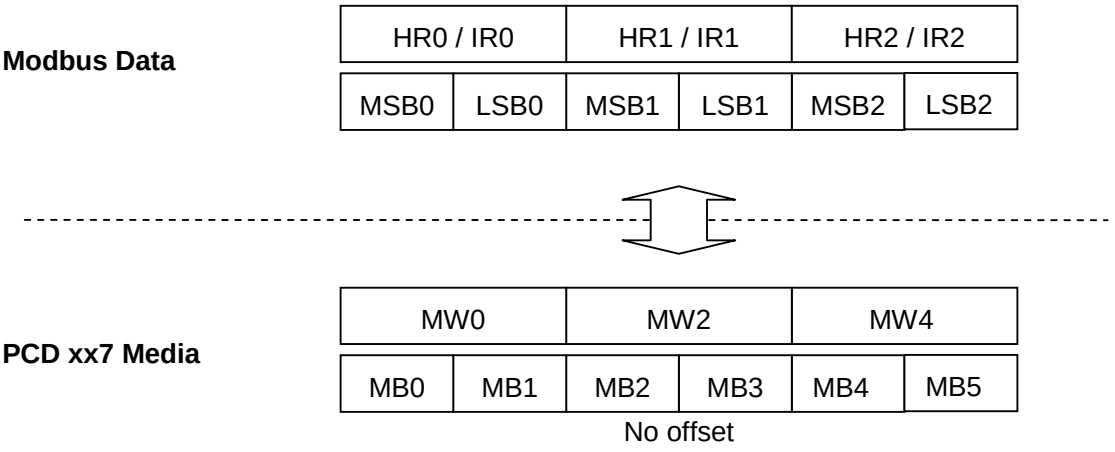
Modbus uses 4 media types: Coils, Discrete Inputs, Holding Registers, Input Registers.

In order to make PCD media accessible on a Modbus server, these Modbus media have to be mapped on PCD Media; the mapping areas and the UIDs enable the user to configure the mapping of Modbus Media on Saia®PCD Media. **At least one UID has to be defined**.

Coils and Discrete Inputs can be mapped only on Merkers, Inputs or Outputs. The xx7 Inputs can only be accessed via Read Requests.

Holding Registers and Input Registers can only be mapped on Merkers, Inputs, Outputs, Timers or Counters.

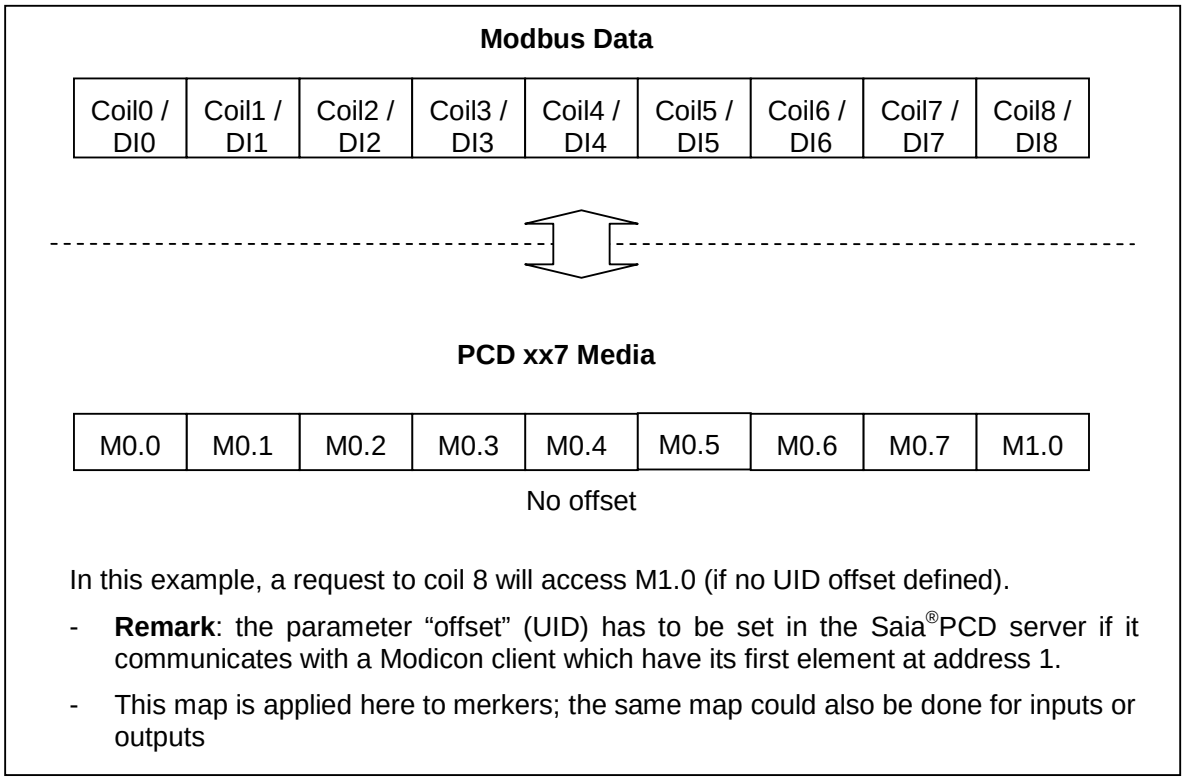
The following schema shows how the data are processed between PCD media and Modbus Frame.



