

GNetPlus[®] Communication Protocol

Basic Package (BINARY VERSION)

Master Query Package (HOST)

Field	Header	Address	Query Function	Data length	DATA BYTES	Error Check	
Desc	SOH	0~255	0~255	0~255		CRC16_Low	CRC16_Hi
Size	1 BYTE	1 BYTE	1 BYTE	1 BYTE	0~255 BYTES	1 BYTE	1 BYTE

Note:

SOH = 01h.

Address = Device Address (Slave Machine ID)

Slave Response Package (DEVICE)

Field	Header	Address	Response Function	Data length	DATA BYTES	Error Check	
Desc	SOH	0~255	ACK / NAK / EVN	0~255		CRC16_Low	CRC16_Hi
Size	1 BYTE	1 BYTE	1 BYTE	1 BYTE	0~255 BYTES	1 BYTE	1 BYTE

Note:

SOH = 01h.

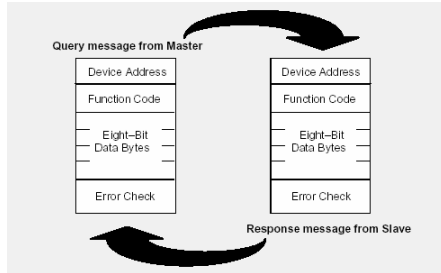
Address = Device Address (Slave Machine ID)

ACK = 06h, Acknowledge (Passive, in response to Master message)

NAK = 15h, Negative Acknowledge (Passive, in response to Master message)

EVN = 12h, Event Message (Active, For One Host to One Device Connection)

The Query - Response Cycle (Passive)



The Query

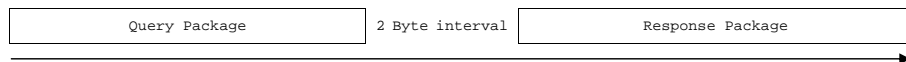
The function code in the query tells the addressed slave device what kind of action to perform. The data bytes contain all additional information that the slave will need to perform the function. The error check field provides a method for the slave to validate the integrity of the message contents.

The Response

If the slave makes a normal response, the function code in the response is an ACK of the function code in the query. The data bytes contain the data collected by the slave, such as register values or status. If an error occurs, the function code is NAK to indicate that the response is an error response, and the data bytes contain a code that describes the error. The error check field allows the master to confirm that the message contents are valid.

Avoiding Collision And Synchronizing (optional)

1. Send Package: Must have a 2 byte interval (by baudrate) after end of receiving.
2. Receive Package: If package is cut off (unfinished) and exceeds the interval, receiver can discard and reset the unfinished package buffer.



Byte sequence for data type INTEGER or LONG in "DATA BYTES" field

Example for data type integer 261Bh (2 BYTES)

SOH	Address	Function	Data Length	(MSB)	Data Bytes	(LSB)
			2	26h	1Bh	

Example for data type long 261B3C27h (4 BYTES)

SOH	Address	Function	Data Length	(MSB)	Data Bytes	(LSB)
			4	26h	1Bh	3Ch 27h

Example for data structure:

Structure _example

Byte bCode = 2Ch;

Int nValue = 261Bh;

Long lValue = 261B3C27h;

SOH	Address	Function	Data Length	(MSB)	Data Bytes			(LSB)		
				bCode	nValue	lValue				
			7	2Ch	26h	1Bh	26h	1Bh	3Ch	27h

Generating a CRC

Step 1 Load a 16-bit register with FFFF hex (all 1's). Call this the CRC register.

Step 2 Exclusive OR the first eight-bit byte of the message with the low order byte of the 16-bit CRC register, putting the result in the CRC register.

Step 3 Shift the CRC register one bit to the right (toward the LSB), zero-filling the MSB. Extract and examine the LSB.

Step 4 If the LSB is 0, repeat Step 3 (another shift). If the LSB is 1, Exclusive OR the CRC register with the polynomial value A001 hex (1010 0000 0000 0001).

Step 5 Repeat Steps 3 and 4 until eight shifts have been performed. When this is done, a complete eight-bit byte will have been processed.

Step 6 Repeat Steps 2 ... 5 for the next eight-bit byte of the message. Continue doing this until all bytes have been processed.

Result The final contents of the CRC register is the CRC value.

Step 7 When the CRC is placed into the message, its upper and lower bytes must be swapped as described below.

