123 rue de Bellevue, 92100 Boulogne Billancourt
France
Tél : 33-(0) 1 41 41 00 02 - www.nano-sense.com

E5000 Modbus Firmware Update

Ver	Date	Update
V1	06/01/2021	Initial version
V2	18/02/2021	Command 0xED removed, CRC command changed
V3	05/03/2021	Command 0x7F modified
V4	06/04/2021	Reset command 0xF5 added
V5	09/04/2021	Command 0x5E to select the firmware area depending on device target (MCU, …)
V6	07/07/2021	Correction of the 0x5E command description
V7	07/10/2021	Minimum timing added
V8	21/12/2021	Adding MOX module firmware update commands

Sommaire

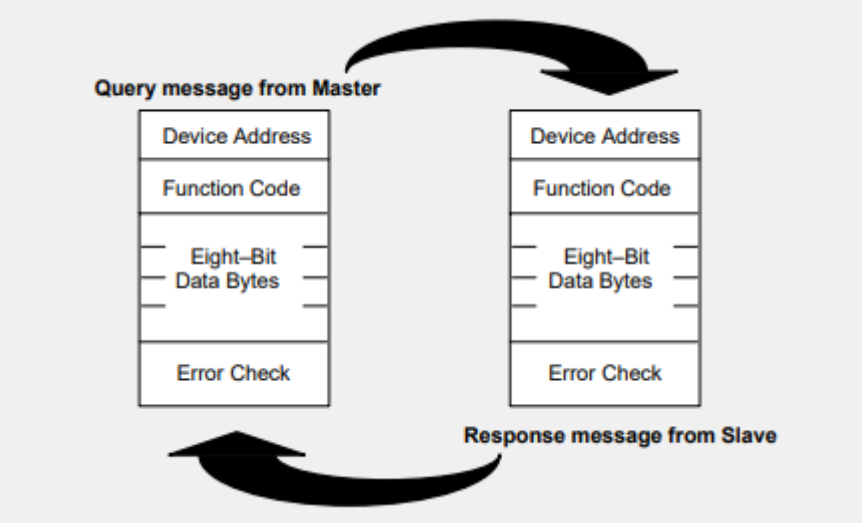
EP5000 Modbus Firmware Update Protocol

The Modbus protocol allows a master unit to access up to 255 slave units connected on a single bus. Each slave is assigned an address that distinguishes each from other slaves connected to the bus.

The bus address is settable by NFC. By default, when there is no NFC, the address is 1.

Transactions can be only initiated by the master and are of two types:
- Question / answer One slave is addressed
- Broadcast / no answer All slaves are addressed, but they shall not reply

The query / response cycle:



Function codes used for firmware update

CODE	MEANING	ACTION
04	READ INPUT REGISTER	Obtains current binary value in one or more input registers.
16	PRESET MULTIPLE REGISTERS	Place specific binary values into a group of holding registers.
18	PROGRAM	Program

Working principle

The firmware **binary** file is store in the master.

First of all, the master has to select the destination flash area.

The master send command to erase the slave firmware area storage flash and wait at least 15 seconds (erasing time is 12s).

The master verifies if the erasing has ended without error by sending the command 0xEC

The master sends packets containing maximum 150 bytes of the firmware file to the slave.