In this example, a request to Holdin Register 2 will access MW4 (if no UID offset defined).

- **Remark**: the parameter “offset” (UID) has to be set in the Saia®PCD server if it communicates with a Modicon client which have its first element at address 1.
- This map is applied to merkers; the same map could also be done for inputs or outputs

Figure 1: Server Map of Modbus Holding Registers / Input Registers on xx7 Media (Merkers, Inputs, Outputs), assuming that HR0 / IR0 is mapped on MW 0



In this example, a request to coil 8 will access M1.0 (if no UID offset defined).

- **Remark**: the parameter "offset" (UID) has to be set in the Saia®PCD server if it communicates with a Modicon client which have its first element at address 1.
- This map is applied here to merkers; the same map could also be done for inputs or outputs

Figure 2: Server Map of Modbus Coils / Discrete Inputs on xx7 Media (Merkers, Inputs, Outputs), assuming that Coil0 / DI0 is mapped on M0.0

Default Mapping

		Modbus Request			PCD		
AccessType	**Area Processing**	**MediaType**	**Start Addr.**	**Range**	**MediaType**	**Start Addr.**	**Range**
ReadWrite	NO_PROCESSING	MB_HOLDING_REG_MEDIA	1	10000	PCD_XX7_MERKER	0	20000
ReadWrite	SWAP_WORDS	MB_HOLDING_REG_MEDIA	10001	10000	PCD_XX7_MERKER	0	20000
ReadOnly	NO_PROCESSING	MB_INPUT_REG_MEDIA	1	10000	PCD_XX7_COUNTER	0	10000
ReadOnly	NO_PROCESSING	MB_INPUT_REG_MEDIA	10001	10000	PCD_XX7_TIMER	0	10000
ReadWrite	NO_PROCESSING	MB_COILS_MEDIA	1	10000	PCD_XX7_MERKER	0	10000
ReadWrite	NO_PROCESSING	MB_COILS_MEDIA	10001	10000	PCD_XX7_OUTPUT	0	10000
ReadOnly	NO_PROCESSING	MB_DISCR_INPUT_MEDIA	1	10000	PCD_XX7_MERKER	0	10000
ReadOnly	NO_PROCESSING	MB_DISCR_INPUT_MEDIA	10001	10000	PCD_XX7_INPUT	0	10000

3 Modbus System Functions Library

3.1 FB Parameters

Name	Used	Description
AccessType	InitMap	Defines the way the mapping area can be accessed (Read/Write). See 3.8
Baudrate	Init_Slave	Defines Baudrate used for serial communication in Bauds (from 1200 to 115200)
ChannelID	OpenIPChannel OpenSRChannel SendReadReq SendWriteReq	A client station can implement one or many logical channels. The ChannelID is used for creating or modifying a channel, or for sending a request to a remote partner. Range authorized: 1-10
ClearDiag	GetUID_Diag GetServer_Diag	Defines which diagnostic fields have to be cleared after being read: Bit 0 = clear Diagnostic Register Bit 1 = clear Request Counter (Only for GetUID_Diag)
CloseTimeout	OpenIPChannel	Timeout to close a connection after an idle time (in seconds) – Only used in TCP/UDP

Data	SendReadReq
	SendWriteReq
		Read Request: defines at which which location on the PCD the requested data will be copied.
		Write Request: Defines from which location in the PCD the data to send will be taken
		The Data shall be built as an ***ANY*** pointer. It can point to DB, merker, input and output areas.
		Examples:
		`p#DB10.DBX0.0 BYTE 20`
		DB10 area of 20 bytes, starting at offset 0.
		`p#M100.0 DINT 20`
		merker area of 20 double words (= 80 bytes), starting at offset 100.
		`p#E150.1 BOOL 9`
		input area of 9 bits, starting at bit 1 from byte 150.
		Remark: the type BOOL is only allowed in case of binary access (Coils or DI).

