KNX *ObjectServer* protocol
Version 1.1

WEINZIERL ENGINEERING GmbH
Bahnhofstr. 6
84558 Tyrlaching
Tel. +49 8623 / 987 98 - 03
Fax +49 8623 / 987 98 - 09
E-Mail: info@weinzierl.de
Web: www.weinzierl.de

Document history

Document status	**Date**	**Editor**
Draft	23. November 2006	Y.Kyselytsya
Revision	20. February 2007	Y.Kyselytsya
Update	24. April 2007	Y.Kyselytsya
Add discovery	30. July 2007	Y.Kyselytsya
Add FT1.2 protocol description	07. October 2008	Y.Kyselytsya

Contents

1. What is an *ObjectServer*?

The *ObjectServer* is a hardware component, which is connected to the KNX bus and represents it for the client as set of the defined “objects”. These objects are the server properties (called “items”), KNX datapoints (known as “communication objects” or as “group objects”) and KNX configuration parameters (Fig. 1). The communication between server and clients is based on the *ObjectServer* protocol that is normally encapsulated into some other communication protocol (e.g. FT1.2, IP, etc.).

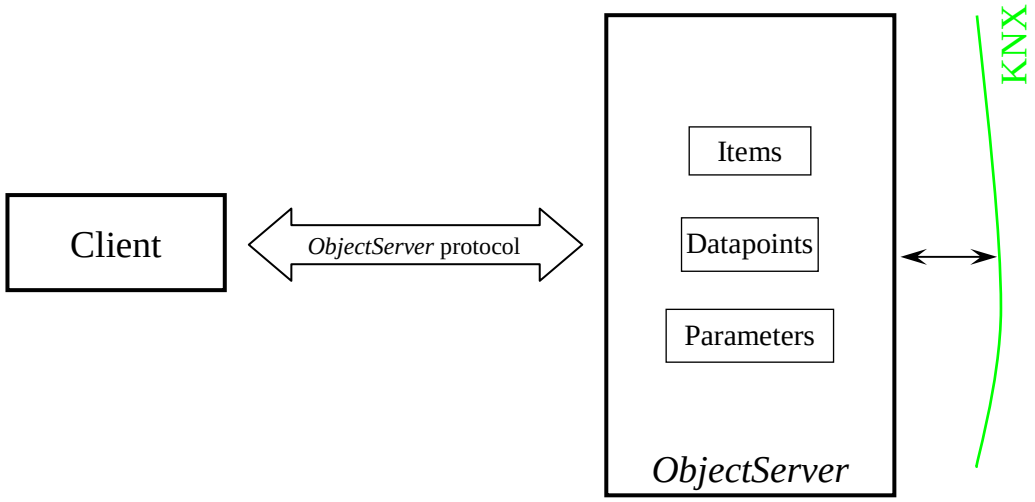

Figure 1. Communication between *ObjectServer* and Client

2. Communication protocol

How is mentioned above, the communication between the server and the client is based on an *ObjectServer* protocol and consists of the requests sent by client and the server responses. To inform the client about the changes of datapoint's value an indication is defined, which will be sent asynchronously from the server to the client. In this version of the protocol are defined following services:

- GetServerItem.Req/Res
- SetServerItem.Req/Res
- GetDatapointDescription.Req/Res
- GetDescriptionString.Req/Res
- GetDatapointValue.Req/Res
- DatapointValue.Ind
- SetDatapointValue.Req/Res
- GetParameterByte.Req/Res

2.1. GetServerItem.Req

This request is sent by the client to get one or more server items (properties). The data packet consists of four bytes:

Offset	Field	Size	Value	Description
+0	MainService	1	0xF0	Main service code
+1	SubService	1	0x01	Subservice code
+2	StartItem	1		ID of first item
+3	NumberOfItems	1		Maximal number of items to return

As response the server sends to the client the values of the all supported items from the range [StartItem … StartItem+NumberOfItems-1].

