SAPHIR

SIGNAL PHYSIQUE & INSTRUMENTATION

GModBus Over Serial Line

USER GUIDE

GModBus +

Over Serial Line

COMPATIBLE WITH NATIONAL INSTRUMENTS LabVIEW SILVER PRODUCT

www.saphir.fr

Table of contents

1. Foreword

GModBus over Serial Line protocol is a communication protocol based on a master – slave architecture. The network will connect one master to one or several slaves with a RS 485 link.

Through this protocol, only the master can prompt the exchange with the slave by sending a request and waiting for an answer. Therefore only one device can emit through the serial link. No slave can send a message without being asked first. The validity of the communication is controlled by checksum or timeout functions.

GModBus over Serial Line driver encapsulates all these layers in order to make it easy for the developer to insert a computer, as master or slave, within such a bus.

The following functionalities, hidden to the user, are managed:

- Link and Network low layers of ModBus procedure
- Encoding/decoding ASCII or RTU frames
- Frames control through checksum and timeout
- Serial link(s) communication management

2. Configuration

2.1 Software configuration

GModBus over Serial Line driver runs under the following LabVIEW:

- 2010 * and later

And on the following platforms:

- PC under Windows XP and later
- RT system (Real Time)

2.2 Hardware: Network connection

There are 2 ways to connect a computer to a ModBus network:

Point to point link:

Master Slave

The « Point to point » set up enables a platform (set as the master) to connect to a single equipment through the same ModBus network. A classic serial link is used in most cases but a RS 485 link can suit too. The equipment must have a serial communication card. Device wiring is defined in the manufacturer's specific documentation.

On the computer side, refer to the following outlines:

PC RS 232	1	Not used
	2	Rx
	3	Tx
	4	Not used
	5	GND
	6	Not used
	7	Not used
	8	Not used
	9	Not used

Figure 1 : Serial connector SUB D9 for PC

* Contact us for further information concerning former LabVIEW versions.

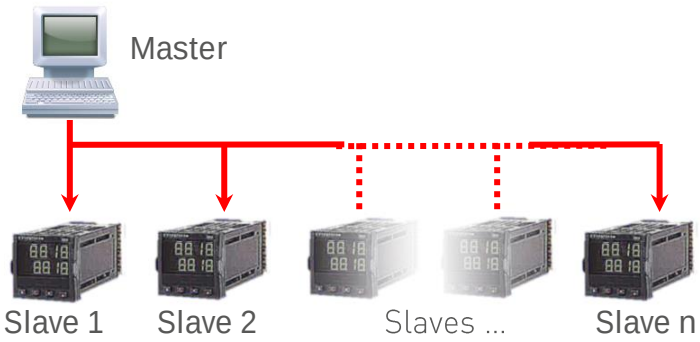
Macintosh RS 232	1	DTR
	2	DSR
	3	Tx
	4	GND
	5	Rx
	6	Not used
	7	Not used
	8	Not used

Macintosh RS 422	1	Not used
	2	Not used
	3	Tx-
	4	GND
	5	Rx-
	6	Tx+
	7	Not used
	8	Rx+

Figure 2: Mini serial connector DIN 8 for Macintosh*

Note: To Connect 2 computers (one master to a slave) through a serial RS 232, use a crossed serial cable.

Multipoint link:



A multipoint link is the connexion of several equipments on the same physical link. The RS 485 standard has to be used. To that end a RS 232/422 ←→ RS 485 adapter or a specific card will be required. In both cases, refer to the manufacturer's documentation to determine the wiring.

The usual connexion will be:

- The master Rx line must be connected to the slave Tx line.
- The master Tx line must be connected to the slave Rx line.

* Apple© does not include mini DIN 8 connector on G3,4 an 5 and iMac.Use USB/Serial (subD9) to connect a ModBus network.

3. GModBus over Serial Line

The driver has been designed to follow different stages:

- Open : initialization of the communication
- Read / Write : read and write tasks
- Close : stops the communication

3.1 GModBus over Serial Line within LabVIEW

GModBus over Serial Line driver installation adds the GModBus over Serial Line palette to LabVIEW functions palette.

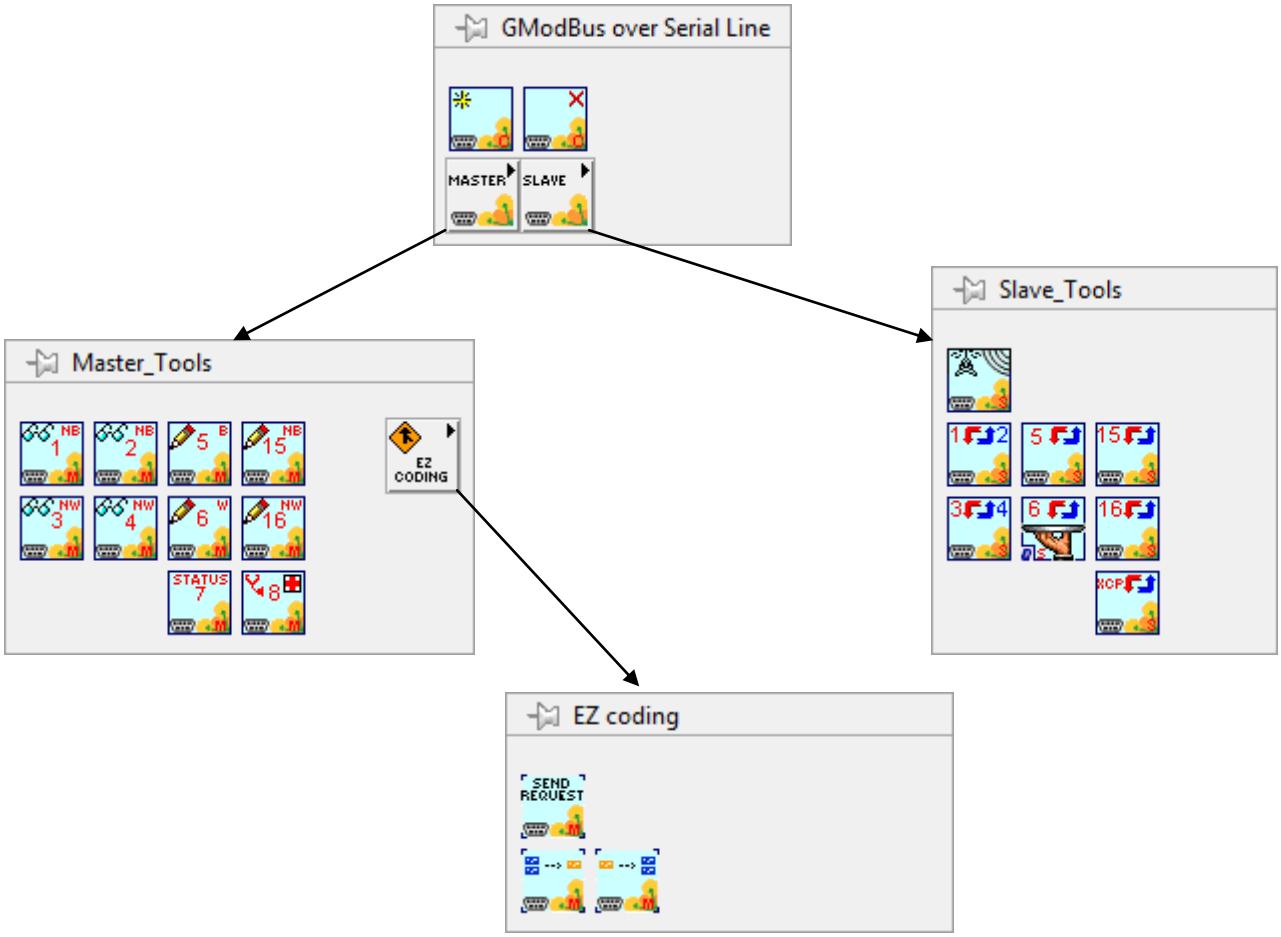


Figure 3: GModBus over Serial Line within LabVIEW palette

4. GModBus over Serial Line components

4.1 Opening/Closing ModBus communication

This chapter deals with the opening and closing of a ModBus connection. The VIs to use are found in the LabVIEW Functions palette by selecting Data Communication > GModBus over Serial Line.