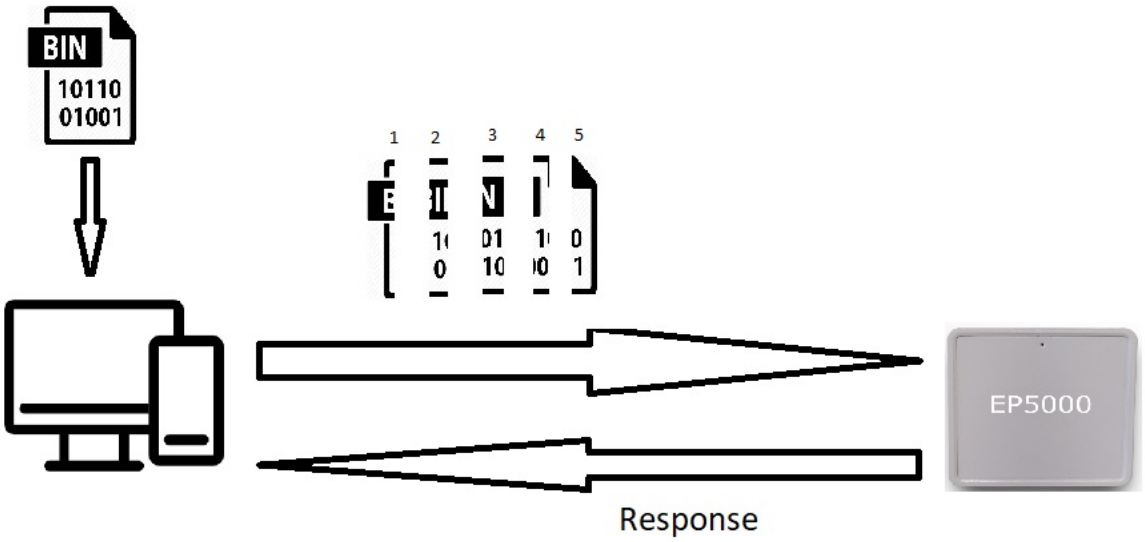
The minimum delay for waiting response from the slave is 500ms.

The master sends the final global CRC value. If the CRC is not sent, it will not be stored in the slave storing flash. In this case, slave's MCU flash will not be authorized to be updated.

When ended, the master sends the end programming command and wait a few seconds while the slave calculate its own CRC.

The master can request the slave calculated CRC to verify. If wrong, it may ask the slave to erase its flash.

The master sends an update command to perform a restart and update the MCU flash.



Communication timing

Firmware under 5.9

Master request REQUEST

Slave answer RESPONSE

← 600ms minimum →

Firmware from 5.9

Master request REQUEST

Slave answer RESPONSE

← 60ms minimum →

See EP5000 ModBus protocol V27 and above for details.

Function 18 (0x12) Program

This function allows to perform the firmware binary file upload to the slave.

Request from the master:

Program: Function = 18

FUNCTION CODE (0x12)	CMD CODE	DATA
8-BIT	8-BIT	N*8-BIT

Response from the slave:

If no errors:

FUNCTION CODE (0x12)	RESPONSE
8-BIT	N*8-BIT

In case of error:

ERROR CODE (0x92)	EXCEPTION CODE
8-BIT	8-BIT

Command codes:

The commands are detailed in logic execution order

Command codes	Description
0x5E	Select the external flash area to be programmed (depending on the device to be updated)
0xEA	ErAse flash. This is the first step before sending data to be programmed. It takes at least 15 seconds while the main MCU will not process any other request.
0xEC	Request the Erase Confirmation (wait more than 15 seconds after having sent the 0xEA command to send this command)
0xDA	Send DAta to be programmed in the flash
~~0xED~~	~~EnD of file transfer~~
0xCC	Send the final CRC
0x7F	Request the last received Frame number received by the slave
0xCF	Request final CRC calculated by the slave (available only after an End of file transfer).
0x5A	Restart and perform the last downloaded firmware programming
0x5F	Restart and perform a factory firmware programming
0xAF	Copy the current firmware as the new factory firmware in the flash
0x70	Sensor action (programming updated firmware …)
0xF5	Reset command

In case the slave returns an error, see exception codes.

