



## COLLABORATIVE ELECTRIC GRIPPERS

### **MPCF SERIES**



## SERIAL COMMUNICATION PROTOCOL

### Version 2.2

Last update date: 19/09/2023



# TABLE OF CONTENTS

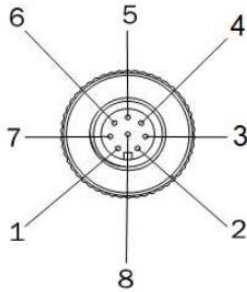
1. INTRODUCTION
2. WIRING INFORMATION
3. COMMUNICATION SETUP AND PROTOCOL DESCRIPTION
4. OPERATING MODES WITH GIMATIC MIDDLEWARE
5. COMMAND REGISTER AND STATUS REGISTER
6. MPCFSerialDemo.exe INTERFACE
7. EXAMPLES
  - 7.1. Example 1: Commanding a movement from base to limit positions
  - 7.2. Example 2: Movement in JOG mode
  - 7.3. Example 3: Movement to Home position in Expert mode
  - 7.4. Example 4: Communication with a generic serial master – Moving to Limit Position with mode 100 and mode 800
  - 7.5. Example 5: Communication with a generic serial master – Interacting directly with CAN objects

# 1.INTRODUCTION

The following documentation aims to illustrate the RS-485 Half Duplex serial communication protocol for the MPCF family's grippers that currently consists of model MPCF3270. It is recommended to first read the "Gimatic MPCF gripper user manual" documentation as a general introduction to the product family and as an overview on commissioning, working principles, configuration, and operating modes. Any MPCF gripper can indeed be tested and configured using the dedicated Windows based application (MPCFApplication.exe) running on a laptop and communicating over an USB to CAN converter of the type PCAN-SUB from Peak-System (<https://www.peak-system.com/>). Using this interface, developed by Gimatic, and a middleware running on board of the drive, users can work offline and easily configure and store into the drive's memory a set of recipes and recall them during on-line operations using the serial protocol without having to deal with the CANopen standard. The middleware is active as standard on MPCF grippers, however for high demanding application in which a real time control of the gripper from an external controller is required, it can be disabled using the MPCFApplication.exe. In this case the gripper becomes a CANopen axis CiA-301, CiA-305 and CiA-402 compliant (more details can be found here: <https://drives.novantamotion.com/emcl/canopen-protocol>). Sections describing wiring and protocol are valid in both the case of CANopen and middleware interaction, however the main purpose of this document is to drive the user through the usage of the middleware by demonstrating how to write a Command register and how to read a Status register. An additional Windows based application is also introduced (MPCFSerialDemo.exe) as a tool to test and to familiarize with the communication protocol. It allows indeed to view the raw data generated by the reading and writing of registers for the case of the most used commands. Additional tests can also be performed by enabling the expert mode granting full access to the Control register. At the end of this manual there are however reported some examples of direct interaction with the CAN objects, therefore bypassing the middleware.

## 2.WIRING INFORMATION

MPCF grippers come with a M12 – 8 pins connector with the following pinout:

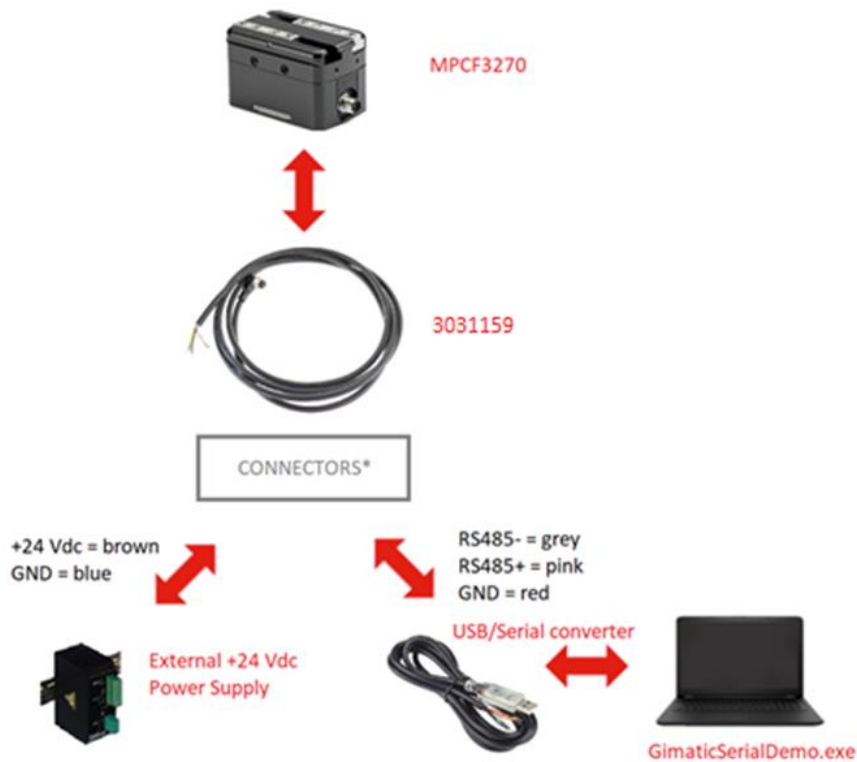


Nr. Pin	Function	Colour
1	Not used	White
2	+24V	Brown
3	CAN-H	Green
4	CAN-L	Yellow
5	RS485-	Grey
6	RS485+	Pink
7	GND (supply)	Blue
8	GND (signal)	Red

*Table 1 - MPCF pinout*

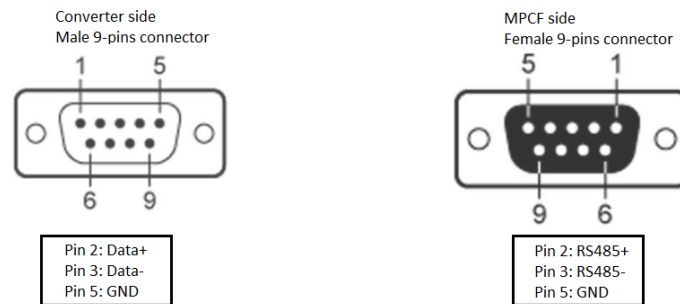
Pin number 5 (grey one) and 6 (pink one) of the M12 gripper connector must be connected to an USB/Serial converter to establish a serial communication with the gripper. Please also connect the power supply pins to an external power source and connect the pin 8 (red one) to the GND cable of the converter to guarantee an appropriate communication, according to the schema below.

A full shielded cable is mandatory to establish the communication between gripper and master. A twisted cable is recommended. As twisted pair is to be consider the serial communication wires: RS485+ and RS485- (and eventually the CANbus communication ones: CAN-H and CAN-L).



*Figure 1 - System wiring with Serial communication*

\* The choice of appropriate connectors is up to the user. Standard D-SUB 9-pins connectors type can be used for example to connect the RS485 lines to the USB/Serial converter. The connection choice is arbitrary, but it is mandatory to respect the one-to-one connection. In the following example pins number 2, 3 and 5 have been used.



*Figure 2 - RS485 arbitrary connection example*

Power supply GND and signal GND (respectively pins 7 and 8 of the M12 gripper connector) must be connected to ensure proper functioning.

### 3.COMMUNICATION SETUP AND PROTOCOL DESCRIPTION

The following table summarizes the default parameters of the RS-485 Half Duplex serial communication channel:

<b>Baudrate</b>	115200 bps
<b>Data bits</b>	8 bits
<b>Parity</b>	None
<b>Stop bits</b>	1 bit
<b>Flux control</b>	None

*Table 2 - RS-485 Serial communication settings*

Two command and data formats are available: ASCII format or Binary format.

The data packet's structure depends on the selected communication format.

#### ASCII format

<b>NODE ID</b>	<b>SPACE</b>	<b>FCT</b>	<b>SPACE</b>	<b>OBJECT</b>	<b>SPACE</b>	<b>VALUE</b>	<b>CR</b>
0-127	0x20(**)	'R' / 'W'	0x20	0 to 0xFFFFFFFF	0x20 (*)(**)	(*)	0x0D (**)

*Table 3 - Data packet's structure in ASCII format*

(\*) Only in write operations

(\*\*) Hexadecimal code for the corresponding ASCII character.

Only node ID, object, and value support hexadecimal format. Other fields do not support hexadecimal format and must be inserted as characters. In addition, the value of the command sent by the controller can be expressed in hexadecimal or decimal format.

