Micus Real Time Software Inc.
5863 Leslie St. Suite 127
Toronto, Ontario
M2H 1J8
Canada
Tel: (416) 493 3623
Fax: (416) 502 9083
www.micus.ca

MACS COMMUNICATION PROTOCOL SPECIFICATION

Revision 4.0

September 16, 2014

Table of Contents

1 Revision History

Revision:	Date:	Description:	Prepared by:
1.0	Dec. 14, 2005	First draft.	M. Bankovitch
2.0	May 14, 2007	Addresses and OIDs extended to 2 bytes. Support for float and double encodings added. Broadcast packets added.	M. Bankovitch
3.0	August 1, 2007	LIST request added.	M. Bankovitch
4.0	September 16, 2014	Support for 64-bit integers added. Unsolicited event reports added. Get Record and Set Record commands added. Operation over cellular network added.	M. Bankovitch

2 Introduction

The latest generation of the embedded Micus Real Time Software Inc. products supports remote monitoring and control via serial communication lines, wired and TCP/IP connections, cellular networks and SMS messages. This document contains the definition of the electrical interfaces and the application layer communication protocol used to monitor and control these products.

Micus Alarm and Control System (MACS) Communication Protocol is an open protocol. Although Micus Real Time Software Inc. maintains the copyright on the protocol definition, there are no restrictions and royalties attached to the protocol use. Other vendors are welcome and encouraged to use the protocol in their products.

The document is structured as follows:

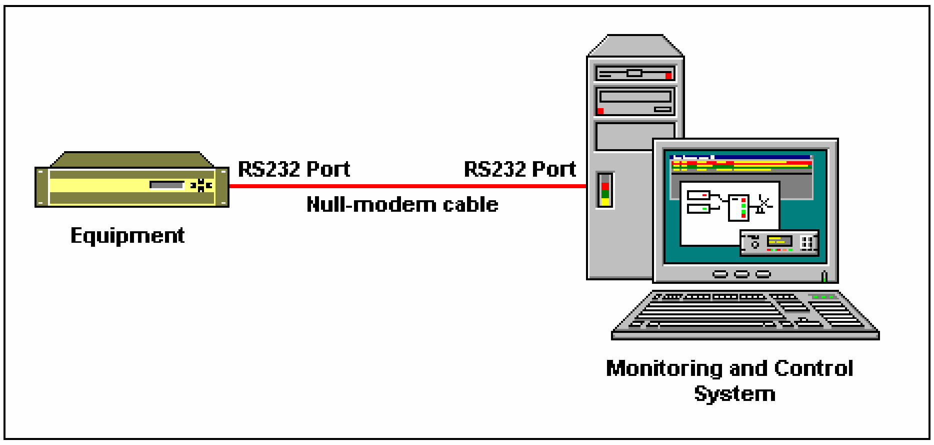
- Section 3 specifies the electrical interfaces and network interfaces used for communication between monitoring and control systems and controllers embedded in the equipment
- Section 4 defines the structure and format of the protocol packets
- Section 5 defines a list of commands used to monitor and control devices attached to the serial lines or network interfaces

The intended audience for this document is:

- System integrators and maintenance personnel
- Software developers

3 Electrical Interfaces

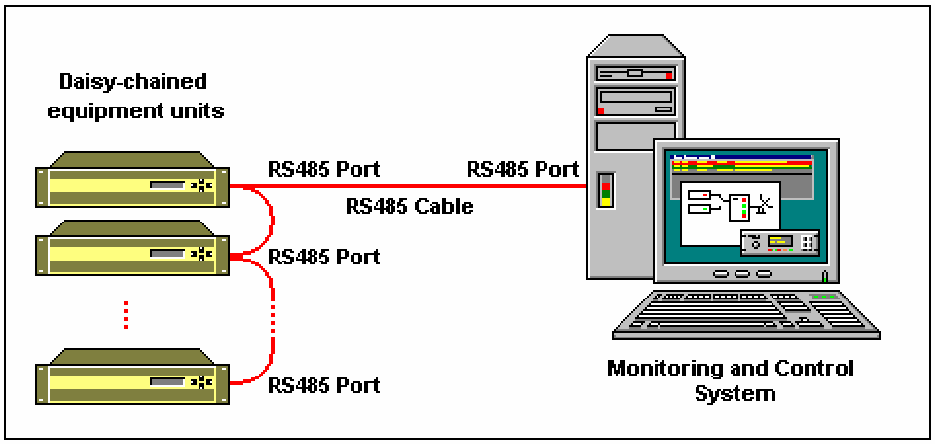
Many Micus Real Time Software Inc. and other manufacturer products that support remote monitoring and control are equipped with RS232 serial interface ports. The RS232 interface allows a single piece of equipment to be connected to the computer via a serial line, as illustrated in the following picture.



Since both the computer and the equipment are DTE devices, a standard RS232 null-modem cable must be used to connect them. If the computer is not equipped with an RS232 port, a USB to serial port adapter can be used.

The RS232 interface also allows connections to the remote sites via telephone lines. In such case the equipment requires an external modem connected to the equipment using a standard RS232 straight cable. In such a scenario it is strongly recommended setting the modem inactivity timeout to 10 to 15 seconds.