DataBits	Init_Slave	Number of databits in a character: 7 or 8
DiagReg	GetUID_Diag GetServer_Diag	Diagnostic register (see 3.10.1)
Done	SendWriteReq	This out bit is set to 0 as long as the job is not finished or if the job could not be started. It is set to 1 when the job is finished successfully.
Error	SendReadReq	This output bit indicates an error, either in a parameter of

	SendWriteReq	the FB call or in the processing of the request.
ExceptionCode	SendReadReq SendWriteReq	Output word that contains the exception code in case of Exception response. If no exception response, it is set to 0.
FunctionCode	SendReadReq SendWriteReq	This parameter defines the type of the request (Read/Write, Coils/Holding Registers, …). See 3.5.
IPAddr	SendReadReq SendWriteReq	IP address of the remote partner. It is coded on a DWORD where each byte corresoponds to a part of the IP address Ex: for 172.23.2.156, the value is 0xAC17029C (0xAC = 172, 0x17 = 23, 0x02 = 2, 0x9C = 156)
Length	SendReadReq SendWriteReq	Number of Modbus elements to Read/Write. Max number of coils / DI to read: 2000 Max number of coils / DI to write: 1968 Max number of HR / IR to read: 125 Max number of HR / IR to write: 123
LineType	Init_Slave	Type of serial line used: - 0: RS232 - 2: RS422 - 3: RS232, no handshake - 4: RS422, no handshake - 7: RS485, auto RTS
MapOptions	InitMap	*Bit 0: Swap Words* The 2 words of a DWordhave to be swapped (either before sending data, or when receiving data). Can be useful with some devices that work word-swapped on 32bits values. *Remark*: the swap can only be executed in HR or IR access, with an even number of these media (1 HR/IR is 16 bits long). *Bits 1-31*: Reserved
NDR	SendReadReq	This out bit is set to 0 as long as the job is not finished or if the job could not be started. It is set to 1 when the job is finished successfully, that is when a correct answer to a ReadRequest has been received (New Data Received).
Options	InitMap InitUID SendReadReq SendWriteReq	InitMap → see MapOptions InitUID → see UIDOptions SendRequest → see SendOptions
Parity	Init_Slave	Character parity: can be ‘E’ (Even), ‘O’ (Odd) or ‘N’ (None)
PartnerMedia	InitMap	This parameter defines the type of Modbus data accessed by the client. See 3.7

PartnerRange	InitMap	Number of **Modbus elements** in the mapping area.
PartnerStart	InitMap	Modbus adddress of the first element of the mapping area.
PCDMedia	InitMap	PCD media type on which the Modbus data with type “PartnerType” will be mapped. See 3.9
PCDRange	InitMap	Number of PCD elements in the mapping area.
PCDStart	InitMap	PCD Start Address corresponding to Partner Start.
Port	InitServer Init_Slave GetServer_Diag OpenIPChannel OpenSRChannel	Number of the port used for the server instance (TCP/UDP: logical port)
Protocol	InitServer Init_Slave GetServer_Diag OpenIPChannel OpenSRChannel	Protocol used for modbus communication. See 3.6
Req	SendReadReq SendWriteReq	A request is sent on a rising edge of this input bit.
ReqCount	GetUID_Diag	Counter of requests received for a UID
RespTimeout	OpenIPChannel OpenSRChannel	Timeout within which a response shall be received (in milliseconds)
Retries	OpenIPChannel OpenSRChannel	Number of retries in case of response timeout (1st try included). Range: 1 - 15
SendOptions	SendReadReq SendWriteReq	Options for sending Request *Bit* 0: Swap Words The 2 words of a DWord have to be swapped (before sending data / when receiving data). Can be useful with some devices that work word-swapped on 32bits values. *Remark*: the swap can only be executed in HR or IR access, with an even number of these media. *Bits 1-31*: Reseved
StartAddr	SendReadReq SendWriteReq	Modbus address of the first element to read/write.
Status	InitMap InitServer InitUID Init_Slave GetServer_Diag GetUID_Diag OpenIPChannel OpenSRChannel SendReadReq	State after the FB has been executed. If 0x8… an error occurred during the call or the request processing. Refer to the error code list (§3.3) When the function has been successfully executed, status is >= 0 For asynchronous calls (Send Read/Write Request): see 0.

	SendWriteReq	
StopBits	Init_Slave	Number of stopbits in a character (1 or 2)
UID	InitUID InitMap GetUID_Diag SendReadReq SendWriteReq	The Unit Identifier (UID) is inserted into the request. When a request is received, the UID is used to find out which Mode has to be used and which processing will be performed on data. More than one UID can be defined on a server station. The UIDs are refered by all server instances on the station. It is recommended to use only one UID on a station in serial mode. In Serial mode, if a request with UID 0 (broadcast UID) is received, all the defined UIDs react to this request with their own processing and mapping. In TCP/UDP, the UID 0 and the UID 255 correspond to the same UID. **To access this UID on a TCP/UDP server, the user has to configure its parameters. It can be done using the InitUID CSF.** It is recommended to avoid accessing a TCP/UDP server with UID 0.
UIDOptions	InitUID	Defines the processing to be performed on data *Bit 0 = DEFMAP*: defines in which mapping table the modbus addresses will be searched DEFMAP=0: only user defined mapping has to be used. DEFMAP=1: first search in user defined mapping. If not found, search in default mapping (see DefaultMapping). *Bit1 = OFFSET* OFFSET =1 for communication with a Modicon client (first element at address 1) OFFSET =0 for other devices with first register at address 0 (Saia®PCD, …).