#### Binary format

Network Address (1 byte)	Function (1 byte)		MEI type (1 byte)		Protocol Control (1 byte)			Reserved Field (1 byte)	Node ID (1 byte)		Index (2 byte)	
Same as Node ID	43 (dec) 0x2B (hex)		13 (dec) 0x0D (hex)		0-1			0	0-127		HIGH BYTE	LOW BYTE
											0x0000 - 0xFFFF	
Sub-index (1 byte)	Starting Address (2 byte)		Number of data (2 byte)		Data (0 - 8 bytes)				CRC (2 bytes)		SYNC bytes (4 bytes)	
0x00 - 0xFF	HIGH BYTE	LOW BYTE	HIGH BYTE	LOW BYTE	LOW BYTE	...	...	HIGH BYTE	HIGH BYTE	LOW BYTE	0x55555555	
	0x0000		0 – 8		0x0 – 0xFFFFFFFFFFFFFFFF				CRC			

*Table 4 - Data packet's structure in Binary format*

All the fields (except the SYNC bytes) are defined in the MODBUS specification, on following there is a brief explanation:

- **Network Address.** This field is defined on the MODBUS over Serial Line and its value is the same as Node ID.
- **Function.** The MODBUS function selected for all frames is the Modbus Encapsulated Interface (MEI).
- **MEI type.** The encapsulated protocol is the CANopen.
- **Protocol Control.** Works like the **FCT Field** of the ASCII format. If the protocol control field contains a **0 value**, it will indicate that the process is a **read process**. If the protocol control field contains a **1 value**, it will indicate that the process is a **write process**.
- **Reserved Field.** This byte is reserved.
- **Node ID.** Contains the Node ID of the controller. It may be a value from 0-127 (both inclusive).
- **Index.** Index of the object to be read or written.
- **Sub-index.** Subindex of the object to be read or written.
- **Starting Address.** Not used.
- **Number of data.** Contains the number of data available in the data field. In case of reading operation can be left the 0 value.
- **Data.** Contains the object value.
- **CRC.** If it is available contains the calculated CRC.
- **SYNC bytes.** Four bytes with 0x55 value are sent to synchronize the communication.

**By default, the Node ID is equal to the decimal value 32 (0x20 in hex format).**

**Please note that in binary format the Data bytes are the only ones to be written backwards.**

**The Binary format is the one chosen to communicate through MPCF gripper** to achieve best performances although the ASCII format is the default one every time the drive is turned on. Configuration packets must be sent every time the gripper is turned on to set the binary mode.

In case of functioning through MPCFSerialDemo.exe application the configuration packets are sent automatically in background once clicked on "Open serial port" button.

In case of functioning without MPCFSerialDemo.exe application, such as all real application like robot and automatic machinery equipped with serial RS-485 connection, the configuration of the communication is borne by the customer. Please always beware about Node ID, equals to 32 (0x20) by default.

The data packets are the following:

- 0x0D (**wake up byte** to establish a communication)
- '0' 'x' '2' '0' ' ' 'W' ' ' '0' 'x' '8' '2' '0' '0' '0' ' ' '0' 'x' '1' '\r' (**enable the Binary format**, note that they are all characters since the drive is in ASCII format by default). The packet writes value 1 to the object with index 0x2000, sub-index 0x08 (<https://drives.novantamotion.com/emcl/0x2000-uart-configuration>) to pass to the binary mode. Note that 0x20 is the Node ID. From this point the data are always sent as byte (in hex format).
- 0x20 0x2B 0x0D 0x01 0x00 0x20 0x20 0x00 0x06 0x00 0x00 0x00 0x01 0x01 0x55 0x55 0x55 0x55 (**set CRC**). The packet writes the value 1 to the index 0x2000, sub-index 0x06 to enable the use of CRC in the communication. Note the Node ID equals to 0x20.



Below an example of a piece of code that show the construction of these three packets. They must be sent every time a serial communication is opened.

```
// 1) Clean \n
Command[0] = 0xd;
Serial1.Write(Command, 0, 1);
System.Threading.Thread.Sleep(100);
// 2) Set Binary Mode
Command[0] = Convert.ToByte('0');
Command[1] = Convert.ToByte('x');
Command[2] = Convert.ToByte('2');
Command[3] = Convert.ToByte('0');
Command[4] = Convert.ToByte(' ');
Command[5] = Convert.ToByte('W');
Command[6] = Convert.ToByte(' ');
Command[7] = Convert.ToByte('0');
Command[8] = Convert.ToByte('x');
Command[9] = Convert.ToByte('8');
Command[10] = Convert.ToByte('2');
Command[11] = Convert.ToByte('0');
Command[12] = Convert.ToByte('0');
Command[13] = Convert.ToByte('0');
Command[14] = Convert.ToByte(' ');
Command[15] = Convert.ToByte('0');
Command[16] = Convert.ToByte('x');
Command[17] = Convert.ToByte('1');
Command[18] = 0xd;
Serial1.Write(Command, 0, 19);
System.Threading.Thread.Sleep(100);
//Set CRC
Command[0] = 0x20; // Node Id
Command[1] = 0x2b;
Command[2] = 0x0d;
Command[3] = 0x01;
Command[4] = 0x00;
Command[5] = 0x20; //Node Id
Command[6] = 0x20;
Command[7] = 0x00;
Command[8] = 0x06;
Command[9] = 0x0;
Command[10] = 0x0;
Command[11] = 0x0;
Command[12] = 0x01;
Command[13] = 0x01;
Command[14] = 0x55;
Command[15] = 0x55;
Command[16] = 0x55;
Command[17] = 0x55;
Serial1.Write(Command, 0, 18);
System.Threading.Thread.Sleep(100);
```

Once configured the communication, one can read and write the objects always sending data packets in Binary format as reported in Table 4.

Examples and more information at the following link:

<https://drives.novantamotion.com/emcl/uart-based-rs-232-and-rs-485>

## 4.OPERATING MODES WITH GIMATIC MIDDLEWARE

The available operating modes for MPCF grippers can be divided into five categories:

- Positioning with position control (modes number 100, 101, 110 and 111)
- Gripping command with force control (modes number 200, 201, 210, 211, 220 and 221)
- Gripping command with force control and pre-positioning (modes number 300, 310 and 320)
- Movement in JOG (modes number 901 and 902)
- Homing (mode number 900).

Each operating mode can be executed referring to a set of parameters grouped into recipes (a maximum of eight different recipes can be configured by the user and stored into the drive). Every recipe consists of a list of parameters that include positions (Base, Offset, Limit, Teach 1 and Teach 2, Virtual switches), position tolerance (value of the maximum position error to consider the jaws of the gripper “in-position”), set point velocity and grip force setpoint (used only in force control modes). Parameters like Home Position, Acceleration and Deceleration are common to all the eight recipes. JOG movements are performed with a fixed setpoint velocity pre-defined into the drive the user cannot modify.

For details about all the possible operating modes and recipe parameters please refer to “Gimatic MPCF gripper user manual”.

## 5.COMMAND REGISTER AND STATUS REGISTER

In order to interact with the gripper, only two 32-bit registers are used: the **Gripper Control Register (0x2C00-0x03)** and the **Gripper Status Register (0x2C00-0x24)**.

The **Gripper Control Register (0x2C00-0x03)** is used to command the gripper (i.e. to select an operating mode or a recipe, to enable or disable the gripper, to start or stop a movement etc). Access to this register by the external master is write-only. The following table shows the structure of this register.

	Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
Byte 3	EXMOD (Msb)	EXMOD	EXMOD	EXMOD	EXMOD	EXMOD	EXMOD	EXMOD
	Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
Byte 2	EXMOD	EXMOD (Lsb)	-	SetPosition	Halt	-	-	-
	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Byte 1	-	-	-	-	RCP_ID (Msb)	RCP_ID	RCP_ID	RCP_ID (Lsb)
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	REG_ID (Msb)	REG_ID (Lsb)	FSTP	TEACH	JOG	REER	STE	EN

Table 5 - Gripper Control Register structure (0x2C00 - 0x03)

The **Gripper Status Register (0x2C00-0x24)** is used to read the gripper status (i.e. to know the operating condition of the gripper). Access to this register by the external master is read-only. The following table shows the structure of this register.

	Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
Byte 3	RCP_ID (Msb)	RCP_ID	RCP_ID (Lsb)	VS4	VS3	VS2	VS1	DTCMP
	Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
Byte 2	RCP_ACT	UD_POS	LM_POS	TH_POS	PR_POS	OF_POS	BS_POS	HM_POS
	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Byte 1	-	-	-	-	-	-	-	-
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	RCP_RDY	MCMP	PositionSet	TEACH	JOG	ERR	-	EN

Table 6 - Gripper Status Register structure (0x2C00 - 0x24)

In case of use of one of free-positioning operation mode or the ones with velocity and/or force remodulation, on follow there are the sub-index to modify on-fly the single parameters.

Index	Sub-index	Parameter
0x2C00	0x25	Base Position
0x2C00	0x26	Offset Position
0x2C00	0x27	Teach Position 1
0x2C00	0x28	Teach Position 2
0x2C00	0x29	Limit Position
0x2C00	0x2A	Tolerance in Position
0x2C00	0x2B	Virtual Switch1 position
0x2C00	0x2C	Virtual Switch2 position
0x2C00	0x2D	Virtual Switch3 position
0x2C00	0x2E	Virtual Switch4 position
0x2C00	0x2F	Tolerance Virtual Switch 1/2/3/4 Position
0x2C00	0x30	Drive Velocity
0x2C00	0x31	Grip Force
0x2C00	0x32	Acceleration
0x2C00	0x33	Deceleration

*Table 7 - Parameters editable on-fly*

Beware about measurement units: unlike parameters regarding positions, velocity and accelerations that maintain the same unit of measurement used by the CAN objects of the drive (i.e. based on encoder counts), **the Grip Force (0x2C00 – 0x31) is expressed in % of rated torque to facilitate user interaction.**

Please see the chapter 7.4 (example 4) reported at the end of this manual to better understand the on-fly execution modes.