If there is a need to interface more than one equipment unit via the same serial line, an optional RS485 port must be installed in each unit and in the computer itself. The RS485 interface allows multiple units to be daisy-chained into a party line. When querying an individual unit, the computer uses unit address, described in the next section of this document. Only the addressed unit replies to the query, while other units on the party line ignore the query. A party line with several daisy-chained units is depicted in the following picture.



MACS communication protocol also supports broadcast packets. When equipment units receive broadcast packets, they process them but do not respond to the originator.

Connecting multiple units into a daisy chain requires special cabling. Wiring details are usually provided with the equipment. Please note that RS485 electrical interface supports up to 32 connected devices.

The equipment usually supports the following serial port settings:

- Standard data rates between 1200 Bd and 115,200 Bd
- Even, odd or no parity
- One start and one stop bit
- This protocol requires eight data bits

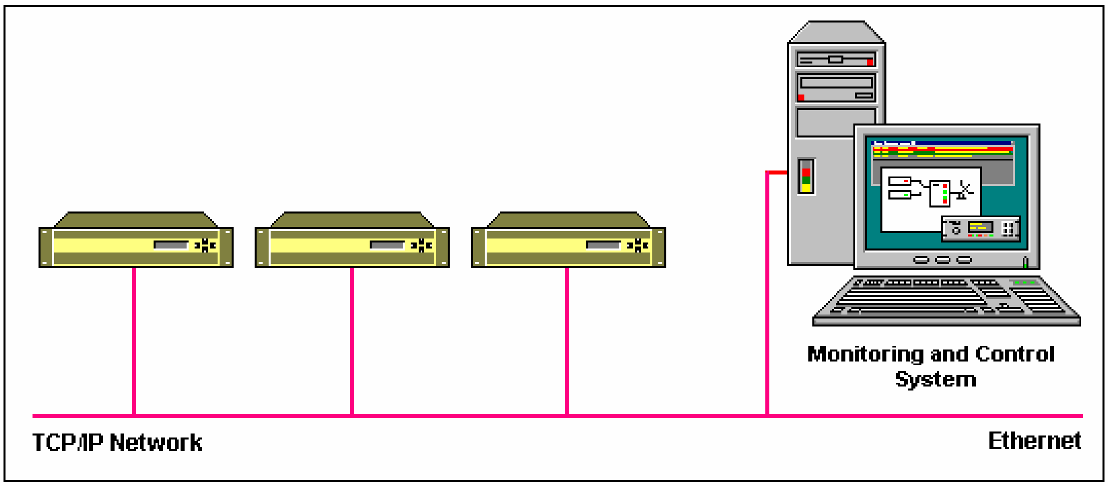
4 Network Interfaces

Many products support TCP/IP connections over a wide area, local area or wireless networks. MACS communication protocol supports TCP/IP network connections. When a TCP or UDP connection is used, MACS protocol packets are simply encapsulated in the TCP or UDP packets.

MACS protocol can be also used over cellular networks. In such case MACS protocol packets may be encapsulated into Short Message Service (SMS) messages.

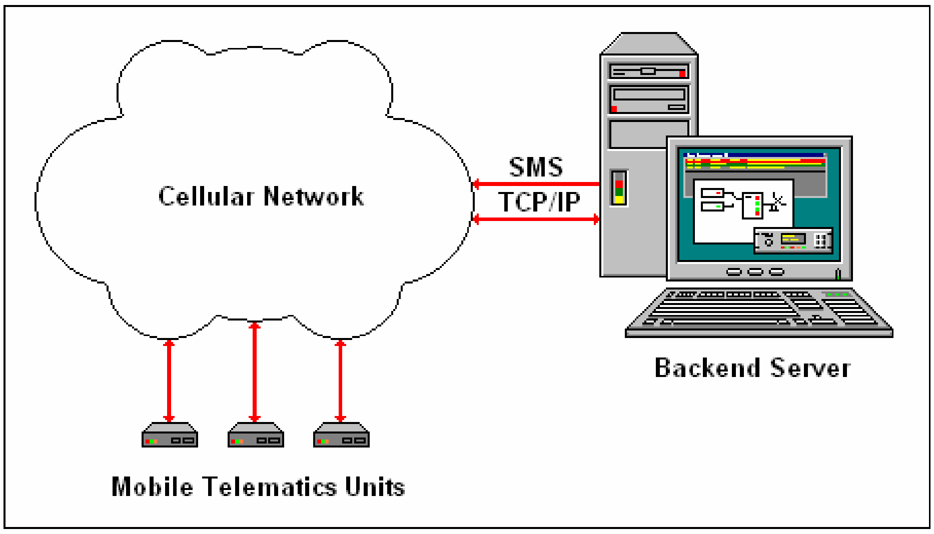
The format and content of MACS protocol packets is exactly the same over serial lines, network connections and SMS messages. While some of the fields, such as source and destination address and checksum, are redundant when using TCP/IP, keeping the same packet structure simplifies the software, and reduces its size.