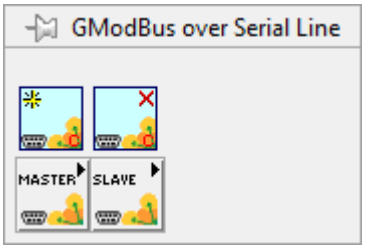


Figure 4: Communication opening and communication closing VIs palette

In *Master* mode like in *Slave* mode, it is imperative for the ModBus communication to be opened before any request transmission. At the end of the communication, the closing step results in releasing the resources properly.

4.1.1 Open

Figure 5: MBV_open.vi

This VI initializes the serial communication of the ModBus network described by ***Network in***. It sends back a ***NetRef out***, single reference to Network in, which is required by the other VIs of ModBus driver managing the same communication.

Network in:

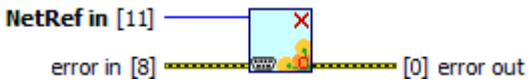
- **Com Port:** serial port used to communicate.
- **BaudRate:** speed of data flow.
- **Parity:** kind of parity calculus which will help to check the validity of the communication (none, odd, even).
- **StopBit:** number of bits associated with the stop of the serial link (1 bit, 1,5 bit or 2 bits).
- **Master:** mode of the opened connection (true = master, false = slave)
- **RTU:** transmission mode of ModBus frames (true = RTU, false = ASCII).
- **TimeOut (ms):** time to deal with the writing and the reading of the requests.
- **LocalSlaveAddress:** associate an address to the slave. This field is useful only during a slave development (cf. §**Erreur ! Source du renvoi introuvable.**).

- *Character time:* inter-character waiting time in RTU mode.
- *Flow control parameters:*
  - *Flow control (0:none):* control type (none, XON/XOFF, RTS/CTS, DTR/DSR).
  - *XOFF byte:* value of the XOFF character
  - *XON byte:* value of the XON character
  - *parity error byte:* parity used with every frame that is transmitted or received.
  - *Valid values include:* (0) Parity None, (1) Parity Odd,(2) Parity Even, (3) Parity Mark, (4) Parity Space.

Mark means that the parity exists and is always 1.

Space means that the parity exists and is always 0.

4.1.2 Close



Closes the Network serial communication associated with *NetRef in*.

It is imperative for the release process to be done properly to free the memory resources of the computer.

4.2 Master tools

This chapter describes the VIs used to realise the master of a ModBus network. These VIs are found in the Functions palette by selecting Data Communication > GModBus over Serial Line > Master_Tools.

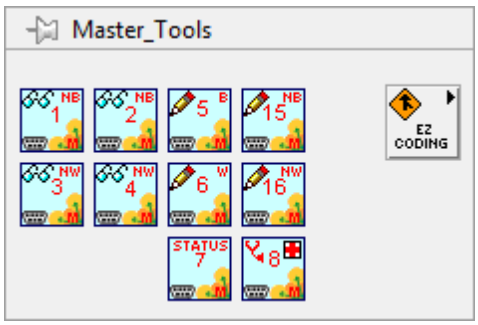


Figure 6: Master_Tools palette

4.2.1 Foreword and writing conventions

The set of VIs that composes the *Master* part of GModBus driver follows the connector model below:

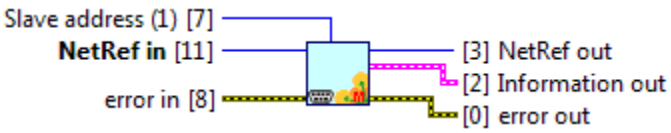


Figure 7: Master VIs connector model

- ***NetRef in / NetRef out: NetRef in*** is the reference to the ModBus network obtained at the opening of the communication (see § **Erreur ! Source du renvoi introuvable.**). ***NetRef out*** is a copy of ***NetRef in***.
- ***error in / error out: error in*** describes the errors that occurred before the VI. The default value corresponds to "no error". If an error occurred before you call the VI, this error **goes** to ***error out*** without the VI executing its function. If an error occurs when the function is running, this error **goes** automatically to ***error out***.
- ***Slave address (1)*** address of the equipment pointed by the ModBus request. Its default value is 1.
- ***Information out*** is a cluster that contains following data :
- ***transmitterAddress:*** same as ***Slave address (1)***.
- ***exceptionCode:*** Code referring to ModBus protocol exceptions (cf. § **Erreur ! ource du renvoi introuvable.**). 0 by default, no exception occurred.
- ***functionCode:*** Number of the request used.
- ***sendFrame:*** Characters sent to slave equipment (This data is given for information, GModBus over Serial Line driver deals with the sending of the frame by itself).
- ***receivedFrame:*** string received by the master (sent by the slave as an answer on the request).
- The connectors respect LabVIEW conventions as follows:
- The label *connectorname (x)* means *x* is the default value associated with this connector if no other value is given to it in Input.
- The connectors which names appear in bold must be wired. Otherwise the caller VI will not be able to run (broken arrow).

4.2.2 Request 1: Reading N output bits

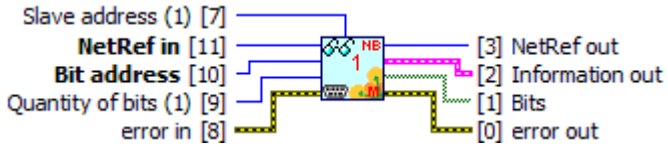


Figure 8: MBV_lecNBitsSortie(1).vi

This VI is used to read consecutive output ***Bits*** defined in the memory of the destination equipment

- ***Bit address:*** address of the first bit
- ***Quantity of bits (1):*** number of bits to read
- ***Bits:*** value of the read bits

4.2.3 Request 2: Reading N input bits

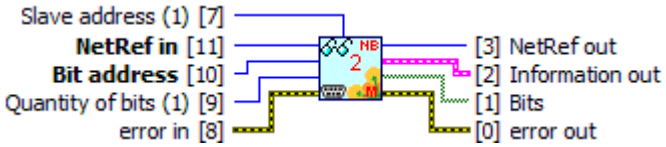


Figure 9: MBV_IecNBitsEntree(2).vi

This VI is used to read consecutive input ***Bits*** defined in the memory of the destination equipment

- ***Bit address:*** address of the first bit
- ***Quantity of bits (1):*** number of bits to read
- ***Bits:*** value of the read bits

4.2.4 Request 3: Reading N output words

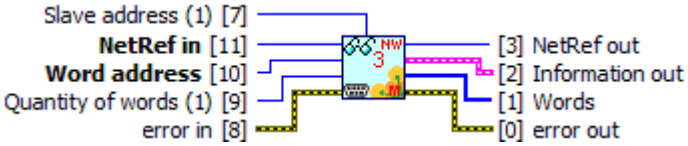


Figure 10: MBV_IecNMotsSortie(3).vi

This VI is used to read consecutive output ***Words*** defined in the memory of the destination equipment

- ***Word address:*** address of the first word
- ***Quantity of Words (1):*** number of words to read
- ***Words*** value of the read words

4.2.5 Request 4: Reading N input words

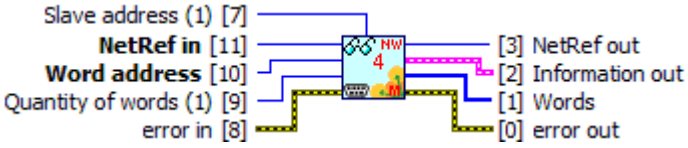


Figure 11: MBV_IecNMotsEntree(4).vi

This VI is used to read consecutive input ***Words*** defined in the memory of the destination equipment