**Gripper Control and Status registers are available only with the Gimatic middleware.**

## 6.MPCFSerialDemo.exe INTERFACE

The Windows based application MPCFSerialDemo.exe developed by Gimatic allows the user to test the serial communication of the MPCF and to execute basic movements using recipes already set through the MPCFApplication.exe application. Only one application must be active at a time and it is not possible to change the recipes parameters (i.e. positions and velocity, values of acceleration, deceleration, virtual limits, etc...) through the MPCFSerialDemo.exe application (use the MPCFApplication.exe instead).

The main purpose of the MPCFSerialDemo.exe is to make possible the testing of the gripper before its integration into an automatic system (like collaborative robots, for example) and its functioning is based on Gimatic middleware.

The serial communication format used as default by the program is the Binary format.

The following picture shows the user interface of MPCFSerialDemo.exe.

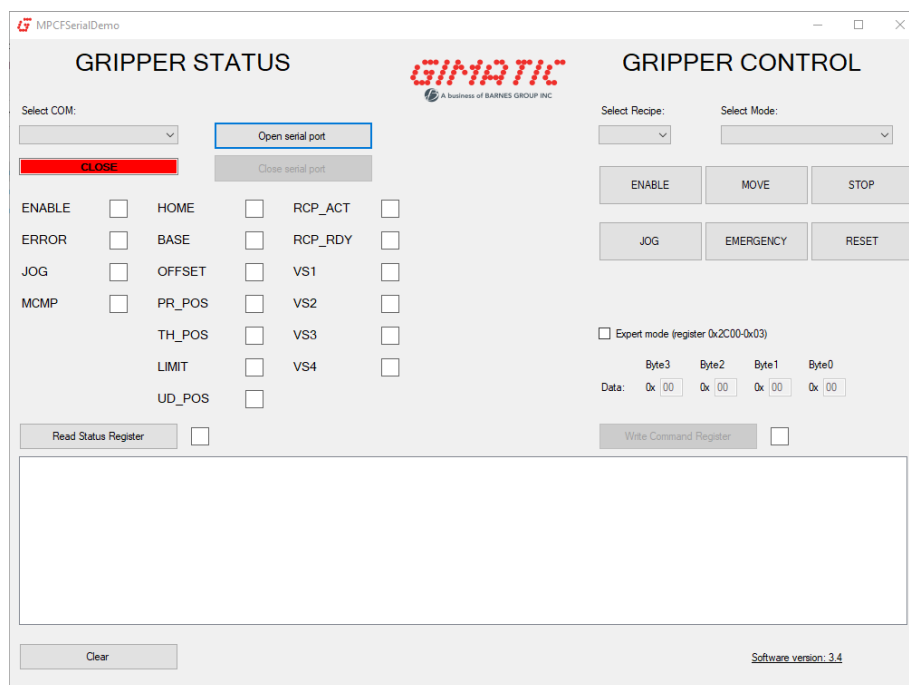


Figure 3 - MPCFSerialDemo.exe interface

This main window is ideally divided into three sections:

- Gripper Status (left side)
- Gripper Command (right side)
- Program Output (bottom side).

In the bottom-right angle of the interface, one can find indicated the software version.

## Gripper Status

Along with some serial communication options, the left side of the application shows the main information contained into the Status Register of the gripper.

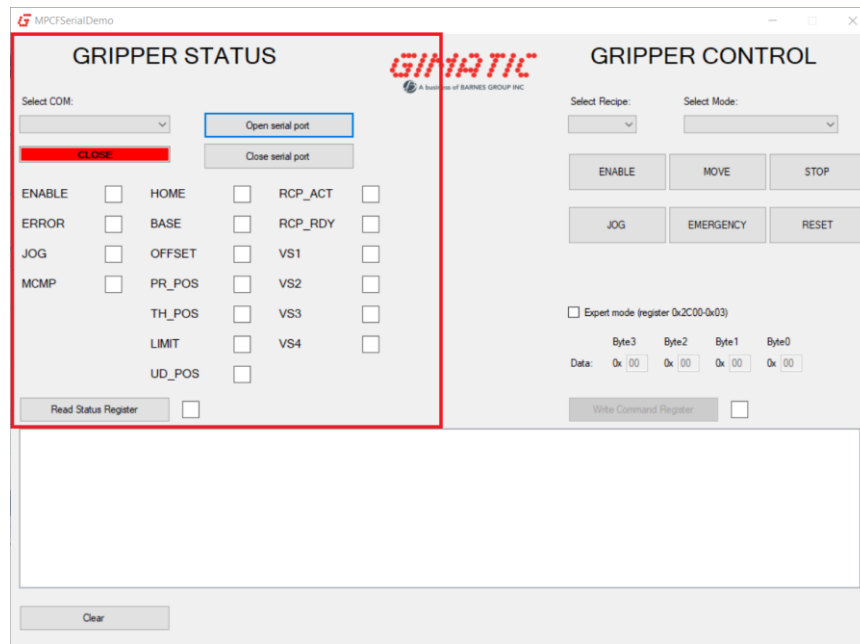


Figure 4 - Detail on Gripper Status

On the top of this section, the user can select the COM port used for the USB to serial converter, open and close the serial communication. A set of “virtual LEDs” are in the central area representing the “real time” value of the main bits of the Gripper Status Register. The status of these LEDs is automatically and continuously updated once the serial port is opened. The automatic reading stops in case the port gets closed by the user by pressing the “Close serial port” button. If the communication port is open, it is possible to manually force a reading of the Status Register by pressing the “Read Status Register” button. In case of a successful read, the LED beside the button blinks green, otherwise it blinks red.

The picture below shows an example of possible state of the gripper status LEDs. In this case port COM5 has been selected and opened, the status register has been read correctly and among all the active LEDs, the ENABLE one shows that the gripper has been enabled (jaws are holding position).

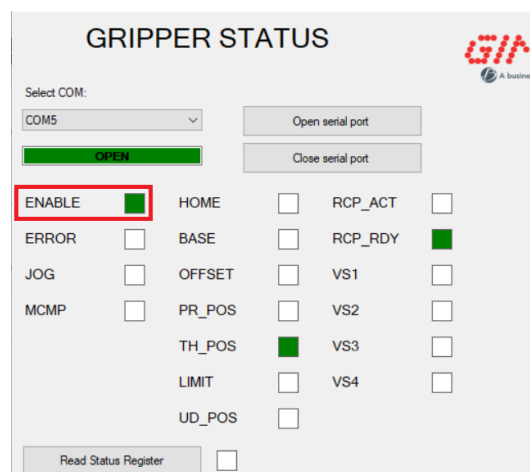


Figure 5 - Example with some LEDs activated

## Gripper Command

By using the controls available on the right side of the application the user can control the gripper.

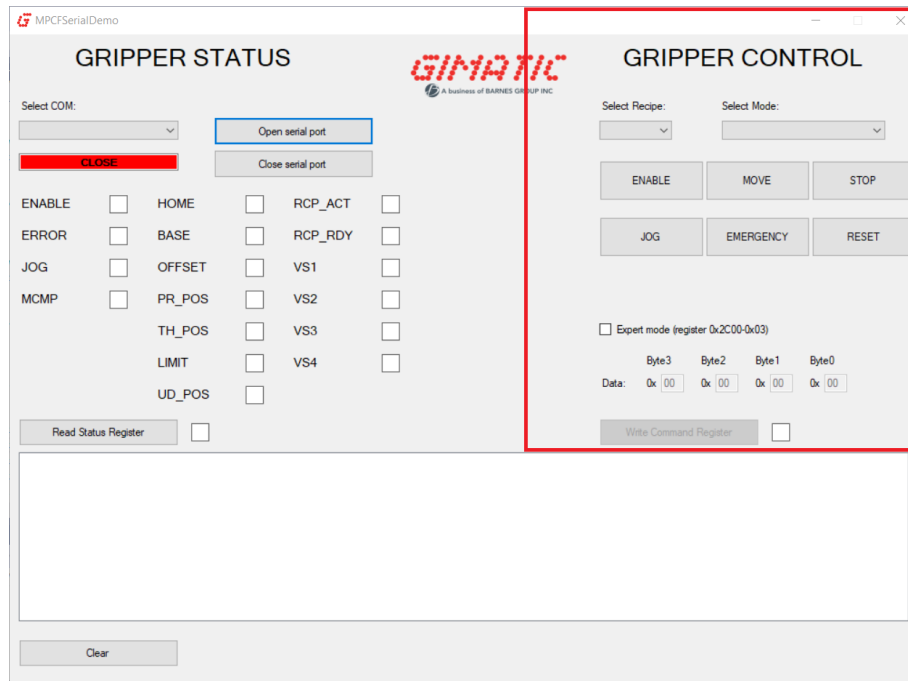


Figure 6 - Detail on Gripper Control

With the appropriate buttons, the user can command some basic operations like:

- Enable the gripper ("ENABLE")
- Select a recipe
- Select an operating mode
- Start the selected movement ("MOVE")
- Stop the current movement ("STOP")
- Make an emergency stop ("EMERGENCY"), different from the previous because the movement is stopped without deceleration ramp
- Move the gripper in JOG mode ("JOG").

