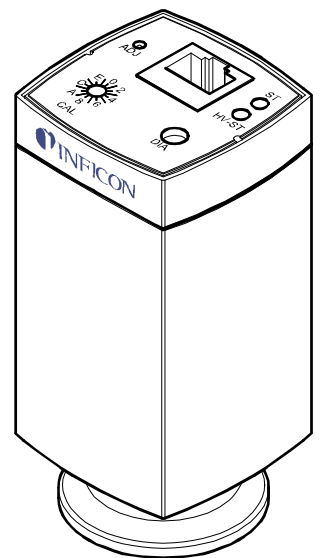


## RS232C

Serial Interface for Cold Cathode and Cold Cathode Pirani Gauges

MAG500, MAG504

MPG500, MPG504



RS232C

## General Information

The Serial Interface allows the communication of the digital INFICON Cold Cathode Gauges (MAG500, MAG504) and the INFICON Cold Cathode Pirani Gauges (MPG500, MPG504) with a PC or another appropriate controller.

The RS232C interface integrated in the gauge allows to digitally transmit measurement values and information on the gauge status as well as to make parameter settings.

## Interface Protocol

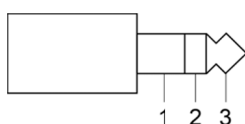
The Serial Interface is used in Master-Slave mode. Without a corresponding request from the Master, the device does not transmit any data.

Instructions to the gauge are transmitted via binary protocol.

### Data format

- binary
- 8 data bits
- 1 stop bit
- no parity bit
- no handshake

### RS232 pin assignment

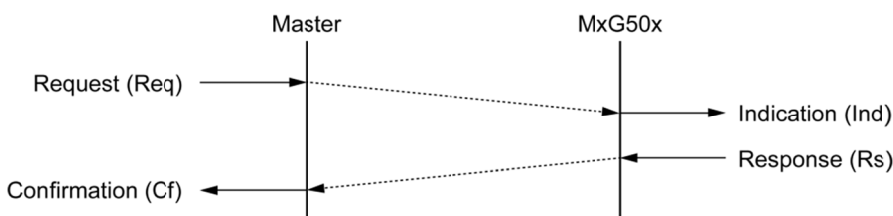


Pin 1	Supply common
Pin 2	RS232, RxD
Pin 3	RS232, TxD

### RS232 address and transmission rate

It is not necessary to set an address of the RS232C interface. For the protocol 0 is always used. The transmission rate can be set as parameter (→ § 5).

### Data Link Layer



The Data Link Layer uses the following terminology:

#### request (Req)

A request is an instruction (to read/or write) transmitted by the Master.

#### indication (Ind)

An indication means that the Slave (gauge) recognizes a request from the Master.

#### response (Rs)

A response is an answer from the Slave to the Master.

#### confirmation (Cf)

A confirmation is an acknowledgement of the response by the Master.

## Protocol Frame

Every protocol layer of the communication protocol is represented in a protocol frame. The maximum length per frame is 64 byte. The data field of the data link layer consists of

- a command (Cmd)
- a parameter identifier (PID)
- a data field (Data).

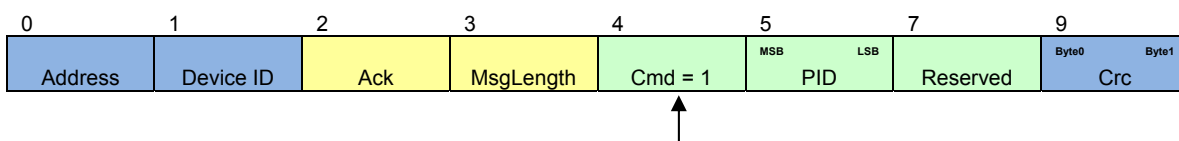
The command determines whether the data transmitted are read or write requests.

## Command (Cmd)

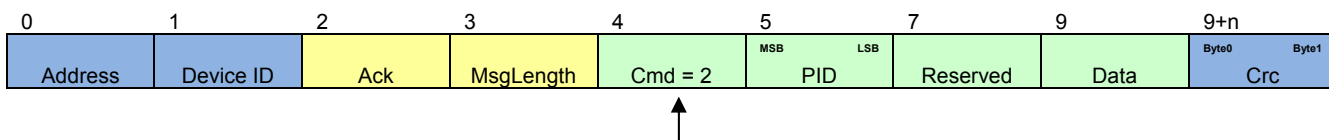
There are four different command types:

- Cmd 1 Read request from the Master
- Cmd 2 Read response from the gauge to a read request
- Cmd 3 Write request
- Cmd 4 Write response from the gauge to a write request