A typical local area (LAN) network connection is depicted in the following picture:



MACS protocol can be also used over cellular networks, to manage mobile devices, such as automotive telematics units. When communicating over a cellular network, data connection is still established using a TCP protocol. However, only the mobile device can originate a TCP connection request. In such case the backend server sends an SMS message to the mobile unit to request such connection. The SMS messages contain the same MACS protocol packets as any other transport media.

A typical cellular network connection is depicted in the following picture:


Cellular Network
SMS
TCP/IP
Backend Server
Mobile Telematics Units

5 Communication Protocol

This section contains the definition of the communication protocol packets used to exchange information between the computer and the equipment units. The exact content of each packet carrying computer commands and equipment responses is defined in the next section.

Computer queries and equipment responses are encapsulated into packets. A packet consists of a header, a trailer and user data. Each packet may contain up to 1024 bytes of user data. This basic structure is depicted in the following diagram.

Header	User Data	Trailer

The header is used to synchronize packet transmission, and to indicate to the receiving end the size of the user data segment. The header also contains information on which entity is sending the packet, and which entity is the recipient. The header always starts with the *Start of Text (STX)* character (02H). The second and third byte in the header contain the source address, the fourth and fifth byte contain the destination address, and the sixth and seventh byte contain the size of the user data segment.

The trailer contains a checksum used to verify packet integrity, followed by the *End of Text (ETX)* character (03H).

Actual queries and responses are contained in user data segment, defined in the next section of this document.

The overall packet structure can be presented as follows:

STX	Source Address	Destination Address	Size	User Data	Checksum	ETX

5.1 STX and ETX Bytes Substitution

Each packet starts with an STX character and ends with an ETX character. These characters are used by the receiving entity to detect start and end of the packet. Upon receiving a packet, the receiving entity discards STX and ETX bytes and processes all other bytes in between. However, since the protocol is used to transmit binary data, it is entirely possible that addresses, user data and checksum contain STX (02H) and/or ETX (03H) binary values. If an address, user data or the checksum contains such bytes, they must be substituted with *two bytes* of the same value. When the receiving entity detects an STX or ETX byte, it discards it. If such byte is immediately followed by another byte of the same value, the receiving entity assumes that the second byte is part of the address, user data or checksum.

5.2 Packet Header

Each packet header consists of seven bytes, excluding STX substitution. The first byte is an STX byte, which marks the beginning of the packet. The second and third byte contain the source address, which uniquely identifies the entity sending the packet. The fourth and fifth byte contain the destination address, which uniquely identifies the entity receiving the packet. The sixth and seventh byte contain the size of the user data segment, which may be up to 1024 bytes long. The size of the user data segment is calculated *before* STX and ETX byte substitution described above.

Addresses and user data size are transmitted in network byte order, which is the MSB byte first, the LSB byte last.

Valid address range is from 0 to 65535, where address 0 is reserved for the broadcast messages, address 1 is by convention reserved for the computer itself, and all other addresses can be assigned to the equipment units.

5.3 Packet Trailer

Each packet trailer consists of two bytes: the checksum byte and the ETX byte. The checksum is calculated as an exclusive OR on all bytes from the source address byte up to the checksum byte, but excluding the checksum byte itself[1]. STX and ETX bytes are not included in the checksum calculation. The checksum is calculated *before* STX and ETX byte substitution. If the calculated checksum value is 02H or 03H then the checksum byte itself must be substituted with two consecutive bytes of the same value.

5.4 Packet Exchange Rules

The equipment may send unsolicited messages, referred to as the event reports, over a serial line or network connection. The connection used for unsolicited messages must be a full duplex connection.

The equipment also responds to the requests received from the computer. Each request from the computer is contained in a single packet. Once the equipment receives a valid request, it responds by sending exactly one packet back to the computer.

Invalid or corrupted packets are discarded. For example, if the checksum is wrong, the packet is not processed and the equipment does not send any response to the computer.

A valid packet from the computer contains a source and a destination address. The source address is that of the computer itself, while the destination address selects the equipment unit. When a party line is used, all daisy-chained units receive the same packet. Each unit compares the destination address with its own pre-configured address[2]. Units that are not addressed ignore the packet. Only the addressed unit responds.

If a broadcast packet is received, all units service the request, but do not respond to the computer.

[1] When verifying the checksum, the receiving entity may calculate the checksum for the addresses, user data size, and user data bytes and then applies the exclusive OR operation between the calculated checksum and the checksum byte found in the packet. If the packet is valid the result must be 0.

[2] The address might be either stored in a non-volatile memory, or set using a DIP switch.

When formatting a response packet, the addressed unit takes the source address from the request packet and inserts it into the destination address field of the response packet. It then inserts its own address into the source address field.

Once the user data segment is populated with the requested information, the unit calculates and inserts the user data size and the checksum.

While sending a packet, each time an STX or ETX value is encountered within the packet, it gets substituted with two consecutive bytes of the same value. This does not apply to the very first STX and the very last ETX byte.

The maximum packet size is 1033 bytes. At the lowest supported speed of 1200 Bd it takes about 10 seconds to transmit such packet. Thus, if the entire packet is not received within 10 seconds, the incomplete packet is dropped. Timeout may be adjusted to lower values, depending on the exact physical connection being used.