By pressing any of these buttons, the application builds the proper serial command and send it automatically to the gripper. If data are correctly written, the LED beside the "Write Command Register" button blinks green. Otherwise, the LED blinks red. It is also possible to flag the Expert mode check box allowing for a direct writing of the Command Register (0x2C00-0x03). It is recommended to use this mode only in case of a clear understanding of how registers work. By using this Expert mode, data packet is not sent automatically but only once when the "Write Command Register" button is pressed.

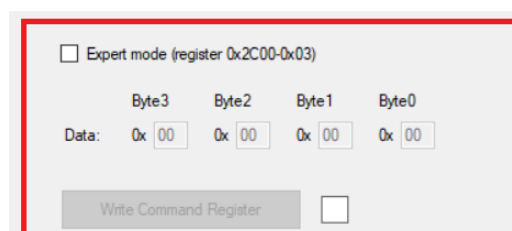
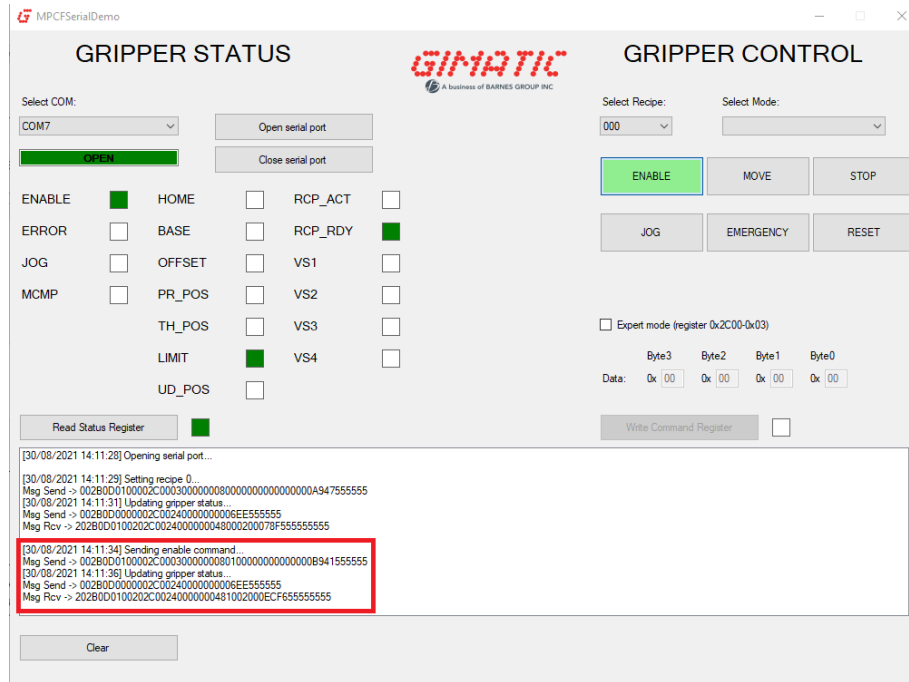


Figure 7 - Expert mode

## Program Output

The bottom part of the window hosts the output of the application and shows the raw data packets written and read by the application (in hex format).

The following picture shows the output corresponding to the Enable command send to the gripper by pressing the “ENABLE” button.



```
[30/08/2021 14:11:34] Sending enable command...
Msg Send -> 002B0D0100002C0003000000008010000000000000B9415555555
[30/08/2021 14:11:36] Updating gripper status...
Msg Send -> 002B0D0000002C00240000000006EE5555555
Msg Rcv -> 202B0D0100202C00240000000481002000ECF655555555
```

Analyzing the data packet sent to enable the gripper, it is structured like follow (values in hex):

00 2B 0D 01 0000 2C00 03 0000 0008 0100000000000000 B941 55555555

Network Address	Function (dec)		MEI type (dec)		Protocol Control			Reserved Field	Node ID		Index (2 byte)	
00	2B (43)		0D (13)		01 (write command)			00	00		HIGH BYTE	LOW BYTE
											2C	00
Sub-index	Starting Address (2 byte)		Number of data (2 byte)		Data (0 - 8 bytes)			CRC (2 bytes)		SYNC bytes (4 bytes)		
03	HIGH BYTE	LOW BYTE	HIGH BYTE	LOW BYTE	LOW BYTE	...	...	HIGH BYTE	HIGH BYTE	LOW BYTE	55555555	
	0000		00	08	01	00...00		00	B9	41		



Instead, the data packet sent to read the gripper status is structured like follow:

00 2B 0D 00 00 00 2C00 24 0000 0000 06EE 55555555

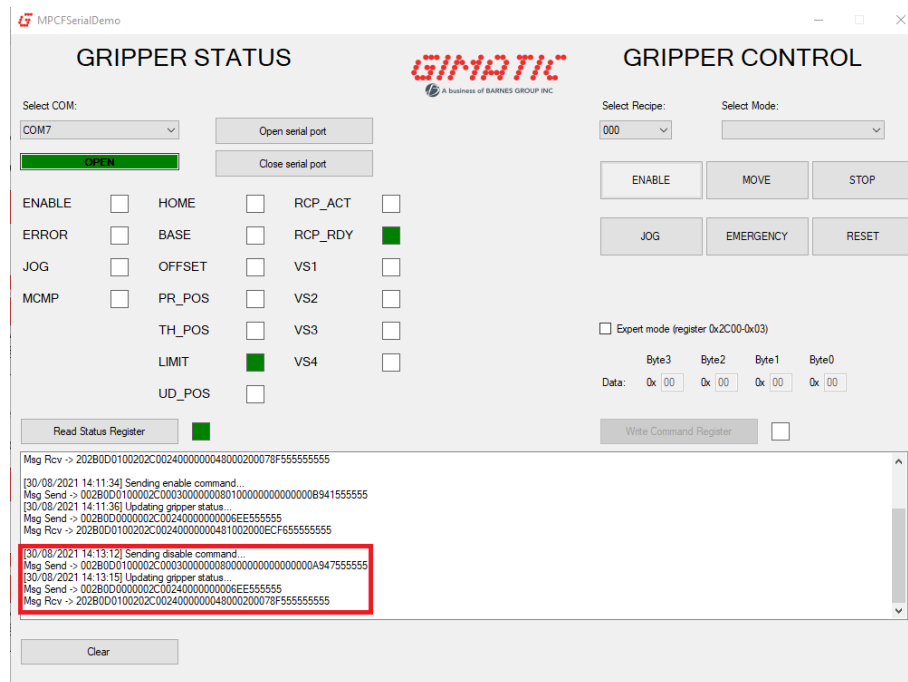
Network Address	Function (dec)		MEI type (dec)		Protocol Control			Reserved Field	Node ID		Index (2 byte)	
00	2B (43)		0D (13)		00 (read command)			00	00		HIGH BYTE	LOW BYTE
											2C	00
Sub-index	Starting Address (2 byte)		Number of data (2 byte)		Data (0 - 8 bytes)				CRC (2 bytes)		SYNC bytes (4 bytes)	
24	HIGH BYTE	LOW BYTE	HIGH BYTE	LOW BYTE	LOW BYTE	...	...	HIGH BYTE	HIGH BYTE	LOW BYTE	55555555	
	0000		00	00	/			06	EE			

The data packet received by the gripper representing the gripper status in real time:

20 2B 0D 01 00 20 2C00 24 0000 0004 81002000 ECF6 55555555

Network Address	Function (dec)		MEI type (dec)		Protocol Control			Reserved Field	Node ID		Index (2 byte)	
20	2B (43)		0D (13)		01 (write command)			00	20		HIGH BYTE	LOW BYTE
											2C	00
Sub-index	Starting Address (2 byte)		Number of data (2 byte)		Data (0 - 8 bytes)				CRC (2 bytes)		SYNC bytes (4 bytes)	
24	HIGH BYTE	LOW BYTE	HIGH BYTE	LOW BYTE	LOW BYTE	...	...	HIGH BYTE	HIGH BYTE	LOW BYTE	55555555	
	0000		00	04	81002000				EC	F6		

The following picture shows the output corresponding to the Disable command send to the gripper by newly pressing the “ENABLE” button (releasing the enable).



```
[30/08/2021 14:13:12] Sending disable command...
Msg Send -> 002B0D0100002C0003000000008000000000000000A94755555555
[30/08/2021 14:13:15] Updating gripper status...
Msg Send -> 002B0D0000002C00240000000006EE55555555
Msg Rcv -> 202B0D0100202C0024000000048000200078F555555555
```

Since the gripper status is continually read, the read data packet is not shown in the output area every time but only once after every write command, like shown in the previous examples.

The “Read Status Register” button can be used to visualize data representing the current content of the Status Register of the gripper.

By pressing the “Clear” button all the data listed in the output data can be removed.