- ***Word address:*** address of the first word
- ***Quantity of Words (1):*** number of words to read

- ***Words*** value of the read words

4.2.6 Request 5: writing an output bit

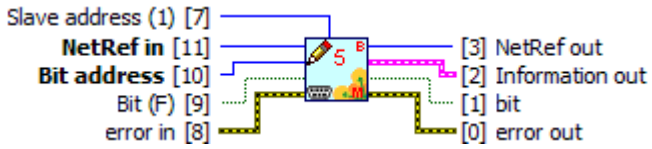


Figure 12: MBV_ecrBitSortie(5).vi

This VI is used to write (at 0 or at 1) an output Bit in the memory of the destination equipment

- ***Bit address***: address of the bit to write
- ***Bit (F):*** value of the bit to write

4.2.7 Request 6: writing an output word

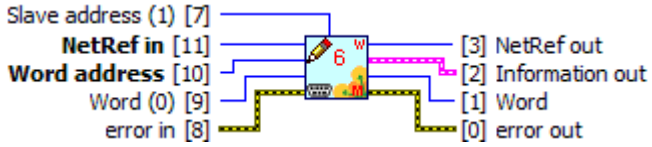


Figure 13: MBV_ecrMotSortie(6).vi

This VI is used to write an output Word in the memory of the destination equipment

- ***Word address:*** address of the word to write
- ***Word (0)***: value of the word to write

4.2.8 Request 15: Writing N output bits

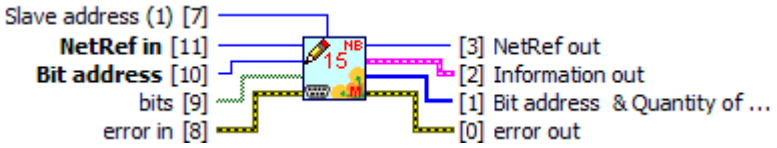


Figure 14: MBV_ecrNBitsSortie(15).vi

This VI permits to write (at 0 or at 1) a group of consecutive output bits in the memory of the destination equipment.

- ***Bit address:*** address of the first bit to write.
- ***Bits:*** array of bits to write

The driver sends groups of 8 bits. If the number of bits written is not a multiple of 8, the driver fills the missing bits at FALSE. Depending on the equipment of destination, these bits can be interpreted or not.

4.2.9 Request 16: Writing N output words

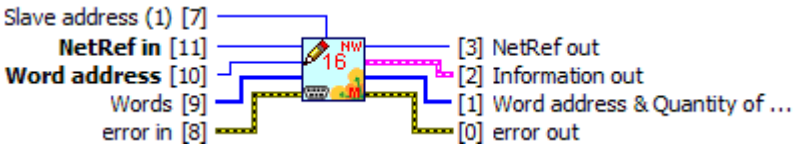


Figure 15: MBV_ecrNmotsSortie(16).vi

This VI is used to write a group of consecutive output words (16 bits) in the memory of the destination equipment.

- ***Word address:*** address of the first word to write.
- ***Words:*** array of words to write.

4.2.10 Request 7: Status reading

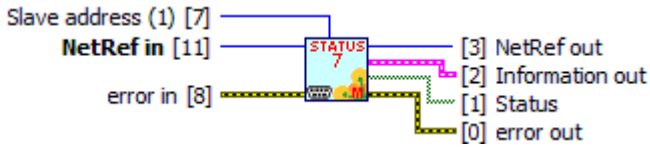


Figure 16: MBV_lecStatusException(7).vi

This VI permits to reach the 8 bits of the equipment status

- ***Status:*** array of bits representing the slave state.

The meaning of the status bits is specific to the equipment used. For more information refer to the manufacturer's documentation.

4.2.11 Request 8: Diagnosis

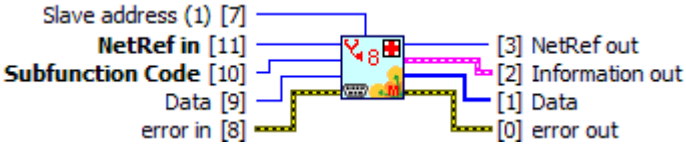


Figure 17: MBV_diagnostic(8).vi

This VI is used to run Master/Slave communication tests or to check the slave is functional.

- ***Subfunction Code:*** type of test to run.
- ***Data:*** data associated with the test if necessary.
- ***Data:*** test result if necessary.

Diagnosis functions are specific to the equipment used. For more information refer to the manufacturer's documentation.

4.2.12 EZ Coding Vis

EZ Coding VIs are to be dropped on an existing VI. They propose a starting architecture to the implementation of a ModBus master.

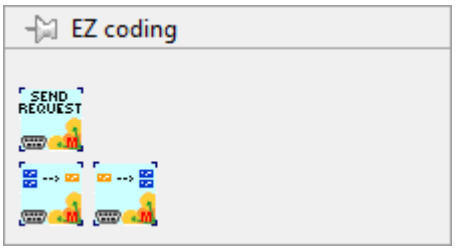


Figure 18: EZ Coding functions palette

Drop this VI into the block diagram to place his content and customized it.

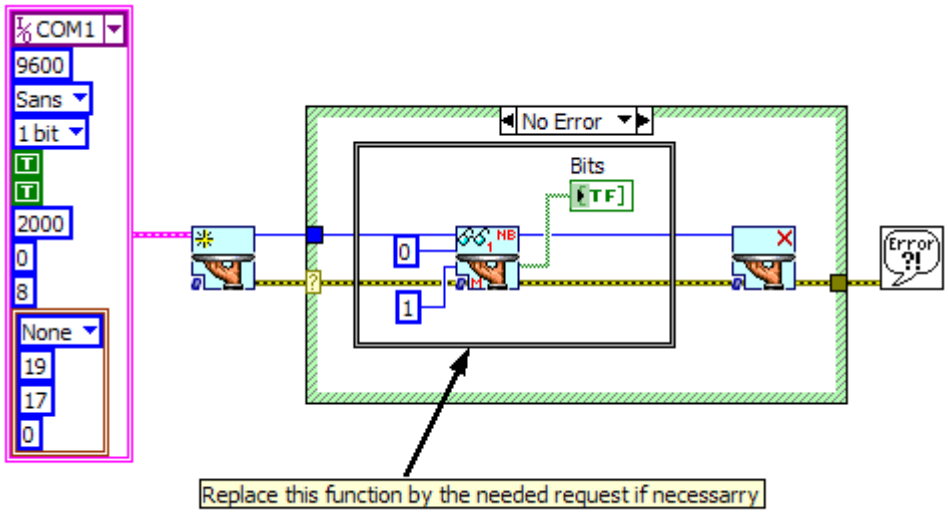


Figure 19: How to implements a simple request to a ModBus Slave

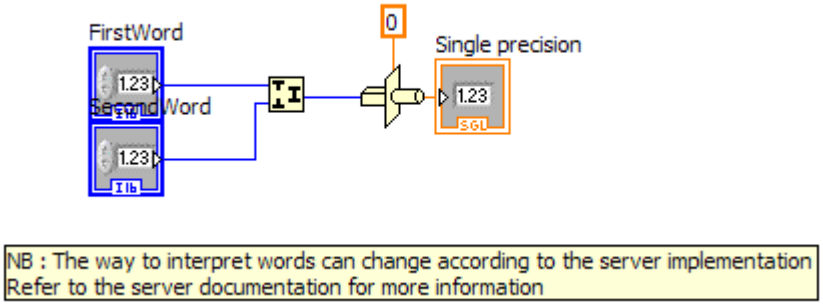


Figure 20: How to interpret 2 words to obtain a single.

