SCIENTIFIC PRODUCTION COMPANY
"DOZA"

DiBus
Description of data exchange protocol of instrument networks for devices produced by SPC "Doza"

FVKM.09.001.0002-01OPP

Content

1 DESCRIPTION OF INSTRUMENT NETWORK OF SPC "DOZA"

The proprietary instrument network of SPC "Doza" (hereinafter – the "network") has a "monomaster" architecture. Slave devices send data to the network only on request of the master device. A master device must perform relevant polling of all devices connected to it through the network.

All packets transmitted over the network have the same structure as follows:

Header (14 byte)						Data block	
Recipient address	Sender address	Packet type	Data type or interface	Data length, byte	Header CRC*	Data block body	Data block CRC*
3 bytes	3 bytes	1 byte	1 byte	2 bytes	4 bytes	0..32767 byte	4 bytes
* - Algorithm of calculation CRC in accordance with 2.3							

"Recipient address" and "sender address" are described in 1.1.

"Packet types" are defined in 1.2.

"Data types" are defined in 1.3.

"Data length" is the amount of information (in bytes) in the field "Data block body".

"CRC of header" is a 4-byte number, obtained by processing of the first ten bytes of the header by the algorithm described in 2.3.

"Data block body" are actual data defined by the transmitted packet type.

"data block CRC" is a 4-byte number, obtained by processing of all bytes of the "Data block body" field by the algorithm described in 2.3, for packets with no data ("data length" = 0) "data block CRC" is not calculated and not included in the packet (In this case, the packet consists of a header only).

Slave sends a packet in response to each command received, thus confirming the receipt. The form of response is strictly defined for each type of command in accordance with 1.2.

If the slave device can not send the correct response to a command, it must send a packet with an error code in accordance with 2.2.

1.1 Recipient address, sender address

The sender address is a unique network address (UNA) of a device that sends a packet.

The recipient address is a UNA of the recipient or reserved template in accordance with 1.1.1.

The UNA is a unique constant, assigned to the device during manufacture. The uniqueness of addresses of different devices is ensured by centralized accounting of assigned ranges.

For each project its address range is allocated at common center.

The address is divided into fields: the 1st byte corresponds to project, the 2nd byte – to the device type and the 3rd byte is the device's serial number. If a series of more than 253 devices is to be manufactured, the next number is allocated for the device type.

The common table of address ranges, registered device types and responsible developers is stored on the SPC "Doza" file server in a folder "ID\\Unique Identifiers.xls". In the above file the procedure for assigning a new address or address range is also described.

1.1.1 Broadcast Addresses

The addresses "0.0.0", "1.1.1" and "255.255.255" are reserved for templates and can not be used in devices.

The address "0.0.0" corresponds to the address template of an unregistered device. Any of non-registered slave devices must respond when it receives a packet with recipient address "0.0.0". For registration procedure, see 2.1.

The address "255.255.255" corresponds to an address template of any device on the network except that of master device - sender.

The address "1.1.1" corresponds to "master" device address.

Template addresses "0.0.0" and "255.255.255" can indicate only "master" as the "recipient address". The template address "1.1.1" can indicate only "slave device" as the "recipient address".

1.2 Packet type

Code	Command	Data length, byte
0	Request for registration of "slave"	1 *
1	Confirmation of command receipt by "slave"	0
2	Confirmation of registration by "slave"	1 *
3	Device error	1 ** (error code)
4	Request "Connected?" (Ping)	0
5	Response "slave" – want to send a packet	2
6	Receive data from "slave"	determined by data type ***
7	Requested data from "slave"	determined by data type ***
8	Transfer data to "slave"	determined by data type ***
9	Redirect	arbitrary ****
10	Request packet from "slave"	0
12	Deregistration of "slave"	0

Note – Additional packet types will be formed when needed based on additional reconciliation.

* - for registration procedure see 2.1.
** - error codes are presented in 2.2.
*** - data types are presented in 1.3.1.
**** - "Redirect" command is described in 1.3.2.