## 7.EXAMPLES

The following sections are dedicated to communication examples showing the interaction steps a user can make with the applications also highlighting the occurring data packet transmission. **Please note that these data packets can change depending on the specific Node ID assigned to the gripper.**

Examples are made both with MPCFSerialDemo.exe and with a generic serial master (like can be a robot in a real application, for example).

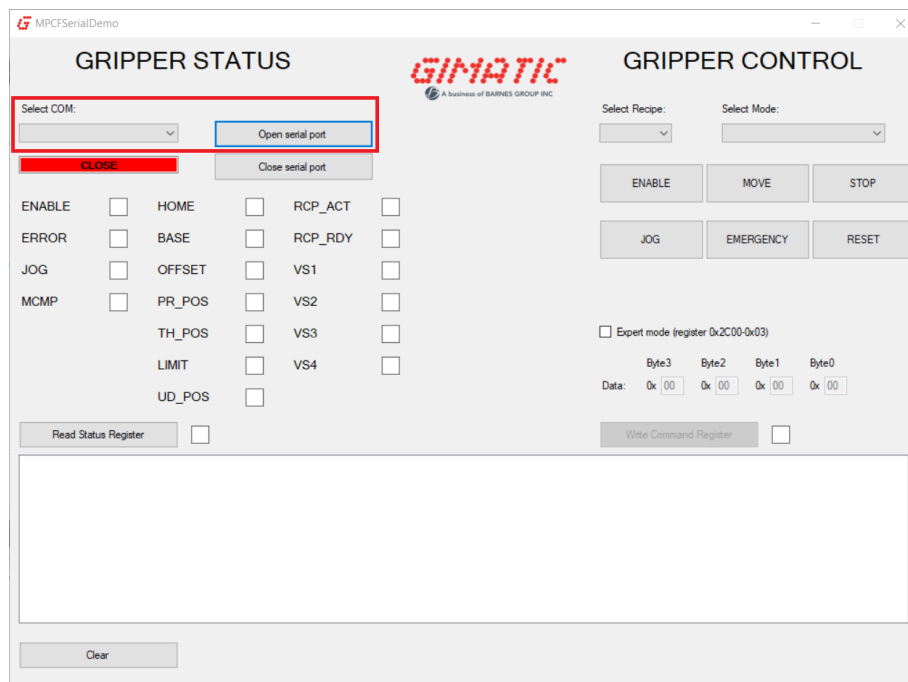
### 7.1 Example 1: Commanding a movement from Base to Limit positions

The following example shows how to command a movement from Base Position to Limit Position of the gripper using recipe 001. This is the sequence of operations:

- A. Opening of the serial port
- B. Selecting the recipe and the operation mode
- C. Enabling the gripper and executing the movement
- D. Closing the serial port and exiting the program.

#### A. Opening of the serial port

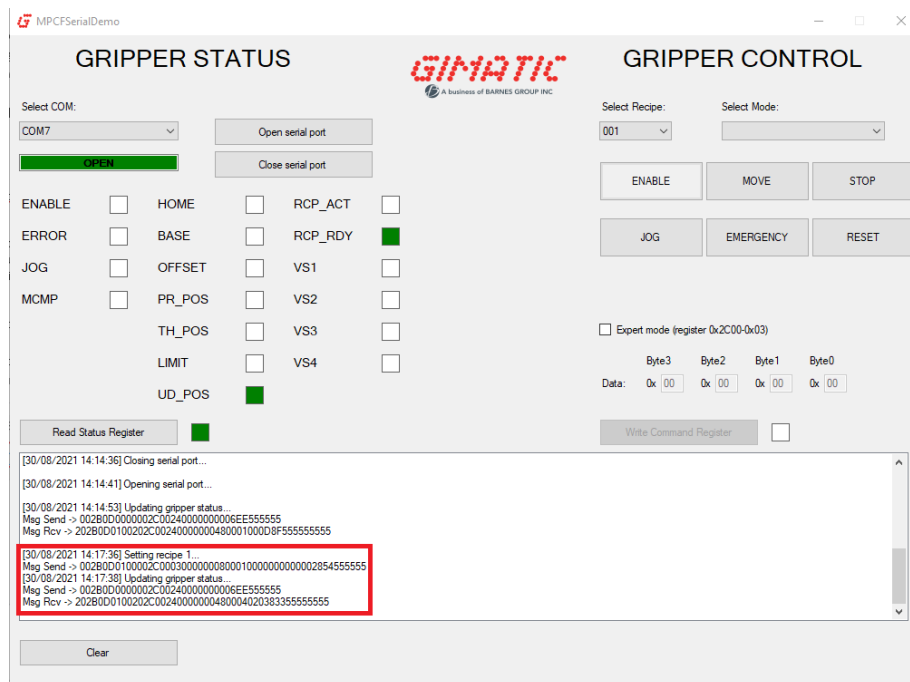
Connect the gripper to the power supply unit and the serial lines to the laptop by means of an USB to serial converter. Check the converter to be correctly identified by the Windows operating system and run the Gimatic “MPCFSerialDemo.exe” application. Select then the proper serial COM port used by the USB to serial converter where the converter and press the “Open serial port” button.



In case the port opens correctly, the serial port rectangular LED turns from red to green color indicating a successful operation (the label changes from “CLOSE” to “OPEN”). At this point all the LEDs indicators of the gripper status register are periodically and automatically updated by the application.

## B. Selecting the recipe and the operation mode

From the gripper control area of the main window, select recipe number 001. This selection generates the transmission of a first data packet from the laptop to the gripper as visible from the program output area (bottom part of the interface).



Anytime a new transmission occurs, few lines of text are added into this program output area: a timestamp with short description of the type of message and the raw bytes of data sent or received (in hex format). Once the recipe selection message has been correctly handled by the gripper, the “RCP\_RDY” LED turns green indicating that the selected recipe has been correctly uploaded into the memory of the gripper.

```
[30/08/2021 14:17:36] Setting recipe 1...
Msg Send -> 002B0D0100002C0003000000008000100000000000285455555555
[30/08/2021 14:17:38] Updating gripper status...
Msg Send -> 002B0D0000002C00240000000006EE55555555
Msg Rcv -> 002B0D0100202C00240000000480004020383355555555
```

Similar behavior occurs when selecting the “P2P\_SPEED\_B-L\_RCP” operation mode (representing a point-to-point motion with speed setpoint from Base Position to Limit Position using parameters from recipe). Once the operation mode message has been correctly handled by the gripper, the “RCP\_ACT” LED turns green indicating that the gripper is ready to perform the selected movement.

**GRIPPER STATUS**

Select COM: COM7

ENABLE <input checked="" type="checkbox"/>	HOME <input type="checkbox"/>	RCP_ACT <input checked="" type="checkbox"/>
ERROR <input type="checkbox"/>	BASE <input type="checkbox"/>	RCP_RDY <input checked="" type="checkbox"/>
JOG <input type="checkbox"/>	OFFSET <input type="checkbox"/>	VS1 <input type="checkbox"/>
MCMP <input type="checkbox"/>	PR_POS <input type="checkbox"/>	VS2 <input type="checkbox"/>
	TH_POS <input type="checkbox"/>	VS3 <input type="checkbox"/>
	LIMIT <input type="checkbox"/>	VS4 <input type="checkbox"/>
	UD_POS <input checked="" type="checkbox"/>	

☐

**GRIPPER CONTROL**

Select Recipe: 001

Select Mode: P2P\_SPEED\_B-L\_RCP

☐ Expert mode (register 0x2C00-0x03)

Byte3 Byte2 Byte1 Byte0  
Data: 0x 00 0x 00 0x 00 0x 00

☐

Msg Rcv -> 202B0D0100202C00240000000480001000D8F555555555  
 [30/08/2021 14:17:36] Setting recipe 1  
 Msg Send -> 002B0D0100002C00030000000080001000000000002854555555  
 [30/08/2021 14:17:38] Updating gripper status  
 Msg Send -> 002B0D0000002C00240000000006EE55555555  
 Msg Rcv -> 202B0D0100202C002400000004800004020383355555555  
 [30/08/2021 14:25:03] Point to point from Base to Limit position selected  
 Msg Send -> 002B0D0100002C00030000000080001001900000000AC6F55555555  
 [30/08/2021 14:25:05] Updating gripper status  
 Msg Send -> 002B0D0000002C00240000000006EE55555555  
 Msg Rcv -> 202B0D0100202C00240000000480000C020B83A555555555

[30/08/2021 14:25:03] Point to point from Base to Limit position selected  
 Msg Send -> 002B0D0100002C00030000000080001001900000000AC6F55555555  
 [30/08/2021 14:25:05] Updating gripper status...  
 Msg Send -> 002B0D0000002C00240000000006EE55555555  
 Msg Rcv -> 202B0D0100202C00240000000480000C020B83A555555555

### C. Enabling the gripper and executing the movement

Press the “ENABLE” button to enable the drive and get the jaws holding their position. The output area of the program shows the transmission data packets sent and received by the application and in case of a successful operation the LED corresponding to the enable bit of the gripper status register turns green.