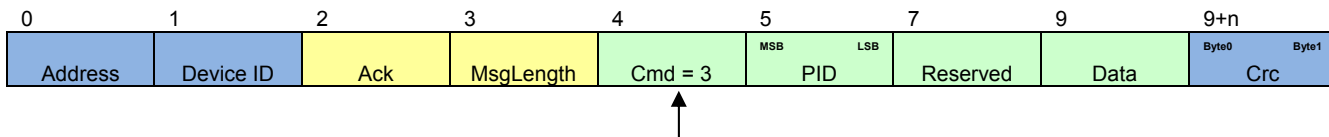
- Cmd 1: Read request from the Master



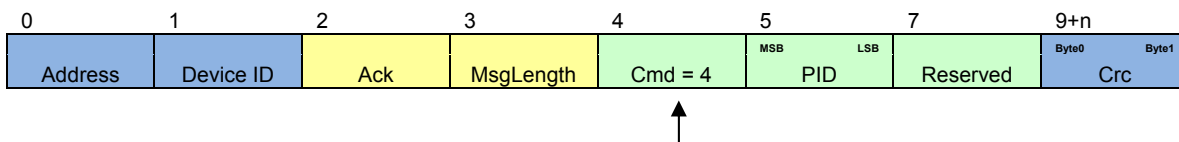
- Cmd 2: Response from the gauge to a read request



- Cmd 3: Write request from a Master



- Cmd 4: Response from the gauge to a write request



## Parameter Identifier (PID)

The parameter identifier (PID) addresses a defined parameter of the device.

## Data field (Data)

The data field contains the data of a request. In the case of a write request it contains the data transmitted from the Master to the gauge, in the case of a read request it contains the data to be transmitted from the gauge to the Master. Data are transmitted in Big Endian.

## Medium Access Layer

The medium access layer contains the following data fields:

- Address for RS232 is 0
- Device ID (Master 0, MPG50x 4, MAG50x 20)
- Header with Ack and Message Length (all APDU data are counted: Cmd, PID, Reserved, Data)
- 16 bit CRC (example → 7)

## Communication Error

If a communication error occurs during transmission the PID is set to 0xFFFF and an error byte is added.

Byte 0, errors

Error Code	Description
1	Access error
2	Value higher max. or lower min.
3	Parameter not found
4	Length error
6	Memory access error
7	Memory access timeout

## Protocol Description

The following parameters can be read via a serial interface.

### Output pressure

PID	Name	Description	Factory setting	Min.	Max.	Type
221	Pressure	Output pressure in integer format (mbar)				LogFixs32en26
222	Pressure	Pressure in vacuum chamber in Real format (in the data unit selected with PID 224)				Real32
224	Data Unit	Changes the pressure unit of all Real32 pressures. 0 mbar 1 Torr 2 Pascal 3 micron 4 Counts	0	0	4	UInt8

### Error

PID	Name	Description	Factory setting	Min.	Max.	Type
228	Device Exception	0 No error 1 Timeout EEPROM memory access 2 EEPROM CRC error 4 EEPROM error 8 Pirani filament rupture 2048 CCIG short circuit	0			UInt32

## General information

PID	Name	Description	Factory setting	Min.	Max.	Type
103	Reset	0 Write The sensor carries out a reset. 1 Write The sensor resets all parameters to their factory settings.		0	1	UInt8
104	Run Hours	Operating hours [0.25 h]				UInt32
207	Serial Number	Serial number			4294967295	UInt32
208	Product Name	Product name	MAG500			String
209	Manufacturers Name	Name of manufacturer	INFICON AG			String
210	Manufacturers Model Number	Model number				String
218	Software Version	Software version				String
227	RS232 Baud Rate	9600, 19200, 38400, 57600	57600	9600	57600	UInt32

## Sensor details

PID	Name	Description	Factory setting	Min.	Max.	Type
223	Active Instance Number	Current sensor 1 CCIG sensor active 2 Pirani sensor active 3 Mixed signal range Pirani and CCIG sensor active				UInt8
33000	Pirani Full Scale*	Pirani fullscale	1000	1e-5	2047	LogFixs32en26
33001	Pirani Overrange Value*	Pirani overrange status output limit	1000	100	1500	LogFixs32en26
255	Pirani Safe State*	Pressure reading in mbar that is output in the event of an error in Pirani mode 0 0 mbar 1 1000 mbar 2 The last valid value is retained. 3 The Pirani Safe State value is output.	0	0	3	UInt8
256	Pirani Safe State Value*	Pressure reading in mbar that is output in the event of an error if the Pirani Safe State is set to 3.	1e-11	0	1000	LogFixs32en26
417	Pirani Adjust Flag*	Executes a manual Pirani sensor adjustment. Set the value to "1" in order for the adjustment to be executed.	0	0	1	UInt8
504	CCIG Safe State	Pressure reading in mbar that is output in the event of an error in CCIG mode. 0 0 mbar 1 1e-2 mbar 2 The last valid value is retained. 3 The CCIG Safe State value is output.	0	0	3	UInt8
505	CCIG Safe State Value	Pressure reading in mbar that is output in the event of an error if the CCIG Safe State is set to 3.	1e-11	0	1e-1	LogFixs32en26