Command 0x5E: Select the flash area to be updated

From master

FUNCTION CODE (0x12)	CMD CODE (0x5E)	FLASH AREA
8-BIT	8-BIT	8-BIT

This command selects the firmware that is to be updated.
This command has to be sent before any other commands

0 → Mainboard MCU firmware
1 → MOX sensor module firmware 1
2 → MOX sensor module firmware 2
3 → MOX sensor module firmware 3
4 → reserved
5 → reserved
6 → reserved
7 → reserved

Response from slave

FUNCTION CODE (0x12)	CMD CODE (0x5E)	FLASH AREA
8-BIT	8-BIT	8-BIT

Command 0xEA: Erase Flash selected area

From master

FUNCTION CODE (0x12)	CMD CODE (0xEA)
8-BIT	8-BIT

Response from slave

FUNCTION CODE (0x12)	CMD CODE (0xEA)
8-BIT	8-BIT

Command 0xEC: Erase confirmation

From master

FUNCTION CODE (0x12)	CMD CODE (0xEC)
8-BIT	8-BIT

Response from slave

FUNCTION CODE (0x12)	CMD CODE (0xEC)	STATUS
8-BIT	8-BIT	8-BIT

Status:

0x00 : Erase ended with an error
0x01 : Erase ended successfully

Command 0xDA: program data in selected area

From master

FUNCTION CODE (0x12)	CMD CODE (0xDA)	PACKET NUMBER	NUMBER OF BYTES	DATA
8-BIT	8-BIT	16-BIT	8-BIT	N * 8-BIT

Response from slave

FUNCTION CODE (0x12)	CMD CODE (0xDA)	PAQUET NUMBER	NUMBER OF BYTES
8-BIT	8-BIT	16-BIT	8-BIT

First packet number is 0x0000.

If the packet number is not the previous one +1, the slave will return an exception code '2' (illegal data address)
If the number of bytes is greater than 150, the slave will return an exception code '3' (illegal data value)

Command 0x7F: last correct packet number received

From master

FUNCTION CODE (0x12)	CMD CODE (0x7F)
8-BIT	8-BIT

Response from slave

FUNCTION CODE (0x12)	CMD CODE (0x7F)	LASTE PACKET NUMBER	DATA SIZE RECEIVED
8-BIT	8-BIT	16-BIT	32-BIT

~~**Command 0xED: End of transfer**~~

~~From master~~

~~FUNCTION CODE (0x12)~~	~~CMD CODE (0xED)~~
~~8-BIT~~	~~8-BIT~~

~~Response from slave~~

~~FUNCTION CODE~~	~~CMD CODE~~

~~(0x12)~~	~~(0xED)~~
~~8 BIT~~	~~8 BIT~~

~~When this command is received the slave calculate its flash CRC.~~

Command 0xCC: send final CRC

From master

FUNCTION CODE (0x12)	CMD CODE (0xCC)	FIRMWARE CRC VALUE
8-BIT	8-BIT	32-BIT

Response from slave

FUNCTION CODE (0x12)	CMD CODE (0xCC)
8-BIT	8-BIT

The CRC transmitted by the master is a CRC32.
When the EP5000 receive this command, it will calculate its own CRC from the external flash.

Command 0xCF: read final CRC of selected area

From master

FUNCTION CODE (0x12)	CMD CODE (0xCF)
8-BIT	8-BIT

Response from slave

FUNCTION CODE (0x12)	CMD CODE (0xCF)	CRC STATE	FLASH CRC
8-BIT	8-BIT	8-BIT	32-BIT

The slave will return the CRC32 calculated from the flash where the uploaded firmware is stored.
It is done only after a upload.
Please wait a few seconds after the end of a firmware upload. While calculating the CRC, the slave will not respond.

CRC state:
0 : CRC calculation is not ended
1 : CRC calculation is ended

Command 0x5A: program MCU flash with last uploaded firmware

From master

FUNCTION CODE (0x12)	CMD CODE (0x5A)
8-BIT	8-BIT

Response from slave

FUNCTION CODE (0x12)	CMD CODE (0x5A)
8-BIT	8-BIT

After receiving this command, the slave restart and update its MCU flash. While processing, it will not perform any other action.