A packet with code "0" is sent by master device for the registration of slaves in accordance with 2.1.

A packet with code "1" ("Receipt confirmation") is sent by slave device in response to all the commands, if they are properly recognized, executed, and do not require a different type of response (commands "0", "2", "4", "8", "12").

A packet with code "2" is sent by master device for the registration of slaves in accordance with 2.1.

A packet with code "3" is sent by slave device in case that the device is unable to execute a command. The "master" sends a packet with code "3" only in certain cases of using the "Redirect" command (code 9) in accordance with 1.3.2. The "slave" must respond with a packet with code "1".

A packet with code "4" is sent by master device to check the connection slaves, as well as survey of devices that are capable of sending active packets. In response to this command a "slave" sends "5", if it wants to send a packet, or "1" otherwise.

A packet with code "5" is sent by slave device in response to a command "4", if the device "wants" to send any packet. In the "data" field the length of a packet is transmitted that the device wants to send. In response to "5", "master" can send a request "10" to receive a packet from slave.

A packet with code "6" is sent by master device to obtain information from the slave devices.

In this case the method for identification of data is determined by the "data type" field in accordance with 1.3. A "slave device" must send a packet with code "7" in response.

A packet with code "7" is sent by a slave device in response to "data request" with code "6". In this case the method for identification of data is determined by the "data type" field in accordance with 1.3.

A packet with code "8" is sent by master device to transfer information for slave devices. In this case the method for identification of data is determined by the "data type" field in accordance with 1.3. A "slave device" must send a packet with code "1" in response.

A packet with code "9" is sent by a slave device or the "master" to transfer information to any external device. In this case the "Data type or interface" field contains the interface number for which data are transmitted. Working with redirected packets is described in 1.3.2.

A packet with code "10" is sent by master device in response to a packet with code "5" to receive information, which a slave device wants to send. In response, a "slave device" must send a packet with code "7" or a packet with code "9", if redirection of data is to be done.

A packet with code "12" is sent by master device for deregistration of "slave devices". In response, a "slave device" should change its internal status on "non-registered" and send a packet with code "1". This command is recommended for use to implement the synchronization of registration statuses in master and slave devices. Alternatively, it is proposed the use several cycles of the command "12" with the recipient address "255.255.255" during the start of "master".

Commands "0", "1", "2" and "4" are recommended for implementation in all network devices with access "by registration".

1.3 Data type or interface

The data type (used for data transmission) is shown in Table 1.3.1

Table 1.3.1 Data type

Code	Data type	Identification
1	Byte, array bytes	by index
2		by name
3	String of one-byte symbols	by index
4		by name
5	Word – unsigned integer, 2 bytes (low byte first) (0-65535)	by index
6		by name
7	ShortInt* – signed integer, 1 byte (-128..127)	by index
8		by name
9	Integer** – signed integer, 2 bytes (low word first) (low byte in words - first) (-32768..32767)	by index
10		by name
11	DWord – unsigned integer, 4 bytes (low byte first, low word first) (low byte in words - first)	by index
12		by name
13	L_Single – unsigned real, 2 bytes (xxxxxxyy yyyyyyyy – x-power, y-mantissa) (low byte first)	by index
14		by name
15	S_Single – unsigned real, 2 bytes (xxxxxyyy yyyyyyyy – x-power, y-mantissa) (low byte first)	by index
16		by name
17	Array of any type	by index
18		by name
19	Contiguous fragment of an array of any type	by index
20		by name
21	ASCII-signed integer	by index
22		by name
23	ASCII-engineering	by index