3.2 Asynchronous functions

The Send Read/Write Request FB's have been implemented to be executed asynchronously. This means the FB call launches a job which will be executed in background while the user program is continuing its execution. This also implies that the FB has to be called again with the same parameters (same instance DB) until the job is finished. As long as the job is not finished, the FB call will return dedicated code (`0x7001/0x7002`). If another asynchronous CSF call is issued while the previous job is not finished, the FB call will return (`0x7000`).

The following table gives the possible combinations FBs output parameters.

Done / NDR	Error	Status	Description
0	0	0x7000	Another asynchronous job is already in execution. Current job has not been submitted.
0	0	0x7001 0x7002	Current job is not yet finished
1	0	0	Current job is finished without error.
0	1	= 0x 8 x x x	Current job is finished with error.
0	0	0x8023	Response Timeout occurred but retry is on.

Table 1: Meaning of OUT flags for asynchronous jobs.

3.3 Error codes

In case of successful processing, all FBs return either 0 (zero) or a positive value. A negative value indicates an error. Below is a list of the error codes:

3.3.1 CSF error codes

Most of these errors can appear when one or more CSF parameters are not correct.

Code	Designation	Description
0x8064	-	
0x8063	DATA_AREA_TOO_LONG	Area length (last element of the area) exceeds las media available
0x8062	DATA_AREA_TOO_SHORT	Area too short for the length specified (SendRequest)
0x8061	DATA_AREA_ERROR	Data area is not valid
0x8060	-	-
0x805F	-	-
0x805E	CHANNEL_NOT_DEFINED	Channel is not defined (Send Request)
0x805D	CHANNEL_BUSY	Tries to send a request on a channel already used for another request
0x805C	SERVER_NOT_CREATED	Server cannot be created (Init Server): max number of servers already reached or server already defined.
0x805B	SERVER_NOT_STARTED	Server cannot start (Init Server)
0x805A	INVALID_FUNCTION_CODE	Function code does not match request type (Read / Write)
0x8059	-	-
0x8058	INVALID_PROTOCOL	Protocol is invalid or does not correspond with FB.
0x8057	INVALID_PARTNER_TYPE	Modbus Media is not a valid type(Init Map)
0x8056	INVALID_PCDMEDIA_TYPE	PCD Media is not a valid type(Init Map)
0x8055	INVALID_PARTNER_ADDR	Partner start address is invalid (exceeds 0xFFFF = 65535)
0x8054	INVALID_PCD_MEDIA_ADDR	PCD Media address is invalid (Init Map): last element of the PCD media defined exceeds last media available
0x8053	INVALID_RANGE	Invalid media range (Init Map): Partner range < PCD range or range = 0.
0x8052	INVALID_AREA_ACCESS_TYPE	Access type is invalid (Init Map).
0x8051	INVALID_NB_OF_MEDIA	Number of media does not match function code (Send Request): > 1 for write single coil or single register, 0 for any request, swap set with an odd number of HR/IR…
0x8050	INVALID_UID	UID out of the range 0-255 (Init UID, Send Request) or UID = 0 for a read request in serial mode or UID nor defined (GetUID_Diag)
0x804F	INVALID_CHANNEL	Channel out of the range 1-10 (Open Channel, Send Request)
0x804E	INCOMPATIBLE_PCDMEDIA	PCD media is incompatible with Modbus media (Init Map: Coils/DI ←→Timers/Counters) or Function code (Send Request: tries to access HR or IR in a BOOL area).
0x804D	INVALID_PORT_CONF	Port configuration is invalid: check parameters (port number, databits, stopbits, baudrate, …)
0x804C	-	-
0x804B	NO_PORT_TO_CLOSE	No serial port to close: for FB494, no channel or server related to the port passed as parameter.
0x804A	SRPORT_START_ERROR	SR port cannot be open (Init Server, Init SR Port). In serial, the port may be already used or port parameters are not correct for the port concerned.
0x8049	INVALID_PORT_NUMBER	SR Port Number is invalid or the port is not open
0x8048	INVALID_RETRY_NB	Number of retries invalid (must be < 16)

3.3.2 ModbusShell error codes

These errors appear when processing CSF.