The defined item IDs are specified in appendix A.

2.2. GetServerItem.Res

This response is sent by the server as reaction to the GetServerItem request. If an error is detected during the request processing server send a negative response that has following format:

Offset	Field	Size	Value	Description
+0	MainService	1	0xF0	Main service code
+1	SubService	1	0x81	Subservice code
+2	StartItem	1		Index of bad item
+3	NumberOfItems	1	0x00	
+4	ErrorCode	1		Error code

The defined error codes are specified in appendix B.

If request can be successfully processed by the server it sends a positive response to the client that has following format:

Offset	Field	Size	Value	Description
+0	MainService	1	0xF0	Main service code
+1	SubService	1	0x81	Subservice code
+2	StartItem	1		As in request
+3	NumberOfItems	1		Number of items in this response
+4	First item ID	1		ID of first item
+5	First item data length	1		Data length of first item
+6	First item data	1-255		Data of first item
…	…		…	…
+N-2	Last item ID	1		ID of last item
+N-1	Last item data length	1		Data length of last item
+N	Last item data	1-255		Data of last item

2.3. SetServerItem.Req

This request is sent by the client to set the new value of the server item.

Offset	Field	Size	Value	Description
+0	MainService	1	0xF0	Main service code
+1	SubService	1	0x02	Subservice code
+2	StartItem	1		ID of first item to set
+3	NumberOfItems	1		Number of items in this request
+4	First item ID	1		ID of first item
+5	First item data length	1		Data length of first item
+6	First item data	1-255		Data of first item
…	…		…	…
+N-2	Last item ID	1		ID of last item
+N-1	Last item data length	1		Data length of last item
+N	Last item data	1-255		Data of last item

The defined item IDs are specified in appendix A.

2.4. SetServerItem.Res

This response is sent by the server as reaction to the SetServerItem request. If an error is detected during the request processing server send a negative response that has following format:

Offset	Field	Size	Value	Description
+0	MainService	1	0xF0	Main service code
+1	SubService	1	0x82	Subservice code
+2	StartItem	1		Index of bad item
+3	NumberOfItems	1	0x00	
+4	ErrorCode	1		Error code

The defined error codes are specified in appendix B.

If request can be successfully processed by the server it sends a positive response to the client that has following format:

Offset	Field	Size	Value	Description
+0	MainService	1	0xF0	Main service code
+1	SubService	1	0x82	Subservice code
+2	StartItem	1		As in request
+3	NumberOfItems	1	0x00	
+4	ErrorCode	1	0x00	

2.5. GetDatapointDescription.Req

This request is sent by the client to get the description(s) of the datapoint(s). The data packet consists of four bytes:

Offset	Field	Size	Value	Description
+0	MainService	1	0xF0	Main service code
+1	SubService	1	0x03	Subservice code
+2	StartDatapoint	1		ID of first datapoint
+3	NumberOfDatapoints	1		Maximal number of descriptions to return

As response the server sends to the client the descriptions of the all datapoints from the range [StartDatapoint … StartDatapoint+NumberOfDatapoints-1].

2.6. GetDatapointDescription.Res

This response is sent by the server as reaction to the GetDatapointDescription request. If an error is detected during the request processing, the server sends a negative response with the following format:

Offset	Field	Size	Value	Description
+0	MainService	1	0xF0	Main service code
+1	SubService	1	0x83	Subservice code
+2	StartDatapoint	1		As in request
+3	NumberOfDatapoints	1	0x00	
+4	ErrorCode	1		Error code

The defined error codes are specified in appendix B.

If request can be successfully processed by the server it sends a positive response to the client with the following format:

Offset	Field	Size	Value	Description
+0	MainService	1	0xF0	Main service code
+1	SubService	1	0x83	Subservice code
+2	StartDatapoint	1		As in request
+3	NumberOfDatapoints	1		Number of descriptions in this response
+4	First DP value type	1		Value type of first datapoint
+5	First DP config flags	1		Configuration flags of first datapoint
…	…		…	…
+N-1	Last DP value type	1		Value type of last datapoint
+N	Last DP config flags	1		Configuration flags of last datapoint

The defined types of the datapoint value are specified in appendix C.