Note: *GNetPlus CRC16 is for fields from "Address" to "Data Bytes".*

Placing the CRC into the communication package

When the 16-bit CRC (two eight-bit bytes) is transmitted in the message, the low order byte will be transmitted first, followed by the high order byte:

CRC16 Generation Function (C Code)

```
/* Table of CRC values for high-order byte */
static 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, 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
```

```
};
```

```
/* Table of CRC values for low-order byte */
```

```
static 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
```

```
};
```

```
/* GNetPlus CRC16 Generation Function */
```

```
unsigned short GNetPlus_CRC16(puchMsg, usDataLen)
```

```
unsigned char *puchMsg ; /* message to calculate CRC upon */
```

```
unsigned short usDataLen ; /* quantity of bytes in message */
```

```
{
```

```
    unsigned char uchCRCHi = 0xFF ; /* high CRC byte initialized */
```

```
    unsigned char uchCRCLo = 0xFF ; /* low CRC byte initialized */
```

```

unsigned uIndex ;                               /* will index into CRC lookup*/
/* table */
while (usDataLen-- )                            /* pass through message buffer */
{
    uIndex = uchCRChi ^ *puchMsgg++ ; /* calculate the CRC */
    uchCRChi = uchCRCLo ^ auchCRChi[uIndex] ;
    uchCRCLo = auchCRCLo[uIndex] ;
}
return (uchCRChi << 8 | uchCRCLo) ;
}

```

Common Query Function Code Table (00h~1Fh)

Desc	Query (Master)			Response (Slave)		
	Func	Len	Data Bytes	Func	Len	Data Bytes
Polling	00h	0		ACK	n	Return OEM Status
Get Version	01h	0		ACK	n	Return OEM Version String
Set Slave Addr	02h	1	New Address (1~255)	ACK	0	
Logon	03h	n	OEM Password	ACK	0	
Logoff	04h	0		ACK	0	
Set Password	05h	n	OEM New Password	ACK	0	
Reserve	06h					
Set Date/Time	07h	7	GNET_DATETIME Structure ¹	ACK	0	
Get Date/Time	08h	0		ACK	7	GNET_DATETIME Structure
Get Register	09h	3	Reg.Address ² + Reg.Len ³	ACK	n	Reg.Block
Set Register	0Ah	n	Reg.Address + Reg.Buffer	ACK	0	
Record Count	0Bh	0		ACK	2	Record Count (Integer)
Get First Record	0Ch	0		ACK	n	First Record
Get Next Record	0Dh	0		ACK	n	Next Record
Erase All Records	0Eh	0		ACK	0	
Add Record	0Fh	n	Record	ACK	0	
Recover All Records	10h	0		ACK	0	
DO	11h	2	DO# + STATUS(0 or 1)	ACK	0	
DI	12h	1	DI#	ACK	1	DI STATUS
Reserve	13h					
Reserve	14h					
Reserve	15h					
Reserve	16h					
Reserve	17h					
Reserve	18h					
Reserve	19h					
Get Time Adjust ^{DEM}	1Ah	0		ACK	4	Time Adjust Value (long)
Echo	1Bh	N	Any Data	ACK	n	Master Data
Set Time Adjust ^{DEM}	1Ch	4	Time Adjust Value (long)	ACK	0	
Debug	1Dh	0		ACK	n	OEM Debug Message
Reset	1Eh	0		ACK	0	
Go To ISP	1Fh	0		ACK	0	

Note:

- GNET_DATETIME Structure (7 BYTES)
 - BYTE Second;
 - BYTE Minute;
 - BYTE Hour;
 - BYTE DayOfWeek;
 - BYTE Day;
 - BYTE Month;
 - BYTE Year;
- Reg.Address: Integer Type(2 Bytes), Range 0000h~FFFFh.
- Reg.Len : Register Block Size (1 Byte), Max.255.

Response NAK Code Table (Common)

Func	Len	Data Bytes	Description
NAK	1	E0h	Access Denied
NAK	1	E4h	Illegal Query Code
NAK	1	E6h	Overrun, Out of record count
NAK	1	E7h	CRC Error
NAK	1	EDh	Out Of Memory Range
NAK	1	EEh	Address Number out of range
NAK	1	EFh	Unknown

Response Event (For Active Slave)

	Active Response (Slave)		
Desc	Func	Len	Data Bytes
Event	12h	n	Customer Event Code or Data