Code	Designation	Description
0x8032	UID_TABLE_FULL	Maximum number of UIDs is allready reached (Init UID)
0x8031	UID_NOT_DEFINED	UID is not defined (Init Map)
0x8030	UID_BUSY	UID is being used (Init UID), cannot modify it
0x802F	CHANNEL_TABLE_FULL	Channel table is full
0x802E	CHANNEL_ALREADY_DEFINED	A channel with same pair Port / Protocol is allready defined (Open Channel)
0x802D	CHANNEL_BUSY	This channel is being used, cannot modify it
0x802C	MAP_TABLE_FULL	Maximum number of mappings reached for the UID (Init Map)
0x802B	MAP_AREA_OVERLAP	At least 2 map areas overlap (Init Map)
0x802A	CONNECTION_TABLE_FULL	Maximum number of “connections” is reached (Send Request). TCP/UDP: connection = {Port-Protocol-IPAddress} Serial: connection = {Port-Protocol}
0x8029	CONNECTION_NOT_READY	Connection is not ready for sending (Send Request). (internal error)
0x8028	DRIVER_ERROR	Error caused by Modbus driver (Internal error)
0x8027	CLIENT_START_ERROR	Client cannot start (no more socket available, …)
0x8026	CLIENT_SEND_ERROR	Error while sending (in RTU/ASCII, check handshake)
0x8025	CLIENT_RESPONSE_ERROR	Error in response
0x8024	CLIENT_RESPONSE_EXCEPTION	Exception response received
0x8023	RESPONSE_TIMEOUT	Response not received after a timeout
0x8022	NO_RESPONSE_AFTER_RETRIES	Response not received after retries
0x8021	CLIENT_STATUS_ERROR	Invalid client status (internal error)
0x8020	SERIAL_ERROR	Error in serial communication (parity, framing, break, CRC,…)
0X801F	UNKNOWN_ERROR	Unknown error (internal error)
0X801E	MEDIA_ACCESS_ERROR	Error while accessing media (internal error)

3.3.3 ModbusDriver error codes

Code	Designation	Description
0x8FA0	-	-
0x8F9F	UID_TABLE_FULL	UID table is full
0x8F9E	UID_NOT_DEFINED	Trying to access a non existing UID
0x8F9D	UID_BUSY	UID is busy
0x8F9C	INVALID_PARAMETER	Call with invalid parameters
0x8F9B	SI_ALREADY_DEFINED	Trying to create an already defined server instance
0x8F9A	SI_TABLE_FULL	Server instance table full
0x8F99	SI_NOT_DEFINED	Server instance not found in table
0x8F98	CLII_ALREADY_DEFINED	Client/Connection instance already defined
0x8F97	CLII_TABLE_FULL	Client/Connection table full
0x8F96	CLII_DRIVER_NOT_READY	Driver is not ready (Send Request)
0x8F95	TOO_MANY_CI	Too many client instances for this server instance
0x8F94	ADDRESS_ERROR	Address error in media processing
0x8F93	QUANTITY_ERROR	Error in the quantity of modbus media to read / write (Send Request): exceeds limit or < 0
0x8F92	INVALID_MEDIA	Invalid media for this function
0x8F91	MEDIA_ACCESS_ERROR	Media cannot be accessed correctly

0x8F90	CLIENT_INSTANCE_NOT_FOUND	Client/Connection instance does not exist
0x8F8F	CLIENT_NO_RESSOURCE	No resource available (Send Request)
0x8F8E	PORT_ID_ERROR	Port already initialized or used
0x8F8D	INVALID_PORT_STATE	Port not ready (Serial) (Send Request)
0x8F8C	UNKNOWN_ERROR	Unknown error

3.4 Exception Codes (sent from Server)

ILLEGAL_FUNCTION_CODE	1	Function code not valid or does not correspond to Modbus media type
ILLEGAL_DATA_ADDRESS	2	Modbus media at received address cannot be accessed (check UID / see diagnostic)
ILLEGAL_DATA_VALUE	3	Number of data in the request is invalid
SERVER_FAILURE	4	PCD Media could not be accessed (see diagnostic) or UID not configured (in particular UID 255 in TCP/UDP)
ACKNOWLEDGE	5	
SERVER_BUSY	6	Server is already busy when trying to access it
GATEWAY_PATH_NOT_FOUND	10	UID not defined
GATEWAY_TARGET_PROBLEM	11	

3.5 Function Codes

```
READ_COILS              0x01
READ_DISC_INPUT         0x02
READ_HOLD_REG           0x03
READ_INPUT_REG          0x04
WRITE_SINGLE_COIL       0x05
WRITE_SINGLE_REG        0x06
WRITE_MULTIPLE_COILS    0x0F
WRITE_MULTIPLE_REGS     0x10
```

3.6 Protocols

```
MODBUS_TCP                  0
MODBUS_UDP                  1
MODBUS_RTU                  2
MODBUS_ASCII                3
```

3.7 Modbus Data Types

```
MB_NO_MEDIA                 0
MB_COILS_MEDIA              1
MB_DISCR_INPUT_MEDIA        2
MB_HOLDING_REG_MEDIA        3
MB_INPUT_REG_MEDIA          4
```

3.8 Modbus Area Access Types (Server)

```
READ_WRITE              0
READ_ONLY               1
WRITE_ONLY              2
NO_ACCESS               3
```

3.9 PCD Media Types

```
XX7_NO_MEDIA            0
XX7_MERKER              1       4096 MerkerBytes available
XX7_TIMER               2       512 Timers available
XX7_COUNTER             3       512 Counters available
XX7_INPUT               4       256 InputBytes available
XX7_OUTPUT              5       256 OutputBytes available
```

3.10 Diagnostic

The diagnostic register is used to report the state of the UIDs (UID Diagnostic) and the Channels (Channel Diagnostic). It is also used report specific server errors.