The coding of the datapoint configuration flags is following:

Bit	Meaning	Value	Description
1 - 0	Transmit priority	00	System priority
		01	Alarm priority
		10	High priority
		11	Low priority
2	Datapoint communication	0	Disabled
		1	Enabled
3	Read from bus	0	Disabled
		1	Enabled
4	Write from bus	0	Disabled
		1	Enabled
5	Reserved	0	
6	Clients transmit request	0	Ignored
		1	Processed
7	Update on response	0	Disabled
		1	Enabled

2.7. GetDescriptionString.Req

This request is sent by the client to get the human-readable description string(s) of the datapoint(s). The data packet consists of four bytes:

Offset	Field	Size	Value	Description
+0	MainService	1	0xF0	Main service code
+1	SubService	1	0x04	Subservice code
+2	StartString	1		ID of first string
+3	NumberOfStrings	1		Maximal number of strings to return

As response server sends to the client the description strings of the all datapoints from the range [StartString … StartString+NumberOfStrings-1].

Note: This service is optional and could be not implemented in some servers.

2.8. GetDescriptionString.Res

This response is sent by the server as reaction to the GetDescriptionString request. If an error is detected during the processing of the request, the server sends a negative response with the following format:

Offset	Field	Size	Value	Description
+0	MainService	1	0xF0	Main service code
+1	SubService	1	0x84	Subservice code
+2	StartString	1		As in request
+3	NumberOfStrings	1	0x00	
+4	ErrorCode	1		Error code

The defined error codes are specified in appendix B.

If request can be successfully processed by the server it sends a positive response to the client with the following format:

Offset	Field	Size	Value	Description
+0	MainService	1	0xF0	Main service code
+1	SubService	1	0x84	Subservice code
+2	StartString	1		As in request
+3	NumberOfStrings	1		Number of strings in this response
+4	First DP description string	StrLen		Description string of first datapoint
…	…		…	…
+N	Last DP description string	StrLen		Description string of last datapoint

2.9. GetDatapointValue.Req

This request is sent by the client to get the value(s) of the datapoint(s). The data packet consists of four bytes:

Offset	Field	Size	Value	Description
+0	MainService	1	0xF0	Main service code
+1	SubService	1	0x05	Subservice code
+2	StartDatapoint	1		ID of first datapoint
+3	NumberOfDatapoints	1		Maximal number of datapoints to return

As response server sends to the client the values of the all datapoints from the range [StartDatapoint … StartDatapoint+NumberOfDatapoints-1].

2.10. GetDatapointValue.Res

This response is sent by the server as reaction to the GetDatapointValue request. If an error is detected during the processing of the request, the server sends a negative response with the following format:

Offset	Field	Size	Value	Description
+0	MainService	1	0xF0	Main service code
+1	SubService	1	0x85	Subservice code
+2	StartDatapoint	1		Index of the bad datapoint
+3	NumberOfDatapoints	1	0x00	
+4	ErrorCode	1		Error code

The defined error codes are specified in appendix B.

If request can be successfully processed by the server, it sends a positive response to the client with the following format:

Offset	Field	Size	Value	Description
+0	MainService	1	0xF0	Main service code
+1	SubService	1	0x85	Subservice code
+2	StartDatapoint	1		As in request
+3	NumberOfDatapoints	1		Number of datapoints in this response
+4	First DP ID	1		ID of first datapoint
+5	First DP state/length	1		State/length byte of first datapoint
+6	First DP value	1-14		Value of first datapoint
…	…		…	…
+N-2	Last DP ID	1		ID of last datapoint
+N-1	Last DP state/length	1		State/length byte of last datapoint
+N	Last DP value	1-14		Value of last datapoint