(continued)

(concluded)

PID	Name	Description	Factory setting	Min.	Max.	Type
503	CCIG Full Scale	CCIG full scale	1E-2	1e-11	1e-1	LogFixs32en26
506	CCIG Overrange Value	If the pressure exceeds this value the measurement is invalid.	1E-2	1e-11	5e-2	LogFixs32en26
507	CCIG Underrange Value	If the pressure is below this value the measurement is invalid.	5E-9	1e-11	1e-1	LogFixs32en26

\*Parameter not available for MAG50x

## Data Format

LogFixs32enXX

This is the logarithmic pressure value in mbar which has to be multiplied or divided by a factor of  $2^{XX}$ , in order to reach the result.

Example LOGFixs32en26

The Pirani safe state value should be adjusted to 10 mbar. The value is logarithmized and multiplied with  $2^{26}$ . The value 67108864 is then transmitted.

## Examples

Read request (reading a pressure)

The following examples show how read and write requests are made.

The required parameter has PID 221.

From Master to MxG50x:

0	1	2	3	4	5	7	9	9+n
Address	Device ID	Ack	MsgLength	Cmd	MSB PID LSB	Reserved	Data	Byte0 Crc Byte1
0x00	0x00	0x00	0x05	0x01	0x00DD	0x0000	-	0xAB21

From MxG50x to Master:

0	1	2	3	4	5	7	9	9+n
Address	Device ID	Ack	MsgLength	Cmd	MSB PID LSB	Reserved	Data	Byte0 Crc Byte1
0x00	0x04	0x01	0x09	0x02	0x00DD	0x0000	0x375A05BF	0xD9BB

Write request (setting a unit)

The required parameter has PID 224. To set the unit to Torr, 1 must be written.

From Master to MxG50x:

0	1	2	3	4	5	7	9	9+n
Address	Device ID	Ack	MsgLength	Cmd	MSB PID LSB	Reserved	Data	Byte0 Crc Byte1
0x00	0x00	0x00	0x06	0x03	0x00E0	0x0000	0x01	0x346D

From MxG50x to Master:

0	1	2	3	4	5	7	9	9+n
Address	Device ID	Ack	MsgLength	Cmd	MSB PID LSB	Reserved	Data	Byte0 Crc Byte1
0x00	0x04	0x01	0x05	0x04	0x00E0	0x0000	-	0x94EA

## Calculating CRC

The data packages are secured with CRC16 in Little Endian with the byte order: byte[0], byte[1].

CRC polynomial            0x1021  
CRC initial value            0xFFFF

## CRC Example Code

To illustrate how the checksum can be calculated and verified, an example code in C# is given below:

```
using System;

class Program
{
    static void Main()
    {
        // the following test array is a valid protocol frame with crc16 at the end
        byte[] arr = new byte[] { 0x00, 0x00, 0x00, 0x05, 0x01, 0x00, 0xDD, 0x00, 0x00, 0xAB, 0x21};
        UInt16 crc;
        Boolean b;
        // Calculate the crc of the test array arr, of course without crc (therefore length minus 2)
        crc = Crc16.Create(arr, (Byte)(arr.Length - 2));
        // Check, if the test array has a correct crc at the end (it is correct, therefore the returnvalue is
true)
        b = Crc16.Check(arr, (Byte)arr.Length);
    }
}

public class Crc16
{
    // initial value for crc16
    public static UInt16 inital = 0xFFFF;

    // function to create a crc16
    public static UInt16 Create(Byte[] buffer, Byte length)
    {
        UInt16 crc16 = new UInt16();
        UInt16 i = 0;

        // Initial Value for CRC calculation
        crc16 = Crc16.inital;

        while (i < length)
        {
            crc16 = (UInt16)((Crc16.crc16Tab[(crc16 ^ buffer[i++]) & (Byte)0xFF]) ^ (crc16 >> 8));
        }
        return crc16;
    }

    // function to check a buffer with a crc16 at the end
    public static Boolean Check(Byte[] buffer, Byte length)
    {
        UInt16 crc16 = Crc16.inital;
        UInt16 i = 0;
        // calculate crc for the buffer without crc
        while (i < length)
        {
            crc16 = (UInt16)((Crc16.crc16Tab[(crc16 ^ buffer[i++]) & (Byte)0xFF]) ^ (crc16 >> 8));
        }
        if (crc16 == 0)
        {
            return true;
        }
        return false;
    }
}

// crc array
public static UInt16[] crc16Tab =
{
    0x0000, 0x1189, 0x2312, 0x329B, 0x4624, 0x57AD, 0x6536, 0x74BF,
    0x8C48, 0x9DC1, 0xAF5A, 0xBED3, 0xCA6C, 0xDBE5, 0xE97E, 0xF8F7,
    0x1081, 0x0108, 0x3393, 0x221A, 0x56A5, 0x472C, 0x75B7, 0x643E,
    0x9CC9, 0x8D40, 0xBFDB, 0xAE52, 0xDAED, 0xCB64, 0xF9FF, 0xE876,
    0x2102, 0x308B, 0x0210, 0x1399, 0x6726, 0x76AF, 0x4434, 0x55BD,
    0xAD4A, 0xBCC3, 0x8E58, 0x9FD1, 0xEB6E, 0xFAE7, 0xC87C, 0xD9F5,
    0x3183, 0x200A, 0x1291, 0x0318, 0x77A7, 0x662E, 0x54B5, 0x453C,
    0xBDCB, 0xAC42, 0x9ED9, 0x8F50, 0xFBEE, 0xEA66, 0xD8FD, 0xC974,
    0x4204, 0x538D, 0x6116, 0x709F, 0x0420, 0x15A9, 0x2732, 0x36BB,
    0xCE4C, 0xDFC5, 0xED5E, 0xFCD7, 0x8868, 0x99E1, 0xAB7A, 0xBAF3,
    0x5285, 0x430C, 0x7197, 0x601E, 0x14A1, 0x0528, 0x37B3, 0x263A,
    0xDECD, 0xCF44, 0xFDDF, 0xEC56, 0x98E9, 0x8960, 0xBBFE, 0xAA72,