In the diagnostic register, all the bits but “Retry count” and “Response timeout” are cumulative (that is the user has to clear the register to reset them using option in FB 504 or FB505).

3.10.1 Diagnostic Register

	Bit	Designation	Description	Protocol	Used for
RECEPTION	0	*Overrun Error*	Overrun of the internal reception buffer	Serial	Server
	1	*Parity Error*	Parity error in the received character	Serial	Server
	2	*Framing Error*	Incorrect baudrate	Serial	Server
	3	*Break Error*	Break on the serial line	Serial	Server
	4	*CRC Error*	CRC of the frame is incorrect	Serial	Server
	5	-	-		
	6	-	-		
	7	-	-		
	8	*Length Error*	Telegram with invalid length received	Serial	Server
	9	*Media Access Error*	Error while accessing PCD Media	All	UID
	10	-	-		
	11	*Server Start Error*	Server could not start	All	Server
	12	*Range Error*	Error in the address or number of Modbus elements to access (see below).	All	UID
	13	*Value Error*	A value in the telegram is invalid (Media type, …)	All	UID
	14	*Area access error*	Area not accessible (UID busy, area access not allowed)	All	UID
	15	-	-		
TRANSMISSION	16	-	-	-	-
	17				
	18				
	19				
	20	-	-	-	-
	21	-	-	-	-
	22	-	-	-	-
	23	-	-	-	-
	24	-	-	-	-
	25	-	-	-	-
	26	-	-	-	-
	27	-	-	-	-
	28	-	-	-	-
	29	-	-	-	-
	30	-	-	-	-
	31	-	-	-	-

A "**Range Error**" may have different causes:

- No corresponding mapping found: no mapping area corresponding to data start address and number of Modbus elements could be found in User defined mapping areas (and also in default mapping areas if DEFMAP = 1).

- Number of PCD media to access exceeds PCD range of the area: for example when trying to access 8 HR/IR at offset 2 in an area containing 16 bytes.

If this error occurs, check address of modbus element requested and mapping areas and make sure that the requested address can be accessed.

A "**Media Error**" can occur if number of PCD media to access exceeds the limits defined in 3.9 or if an error occurs while accessing PCD media (PCD media invalid, address of PCD media invalid…).

3.11 Modbus Limitations

Servers: maximum 4 on a PCD system (a server is defined by a unique {port / protocol} pair) – For example: 1 server TCP port 502, 1 server TCP port 503, 1 server UDP port 502 and 1 server RTU on serial port 2.

UIDs: maximum 10 on a PCD system: 10 = 9 + UID 255 (UID from 0 to 254 are authorized; UID 255 exists by default on server **but it must be configured before being accessed**).

Mapping Areas: maximum 10 per UID – If more than 10 mapping areas are necessary, an additional UID can be defined (example: define a UID for Coils/Discrete Inputs and another for Holding/Input Registers).

Connections: 26 connections can be open simultaneously. From these 26, maximum 10 connections can be open on client side. The remaining connections can be used by server. Remark: a connection is defined by a unique triplet {Port –Protocol – IP Address} in TCP / UDP and only pair Port / Protocol) in serial mode). The connections are open at the first Modbus request. In TCP/UDP, the connections are closed automatically after a user defined time.

4 Modbus FB Description

4.1 FB OpenIPChannel (FB 490)

This function allows creating a TCP/UDP channel on client side.

Multiple use of channel with same pair Port/Protocol is not allowed.

```
CALL FB 490 , DB490
     ChannelID      := [IN]   [INT]    See ChannelID parameter
     Port           := [IN]   [INT]    See Port parameter
     Protocol       := [IN]   [INT]    See Protocol parameter
     CloseTimeout   := [IN]   [DINT]   See CloseTimeout parameter
     RespTimeout    := [IN]   [DINT]   See RespTimeout parameter
     Retries        := [IN]   [INT]    See Retries parameter
     Status         := [OUT]  [WORD]   See Status parameter
```

The FB 490 (“OpenIPChannel”) will call the SFB 490. The FB call requires a (multi-)instance data block, referenced here has DB490.

4.2 FB OpenSRChannel (FB 491)

This function allows creating a Serial channel on client side.

Multiple use of channel with same pair Port/Protocol is not allowed.