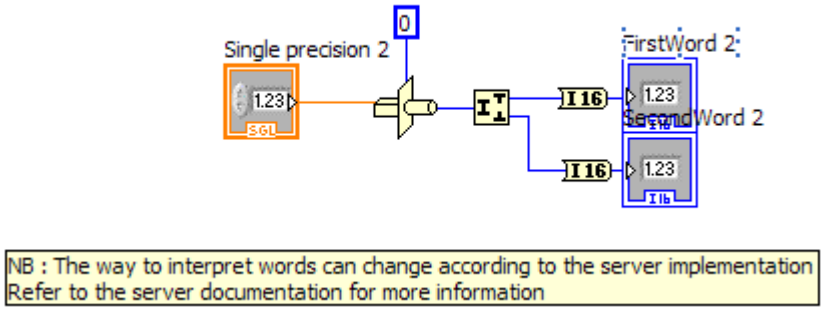


Figure 21: How to interpret a single to obtain 2 words

4.3 Slave Tools

This chapter describes the Vis to use to realise a slave for a ModBus network. These Vis are found in the Functions palette by selecting Data Communication > GModBus over Serial Line > Slave_Tools.

Figure 22: Slaves Vis palette

4.3.1 "Listening" a ModBus Network

In *Slave* mode, the computer never initiates the communication. It scans the network to get back the requests that are sent to it. These tasks are done with the VI MBV_listenRequest.vi.

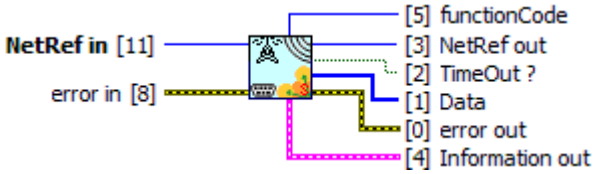


Figure 23: MBV_listenRequest.vi

This VI scans ModBus network and returns information about the requests sent by the master.

- *TimeOut ?:* returns TRUE if the computer did not receive any request, returns FALSE otherwise.
- *FunctionCode:* number of the received request.
- *Data:* data contained in the request.
- *Information out:*
- *receivedFrame:* Characters received by the slave (and sent by the master).
- *exceptionCode:* Code refering to ModBus protocol exceptions (cf. § **Erreur ! ource du renvoi introuvable.**). By default 0, no exception occured.
- *CRC / LRC error:* returns TRUE if an error occured during the frame reception.

4.3.2 Request management

The set of VIs that composes the *Slave* part of GModBus over Serial Line driver follows the connector model below:

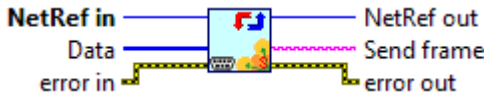


Figure 24: Slave VIs connector model

- *NetRef in / NetRef out : NetRef in* is the reference to the ModBus network that you get when you open the communication (cf. § **Erreur ! Source du envoi introuvable.**). *NetRef out* is a copy of *NetRef in*.
- *error in / error out : error in* describes the errors that occurred before the VI. The default value corresponds to "no error". If an error occurred before this VI is called, this error goes to *error out* without the VI executing its function. If an error occurs while the function is running, it automatically goes to *error out*.
- *Data* describes data received by the slave (cf. § **Erreur ! Source du renvoi ntrouvable.**).
- *Send frame* describes the frame sent by the slave to the master (This data is given for information, *GModBus over Serial Line* driver deals with the sending of the frame by itself).

4.3.2.a Answer request 1 and 2 (reading N input or output bits)

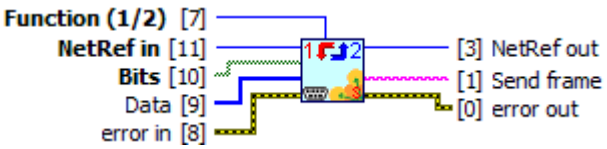


Figure 25: MBV_repLectureNBits(1_2).vi

This VI permits to answer requests 1 or 2 sent by the master through ModBus network.

- *Function (1/2):* request to deal with.
- *Bits:* array containing the values of the slave registers group.

4.3.2.b Answer request 3 and 4 (reading N input or output words)

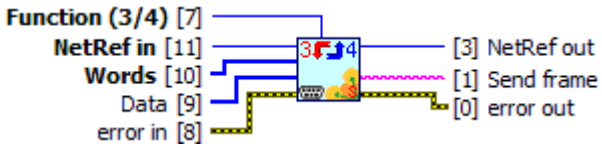


Figure 26: MBV_repLectureNMots(3_4).vi

This VI permits to answer requests 3 or 4 sent by the master through ModBus network.

- *Function (3/4):* request to deal with.
- *Words:* array containing the values of the slave registers group.

4.3.2.c Answer request 5 (writing an output bit)

Figure 27: MBV_repEcritureBit(5).vi

This VI permits to answer request 5 sent by the master through ModBus network.

- *Bits in*: array defining the values of the slave registers.
- *Bits out*: array containing the values of slave registers after the request has been done.

4.3.2.d Answer request 6 (writing an output word)

Figure 28: MBV_repEcritureMot(6).vi

This VI permits to answer request 6 sent by the master through ModBus network.

- *Words in:* array defining the values of the slave registers group.
- *Words out:* values of slave register when the request has been done.

4.3.2.e Answer request 15 (writing N output bits)

Figure 29: MBV_repEcritureNBits(15).vi

This VI permits to answer request 15 sent by the master through ModBus network.

- *Bits in:* array defining the values of the slave registers.
- *Bits out:* values of slave register when the request has been done.

4.3.2.f Answer request 16 (writing N output words)

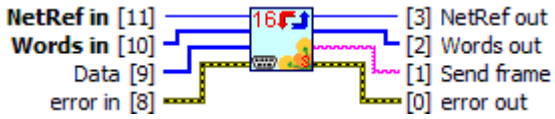


Figure 30: MBV_repEcritureNMots(16).vi

This VI permits to answer request 16 sent by the master through ModBus network.

- ***Words in:*** array defining the values of the slave registers group.
- ***Words out:*** values of slave register when the request has been done.

4.3.2.g Return an exception code

Figure 31: MBV_repException.vi

This VI returns an exception code to the master of ModBus network.

- ***Function:*** request that generated the exception code.
- ***Exception Code:*** exception code to send (see§ **Erreur ! Source du renvoi ntrouvable.**).

5. Tools

This chapter describes tools to quickly simulate a ModBus master or slave. You will find them in the LabVIEW menu bar.