5.5 Encoding Rules

The user data segment usually contains a list of parameters and their respective values. The protocol allows the computer to query or to set over 140 numeric parameters within a single packet. The number of text parameters that can be included into a single packet depends on the length of each character string. Such a large number of parameters transmitted in a single packet is achieved by formatting the user data segment as follows.

The user data segment always starts with an opcode. The opcode is a byte that identifies the command or response contained in the packet. A list of valid opcodes is defined in the next section.

The next byte in the user data segment indicates how many parameters to expect in the parameter list.

The remaining part of the user data segment contains a list of parameters. The structure of the user data segment is shown in the following diagram:

Opcode	Parameter Count	Parameter List

The values of the parameters in the list must have one of the following *data types*:

- Boolean numbers
- Unsigned integer numbers
- Signed integer numbers
- Single-precision floating point numbers
- Double-precision floating point numbers
- Text strings

The protocol supports the following data types:

Data Type	Identifier
Analog 32-bit Integer Input	1
Analog 32-bit Integer Output	2
Single-precision Floating Point Input	3
Single-precision Floating Point Output	4
Boolean (Digital) Input	5
Boolean (Digital) Output	6
Text Input	7
Text Output	8
Double-precision Floating Point Input	9
Double-precision Floating Point Output	10
Analog 64-bit Integer Input	11
Analog 64-bit Integer Output	12

Note that protocol packet encoding does not indicate the semantics of the parameters being transmitted. This means that the software querying or setting parameter values must be aware of how to interpret the values associated with any given parameter ID.

The valid parameter set is equipment-specific. Therefore, a list of valid parameter IDs, associated data types and numerical ranges must be explicitly defined in the equipment-specific documentation.

5.5.1 Boolean and 32-Bit Integer Numbers

Boolean and 32-bit integer parameters are always encoded using 7 bytes. The first two bytes contain a number from 1 to 65535, which uniquely identifies the parameter within a given type of equipment. This means that protocol supports up to 65535 parameters within a given equipment unit.

The parameter ID is followed by one byte that uniquely identifies the data type being transmitted.

The remaining four bytes contain parameter value. Parameter ID and value are encoded by transmitting the *most significant byte (MSB)* first and the *least significant byte (LSB)* last, using the natural byte order.

Parameter ID	Data Type	MSB	…	…	LSB

Boolean parameters take value of 0 or 1.

Unsigned 32-bit integers have range from 0 to 4,294,967,295.

Signed 32-bit integers have range from -2,147,483,648 to 2,147,483,647.

5.5.2 64-Bit Integer numbers

64-bit integer parameters are always encoded using 11 bytes. The first two bytes contain a number from 1 to 65535, which uniquely identifies the parameter within a given type of equipment. This means that protocol supports up to 65535 parameters within a given equipment unit.

The parameter ID is followed by one byte that uniquely identifies the data type being transmitted.

The remaining eight bytes contain parameter value. Parameter ID and value are encoded by transmitting the *most significant byte (MSB)* first and the *least significant byte (LSB)* last, using the natural byte order.

Parameter ID	Data Type	MSB	…	…	LSB

Unsigned 64-bit integers have range from 0 to 18,446,744,073,709,551,615.

Signed 64-bit integers have range from −9223372036854775807 to +9223372036854775807.

5.5.3 Single-Precision Floating Point Numbers

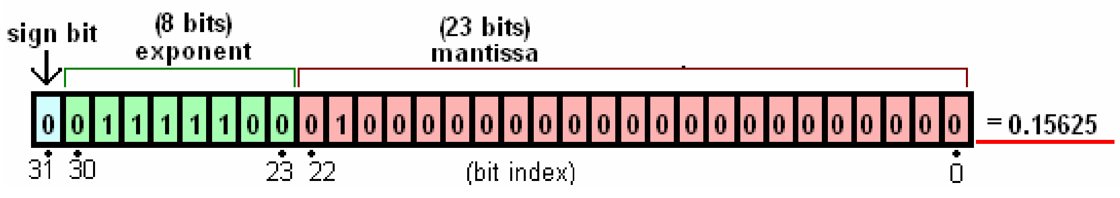
Single-precision floating point numbers are encoded using 7 bytes. The first two bytes contain a number from 1 to 65535, which uniquely identifies the parameter within a given type of equipment. This means that protocol supports up to 65535 parameters within a given equipment unit.

The parameter ID is followed by one byte that uniquely identifies the data type being transmitted.

The remaining four bytes contain parameter value. Parameter ID and value are encoded by transmitting the *most significant byte (MSB)* first and the *least significant byte (LSB)* last, using the natural byte order.

Parameter ID	Data Type	MSB	…	…	LSB

A single-precision floating point number is stored into four bytes of memory as depicted in the following diagram:



Single-precision floating point numbers offer 7 significant digits within the range of +/- 3.4 E +/- 38.