Code	Data type	Identification
24		by name
25	Single (IEEE-754 Single) – real, 4 bytes (low byte (bits 0..7) first) see Appendix 1.	by index
26		by name
27	M_Single – real, 4 bytes	by index
28	[bit31] sxxxxxxx xxxxxxxx xxxxxxxx eeeeeeee [bit0] s - sign of the mantissa (1 bit) x - mantissa (23 bits, normalization is not necessary) e – decimal order + 127 (8 bit) low byte is transmitted first	by name
29	String Unicode (two-byte symbols)	by index
30		by name
31	Long_DateTime – Date and Time	by index
32	[bit63] *yy mm dd hh mm ss msec* [bit0] yy – year (1 byte) mm – month (1 byte) dd – day of the month (1 byte) hh – hours (1 byte) mm – minutes (1 byte) ss – seconds (1 byte) msec – milliseconds (2 byte (word))	by name
33	DiBUS_address	by index
34	[bit23] *AA BB CC* [bit0] AA – 1st byte (project-type) (1 byte) BB – 2nd byte (type) (1 byte) CC – 3rd byte (serial number) (1 byte)	by name
125	Element of user-defined type	by index
126		by name
128	"Parametric" request / response ***	

Note – Additional packet types will be formed when needed based on additional reconciliation.
* - Negative numbers in ShortInt are presented in additional code (80h=-1…FFh=-1)
** - Negative numbers in Integer are presented in additional code
*** - Parametric protocol is described in 2.4

1.3.1 Fundamental types

For types 1÷24 identification of variables is performed in two different ways.

The identification by index assumes one additional byte in the data field preceding the data, which implements the idea of a variable number. Thus, the access by index allows referring to up to 256 elements of each type.

The identification by name assumes one additional string identifier in the data field preceding the data, which is the name of the variable. The variable name can only contain Latin letters, digits and underscores. The maximum size of a string identifier is 16-byte ASCII-symbols, together with a symbol #0 completing a line. Thus, using access by name it is possible to name any number of elements of each type.

Uppercase and lowercase letters of the string identifier are different. However, it is not recommended to name different variables in a system using identifiers that differ only by case of symbols.

The A string identifier must begin from the beginning of "data" and end with #0 (byte code 0). Data must follow the "#0".

In detail:

For "1" – array size = "data size"-1, if this type is used as an array. A reference to this type (in types 17, 18, 19, 20, 125, 126) assumes reference to byte.

For "2" – array size = "data size" - " length of the string-identifier", if this type is used as an array. A reference to this type (in types 17, 18, 19, 20, 125, 126) assumes reference to byte.

For "3" and "4" – string of one-byte symbols is any number of one-byte symbols, ending with one-byte symbol with code 0.

For "5" ÷ "12" – standard types for ObjectPascal.

For "13" and "14" – L_Single – unsigned real, 2 bytes (xxxxxxyy yyyyyyyy – x-power, y-mantissa) (low byte first).

Examples: *6Fh 3Dh = 03D6Fh=00111101 01101111 b=3.67·10^15;*
93h F7h = 0F793h=11110111 10010011=9.15·10^-3

For "15" and "16" – S_Single – unsigned real, 2 bytes (xxxxxyyy yyyyyyyy – x-power, y-mantissa) (low byte first)

For "17" and "18" – index or name (array identifier) is followed by one byte of type number. This byte characterizes the type of each array element. Correspondence of this byte value to the element type can be obtained from the Table 1.3.1. It is recommended to choose odd value of types (element of specific type by index). It is followed by the values of specified type with no additional separators.

It is acceptable to use complex structures, such as records, as type of elements of an array (type 125). At that the data characterizing record type are presented in the packet prior to the values, in a single copy (see description of type "125").

Example 1: *[array with index 7]:=((1,1), (2,2))*
data – array of structures including two elements with types of fields 1,5.

Header – field "Data type or interface": 11h
Data block: 07h 7Dh 02h 01h 05h 01h 01h 00h 02h 02h 00h
Example 2: *[array with name "DOSE"]:=((1,1), (2,2));*
data –array of structures including two elements with types of fields 5,5.