Command 0x5F: program MCU flash with the factory firmware

From master

FUNCTION CODE (0x12)	CMD CODE (0x5F)
8-BIT	8-BIT

Response from slave

FUNCTION CODE (0x12)	CMD CODE (0x5F)
8-BIT	8-BIT

After receiving this command, the slave restart and update its MCU flash. While processing, it will not perform any other action.

Command 0xAF: update the factory firmware stored in flash with the current one

From master

FUNCTION CODE (0x12)	CMD CODE (0xAF)
8-BIT	8-BIT

Response from slave

FUNCTION CODE (0x12)	CMD CODE (0xAF)
8-BIT	8-BIT

After receiving this command, the slave restart and update its MCU flash. While processing, it will not perform any other action.

Command 0x60: program sensors flash

From master

FUNCTION CODE (0x12)	CMD CODE (0x60)	SENSOR NUM	ACTION CODE
8-BIT	8-BIT	8-BIT	8-BIT

Sensors flash action:

SENSOR NUM:

0 → MOX module Firmware 1
1 → MOX module Firmware 2
2 → MOX module Firmware 3
3 → reserved
4 → reserved
5 → reserved
6 → reserved

ACTION CODE:

0x44: start the sensor module update (upload the firmware to the sensor and proceed the MCU update)
0x5A: program sensor flash with uploaded firmware
0x5F: program sensor flash with factory firmware
0xAF: update factory firmware with uploaded firmware

Note that for the MOX sensor, the sensor number is important only for command 0x44 as it select the firmware to upload from the main EP5000 flash to the sensor. The sensor only stored one factory firmware and one update firmware, so, for commands 0x5A, 0x5F and 0xAF, sensor numbers 0, 1 and 2 gives the same result.

Response from slave

FUNCTION CODE (0x12)	CMD CODE (0x60)
8-BIT	8-BIT

Command 0xF5: reset probe

From master

FUNCTION CODE (0x12)	CMD CODE (0xF5)
8-BIT	8-BIT

Response from slave

FUNCTION CODE (0x12)	CMD CODE (0xAF)
8-BIT	8-BIT

Exception codes

If the slave has to return an error in response of a query sent by the master. The frame is:

ERROR CODE (0x92)	EXCEPTION CODE
8-BIT	8-BIT

Code	Exception
0x01	ILLEGAL FUNCTION
0x02	ILLEGAL DATA ADDRESS
0x03	ILLEGAL DATA VALUE
0x04	
0x05	
0x06	
0x07	
0x08	
0x0A	
0x0B	

CRC16 calculation

```
static const unsigned char auchCRCHi[] = {
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01,
0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40
} ;

static const unsigned char auchCRCLo[] = {
0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06, 0x07, 0xC7, 0x05, 0xC5, 0xC4,
0x04, 0xCC, 0x0C, 0x0D, 0xCD, 0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B, 0xC9, 0x09,
0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A, 0x1E, 0xDE, 0xDF, 0x1F, 0xDD,
0x1D, 0x1C, 0xDC, 0x14, 0xD4, 0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12, 0x13, 0xD3,
0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3, 0xF2, 0x32, 0x36, 0xF6, 0xF7,
0x37, 0xF5, 0x35, 0x34, 0xF4, 0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A,
0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38, 0x28, 0xE8, 0xE9, 0x29, 0xEB, 0x2B, 0x2A, 0xEA, 0xEE,
0x2E, 0x2F, 0xEF, 0x2D, 0xED, 0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26,
0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60, 0x61, 0xA1, 0x63, 0xA3, 0xA2,
0x62, 0x66, 0xA6, 0xA7, 0x67, 0xA5, 0x65, 0x64, 0xA4, 0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F,
0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68, 0x78, 0xB8, 0xB9, 0x79, 0xBB,
0x7B, 0x7A, 0xBA, 0xBE, 0x7E, 0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74, 0x75, 0xB5,
0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71, 0x70, 0xB0, 0x50, 0x90, 0x91,
0x51, 0x93, 0x53, 0x52, 0x92, 0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54, 0x9C, 0x5C,
0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B, 0x99, 0x59, 0x58, 0x98, 0x88,
0x48, 0x49, 0x89, 0x4B, 0x8B, 0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C,
0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42, 0x43, 0x83, 0x41, 0x81, 0x80,
0x40
} ;


unsigned short crc16(unsigned char *puchMsg , unsigned short usDataLen)
{
  unsigned char uchCRCHi = 0xFF ; /* high byte of CRC initialized */
  unsigned char uchCRCLo = 0xFF ; /* low byte of CRC initialized */
  unsigned uIndex ; /* will index into CRC lookup table */

  unsigned short usVal1;
  unsigned short usVal2;


  while (usDataLen--)/* pass through message buffer */
  {
```



```
    uIndex = uchCRCHi ^ *puchMsg; //++ ; /* calculate the CRC */
    puchMsg++ ;
    uchCRCHi = uchCRCLo ^ auchCRCHi[uIndex];
    uchCRCLo = auchCRCLo[uIndex] ;
   }

  usVal1 = uchCRCHi;
  usVal2 = uchCRCLo;

  usVal1 = usVal1 << 8;
  usVal1 = usVal1 | usVal2;

  return usVal1; //( ((unsigned short)uchCRCHi << 8) | (unsigned short)uchCRCLo) ;
 }
```