**GRIPPER STATUS**

Select COM: COM7

ENABLE <input checked="" type="checkbox"/>	HOME <input type="checkbox"/>	RCP_ACT <input checked="" type="checkbox"/>
ERROR <input type="checkbox"/>	BASE <input type="checkbox"/>	RCP_RDY <input checked="" type="checkbox"/>
JOG <input type="checkbox"/>	OFFSET <input type="checkbox"/>	VS1 <input type="checkbox"/>
MCMP <input type="checkbox"/>	PR_POS <input type="checkbox"/>	VS2 <input type="checkbox"/>
	TH_POS <input type="checkbox"/>	VS3 <input type="checkbox"/>
	LIMIT <input type="checkbox"/>	VS4 <input type="checkbox"/>
	UD_POS <input checked="" type="checkbox"/>	

☐

**GRIPPER CONTROL**

Select Recipe: 001

Select Mode: P2P\_SPEED\_B-L\_RCP

☐ Expert mode (register 0x2C00-0x03)

Byte3 Byte2 Byte1 Byte0  
Data: 0x 00 0x 00 0x 00 0x 00

☐

Msg Rcv -> 202B0D0100202C00240000000480004020383355555555  
 [30/08/2021 14:25:03] Point to point from Base to Limit position selected  
 Msg Send -> 002B0D0100002C00030000000080001001900000000AC6F55555555  
 [30/08/2021 14:25:05] Updating gripper status  
 Msg Send -> 002B0D0000002C00240000000006EE55555555  
 Msg Rcv -> 202B0D0100202C00240000000480000C020B83A555555555  
 [30/08/2021 14:26:28] Sending enable command  
 Msg Send -> 002B0D0100002C00030000000080001001900000000BC95555555  
 [30/08/2021 14:26:31] Updating gripper status  
 Msg Send -> 002B0D0000002C00240000000006EE55555555  
 Msg Rcv -> 202B0D0100202C0024000000048100C0202C39555555555

```

[30/08/2021 14:26:28] Sending enable command...
Msg Send -> 002B0D0100002C0003000000080101001900000000BC6955555555
[30/08/2021 14:26:31] Updating gripper status...
Msg Send -> 002B0D0000002C00240000000006EE55555555
Msg Rcv -> 202B0D0100202C0024000000048100C0202C3955555555

```

Now, press the “MOVE” button to start the movement. The output area of the program shows the transmission data packets sent and received by the application and when the jaws reach the Limit Position the “MCMP” LED turns green indicating that the movement has been completed correctly.

```

[30/08/2021 14:39:29] Sending motion start command...
Msg Send -> 002B0D0100002C00030000000803010019000000009C6555555555
[30/08/2021 14:39:31] Updating gripper status...
Msg Send -> 002B0D0000002C00240000000006EE55555555
Msg Rcv -> 202B0D0100202C002400000004C100A0206C2155555555

```

Select the “P2P\_SPEED\_L-B\_RCP” operation mode, release the “MOVE” button and then press it again to eventually perform a backward motion from Limit position to Base Position. Try also to stop the motion before completing the movement command by pressing the “STOP” button.

Other movements can be performed by simply selecting a different operation mode.

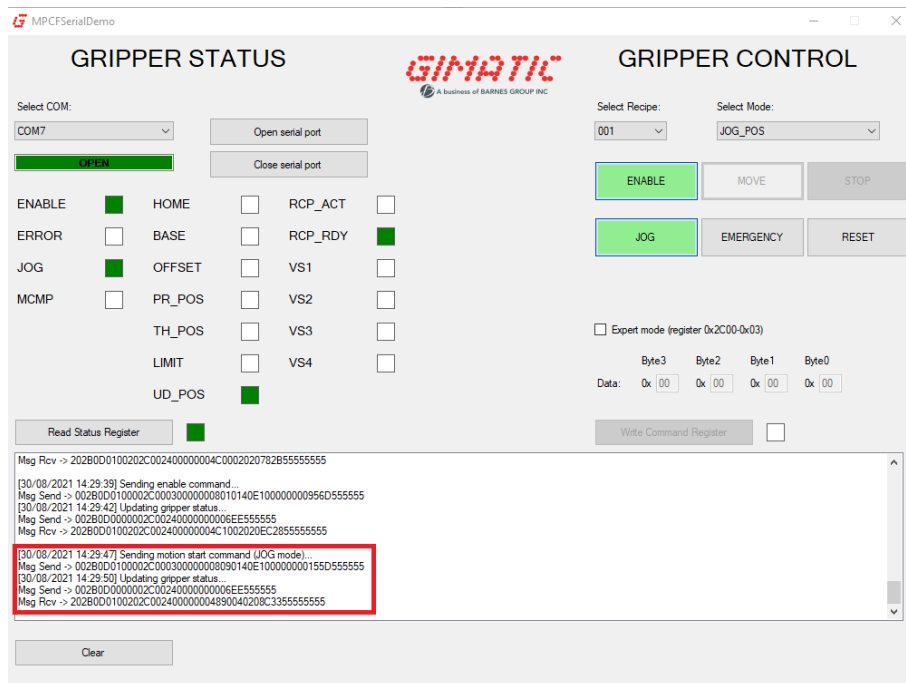
#### D. Closing the serial port and exiting the program

To exit the program, close the serial port and then close the program by left click on the “X” icon on the top right part of the interface. The program can be closed keeping the gripper in the enable state (i.e. holding the position and/or a part gripped).

## 7.2 Example 2: Movement in JOG mode

Among all the movement options it is also possible to command the jaws in JOG mode. Once the gripper is in enabled state, either select the “JOG\_POS” or “JOG\_NEG” operation mode to command the jaws in positive or negative direction respectively and press and hold the “JOG” button to start the movement at a constant speed.

Please note that when a JOG mode is selected buttons “MOVE” and “STOP” are disabled. Simply release the “JOG” button to stop the movement. In JOG mode the recipe uploaded during this operating mode is not relevant because the velocity setpoint is stored into the memory of the gripper.



```
[30/08/2021 14:29:47] Sending motion start command (JOG mode)...\nMsg Send -> 002B0D0100002C0003000000008090140E10000000155D55555555\n[30/08/2021 14:29:50] Updating gripper status...\nMsg Send -> 002B0D0000002C00240000000006EE55555555\nMsg Rcv -> 202B0D0100202C002400000004890040208C3355555555
```

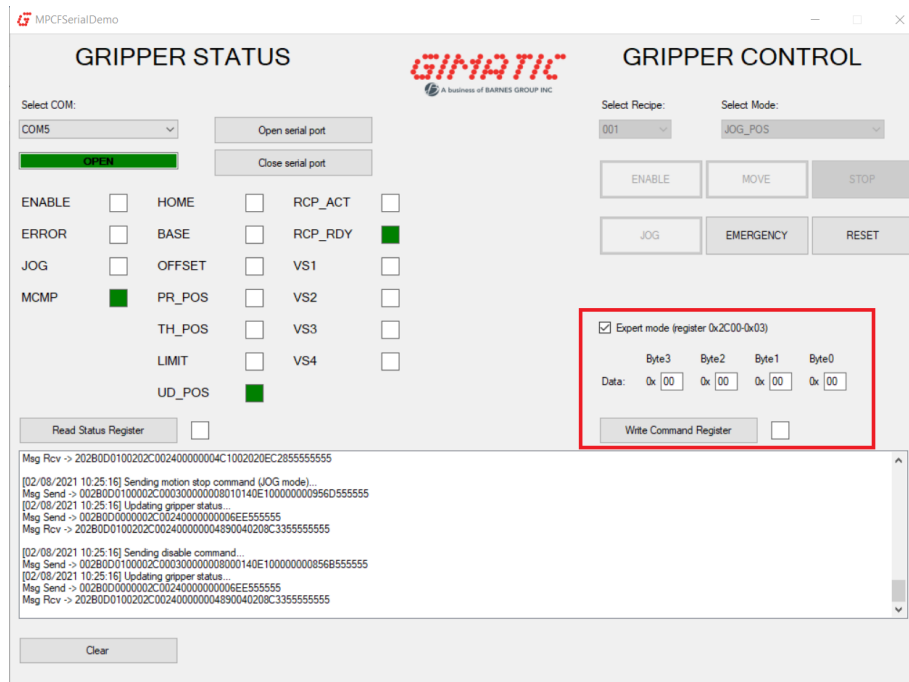
## 7.3 Example 3: Movement to Home position in Expert mode

This example aims to expose the procedure to operate in Expert mode. The intention is to make a movement to Home Position using the available bytes of the Gripper Control Register (0x2C00-0x03).

Main steps are:

- Defining the data packet that sets the recipe (in this case recipe 003) and the operation mode (in this case Home)
- Enabling the gripper
- Starting the movement
- Disabling the gripper.

Check the “Expert mode” checkbox to enable the Expert mode. All the standard control buttons are disabled as the commands can now only be sent over serial by entering the proper data bytes into the active edit boxes (the highlighted red rectangle in following image).



#### A. Defining the data packet that sets the recipe and the operation mode

Referring to the structure of the Gripper Control Register, the 4 less significant bits of the byte 1 represent the identification number of the recipe (“RCP\_ID”, bits 8, 9, 10 and 11).

	Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
Byte 3	EXMOD (Msb)	EXMOD	EXMOD	EXMOD	EXMOD	EXMOD	EXMOD	EXMOD
	Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
Byte 2	EXMOD	EXMOD (Lsb)	-	SetPosition	Halt	-	-	-
	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Byte 1	-	-	-	-	RCP_ID (Msb)	RCP_ID	RCP_ID	RCP_ID (Lsb)
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	REG_ID (Msb)	REG_ID (Lsb)	FSTP	TEACH	JOG	REER	STE	EN

In this example, we want to set the recipe 003 corresponding to 0011 in binary representation.

The operation mode is set by assigning the proper binary representation to bits “EXMOD”. In this example the Homing movement is selected presented by number 900 which corresponds to 0011 1000 0100 in binary representation. Please note that for the execution mode 10 bits are splitted over byte 2 (for the 2 less significant bits) e byte 3 (for the 8 most significant bits).



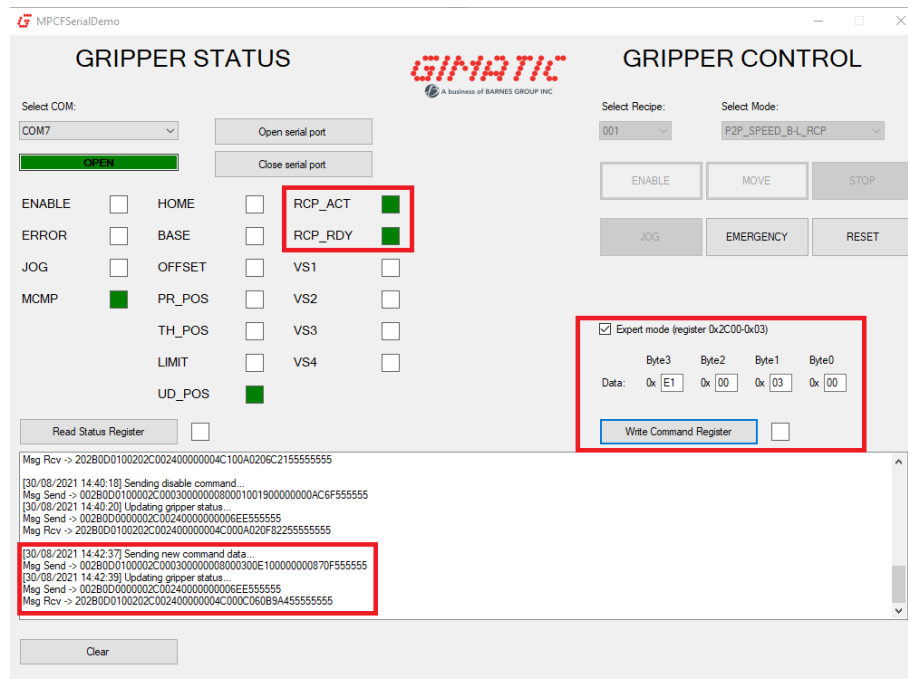
Following above description, the following bits can be set into the Gripper Control Register:

Byte 3	1	1	1	0	0	0	0	1
Byte 2	0	0	0	0	0	0	0	0
Byte 1	0	0	0	0	0	0	1	1
Byte 0	0	0	0	0	0	0	0	0

and in hexadecimal format:

Byte 3	0xE1
Byte 2	0x00
Byte 1	0x03
Byte 0	0x00

The calculated values can now be inserted into the appropriate box and by clicking on “Write Command Register” button the gripper is configured accordingly.



## B. Enabling the gripper

To enable the gripper, it sufficient to set bit “EN” (bit 0 of the Gripper Control Register).

The new bytes in hexadecimal format are:

Byte 3	0xE1
Byte 2	0x00
Byte 1	0x03
Byte 0	0x01

By writing this new value to the Gripper Control Register, the gripper enters the enabled state as shown by the “ENABLE” LED turning green.

### C. Starting the movement

A this point it is possible to start the movement by setting to 1 bit “STE” (bit 1 of the Gripper Control Register).

Byte 3	1	1	1	0	0	0	0	1
Byte 2	0	0	0	0	0	0	0	0
Byte 1	0	0	0	0	0	0	1	1
Byte 0	0	0	0	0	0	0	1	1

The new value of the register in hexadecimal format is:

Byte 3	0xE1
Byte 2	0x00
Byte 1	0x03
Byte 0	0x03

**GRIPPER STATUS**

Select COM: COM7

**GRIPPER CONTROL**

Select Recipe: 001

☒ Expert mode (register 0x2C00-0x03)

Byte3 Byte2 Byte1 Byte0  
Data: 0x E1 0x 00 0x 03 0x 03

Log:

```
[30/08/2021 14:48:54] Updating gripper status...
Mag Send -> 002B0D0000002C00240000000006EE555555
Mag Rcv -> 202B0D0100202C00240000000048100C060ADBA55555555

[30/08/2021 14:48:58] Updating gripper status...
Mag Send -> 002B0D0000002C00240000000006EE555555
Mag Rcv -> 202B0D0100202C00240000000048100C060ADBA55555555

[30/08/2021 14:49:03] Sending new command data...
Mag Send -> 002B0D0100002C00030000000008030300E100000000B705555555
Mag Send -> 002B0D0100002C00030000000008030300E100000000B705555555
Mag Send -> 002B0D0000002C00240000000006EE555555
Mag Rcv -> 202B0D0100202C0024000000004C10081602BA2555555555
```

In case of a successful completion of the movement the “MCMP” and “HOME” LEDs turn green.

The same result could have been obtained with a single command by directly writing the following values into the Gripper Control Register:

Byte 3	0xE1
Byte 2	0x00
Byte 1	0x03
Byte 0	0x03

#### D. Disabling the gripper

The gripper can be disabled by setting Byte 0 to the value 0x00, the serial port and the program can then be closed.

**GRIPPER STATUS**

Select COM: COM7

**GRIPPER CONTROL**

Select Recipe: 001

☒ Expert mode (register 0x2C00-0x03)

Byte3 Byte2 Byte1 Byte0  
Data: 0x E1 0x 00 0x 03 0x 00

Log:

```
[30/08/2021 14:49:03] Sending new command data...
Mag Send -> 002B0D0100002C00030000000008030300E100000000B705555555
Mag Send -> 002B0D0100002C00030000000008030300E100000000B705555555
Mag Send -> 002B0D0000002C00240000000006EE555555
Mag Rcv -> 202B0D0100202C0024000000004C10081602BA2555555555

[30/08/2021 14:49:53] Sending new command data...
Mag Send -> 002B0D0100002C00030000000008030300E100000000B705555555
Mag Send -> 002B0D0100002C00030000000008030300E100000000B705555555
Mag Send -> 002B0D0000002C00240000000006EE555555
Mag Rcv -> 202B0D0100202C0024000000004C0008160BFA1555555555
```

## 7.4 Example 4: Communication with a generic serial master – Moving to Limit Position with mode 100 and mode 800

This example aims to show two different things:

- Data packets construction without MPCFSerialDemo.exe.
- The execution of a free-positioning operation mode (in this case mode 800) and therefore underline the difference between it and a standard operating mode (in this case mode 100).

### A. Configure the communication and setting the Binary format

First of all, one must setup the serial communication as reported in chapter 3. Please remember that since we are avoiding the MPCFSerialDemo.exe, the switching to the Binary format has to be done by the user.

The following data packets must be sent:

- 0D: **wake up byte** to establish a communication.
- '0' 'x' '2' '0' ' ' 'W' ' ' '0' 'x' '8' '2' '0' '0' '0' ' ' '0' 'x' '1' '\r': **enable the Binary format**.
- 20 2B 0D 01 00 20 20 00 06 00 00 00 01 01 55 55 55 55: **set CRC**.

### B. Execution of mode 100 with recipe 001

The movement is the same reported in chapter 7.1 (example 1), the only difference is that here the data packets must be prepared by the user and are not automatically prepared by the application. The result must be the same.

The following packets are sent:

- Select mode, recipe and enable the gripper. Register 0x2C00, sub-index 0x03 (Gripper Control Register), value: 0x19000101  
Message send → 00 2B 0D 01 00 00 2C 00 03 00 00 00 08 01 01 00 19 00 00 00 00 BC 69 55 55 55 55  
In one fell swoop, we set the mode (100), the recipe (001) and the gripper status (enabled)
- Start movement. Register 0x2C00, sub-index 0x03 (Gripper Control Register), value: 0x19000103  
Message send → 00 2B 0D 01 00 00 2C 00 03 00 00 00 08 03 01 00 19 00 00 00 00 9C 65 55 55 55 55  
The STE bit has been activated and the movement starts.
- Disable the STE bit (is like when one disables the MOVE button in app). Register 0x2C00, sub-index 0x03 (Gripper Control Register), value: 0x19000101  
Message send → 00 2B 0D 01 00 00 2C 00 03 00 00 00 08 01 01 00 19 00 00 00 00 BC 69 55 55 55 55

### C. Execution of mode 800 with parameters of recipe 001

- Select mode, recipe and enable the gripper. Register 0x2C00, sub-index 0x03 (Gripper Control Register), value: 0xC8000101  
Message send → 20 2B 0D 01 00 20 2C 00 03 00 00 00 04 01 01 00 C8 F1 4E 55 55 55 55
- Set the on-fly Limit position value. Register 0x2C00, **sub-index 0x29**, value: 70000 increments (example, equals to  $70000/3200 = 21.88$  mm), 0x011170  
Message send → 20 2B 0D 01 00 20 2C 00 29 00 00 00 03 70 11 01 7F D3 55 55 55 55
- Start movement. Register 0x2C00, sub-index 0x03 (Gripper Control Register), value: 0x03 (changes only the last byte to activate the STE bit)  
Message send → 20 2B 0D 01 00 20 2C 00 03 00 00 00 01 03 F9 59 55 55 55 55

In the first case, the Limit Position value was the one stored in the recipe, in the second case the value was the one set in the RAM (this is why is called “on-fly” value, it is a momentary value). It allows to move to a different position without change the value permanently stored in the recipe.