Header – field "Data type or interface": 12h
Data block: 44h 4fh 53h 45h 00h 7Dh 02h 05h 05h 01h 00h 01h 00h 02h 00h 02h 00h

For "19" – the index-identifier of an array is followed by 1 byte of the type number. This byte defines the type of each array element. Correspondence of this byte value to the type of elements can be obtained from the Table 1.3.1. It is recommended to choose odd value of types (element of specific type by index). The byte of type is followed by four additional bytes. The first 2 bytes contain the starting index of the array, the second pair of bytes is the number of elements (in this pair the first byte is the low byte). It is followed by the values of specified type with no additional separators.

It is acceptable to use complex structures, such as records, as type of elements of an array (type 125). At that the data characterizing record type are presented in the packet prior to the values, in a single copy (see description of type "125").

Indexing of elements in the array is arbitrary and is determined by the user. It is recommended

to assign the index "0" to the first element of an array.

Example: *[array with index 4]:=(10,11,12,13,14,15,16,17,18,19,20)*
data – array of elements of type 5 (WORD) of 11 elements.
In the packet 5 elements of an array are transferred , starting from the element with index 3.

Header – field "Data type or interface": 13h
Data block: 04h 05h 03h 00h 05h 00h 0Dh 00h 0Eh 00h 0Fh 00h 10h 00h 11h 00h

For "20" – the name of the array identifier is followed by the number of type of array elements (as well as for "19", it may be a number of simple type or the number of a type and the description of a structure). It is followed by a string identifier of the initial element, further followed by a string identifier of the number of elements. The format of strings is similar to the type "21". Further elements follow, as for the type "19".

Example: *[array with name "DOSE"]:=(10,11,12,13,14,15,16,17,18,19,20)*
data – array of elements of type 5 (WORD) of 11 elements.
In the packet 5 elements of an array are transferred , starting from the element with index 3.

Header – field "Data type or interface": 14h
Data block: 44h 4fh 53h 45h 00h 05h 33h 00h 35h 00h 0Dh 00h 0Eh 00h 0Fh 00h 10h 00h 11h 00h

For "21" and "22" – ASCII-integer is a string (as in "3") containing only ASCII-codes of digits with a possible sign as leading symbol. No sign means "+".

Examples:
"45676" = 45676
"-145568" = -145568
"+7" = 7

For "23" and "24" – ASCII- engineering is a string (as in "3") containing only ASCII-codes of digits, symbols ".", "E" and symbols of sign ("+" or "-") in specific format:
[-,+]X.X[X…]E[-,+]Y[Y…]
- optional elements are in square brackets;
- "…" denotes an arbitrary number of elements;
- "X" - digit of mantissa;
- "Y" - digit of power.

Examples:
"4.5676E-5" =0.045676
"-1.4E56" =$-1.4 \cdot 10^{56}$
"+7.0E2" =700.0

For "25" and "26" – Single (IEEE-754 Single) – real, 4 bytes (low byte (bits 0..7) first). This format is described in detail in the Appendix 1.
For "27" and "28" – M_Single – real, 4 bytes (low byte (bits 0..7) transmitted first).
[bit31] sxxxxxxx xxxxxxxx xxxxxxxx eeeeeeee [bit0]
s - sign of the mantissa (1 bit)

x - mantissa (23 bits, normalization is not necessary)
e - decimal power + 127 (8 bit)

Examples: *80h 00h 04h 7Eh = -4·10^-1;*
00h 00h FFh 7Fh = 255

For "29" and "30" – Unicode string (string of two-byte symbols). It is any number of two-byte symbols, ending with two-byte symbol with code 0.

For "31" and "32" – Long_DateTime – Date and Time.
[bit63] yy mm dd hh mm ss msec [bit0]
yy – year (1 byte)
mm – month (1 byte)
dd – day of month (1 byte)
hh – hours (1 byte)
mm – minutes (1 byte)
ss – seconds (1 byte)
msec – milliseconds (2 byte (word))
Low byte is transmitted first.