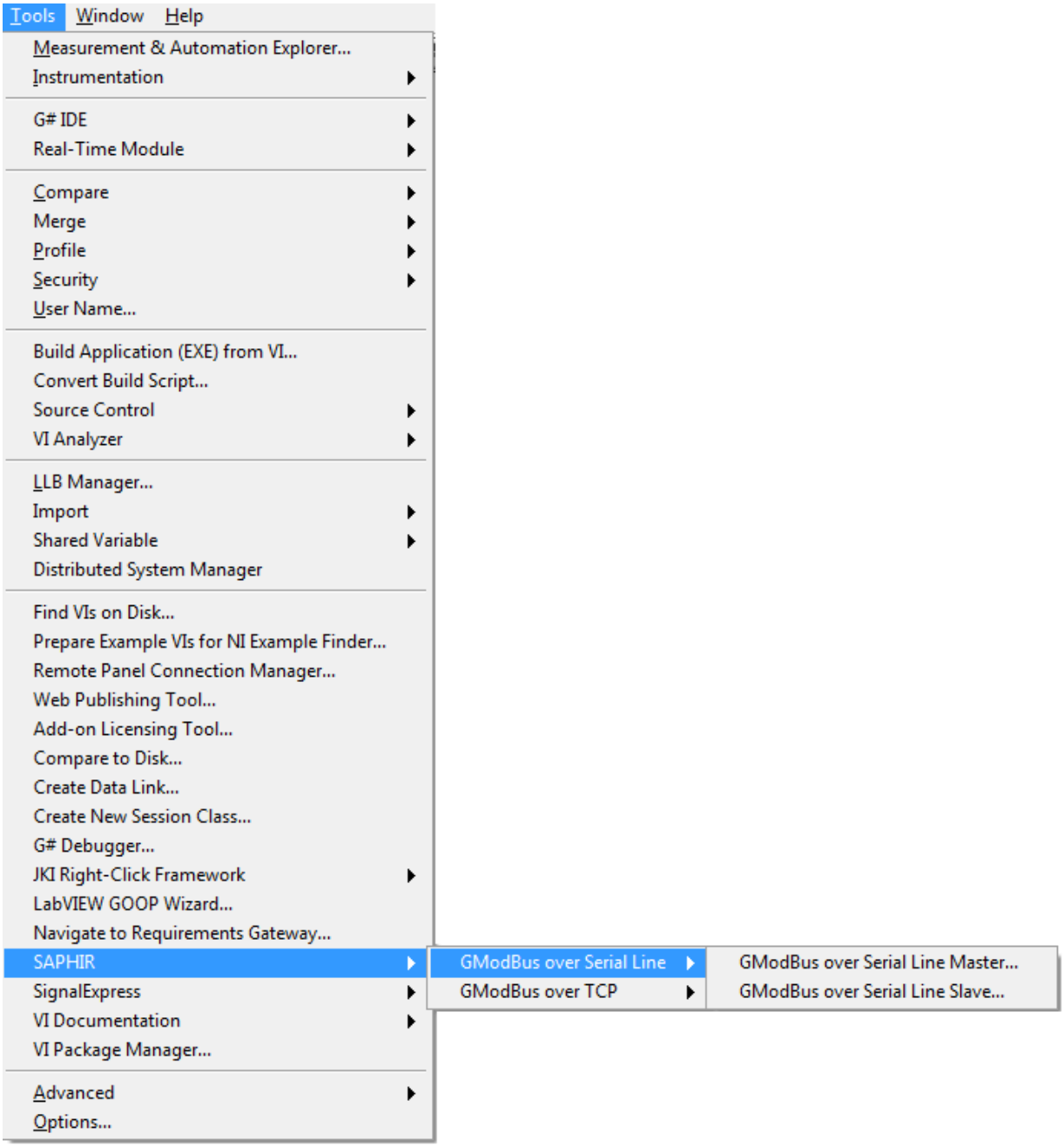


Figure 32: Tools menu

5.1.1 Master

The master application will quickly test the communication with a slave through ModBus network.

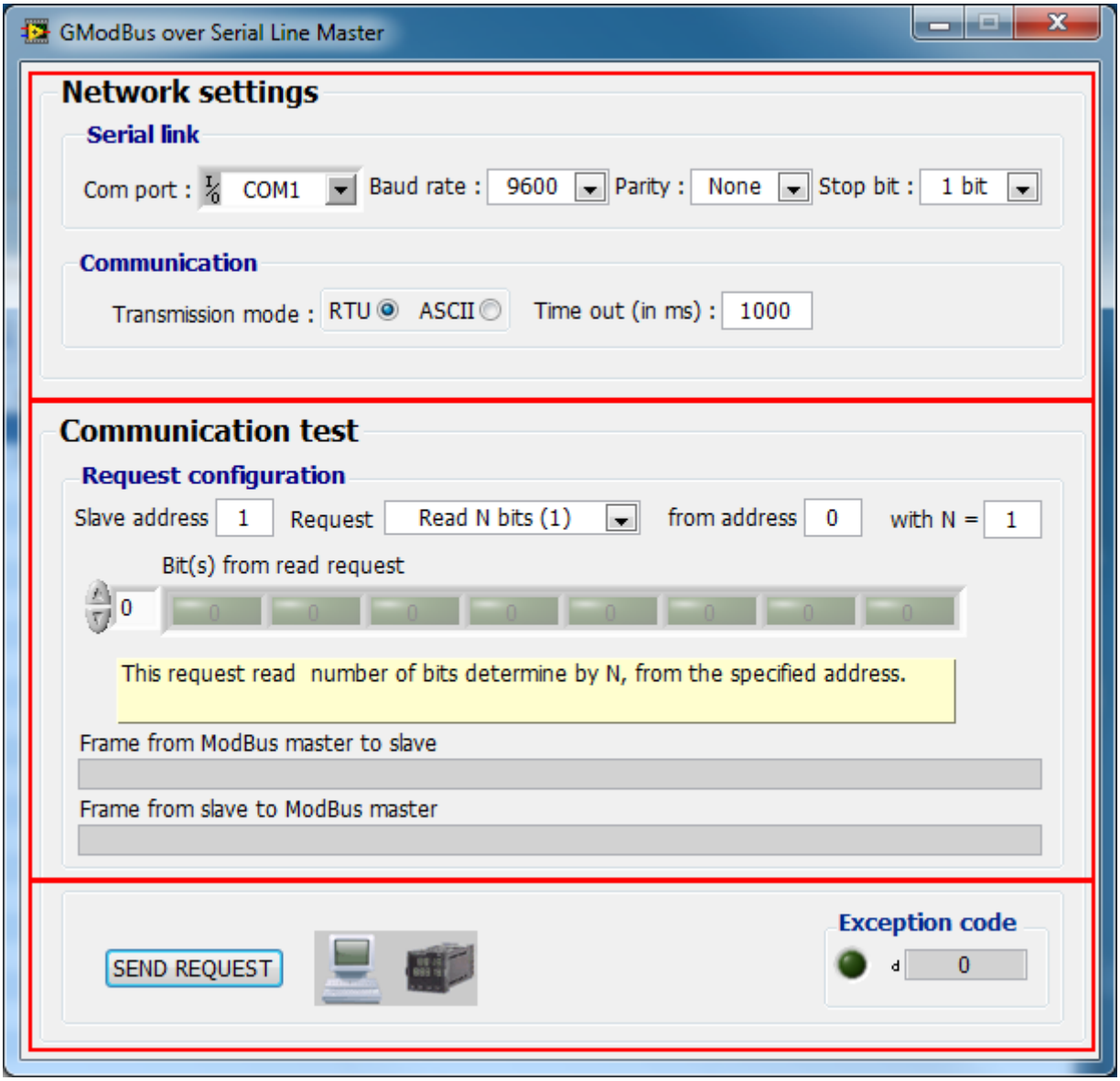


Figure 33: Master window

The interface falls into three sections:

Network settings

- The serial link section defines:
  - *Com Port:* number or name of the serial port used to communicate. the list of serial ports available on the computer is automatically updated
  - *BaudRate:* speed of data flow (must be coherent with the slave(s)).

- *Parity:* kind of parity calculus which will help to check the validity of the communication (none, odd, even) (must be coherent with the slave(s)).
- *StopBit:* number of stop bits (1 bit, 1,5 bit or 2 bits) (must be coherent with the slave(s)).

The communication parameters permit to choose:

- The *transmission mode*: RTU or ASCII
- The *time out (in ms):* time granted to the slave to answer the request of the master. If this period of time is over, it generates a Time Out error.