REMARK: It is not allowed to use 2 Modbus masters on the same serial network. The correct behaviour of the network in case 2 masters communicate simultaneously on the bus cannot be guaranteed.

```
CALL FB 491 , DB491
     ChannelID      := [IN]   [INT]    See ChannelID parameter
     Port           := [IN]   [INT]    See Port parameter
     Protocol       := [IN]   [INT]    See Protocol parameter
     Baudrate       := [IN]   [DINT]   See Baudrate parameter
     DataBits       := [IN]   [INT]    See DataBits parameter
     Parity         := [IN]   [CHAR]   See Parity parameter
     StopBits       := [IN]   [INT]    See StopBits parameter
     LineType       := [IN]   [INT]    See LineType parameter
     RespTimeout    := [IN]   [DINT]   See RespTimeout parameter
     Retries        := [IN]   [INT]    See Retries parameter
     Status         := [OUT]  [WORD]   See Status parameter
```

The FB 491 (“OpenSRChannel”) will call the SFB 491. The FB call requires a (multi-)instance data block, referenced here has DB491.

4.3 FB SendReadReq (FB 492)

This function allows sending a Modbus Read request.

```
CALL FB 492 , DB492
     Req            := [IN]   [BOOL]   See Req parameter
     ChannelID      := [IN]   [INT]    See ChannelID parameter
```

```
    IPAddr        := [IN]   [DWORD]  See IPAddr parameter
    UID           := [IN]   [INT]    See UID parameter
    FunctionCode  := [IN]   [INT]    See FunctionCode parameter
    StartAddr     := [IN]   [DINT]   See StartAddr parameter
    Length        := [IN]   [INT]    See Length parameter
    Options       := [IN]   [DWORD]  See SendOptions parameter
    NDR           := [OUT]  [BOOL]   See NDR parameter
    Error         := [OUT]  [BOOL]   See Error parameter
    ExceptionCode := [OUT]  [BYTE]   See ExceptionCode parameter
    Status        := [OUT]  [WORD]   See Status parameter
    Data          := [I/O]  [ANY]    See Data parameter
```

The FB 492 (“SendReadReq”) will call the SFB 492. The FB call requires a (multi-)instance data block, referenced here has DB492.

REMARK: to send Read Requests on 2 different channels, do not change the channel ID while a request is being processed. Make sure that the request on a channel is finished (*Error* or *NDR* set) before sending on another channel or use two function calls with 2 different instance DBs.

Moreover, a request cannot be sent on a channel if the previous request on this channel is not finished (*Error* or *NDR* set).

4.4 FB SendWriteReq (FB 493)

This function allows sending a Modbus Write request.

```
CALL FB 493 , DB493
     Req           := [IN]   [BOOL]   See Req parameter
     ChannelID     := [IN]   [INT]    See ChannelID parameter
     IPAddr        := [IN]   [DWORD]  See IPAddr parameter
     UID           := [IN]   [INT]    See UID parameter
     FunctionCode  := [IN]   [INT]    See FunctionCode parameter
     StartAddr     := [IN]   [DINT]   See StartAddr parameter
     Length        := [IN]   [INT]    See Length parameter
     Options       := [IN]   [DWORD]  See SendOptions parameter
     Done          := [OUT]  [BOOL]   See Done parameter
     Error         := [OUT]  [BOOL]   See Error parameter
     ExceptionCode := [OUT]  [BYTE]   See ExceptionCode parameter
     Status        := [OUT]  [WORD]   See Status parameter
     Data          := [I/O]  [ANY]    See Data parameter
```

The FB 493 (“SendReadReq”) will call the SFB 493. The FB call requires a (multi-)instance data block, referenced here has DB493.

REMARK: to send Read Requests on 2 different channels, do not change the channel ID while a request is being processed. Make sure that the request on a channel is finished (*Error* or *Done* set) before sending on another channel or use two function calls with 2 different instance DBs.

Moreover, a request cannot be sent on a channel if the previous request on this channel is not finished (*Error* or *Done* set).

4.5 FB CloseSRPort (FB 494)

This function allows closing a serial Port and all the related instances (Channel, Server).

```
CALL FB 49 , DB494
     Port           := [IN]   [INT]     See Port parameter
     Status         := [OUT]  [WORD]    See Status parameter
```

The FB 494 (“CloseSRPort”) will call the SFB 494. The FB call requires a (multi-)instance data block, referenced here has DB494.

REMARK: If the port closed is related to a server, the requests coming on this port cannot be answered anymore. If the port closed is related to a channel, all the FB493 and FB492 call with this channel as parameter will return an error. For sending / receiving on the serial port that has been closed, the appropriate initialization functions have to be called (*Init_Slave* , *OpenSRChannel*).

4.6 FB InitServer (FB 500)

This function allows creating a TCP/UDP server instance.