CRC32 calculation

Exemple of CRC32 calculation

```
long crc32; /* 32 bit data */
#define POLYNOME_CRC  0x4C11DB7



void calculCrc32 ( unsigned long Value )
{
	unsigned char index;
	unsigned long temp;
	tLongWord LongWord;
	unsigned char buffer [ 4 ];

	LongWord.LongWord = Value;

	buffer [ 0 ] = LongWord.Byte.High;
	buffer [ 1 ] = LongWord.Byte.MiddleHigh;
	buffer [ 2 ] = LongWord.Byte.MiddleLow;
	buffer [ 3 ] = LongWord.Byte.Low;

	for (index = 0; index < 4; index++)
	{
		temp = ((crc32 ^ buffer[index]) & 0xff);


		// read 8 bits one at a time
		for (int i = 0; i < 8; i++)
		{
			if ((temp & 1) == 1)
			{
				temp = (temp >> 1) ^ POLYNOME_CRC;
			}
			else
			{
				temp = (temp >> 1);
			}

		}

		crc32 = (crc32 >> 8) ^ temp;
	}
}



unsigned long ComputeFlashCrc ( unsigned long Adresse, unsigned long Size )
{
	unsigned long FlashCrc;
	unsigned short CptPage;
	unsigned long SizeComputeCrc;
	unsigned short IndexDataPage;
```



```
	CptPage = 0;
	crc32 = 0xFFFFFFFF;

	while ( Size > 0 )
	{

		FlashSpi_Read ( Adresse + (CptPage * FLASH_SPI_PAGE_SIZE),
				FLASH_SPI_PAGE_SIZE, (unsigned char *) McuMemoryBuffer );


		/* Calcul de la taille des données à utiliser pour le calcul du CRC */
		if ( Size > FLASH_SPI_PAGE_SIZE )
		{
			SizeComputeCrc = FLASH_SPI_PAGE_SIZE;
		}
		else
		{
			SizeComputeCrc = Size;
		}

		/* Mise à jour de la taille de données restante */
		if ( Size >= SizeComputeCrc )
		{
			Size -= SizeComputeCrc;
		}
		else
		{
			Size = 0;
		}

		/* Calcul du CRC */
		for ( IndexDataPage = 0; IndexDataPage < (SizeComputeCrc / 4 ); IndexDataPage++)
		{
			calculCrc32 (  McuMemoryBuffer [ IndexDataPage ]);
		}


		CptPage++;

	}


	crc32 = crc32 ^ 0xffffffff;


	return crc32;
}
```