Communication test

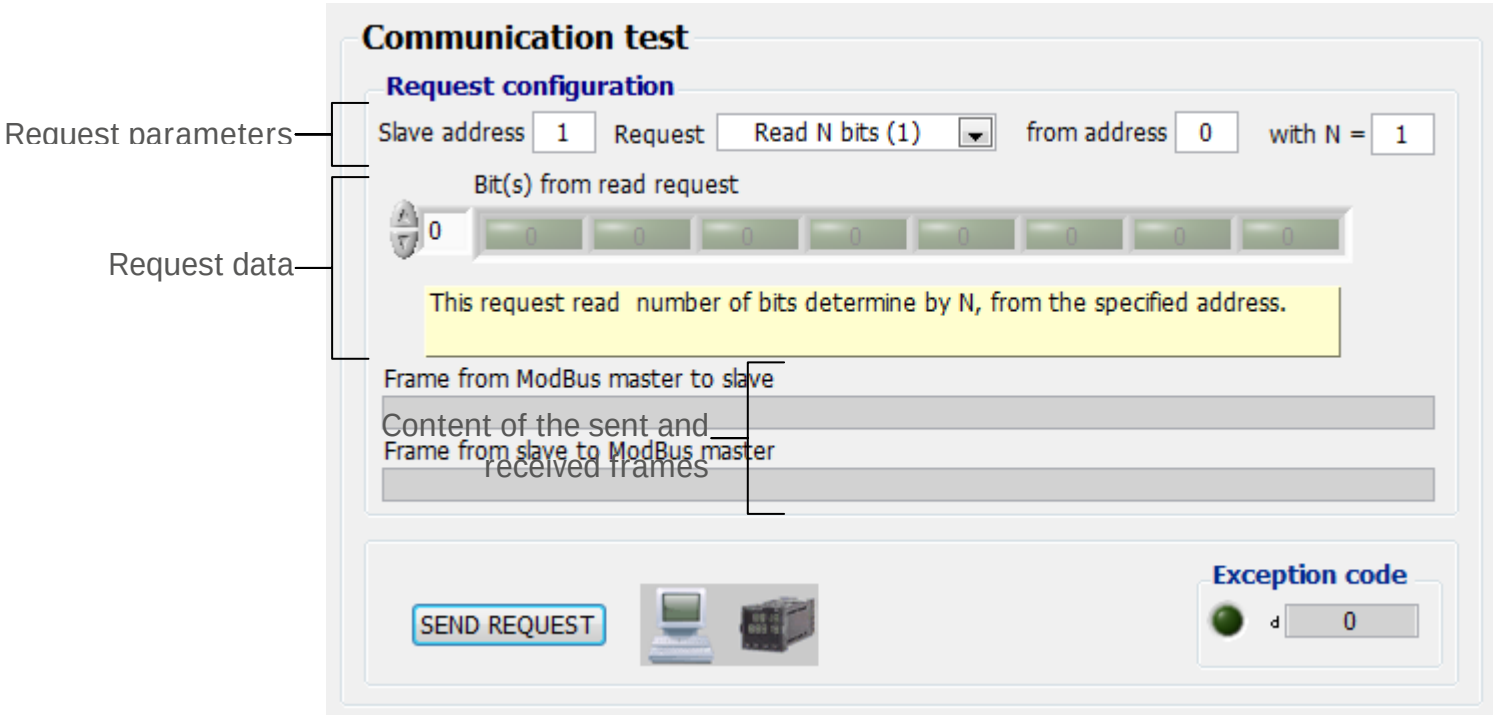


Figure 34: Communication test interface in Master mode

All the types of *Request* of GModBus driver are managed:

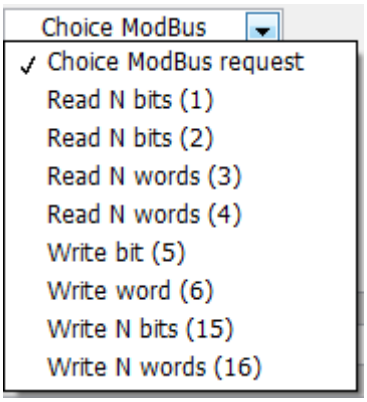


Figure 35: Requests choice

The first register to read or write is defined by *From address.*

The field *With N =* is only available for requests 1, 2, 3 and 4. It represents the number of bit(s) or word(s) to read or to write.

The data zone, located below the request parameters, permits to determine the values to write during the use of writing requests.

When all the settings are done, click on *SEND REQUEST* button to start the communication with the slave. The content of the frames sent and received by the *Master* is displayed below the data zone.

The *Exception code* refers to ModBus protocol exceptions (see §**Erreur ! Source du renvoi introuvable.**).

5.1.2 Slave

The Slave application simulates a slave of witch registers are represented with an array of 100 bits and an array of 100 words.

A ModBus master can read or write these tables.

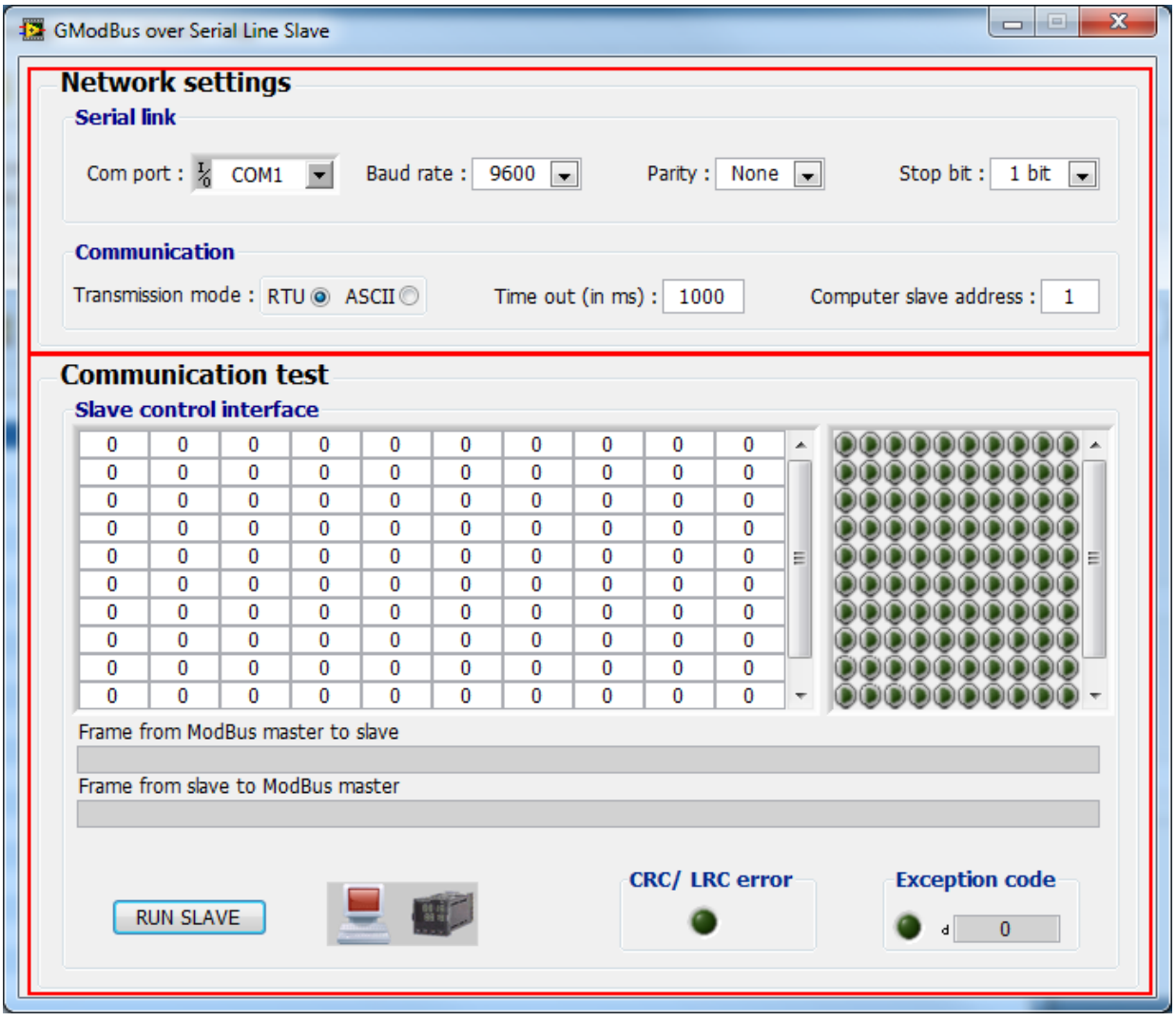


Figure 36: Slave window

Two sections compose this interface:

Network settings:

The *serial link* and *communication* settings are to be coherent with this parameters in *Master* mode (see § **Erreur ! Source du renvoi introuvable.**).