The state/length byte is coded as follow:

Bit	Meaning	Value	Description
7	Update flag	0	Value was not updates
		1	Value is updated from bus
6	Data request flag	0	Idle/response
		1	Data request
5 - 4	Transmission status	00	Idle/OK
		01	Idle/error
		10	Transmission in progress
		11	Transmission request
3-0	Value length	1-14	Length in bytes of datapoint value

The KNX datapoints with the length less than one byte are coded into the one byte value as folow:

	7	6	5	4	3	2	1	0
1-bit:	0	0	0	0	0	0	0	x
2-bits:	0	0	0	0	0	0	x	x
3-bits:	0	0	0	0	0	x	x	x
4-bits:	0	0	0	0	x	x	x	x
5-bits:	0	0	0	x	x	x	x	x
6-bits:	0	0	x	x	x	x	x	x
7-bits:	0	x	x	x	x	x	x	x

2.11. DatapointValue.Ind

This indication is sent asynchronously by the server if the datapoint(s) value is changed and has following format:

Offset	Field	Size	Value	Description
+0	MainService	1	0xF0	Main service code
+1	SubService	1	0xC1	Subservice code
+2	StartDatapoint	1		ID of first datapoint
+3	NumberOfDatapoints	1		Number of datapoints in this indication
+4	First DP ID	1		ID of first datapoint
+5	First DP state/length	1		State/length byte of first datapoint
+6	First DP value	1-14		Value of first datapoint
…	…		…	…
+N-2	Last DP ID	1		ID of last datapoint
+N-1	Last DP state/length	1		State/length byte of last datapoint
+N	Last DP value	1-14		Value of last datapoint

For the coding of the state/length byte see the description of the GetDatapointValue request.

For the coding of the datapoint value see the description of the GetDatapointValue response.

2.12. SetDatapointValue.Req

This request is sent by the client to set the new value(s) of the datapoint(s) or to request/transmit the new value on the bus. It also can be used to clear the transmission state of the datapoint.

Offset	Field	Size	Value	Description
+0	MainService	1	0xF0	Main service code
+1	SubService	1	0x06	Subservice code
+2	StartDatapoint	1		ID of first datapoint to set
+3	NumberOfDatapoints	1		Number of datapoints to set
+4	First DP ID	1		ID of first datapoint
+5	First DP cmd/length	1		Command/length byte of first datapoint
+6	First DP value	1-14		Value of first datapoint
…	…		…	…
+N-2	Last DP ID	1		ID of last datapoint
+N-1	Last DP cmd/length	1		Command/length byte of last datapoint
+N	Last DP value	1-14		Value of last datapoint

The command/length byte is coded as follow:

Bit	Meaning	Value	Description
7-4	Datapoint command	0000	No command
		0001	Set new value
		0010	Send value on bus
		0011	Set new value and send on bus
		0100	Read new value via bus
		0101	Clear datapoint transmission state
		0110	Reserved
		…	
		1111	Reserved
3-0	Value length	1-14	Length in bytes of datapoint value

The datapoint value length must match with the value length, which is selected in the ETS project database.

The value length “zero” is acceptable and means: “no value in frame”. It can be used for instance to clear the transmission state of the datapoint or to send the current datapoint value on the bus or similar.

2.13. SetDatapointValue.Res

This response is sent by the server as reaction to the SetDatapointValue request. If an error is detected during the processing of the request, the server sends a negative response with the following format:

Offset	Field	Size	Value	Description
+0	MainService	1	0xF0	Main service code
+1	SubService	1	0x86	Subservice code
+2	StartDatapoint	1		Index of bad datapoint
+3	NumberOfDatapoints	1	0x00	
+4	ErrorCode	1		Error code

The defined error codes are specified in appendix B.