5.5.4 Double-Precision Floating Point Numbers

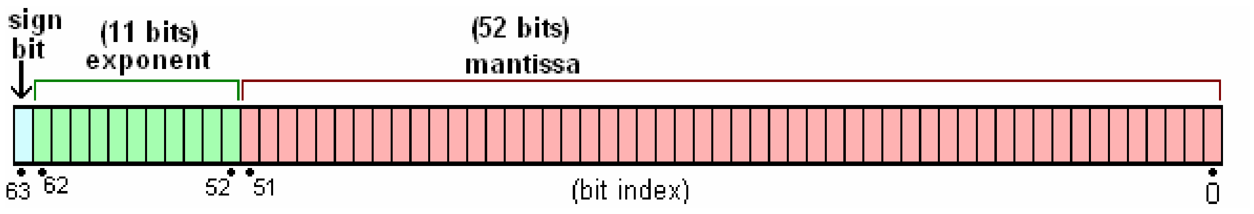
Double-precision floating point numbers are encoded using 11 bytes. The first two bytes contain a number from 1 to 65535, which uniquely identifies the parameter within a given type of equipment. This means that protocol supports up to 65535 parameters within a given equipment unit.

The parameter ID is followed by one byte that uniquely identifies the data type being transmitted.

The remaining eight bytes contain parameter value. Parameter ID and value are encoded by transmitting the *most significant byte (MSB)* first and the *least significant byte (LSB)* last, using the natural byte order.

Parameter ID	Data Type	MSB	…	…	…	…	…	…	LSB

A double-precision floating point number is stored into eight bytes of memory as depicted in the following diagram:



Double-precision floating point numbers offer 15 significant digits within the range of +/- 1.7 E +/- 308.

5.5.5 Text Strings

Text strings have variable length. Thus, the encoding must indicate how many bytes to read. This is achieved by adding the length byte after the parameter ID and data type.

The first two bytes contain a number from 1 to 65535, which uniquely identifies the parameter within a given type of equipment. This means that protocol supports up to 65535 parameters within a given equipment unit.

The parameter ID is followed by one byte that uniquely identifies the data type being transmitted.

The next byte indicates the length of the text string. The length byte is followed by the actual ASCII characters.

Parameter ID	Data Type	Length	ASCII	ASCII	…

The length of the text strings is limited to 254 characters.

5.5.6 Lists of Values

Many applications use arrays or linked lists of parameters of the same type. Transmitting such a list can be optimized by encoding the ID and data type only once for the entire list. In such case, the parameter ID identifies the entire list.

The first two bytes contain a number from 1 to 65535, which uniquely identifies the list of parameters within a given type of equipment.

The parameter ID is followed by one byte that uniquely identifies the data type that applies to all values being transmitted in the list.

The next byte indicates the number of values in the list. The length byte is followed by a list of values. The maximum number of values in the list is 255.

Parameter ID	Data Type	List Count	Value 1	Value 2	…

Lists can be used only for the numerical data types. They do not apply to text strings.

6 Command Set

Using the MACS communication protocol, the computer can query the equipment status, read configuration and operational parameters and statistics, and set configuration parameters to the desired values.

The exact set of parameters depends on the equipment itself. This protocol definition is generic and applies to all products using the protocol. Details on the parameter set applicable to any given type of equipment must be provided in the equipment-specific documentation.

MACS communication protocol supports only the following commands[3]:

- Get Command
- Set Command
- Get Record Command
- Set Record Command
- List Command

These commands are sufficient to monitor and configure the equipment and to execute any remote operations in the equipment. For example, to query equipment status, the computer will *get* a list of relevant parameters. To tune the equipment, the computer will *set* the frequency and gain parameters. To reset the equipment, the computer may *set* the Boolean reset parameter to 1. To execute a program in the remote equipment, the computer will *set* a text parameter to the name of the program to execute.

Since each command may contain a substantially long list of parameters, a single command and a single response should suffice for most of the queries.

The protocol also supports the following responses:

- Get Response
- Set Response
- Get Record Response
- Set Record Response
- List Response
- Error Response

In addition to the commands and responses, the protocol also allows remote units to send unsolicited event reports, also referred to as notifications. A particular implementation of the protocol can use unsolicited messages only if the underlying physical interface supports it. For example, unsolicited messages cannot be used over the RS 485 interface. There is no response to unsolicited event reports.

Each of the above commands and responses is identified by a unique opcode, which appears as the first byte in every user data segment. The opcodes are defined as follows:

[3] It is envisioned that the next protocol revision will also support the Get Next command.

#	Opcode Name	Opcode Value
1.	Get Command	11H
2.	Set Command	12H
3.	Get Response	13H
4.	Set Response	14H
5.	Get Record Command	19H
6.	Set Record Command	1BH
7.	Get Record Response	1AH
8.	Set Record Response	1CH
9.	List Command	16H
10.	List Response	17H
11.	Event Report	18H
12.	Error Response	15H

The next byte after the opcode indicates how many parameters to expect and the remaining bytes in the user data segment contain the actual list of parameters.

6.1 Get Command and Response

The *Get Command* and the associated response are used to query equipment status, statistics and configuration parameters. The computer can use a single packet to query any number of parameters, as long as their IDs and values can fit into the number of bytes available to the user data segment. If the size of the list of parameters and values exceeds the available space, the equipment returns an appropriate error response.