For "33" and "34" – DiBUS_address – device's address in the DiBUS format.
[bit23] AA BB CC [bit0]
AA – 1st byte (project-type) (1 byte)
BB – 2nd byte (type) (1 byte)
CC – 3rd byte (serial number) (1 byte)
Low byte is transmitted first.

For "125" and "126" – after the index and name ID of the user-defined type is followed – size of the "record header " (number of fields in a record). The "record header " is a sequence of bytes, describing the type of each element. Correspondence of this byte value to the element type can be obtained from the Table 1.3.1. It is recommended to choose odd value of types. It is followed by the values of specified type with no additional separators. After the "record header" the record's value follows, which are defined by element type, with no delimiters and in accordance with specified types.

Example: *a structure containing 3 fields of types 5,1,7.*
data: array element with index 1.
Fields of the element (structure) are equal to 1 (WORD), 2 (BYTE), 0 (SHORTINT)

Header – field "Data type or interface": 7Dh
Data block: 01h 03h 05h 01h 07h 01h 00h 02h 00h 00h

This example is described in Pascal as follows:

```
type
    TData = packed record
          Value1: Word;
          Value2: Byte;
         Value3: ShotInt;
    end;
var
    Val: array [0..255] of TData;
```

Val[1]:= (1,2,0);

For "128" – the text information exchange protocol is defined by data, as described in 2.4.

When querying data (command "6"), the packet includes only identifier or index of a variable. When values are returned / set (commands "7" and "8") the values of variables immediately follow their identifier / index.

In commands which do not use field "data type or interface" (commands "0", "1", "2", "3", "4", "5", "10" and "12") the value of this field is reserved and must be equal to "0".

1.3.2 Packet "Redirect"

A "Redirect" packet contains a packet in its body, which is addressed to another device**.**

<table>
<tr><td>Header</td><td>CRC</td><td colspan="4">Data</td><td>CRC</td></tr>
<tr><td></td><td></td><td>Header</td><td>CRC</td><td>Data</td><td>CRC</td><td></td></tr>
<tr><td></td><td></td><td colspan="4">Any packet format, the accepted by the driver interface</td><td></td></tr>
</table>

Note – The header includes the interface number

Interface (used when sending "Redirect" packets) (packet type "9")

Code	Interface *
0	The same interface
1-254	Determined by the device manufacturer
255	All available interfaces
Note - Additional packet types will be formed when needed based on additional reconciliation. * - Interface is a device that provides data transmission channel (serial port, network card or any other).	

The "slave device" supporting the "Redirect" command, after receiving the corresponding packet, must sent a packet to the interface number specified in the byte "interface", which is contained in the packet body. The interface and the address from which the request was received are stored. When the answer arrives in response to a packet sent via specified interface, this answer is sent to the previously stored address, included in the body of a new "Redirect" packet. In case the answer is not received within the maximum waiting period for this interface, it is advisable to clear stored interface with the address, without sending alerts. In case of previously known possibility of exceeding the response time from the requested interface, the procedure for active sending of packets should be used (a combination of commands "4", "5", "10" and "9"). Similarly, data can be transferred through "slave device" as a gateway to other networks built on other physical basis. In case the "Redirect" to specified interface is not supported, or the driver of the specified interface finds an error in structure, the "slave device" sends to the "master" an error code "10" in accordance with 2.2.

"master", supporting the "Redirect" command, in case of receiving the corresponding packet, must sent a packet to the interface number specified in the byte "interface", which is contained in the packet body. The interface and the address from which the request was received are stored. When the answer arrives in response to a packet sent via specified interface, this answer is sent to the previously stored address, included in the body of a new "Redirect" packet. In case the answer is not received within the maximum waiting period for this interface, it is advisable to clear stored interface with the address, without sending alerts. Similarly, a "slave device" can transfer data through "master" as a gateway to other networks built on other physical basis. In case the "Redirect" to specified interface is not supported, or the driver of the specified interface finds an error in structure, "master" sends to the "slave device" an error code "10" in accordance with 2.2.