The *Computer slave address:* Determines the slave address allocated to the computer on ModBus network.

Communication test:

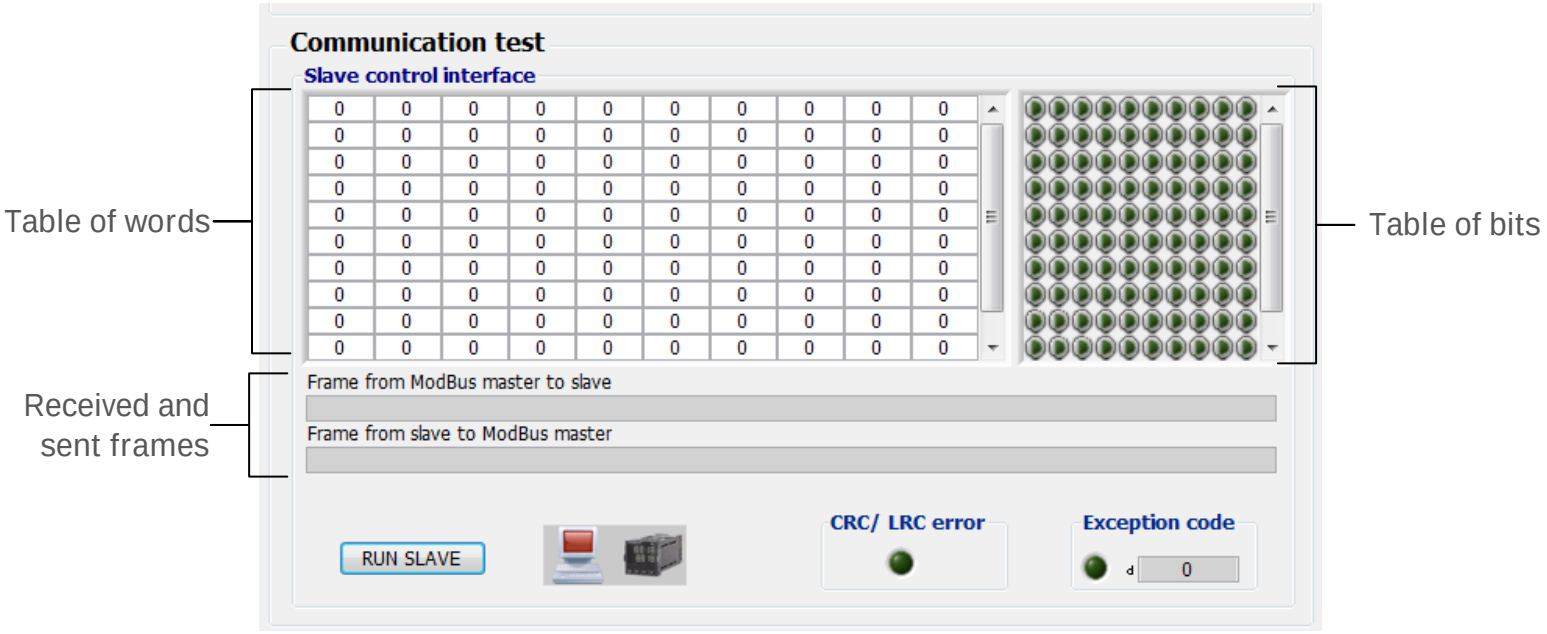


Figure 37: Communication test interface of the slave

When clicking on the *RUN SLAVE* button the slave waits for the master to send a request (listening state). This change of state is notified by the computer icon  which becomes green  .

The settings of ModBus network can't be modified when the slave is listening.

The received and sent frames are displayed in hexadecimal code below the register tables.

The slave doesn't stop instantaneously; the *time out* determines the maximum time it could take.

The *Exception code* refers to ModBus protocol exceptions (cf. §**Erreur ! Source du envoi introuvable.**).

6. GModBus over Serial Line

After the download and installation of GModBus over Serial Line toolkit, an activation window will pop up at LabVIEW launching. You can also go to help > Activation Add-ons, you'll get the following window. Follow the steps of the add-ons activation.

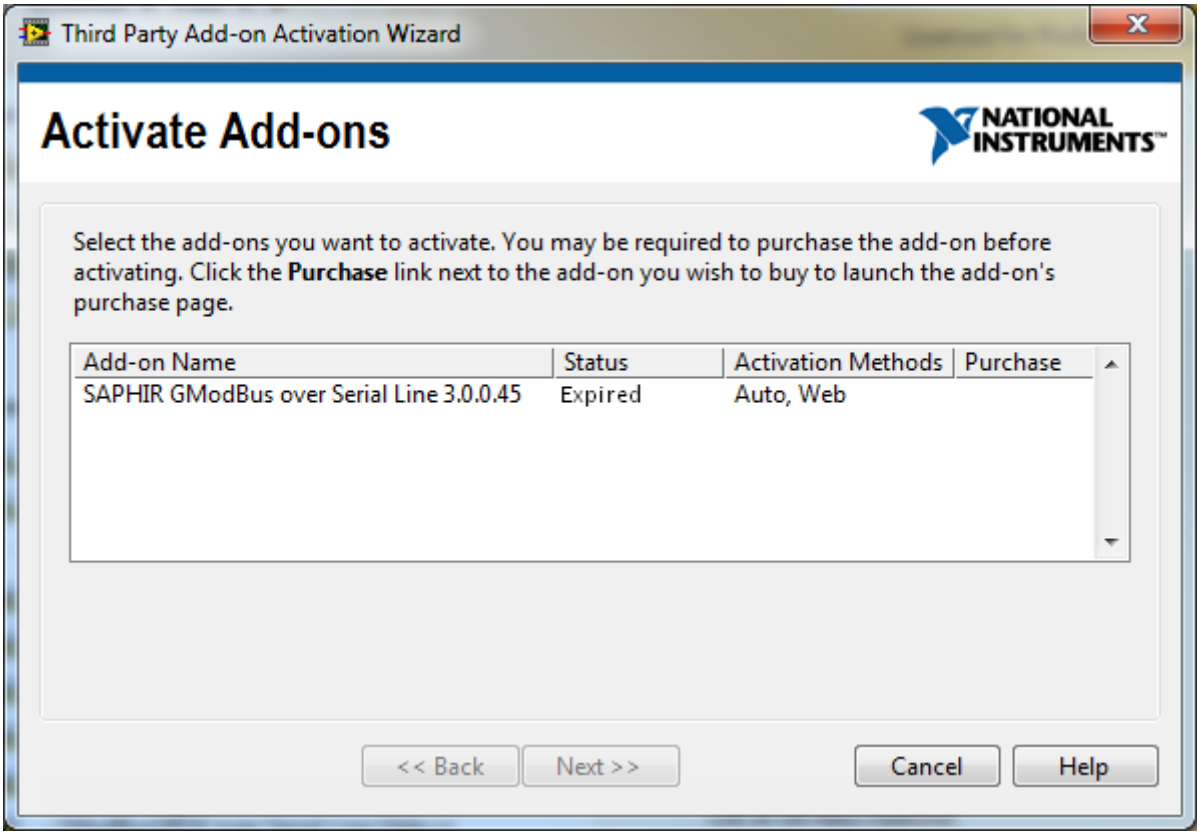


Figure 38: Add-ons activation

You can try GModBus over Serial Line during 30 days. After this period toolkit's VIs will become broken. To activate the toolkit after this period, simply go to Help menu and select Activate Add-ons...