The parameters being queried do not have to be of the same data type. A query can contain any mixture of valid Boolean, integer, floating point and text parameters. A typical *get* command requesting a list of parameters has the following form:

STX	SRC	DST	SIZE	11H	n	PID1	DT1	PID2	DT2	…	PIDn	DTn	CHK	ETX

Where:

STX	byte marks the start of the packet
SRC	contains two-byte source address
DST	contains two-byte destination address
SIZE	contains two-byte size of the user data
11H	is the *get* command opcode
n	byte contains the parameter count
PID1 … PIDn	represent two-byte parameter IDs
DT1 … DTn	Represent a single-byte data type
CHK	byte contains the checksum
ETX	byte marks the end of the packet

The equipment checks the consistency of the command and may detect and return one of the following errors:

- Invalid size
- Invalid parameter count
- Invalid parameter ID

If there are no errors in the *get* command packet, the equipment returns a list of requested parameters with their respective values:

STX	SRC	DST	SIZE	13H	n	PID1	DT1	VAL1	PID2	DT2	VAL2	…	PIDn	DTn	VALn	CHK	ETX

Where:

STX	byte marks the start of the packet
SRC	contains two-byte source address
DST	contains two-byte destination address
SIZE	contains two-byte size of the user data
13H	is the *get* response opcode
n	byte contains the parameter count
PID1 … PIDn	represent two-byte parameter IDs
DT1 … DTn	Represent single-byte data types
VAL1 … VALn	represent values associated with PID1 … PIDn
CHK	byte contains the checksum
ETX	byte marks the end of the packet

If the equipment cannot fit the entire list of parameters into the response packet, it returns a “to many parameters” error.

The following example illustrates a *get* command, which queries the equipment name, current frequency, gain and operational temperature. Please note that the parameters used in this example are fictitious and do not correspond to any actual equipment.

Computer address: 1

Unit address: 2

Parameters:

Parameter Name	Parameter ID	Data Type	Present Value
Equipment name	04H	Text string	MICUS
Frequency	17H	Single-precision	91,120 kHz
Gain	18H	Double-precision	24.78 dB
Serial port data rate	0BH	Integer	9600

To query these parameters, the computer will send a packet that contains the following *get* command:

02 00 01 00 02 02 00 0E 11 04 00 04 08 00 17 04 00 18 0A 00 0B 02 02 1C 03

⬜ – header and trailer

🟦 – parameter ID

🟪 – data type

Note that destination address and last data type contain STX substitution bytes.

The equipment will respond with a *get* response as follows:

02 00 02 02 00 01 00 24 13 04 00 04 08 05 4D 49 43 55 53 00 17 04 42 B6 3D 71

00 18 0A 14 7A E1 48 40 38 C7 AE 00 0B 02 02 00 00 25 80 BB 03

🟨 – parameter value

Note that the above picture represents a single continuous packet, with highlighted parameter IDs and their associated values.

6.2 Get Record Command and Response

The *Get Record Command* and the associated response are used to atomically query a record that contains a consistent set of parameters. The structure of the command and response packets is the same as for *Get Command,* except for the opcodes:

19H	*get record command* opcode
1AH	*get record response* opcode

The difference between *Get Command* and *Get Record Command* is in the way the remote unit services the request. The *Get Record Response* returns a set of matching parameters, correlated in time.

For example, if *Get Record Command* is used to query geographical coordinates the protocol guaranties that longitude and latitude come from the same positioning sample.

6.3 Set Command and Response

The *Set Command* and the associated response are used to set equipment configuration and operational parameters. The computer can use a single packet to set any number of parameters, as long as their IDs, data types and values can fit into the number of bytes available to the user data segment.

The parameters being set do not have to be of the same data type. A *set* command can contain any mixture of valid Boolean, integer, floating point and text parameters. A typical *set* command for a list of parameters has the following form:

STX	SRC	DST	SIZE	12H	n	PID1	DT1	VAL1	PID2	DT2	VAL2	…	PIDn	DTn	VALn	CHK	ETX

Where:

STX	byte marks the start of the packet
SRC	contains two-byte source address
DST	contains two-byte destination address
SIZE	contains two-byte size of the user data
12H	is the *set* command opcode
n	byte contains the parameter count
PID1 … PIDn	represent two-byte parameter IDs
DT1 … DTn	represent two-byte data types
VAL1 … VALn	represent values associated with PID1 … PIDn
CHK	byte contains the checksum
ETX	byte marks the end of the packet

The equipment checks the consistency of the command and may detect and return one of the following errors:

- Invalid size
- Invalid parameter count
- Invalid parameter ID
- Invalid parameter data type
- Invalid parameter value
- Read-only parameter

The equipment first validates the entire packet and then sets parameter values. If any errors are encountered, none of the parameter values is changed.

If there are no errors in the *set* command packet, the equipment sets the parameter values as requested, reads back the new values and sends them to the computer. The computer typically uses the *set* response to verify the newly set parameter values. The structure of the response packet is exactly the same as the *set* command, except that it contains opcode 14H.

The following example illustrates a *set* command, which sets the frequency to 91.12 MHz and gain to 24.78 dB. Please note that the parameters used in this example are fictitious and do not correspond to any actual equipment.