Multiple use of server with same pair Port/Protocol is not allowed.

```
CALL FB 500 , DB500
     Protocol       := [IN]   [INT]     See Protocol parameter
     Port           := [IN]   [INT]     See Port parameter
     Status         := [OUT]  [DWORD]   See Status parameter
```

The FB 500 (“InitServer”) will call the SFB 500. The FB call requires a (multi-)instance data block, referenced here has DB500.

4.7 FB Init_Slave (FB 501)

This function allows creating a serial server instance.

Multiple use of server with same Port is not allowed.

```
CALL FB 501 , DB501
     Protocol       := [IN]   [INT]     See Protocol parameter
     Port           := [IN]   [INT]     See Port parameter
     Baudrate       := [IN]   [DINT]    See Baudrate parameter
     DataBits       := [IN]   [INT]     See DataBits parameter
     Parity         := [IN]   [CHAR]    See Parity parameter
     StopBits       := [IN]   [INT]     See StopBits parameter
     LineType       := [IN]   [INT]     See LineType parameter
     Status         := [OUT]  [DWORD]   See Status parameter
```

The FB 501 (“Init_Slave”) will call the SFB 501. The FB call requires a (multi-)instance data block, referenced here has DB501.

4.8 FB InitUID (FB 502)

This function creates or modifies a unit identifier which will be referred by all the server instances. When a request is received, the UID is used to find out which Mode has to be used and which processing will be performed on data. More than one UID can be defined on a server station.

When trying to create a UID with an already defined UID number, the “old” one is overwritten (if not currently in use, see RBSY).

```
CALL FB 502 , DB502
     UID           := [IN]   [INT]    See UID parameter
     Options       := [IN]   [DWORD]  See UIDOptions parameter
     Status        := [OUT]  [DWORD]  See Status parameter
```

The FB 502 (“InitUID”) will call the SFB 502. The FB call requires a (multi-)instance data block, referenced here has DB502.

4.9 FB InitMap (FB 503)

This function allows defining a mapping area.

It is not allowed to create an area which overlaps an other one (same Modbus media type and overlapping Modbus address ranges)

```
CALL FB 503 , DB503
     PartnerMedia  := [IN]   [INT]    See PartnerMedia parameter
     PartnerStart  := [IN]   [DINT]   See PartnerStart parameter
     PartnerRange  := [IN]   [DINT]   See PartnerRange parameter
     PCDMedia      := [IN]   [INT]    See PCDMedia parameter
     PCDStart      := [IN]   [DINT]   See PCDStart parameter
     PCDRange      := [IN]   [DINT]   See PCDRange parameter
     AccessType    := [IN]   [INT]    See AccessType parameter
     UID           := [IN]   [INT]    See UID parameter
     Options       := [IN]   [DWORD]  See MapOptions parameter
     Status        := [OUT]  [DWORD]  See Status parameter
```

The FB 503 (“InitMap”) will call the SFB 503. The FB call requires a (multi-)instance data block, referenced here has DB503.

Remark:

- **For coils or DI**: PCD element x0.0 has the address 0, PCD element x0.1 has the address 1, ……, element x1.0 has the address 8, element x1.1 has the address 9, etc... where “x” can represent Merkers, Inputs or Outputs (see Figure 2 above).
- **For HR or IR**: in case of Merkers, Inputs or Outputs access, *PCDRange* is given in Bytes and *PCDStart* correspond to the first MW, IW or OW of the area; the access is done wordwise (see Figure 1 above). In case of timer or counter, *PCDstart* is the number of the first counter or timer in this area and *PCDRange* is the number of timers or counters in the area; access to “nth” HR / IR of the area corresponds to “nth” timer or counter of the area.

4.10 FB GetUID_Diag (FB 504)

This function allows getting the diagnostic information related to a UID.

```
CALL FB 504 , DB504
     UID           := [IN]   [INT]    See UID parameter
     ClearDiag     := [IN]   [BYTE]   See ClearDiag parameter
     DiagReg       := [OUT]  [DWORD]  See DiagReg parameter
     ReqCounter    := [OUT]  [DWORD]  See ReqCount parameter
     Status        := [OUT]  [DWORD]  See Status parameter
```

The FB 504 (“GetUID_Diag”) will call the SFB 504. The FB call requires a (multi-)instance data block, referenced here has DB504.

4.11 FB GetServer_Diag (FB 505)

This function allows getting the diagnostic information related to a server instance.

```
CALL FB 505 , DB505
     Port          := [IN]   [INT]    See Port parameter
     Protocol      := [IN]   [INT]    See Protocol parameter
     ClearDiag     := [IN]   [BYTE]   See ClearDiag parameter
     DiagReg       := [OUT]  [DWORD]  See DiagReg parameter
     Status        := [OUT]  [DWORD]  See Status parameter
```

The FB 505 (“GetServer_Diag”) will call the SFB 505. The FB call requires a (multi-)instance data block, referenced here has DB505.