If request can be successfully processed by the server, it sends a positive response to the client with the following format:

Offset	Field	Size	Value	Description
+0	MainService	1	0xF0	Main service code
+1	SubService	1	0x86	Subservice code
+2	StartDatapoint	1		As in request
+3	NumberOfDatapoints	1	0x00	
+4	ErrorCode	1	0x00	

2.14. GetParameterByte.Req

This request is sent by the client to get the parameter byte(s). A parameter is free-defined variable of the 8-bits length, which can be set and programmed by the **E**ngineering **T**ool **S**oftware (ETS). Up to 256 parameter bytes per server can be defined.

The data packet of the GetParameterByte request consists of four bytes:

Offset	Field	Size	Value	Description
+0	MainService	1	0xF0	Main service code
+1	SubService	1	0x07	Subservice code
+2	StartByte	1		Index of first byte
+3	NumberOfBytes	1		Maximal number of bytes to return

As response the server sends to the client the values of the all parameters from the range [StartByte … StartByte+NumberOfBytes-1].

2.15. GetParameterByte.Res

This response is sent by the server as reaction to the GetParameterByte request. If an error is detected during the request processing server send a negative response that has following format:

Offset	Field	Size	Value	Description
+0	MainService	1	0xF0	Main service code
+1	SubService	1	0x87	Subservice code
+2	StartByte	1		Index of the bad parameter
+3	NumberOfBytes	1	0x00	
+4	ErrorCode	1		Error code

The defined error codes are specified in appendix B.

If request can be successfully processed by the server it sends a positive response to the client that has following format:

Offset	Field	Size	Value	Description
+0	MainService	1	0xF0	Main service code
+1	SubService	1	0x87	Subservice code
+2	StartByte	1		As in request
+3	NumberOfBytes	1		Number of bytes in this response
+4	First byte	1		First parameter byte
…	…		…	…
+N	Last byte	1		Last parameter byte

3. Encapsulating of the *ObjectServer* protocol

The *ObjectServer* protocol has been defined to achieve the whole functionality also on the smallest embedded platforms and on the data channels with the limited bandwidth. As a result of this fact the protocol is kept very slim and has no connection management, like the connection establishment, user authorization, etc. Therefore it is advisable und mostly advantageous to encapsulate the *ObjectServer* protocol into some existing transport protocol to get a powerful solution for the easy access to the KNX datapoints and directly to the KNX bus.

3.1. FT1.2

The encapsulating of the *ObjectServer* protocol into the FT1.2 (known also as PEI type 10) protocol is simple and consists in the integration of the *ObjectServer* protocol frames into the FT1.2 frames as is shown in figure 2.

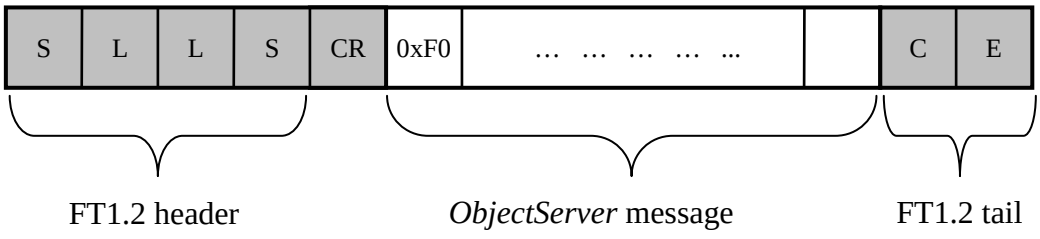

Figure 2. Integration of the *ObjectServer* message into the FT1.2 frame

The short description of the FT1.2 protocol can be found in appendix D.

3.2. KNXnet/IP

The clients that communicate over the KNXnet/IP protocol with the *ObjectServer* should use the “Core” services of the KNXnet/IP protocol to discovery the servers, to get the list of the supported services and to manage the connection. If the *ObjectServer* protocol is supported by the KNXnet/IP server, a service family with the ID=0xF0 is present in the device information block (DIB) “supported service families”. The same ID (0xF0) should be used by the client to set the “connection type” field of the connect request.