Computer address:	1
Unit address:	2

Parameters:

Parameter Name	Parameter ID	Data Type	Present Value
Frequency	17H	Single-precision	91,120 kHz
Gain	18H	Double-precision	24.78 dB

To set these parameters, the computer will send a packet that contains the following *set* command:

02 00 01 00 02 02 00 14 12 02 02 00 17 04 42 B6 3D 71 00 18 0A 14 7A E1 48 40 38 C7 AE 68 03

⬜ – header and trailer

🟦 – parameter ID

🟪 – data type

🟨 – parameter value

The equipment will respond with a *set* response as follows:

02 00 02 02 00 01 00 14 14 02 02 00 17 04 42 B6 3D 71 00 18 0A 14 7A E1 48 40 38 C7 AE 6E 03

6.4 Set Record Command and Response

The *Set Record Command* and the associated response are used to atomically set a consistent set of parameters, with the values that belong to the same record. The structure of the command and response packets is the same as for *Set Command,* except for the opcodes:

1BH	*set record command* opcode
1CH	*set record response* opcode

The difference between *Set Command* and *Set Record Command* is in the way the remote unit services the request. The *Set Record Response* sets a list of parameters atomically, at the same time.

6.5 List Command and Response

The *List Command* and the associated response are used to query a list of values of the same numerical type, identified by a single parameter ID that applies to the entire list. The computer can use a single packet to query more than one list, as long as their IDs and values can fit into the number of bytes available to the user data segment. If the size of the lists exceeds the available space, the equipment returns an appropriate error response.

The *List Command* is typically used to query an array or a linked list of numerical values. Such a list is identified by a single parameter ID and all values on the list must be of the same type. A typical *list* command has the following form:

STX	SRC	DST	SIZE	16H	n	PID1	DT1	PID2	DT2	…	PIDn	DTn	CHK	ETX

Where:

STX	byte marks the start of the packet

SRC	contains two-byte source address
DST	contains two-byte destination address
SIZE	contains two-byte size of the user data
16H	is the *list* command opcode
n	byte contains the parameter count
PID1 … PIDn	represent two-byte parameter IDs which identify the requested lists
DT1 … DTn	Represent a single-byte data type
CHK	byte contains the checksum
ETX	byte marks the end of the packet

The equipment checks the consistency of the command and may detect and return one of the following errors:

- Invalid size
- Invalid parameter count
- Invalid parameter ID

If there are no errors in the *get* command packet, the equipment returns the requested lists, each of which consists of the ID, data type, value count and a list of values:

STX	SRC	DST	SIZE	17H	n	PID1	DT1	CNT1	VAL1$_0$	VAL1$_1$	…	…	PIDn	DTn	CNTn	VALn$_0$	VALn$_1$	…	CHK	ETX

Where:

STX	byte marks the start of the packet
SRC	contains two-byte source address
DST	contains two-byte destination address
SIZE	contains two-byte size of the user data
17H	is the *list* response opcode
n	byte contains the parameter count
PID1 … PIDn	represent two-byte parameter IDs which identify the requested lists
DT1 … DTn	represent a single-byte data type
CNT1 … CNTn	represent a single-byte value count for each list
VALn$_m$	represent requested numerical values
CHK	byte contains the checksum
ETX	byte marks the end of the packet

If the equipment cannot fit the entire list of parameters into the response packet, it returns a “to many parameters” error.

A *Get Command* performed on a parameter that contains a list returns the number of values in the list. The value count is encoded using the same data type assigned to the elements of the list.

The following example illustrates a *list* command, which queries an array of integer values. Please note that a list used in this example is fictitious and does not correspond to any actual equipment.

Computer address: 1

Unit address: 3

List parameter ID: 31H

To query the list, the computer will send a packet that contains the following *get* command:

02 00 01 00 03 03 00 05 16 01 00 31 02 02 23 03

⬜ – header and trailer

🟦 – parameter ID

🟪 – data type

Note that destination address and last data type contain STX substitution bytes.

The equipment will respond with a *list* response as follows:

```
02 00 03 03 00 01 00 D3 17 01 00 31 02 02 32
00 00 00 04 00 00 00 08 00 00 00 0C 00 00 00 10
00 00 00 14 00 00 00 18 00 00 00 1C 00 00 00 20
00 00 00 24 00 00 00 28 00 00 00 2C 00 00 00 30
00 00 00 34 00 00 00 38 00 00 00 3C 00 00 00 40
00 00 00 44 00 00 00 48 00 00 00 4C 00 00 00 50
00 00 00 54 00 00 00 58 00 00 00 5C 00 00 00 60
00 00 00 64 00 00 00 68 00 00 00 6C 00 00 00 70
00 00 00 74 00 00 00 78 00 00 00 7C 00 00 00 80
00 00 00 84 00 00 00 88 00 00 00 8C 00 00 00 90
00 00 00 94 00 00 00 98 00 00 00 9C 00 00 00 A0
00 00 00 A4 00 00 00 A8 00 00 00 AC 00 00 00 B0
00 00 00 B4 00 00 00 B8 00 00 00 BC 00 00 00 C0
00 00 00 C4 00 00 00 C8 0A 03
```

🟨 – parameter value

🟩 – value count

Note that the above picture represents a single continuous packet, with highlighted parameter IDs and their associated values.