7. GModBus support

The "Online Support & **Resources**" menu opens the following SAPHIR community page:
http://decibel.ni.com/content/groups/saphir-toolkit

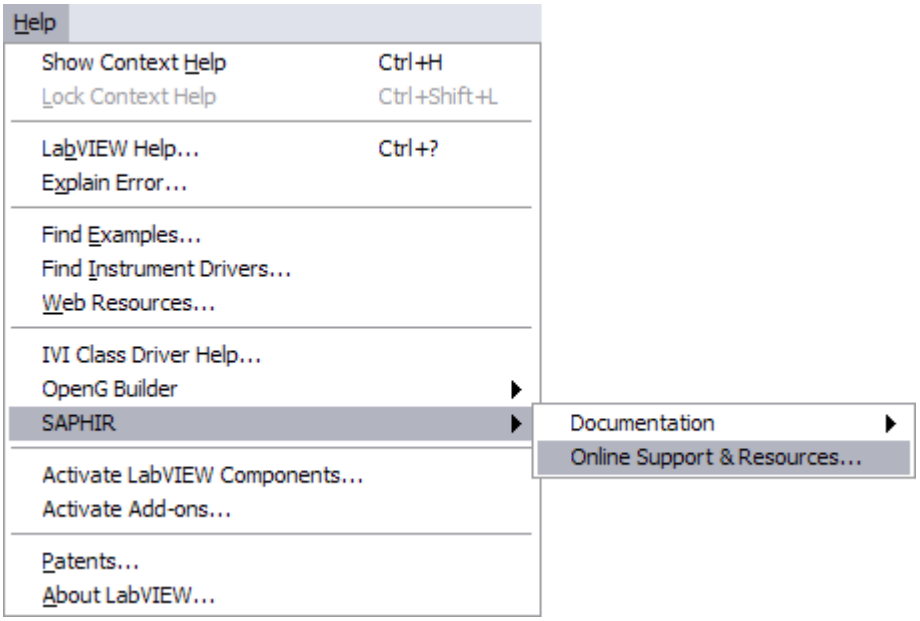


Figure 39: Online support & resources

8. Specific errors to GModBus

8.1 Specific errors to GmodBus driver

Following errors can be generate by GModBus over Serial Line functions

Error	Explanation
6200	wrong CRC/LRC
6201	Timeout
6202	Unable to create reference
6203	Unable to read reference, reference may not exist
6204	Unable to delete reference, reference may not exist

Table 1 : Errors specific to GModBus driver

8.2 Exception codes of ModBus protocol

Following exception codes are specific to ModBus protocol.

Decimal Codes	Explanation
1	Not implemented function
2	Out of limits address
3	Out of limits data
4	Defective equipment
5	Acquit/release.
6	Busy equipment
7	Impossible to release
8	Memory error

Table 2 : ModBus exception codes

9. Problems resolution

The list below enumerates the most frequent problems encountered when implementing *GModBus* driver:

Wiring :

The most frequent wiring error is the reversal of Rx and Tx wires.

Network setting :

Some parameters of ModBus network (communication speed, stop bit, parity and transmission mode) must be common to the master and the slave.

Serial link busy :

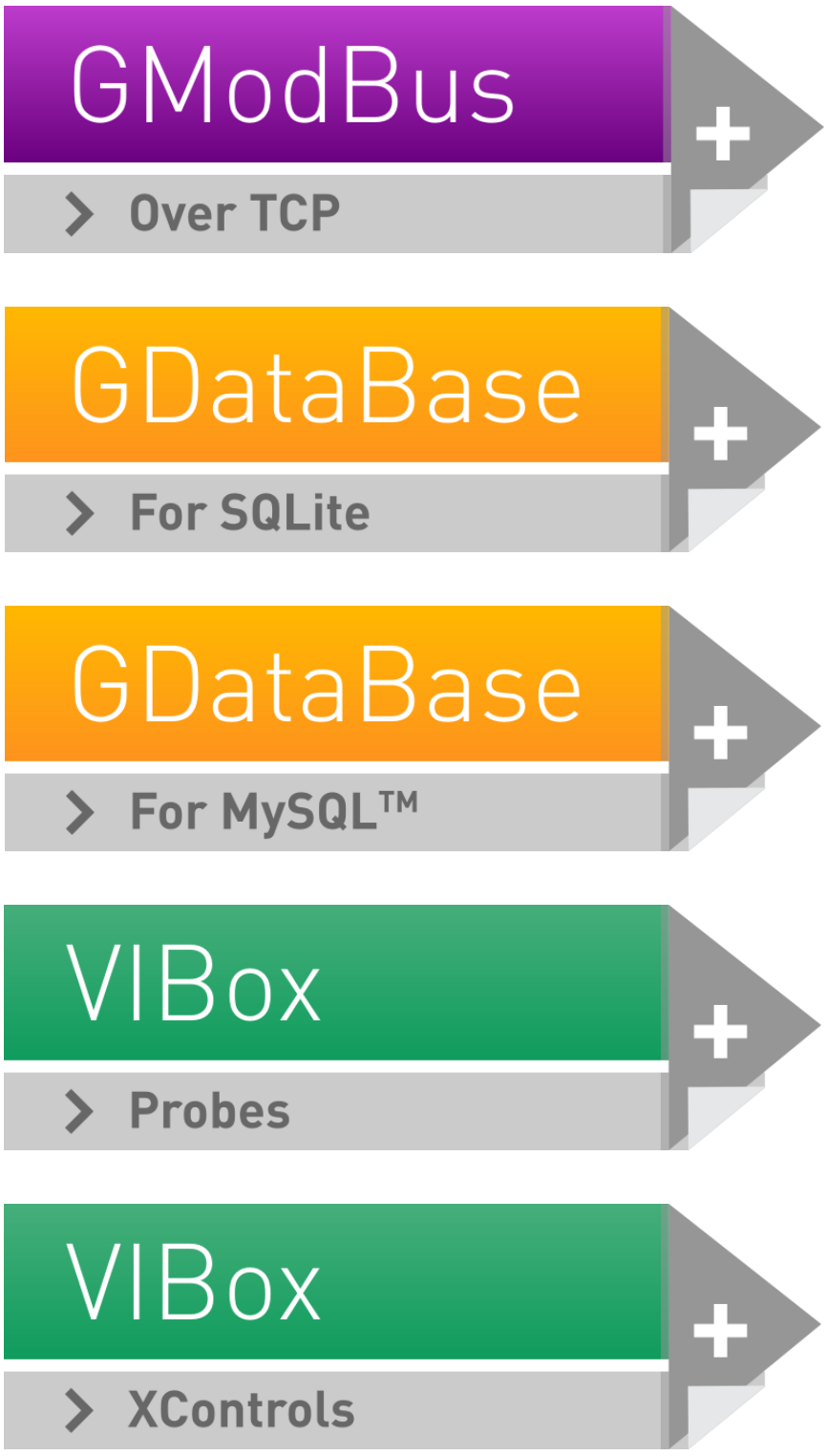
When a program including *GModBus* over Serial Line driver runs, make sure that no other application is communicating on the port used by ModBus network.

Bad requests sequencement :

As the link used is a serial one, you must sequence properly the requests used in the application code. You will not be able to make several requests in parallel.

INDEX

Other add-ons that could be helpful



contact@saphir.fr
+33 (0)4 38 92 15 50
www.saphir.fr