The *ObjectServer* communication procedure is like for the tunneling connection of the KNXnet/IP protocols (see the chapter 3.8.4 of the KNX specification for the details). The communication partners send the requests (ServiceType=0xF080) each other, which will be acknowledge (ServiceType=0xF081) by the opposite side. Each request includes the *ObjectServer* message (figure 3).

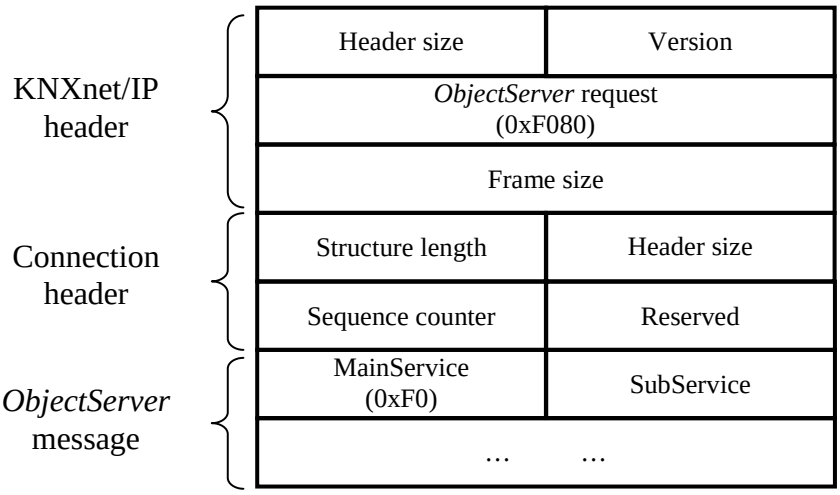

Figure 3. Integration of the *ObjectServer* message into the KNXnet/IP frame

3.3. TCP/IP

The TCP/IP provides the whole required functionality from connection management and maintenance to the data integrity. Therefore it is no auxiliary services or data should be implemented by the *ObjectServer*. The encapsulating of the *ObjectServer* protocol into the TCP/IP is simple and consists in the integration of the *ObjectServer* protocol frames as the TCP data.

Before the client is able to send the requests to the *ObjectServer* he must establish a TCP/IP connection to the IP address and the TCP port of *ObjectServer*.

The default value for the *ObjectServer* port is 12004 (decimal).

4. Discovery procedure

This chapter describes the possibilities to find the installed *ObjectServers* in the local network. This allows the clients to find and to select automatically a definite *ObjectServer* for the communication, alternatively to the manual input from the user. Currently only one discovery procedure is supported, which is based on the KNXnet/IP discovery algorithm. The next chapter describes it briefly. For the full description of the KNXnet/IP discovery algorithm please refer to the KNX handbook Volume 3.8.

4.1. KNXnet/IP discovery algorithm

The KNXnet/IP discovery procedure works in the way showed on the figure 4. The client, which is looking for the installed *ObjectServers*, sends a search request via the multicast on the predefined multicast address 224.0.23.12 and port 3671 (decimal). The *ObjectServers* sends back a search response with the device information block (DIB), which contains among other things the information about the support of the *ObjectServer* protocol.