2 APPENDICIES

2.1 Registration

2.1.1 Full registration

To full operability in the network a device must be registered. For "hot" registration of devices in the network, the "master" device shall periodically send a request with code "0" to the address "0.0.0". In the "data" of this request any varying number is transferred (for example, the request number). In response, any device connected to the network but not registered by the "master" must send a packet containing code "1" and the sender address, set at the factory of manufacturer. The delay before sending this packet is defined in 2.1.3. The "master" must wait for 256·24t, until all possible responses will arrive from the devices. Then the "master device" registers devices at its discretion and sends them confirmation using command "2".

At the same time it sends a "delay option" in the range from 2 to 255. This parameter must be different for connected devices. If broadcast requests are to be sent during operation, "delays" should be defined in such a way as to avoid conflicts due to the duration of the data transmission. The "master" considers that the registration is over when it receives a receipt confirmation in response.

2.1.2 Simplified registration

A "cold" registration is a simplified version of registration. If a device is addressed directly, it must consider itself registered. The "delay" in this case remains uninitialized until the "master" sends a command "2" to direct address. In this case, broadcast commands shall not be used until "distribution of delays".

The simplified registration may be used in conjunction with full registration.

2.1.3 Recommended algorithm for calculating the delay during full registration

For the device's address A.B.C and pseudo-random number X that arrived with a request for registration the recommended delay is calculated according to the following algorithm:

Delay = (lo(A·X) xor lo(B·X·2) xor lo(C·X·4) ·25 + 1) · 24t,

Where t is the time for transfer of 1st byte,

lo – low byte.

2.2 Error codes

Code	Error
1	Unsupported command
2	Unsupported data format
3	Incorrect packet structure
4	Specified variable absent
5	The device is busy and cannot respond right now
6	The device is busy and will respond when ready ...
7	CRC of a header is correct, data CRC is incorrect
10	Incorrect packet "Redirect"
255	Unrecognized error

Error "1" is generated by a "slave device" in case it receives a command that it does not support.

Error "2" is generated by a "slave device" in case it receives a command with data type that it does not support.

Error "3" is generated by a "slave device" in case it receives a command with errors in the packet structure.

Error "4" is generated by a "slave device" in case it receives a command with ID that it does not support.

Error "5" is generated by a "slave device" in case it receives a command to which it can not respond within the allotted time.

Error "6" is generated by a "slave device" in case it receives a command to which it will respond later " independently".

Error "7" is generated by a "slave device" in case it receives a packet, where the checksum does not correspond to accepted data, which may occur, for example, due to noise spike when transmitting a long packet.

Error "10" is generated by a "slave device " or a "master" in case of receiving incorrect content of the "Redirect" command.

Error "255" is generated by a "slave device" in case of reception of the principal impossibility to implement the correct analysis of error. Not recommended for use.

2.3 CRC algorithm

In simple microcontrollers it is extremely difficult to implement the standard CRC algorithms. The calculation in our version of CRC is based on a combination of arithmetic and logical additions and shift.

It is based on algorithm XOR32.

Decompilation from assembler to ObjectPascal Delphi is as follows:

```
function CalculateCRC(Data: array of byte) : DWORD;
  var
    CRC        : DWORD;
    DataSize,
    i          : LongInt;
begin
  CRC := 0;
  i := Low(Data);
  DataSize := High(Data) - Low(Data) + 1;  // SizeOf(Data)
  if DataSize mod 2 = 1 then begin
    CRC := CRC xor DWord(Data[i]);
    Inc(i);
  end;
  while i < DataSize do begin
    CRC := Rol(CRC,5);
    CRC := CRC xor ((DWORD(Data[i]) shl 8) + DWORD(Data[i+1]));
    Inc(i, 2);
  end;
  CalculateCRC := CRC;
end;
```

2.4 Parametric protocol

Data block (taking into account the transport layer checksum)