```

```

0x6306, 0x728F, 0x4014, 0x519D, 0x2522, 0x34AB, 0x0630, 0x17B9,
0xEF4E, 0xFEC7, 0xCC5C, 0xDDD5, 0xA96A, 0xB8E3, 0x8A78, 0x9BF1,
0x7387, 0x620E, 0x5095, 0x411C, 0x35A3, 0x242A, 0x16B1, 0x0738,
0xFFCF, 0xEE46, 0xDCDD, 0xCD54, 0xB9EB, 0xA862, 0x9AF9, 0x8B70,
0x8408, 0x9581, 0xA71A, 0xB693, 0xC22C, 0xD3A5, 0xE13E, 0xF0B7,
0x0840, 0x19C9, 0x2B52, 0x3ADB, 0x4E64, 0x5FED, 0x6D76, 0x7CFF,
0x9489, 0x8500, 0xB79B, 0xA612, 0xD2AD, 0xC324, 0xF1BF, 0xE036,
0x18C1, 0x0948, 0x3BD3, 0x2A5A, 0x5EE5, 0x4F6C, 0x7DF7, 0x6C7E,
0xA50A, 0xB483, 0x8618, 0x9791, 0xE32E, 0xF2A7, 0xC03C, 0xD1B5,
0x2942, 0x38CB, 0x0A50, 0x1BD9, 0x6F66, 0x7EEF, 0x4C74, 0x5DFD,
0xB58B, 0xA402, 0x9699, 0x8710, 0xF3AF, 0xE226, 0xD0BD, 0xC134,
0x39C3, 0x284A, 0x1AD1, 0x0B58, 0x7FE7, 0x6E6E, 0x5CF5, 0x4D7C,
0xC60C, 0xD785, 0xE51E, 0xF497, 0x8028, 0x91A1, 0xA33A, 0xB2B3,
0x4A44, 0x5BCD, 0x6956, 0x78DF, 0x0C60, 0x1DE9, 0x2F72, 0x3EFB,
0xD68D, 0xC704, 0xF59F, 0xE416, 0x90A9, 0x8120, 0xB3BB, 0xA232,
0x5AC5, 0x4B4C, 0x79D7, 0x685E, 0x1CE1, 0x0D68, 0x3FF3, 0x2E7A,
0xE70E, 0xF687, 0xC41C, 0xD595, 0xA12A, 0xB0A3, 0x8238, 0x93B1,
0x6B46, 0x7ACF, 0x4854, 0x59DD, 0x2D62, 0x3CEB, 0x0E70, 0x1FF9,
0xF78F, 0xE606, 0xD49D, 0xC514, 0xB1AB, 0xA022, 0x92B9, 0x8330,
0x7BC7, 0x6A4E, 0x58D5, 0x495C, 0x3DE3, 0x2C6A, 0x1EF1, 0x0F78

```

```
};
```

```
}
```



Notes

Original: English



tira83e1



*LI-9496 Balzers  
Liechtenstein*

*Tel +423 / 388 3111*

*Fax +423 / 388 3700*

*reachus@inficon.com*

*www.inficon.com*