To know the status of the gripper, it is sufficient to read the register 0x2C00, sub-index 0x24 (Gripper Status register) and analyse the status of each bit (exactly as done by the application).

## 7.5 Example 5: Communication with a generic serial master – Interacting directly with CAN objects

On follow there are reported some operations as examples of interacting directly with the CAN objects bypassing the Gimatic middleware. Please remember that the access to the objects is possible with or without the presence of the middleware. However, erasing the middleware makes earning velocity in data transmission making the gripper act like a generic device adhering to the CANopen DS-402 standard. The objects that can be access by users are reported here (CiA 402 object dictionary):

<https://drives.novantamotion.com/emcl/dictionaries#Dictionaries-CiA402objectdictionary>.

### **0x6077 – Read torque actual value (value returned in ‰ rated torque):**

- Message sent to the gripper (please remember that all bytes are in hex format)  
→ 20 2B 0D 00 00 20 60 77 00 00 00 00 00 D8 51 55 55 55 55  
(Mainly 20 is the gripper node, 00 indicates a read operation, 6077 represents the object index to be read, 00 sub-index, D851 are the CRC bytes, the rest of the bytes are easily deducible from Table 4)
- Message received from the gripper  
→ 20 2B 0D 01 00 20 60 77 00 00 00 00 02 2C 00 79 BC 55 55 55 55  
(Mainly 20 is the gripper node, 6077 represents the object index, 00 sub-index, 0002 (2 in decimal) the number of bytes of data received, 2C00 (0x002C, 44 in decimal) the data received i.e. the register value)

Therefore, the result is 44‰ of the rated torque, i.e. 4.4% of rated torque (it’s the same parameter one can see from MPCFApplication.exe). For instance:

Act. Position	54.29	[mm]
Act. Velocity	0.00	[mm/s]
Act. Torque	5.30	[‰]
Temperature	41.30	[°C]

**Note that the data bytes must be write/read backwards.**

### **0x6064 – Read position actual value (value returned in encoder counts):**

- Message sent to the gripper → 20 2B 0D 00 00 20 60 64 00 00 00 00 00 D0 43 55 55 55 55  
(Mainly 20 is the gripper node, 00 indicates a read operation, 6064 represents the object index to be read, 00 sub-index, D043 are the CRC bytes, the rest of the bytes are easily deducible from the Table 4)

- Message received from the gripper  
→ 20 2B 0D 01 00 20 60 64 00 00 00 04 D6 D6 02 00 FF E6 55 55 55 55  
(Mainly 20 is the gripper node, 6064 represents the object index, 00 sub-index, 0004 (4 in decimal) the number of bytes of data received, D6D60200 (0x0002D6D6, 186070 in decimal) the data received i.e. the register value)

Therefore, the result is 186070 increments.

$$186070/3200 = 58.14 \text{ mm}$$

#### **0x606C – Read velocity actual value (value returned in encoder counts / second):**

- Message Sent to the gripper → 20 2B 0D 00 00 20 60 6C 00 00 00 00 90 4A 55 55 55 55  
(Mainly 20 is the gripper node, 00 indicates a read operation, 606C represents the object index to be read, 00 sub-index, 904A are the CRC bytes , the rest of the bytes are easily deducible from the Table 4)
- Message received from the gripper  
→ 20 2B 0D 01 00 20 60 6C 00 00 00 04 40 46 06 00 54 35 55 55 55 55  
(Mainly 20 is the gripper node, 606C represents the object index, 00 sub-index, 0004 (4 in decimal) the number of bytes of data, 40460600 (0x00064640, 411200 in decimal) the data received i.e. the register value)

Therefore, the result is 411200 increments/second.

$$411200/3200 = 128.5 \text{ mm/s}$$

#### **0x6071 – Store target torque (value in ‰ rated torque):**

Desired value: 50% rated torque, equals to 500‰ rated torque and therefore 0x01F4 (hex).

Message Sent to the gripper → 20 2B 0D 01 00 20 60 71 00 00 00 02 F4 01 C9 A2 55 55 55 55

(Mainly 20 is the gripper node, 01 indicates a write operation, 6071 represents the object index to be write, 00 sub-index, 0002 (2 in decimal) the number of bytes to write, F401 are the data bytes (0x01F4), C9A2 are the CRC bytes , the rest of the bytes are easily deducible from the Table 4)

Please note the measurement unit of the torque.

#### **0x6071 – Read target torque (value in ‰ rated torque):**

- Message sent to the gripper → 20 2B 0D 00 00 20 60 71 00 00 00 00 48 54 55 55 55 55  
(Mainly 20 is the gripper node, 00 indicates a read operation, 6071 represents the object index to be read, 00 sub-index, 4854 are the CRC bytes , the rest of the bytes are easily deducible from the Table 4)
- Message received from the gripper  
→ 20 2B 0D 01 00 20 60 71 00 00 00 02 F4 01 C9 A2 55 55 55 55  
(Mainly 20 is the gripper node, 6071 represents the object index, 00 sub-index, 0002 (2 in decimal) the number of bytes of data, F401 (0x01F4, 500 in decimal) the data received i.e. the register value)

Therefore, the result is 500‰ of the rated torque, i.e. 50% of rated torque.

#### **0x60FF – Store target velocity (value in encoder counts / second):**

Desired value: 31.25 mm/s, equals to 100000 increments/second and therefore 0x0186A0 (hex).

Message Sent to the gripper → 20 2B 0D 01 00 20 60 FF 00 00 00 00 03 A0 86 01 34 E3 55 55 55 55

(Mainly 20 is the gripper node, 01 indicates a write operation, 60FF represents the object index to be write, 00 sub-index, 04 (4 in decimal) the bytes of data to write, A08601 are the data bytes (0x0186A0), 34E3 are the CRC bytes , the rest of the bytes are easily deducible from the Table 4)

#### **0x60FF – Store target velocity (value in encoder counts / second):**

Desired value: -31.25 mm/s, equals to -100000 increments/second and therefore 0xFE7960 (hex, 2's complement).

Message Sent to the gripper → 20 2B 0D 01 00 20 60 FF 00 00 00 00 03 60 79 FE BB EE 55 55 55 55

(Mainly 20 is the gripper node, 01 indicates a write operation, 60FF represents the object index to be write, 00 sub-index, 03 (3 in decimal) the bytes of data to write, 6079FE are the data bytes (0xFE7960), BBEE are the CRC bytes , the rest of the bytes are easily deducible from the Table 4)

Please note the 2's complement for the negative value to be write.

#### **0x6060 – Set the modes of operation:**

Desired mode: profile torque, value: 0x04.

Message Sent to the gripper → 20 2B 0D 01 00 20 60 60 00 00 00 00 01 04 D6 FB 55 55 55 55

(Mainly 20 is the gripper node, 01 indicates a write operation, 6060 represents the object index to be write, 00 sub-index, 01 (1 in decimal) the bytes of data to write, 04 is the data byte (0x04), D6FB are the CRC bytes, the rest of the bytes are easily deducible from the Table 4)

#### **Movement example (profile position to 50 mm):**

- Set operation mode to profile position mode. Register 0x6060, value 0x01, message send → 20 2B 0D 01 00 20 60 60 00 00 00 00 01 01 D6 E5 55 55 55 55
- Set target position to 50 mm (50\*3200 = 160000 increments). Register 0x607A, value 0x027100 (160000 in dec) to be write backwards, message send → 20 2B 0D 01 00 20 60 7A 00 00 00 00 03 00 71 02 A1 8C 55 55 55 55
- Enable the gripper. Register 0x6040, value 0x0F, message send → 20 2B 0D 01 00 20 60 40 00 00 00 00 01 0F F4 C1 55 55 55 55
- Start movement. Register 0x6040, value 0x3F, message send → 20 2B 0D 01 00 20 60 40 00 00 00 00 01 3F F4 61 55 55 55 55
- Check the actual position value (read object 0x6077 value as above)







GIMATIC S.R.L.  
Via Enzo Ferrari 2/4 25030 Roncadelle (Bs) Italy  
Tel: +39 030 2584655 Fax: +39 030 2583886  
[info@gimatic.com](mailto:info@gimatic.com)  
[www.gimatic.com](http://www.gimatic.com)