Field	Field length	Position in a packet (byte numbers)
Index of data block - 0 – block is sole and full - 1..255 – serial number of the fragment of full data block (the block is fragmented)	1 byte	0
Type of data block - 0 – block with a set of parameters (1) - 1 – block with a continuous dump of whatever	1 byte	1
Data block body	n-2 byte	2..n-1
CRC	4 byte	n..n+3

(1) – the parameter set is a sequence of ASCII-construct of the following form:

< parameter_ID >[:< parameter_attribute >]=< parameter_value >;

Where
< parameter_ID > is a parameter identifier
< parameter_attribute > is a parameter attribute (optional; it is an extension of the parameter identifier)
< parameter_value > - 1. parameter value - in the case of a correct response to a request;
2. @@ - parameter value is not available (parameter is not supported by the addressee).
3. ?? - request of a parameter value in ASCII form;
4. ??#□ - request of a parameter value as a dump :??## - request of the number of fragments in the dump, the answer is in the format < parameter_ID >[:< parameter_attribute >]=##n;
where n=1..255 (number of fragments), ??#n - request of a fragment No. n, n=1..255

2.5 Active sending of packets from slave devices

Command with code "5" is sent by slave device in response to a command from the "master" with code "4". The "master" accordingly shall perform polling of connected devices using this command.

In the command with code "5" the "slave device" determines the size of the data block that it wants to transfer. After sending commands with code "5" the "slave device" should not change the data intended for transfer. The single-type measurements that must be valid at the time of transfer are the exception from this rule.

The active sending of packets from "slave devices" is convenient for implementing the event-driven logic.

3 PHYSICAL LAYER

3.1 Recommended interface:

The recommended interface is RS-485, half-duplex, 1 stop bit without parity check.

For the family RS-232, RS-485, RS-422:

The basic exchange rate 9600 baud (t – time of transmission of the 1st byte = 1 ms).

Devices that support other data exchange rates, should also support the basic rate of 9600 baud and provide a convenient way for selection of the data exchange rate and setting the basic rate.

3.2 Timing characteristics of the interface

The delay between bytes during transmitting of a packet shall not exceed 3×t.

The minimum allowable delay between packets = 6×t.

3.3 Time delays of slaves responding

The device connected to the network must respond to all commands sent to it directly (not broadcast commands) not earlier than 6×t after receipt of the last bytes of request. The maximum response delay time is 40×t.

If the device must perform some additional actions before a response and the delay due to these actions can may exceed 40×t, than the device must send a response with an error code "5" or "6". The error 5 – "The device is busy and cannot respond right now" is sent in case when the device does not plan to respond independently later, otherwise the error code 6 will be sent – "The device is busy and will respond when ready ..." in accordance with 2.5.

When a device registered in the network receives a packet with the address "255.255.255", it must send a response with a delay determined at the moment of registration in the registration confirmation packet (code 2). The delay should be equal to the "Delay parameter"×24×t, where "t" is the time for transfer of the 1st byte. The "Delay parameter" is transferred to a slave device by the master device as a parameter of the registration confirmation command (code 2) in accordance with 3.2.

When a non-registered network device receives a packet with address "255.255.255" it must respond with a delay of 6×t to 40×t.

When a non-registered network device receives a command "request for registration" (code 0) with address "0.0.0" the delay is calculated as described in 2.1.

When a non-registered network device receives any other packet with address "0.0.0" the response is sent after an arbitrary pseudo-random delay within the range of (1-255)×24t, where "t" is the time for transfer of the 1st byte. The procedure for generating pseudo-random delays shall preclude repetition of the same numbers in different devices.

If the device can not generate pseudo-random numbers, data exchange with it is performed using its direct address only and only after registration. This device should not respond to all other commands (except the request for registration) at the address "0.0.0".

Selecting a delay by the "master" at the registration confirmation (code 2) in accordance with 2.1 is performed based on the principle excluding the same delays in different network devices. In addition, this parameter should not be equal to "0" or "1".