6.6 Event Report

The *Event Report* is an unsolicited message sent by the remote unit, without a query from the computer. Typically, event reports may be time based, or may be sent when there is a change in the status, or some threshold is reached. The remote unit can use a single packet to report changes in any number of parameters, as long as their IDs and values can fit into the number of bytes available to the user data segment:

STX	SRC	DST	SIZE	18H	n	PID1	DT1	VAL1	PID2	DT2	VAL2	…	PIDn	DTn	VALn	CHK	ETX

Where:

STX	byte marks the start of the packet
SRC	contains two-byte source address
DST	contains two-byte destination address
SIZE	contains two-byte size of the user data
18H	is the *event report* opcode
n	byte contains the parameter count
PID1 … PIDn	represent two-byte parameter IDs
DT1 … DTn	Represent single-byte data types
VAL1 … VALn	represent values associated with PID1 … PIDn
CHK	byte contains the checksum
ETX	byte marks the end of the packet

The *event report* format is the same as the *get command* response. The only difference is the value of the opcode.

The computer does not send any packet to acknowledge the receipt of an event report.

6.7 Error Response

Before servicing any request received from the computer, the equipment always performs a consistency check on the received packet and validates the opcode, parameter IDs, data types and values. If any errors are found within the packet, the equipment does not service such request and instead returns an error response.

The error response always contains information on the first error encountered while processing the packet. This means that error responses return information on one error only, as the principal reason why the packet was rejected.

If detected error is related to the parameter list, the response also contains the index of the parameter in question, starting from 1.

The error response has the following form:

STX	SRC	DST	03H	15H	ERR	INDEX	CHK	ETX

Where:

STX	byte marks the start of the packet
SRC	contains two-byte source address
DST	contains two-byte destination address
03H	is two-byte size of the user data, which is always 3
15H	is the *error* response opcode
ERR	byte contains the error code
INDEX	Is the parameter index, starting from 1. If not applicable, this field contains 0
CHK	byte contains the checksum
ETX	byte marks the end of the packet

Possible error codes are defined as follows:

	Error	Code	Index	Description
1	Invalid size	01H	Yes	The value in the size field does not match the actual size of the user data segment.
2	Invalid parameter count	02H	No	The value in the parameter count field does not match the actual number of parameters in the user data segment.
3	Invalid parameter ID	03H	Yes	The parameter ID is invalid for given equipment.
4	Invalid data type	04H	Yes	The requested data type does not match the actual data type.
5	Invalid value	05H	Yes	The requested value is invalid for a given parameter.
6	Read-only parameter	06H	Yes	An attempt was made to set parameter which is read-only.
7	Invalid port address	07H	Yes	Reserved.
8	Invalid port type	08H	Yes	Reserved.
9	Ports not initialized	09H	Yes	Reserved.
10	Points not initialized	0AH	No	Reserved.
11	Invalid bit position	0BH	Yes	Reserved.
12	Lock not created	0CH	No	Reserved.
13	NV memory failed	0DH	No	Failed to save a non-volatile parameter
14	Failed to lock	0EH	No	Failed to service request because a lock could not be obtained within expected time interval.
15	Open file failed	0FH	No	Failed to open a file required to service request.
16	Debugger already open	10H	No	An attempt was made to open internal trace when the trace was already running.
17	Unknown error	11H	Yes	An error has occurred that is not defined in this list
18	Invalid range	12H	Yes	The requested value is out of range for a given parameter.
19	Invalid parameter name	13H	Yes	Could not find a parameter ID associated with a given parameter name.
20	Invalid unit address	14H	No	An out of range number was specified as a unit address.
21	Invalid packet size	15H	No	Calculated packet size does not match the actual

				packet size, or packet exceeds maximum packet size.
22	Unexpected response	16H	No	The actual response opcode does not match the expected response opcode.
23	Serial port error	17H	No	A communication error was encountered while sending or receiving packets over a serial line.
24	No response	18H	No	A timeout has occurred while waiting for the response.
25	Invalid checksum	19H	No	Calculated checksum does not match the checksum byte in the packet.
26	Unsupported request	1AH	Yes	The packet contains an unsupported opcode value, or the requested get or set operation is not applicable for a given parameter.
27	Failed to unlock	1BH	No	Failed to unlock upon servicing request.
28	IPC not created	1CH	No	Failed to create an interprocess communication mechanism, such as shared memory, or a pipe.
29	IPC failed	1DH	No	An error has occurred while using an interprocess communication mechanism, such as shared memory, or a pipe.
30	HW write error	1EH	Yes	Failed to write to a hardware device.
31	HW read error	1FH	Yes	Failed to read from a hardware device.
32	Networking error	20H		A communication error was encountered while sending or receiving packets over a network connection.
33	No memory	21H	No	There is no enough memory to execute command.
34	Callback failed	22H	No	Callback function failed or not found

For example, if a *set* command attempts to set a read-only parameter 1 in the list to a certain value, the equipment will return the following error response:

02 00 02 02 00 01 00 03 03 15 1A 01 0E 03

🟦 – error code

🟨 – error index