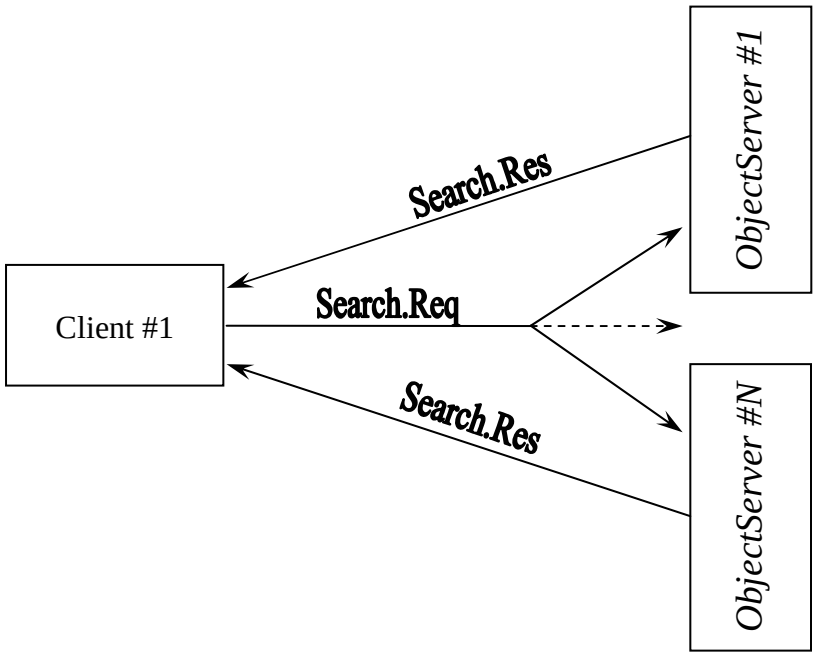

Figure 4. KNXnet/IP discovery

The search request has the length of 14 bytes and its format is presented on figure 5. Most fields are fixed, the client should fill only the fields “IP address” and “IP port”. These fields are used by the *ObjectServer* as destination IP address and port for the search response. For fields, which are longer than one byte, the big-endian format is applied.

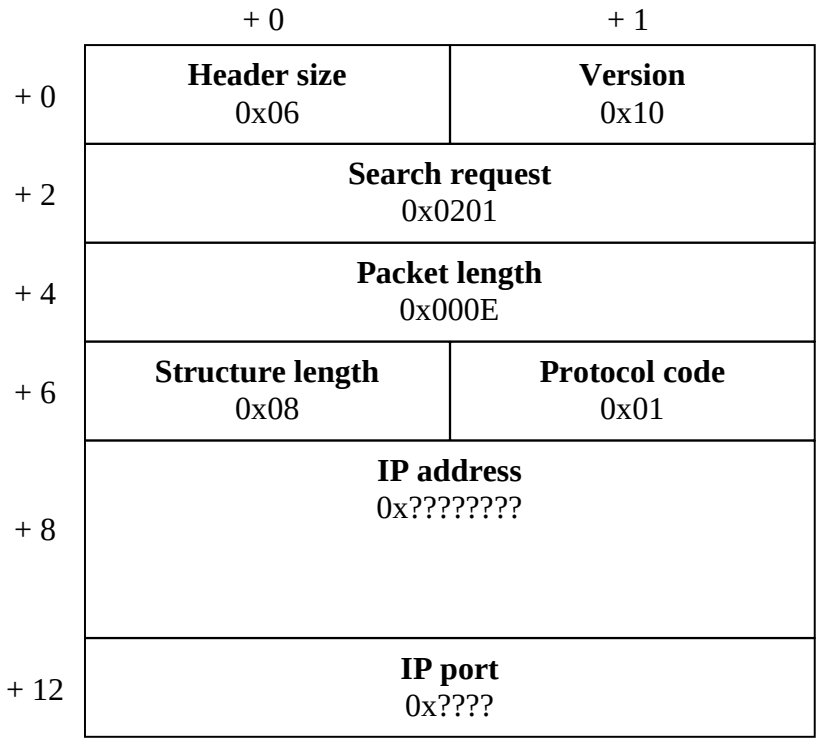

Figure 5. Structure of the Search.Req packet

The search response from the *ObjectServer* has in the version 1.0 of the protocol the length of 84 bytes and its format is presented on figure 6. The support of the *ObjectServer* protocol by the device is indicated through the existence of the manufacturer DIB at the offset +76 bytes in the packet. This manufacturer DIB has the length of 8 bytes.

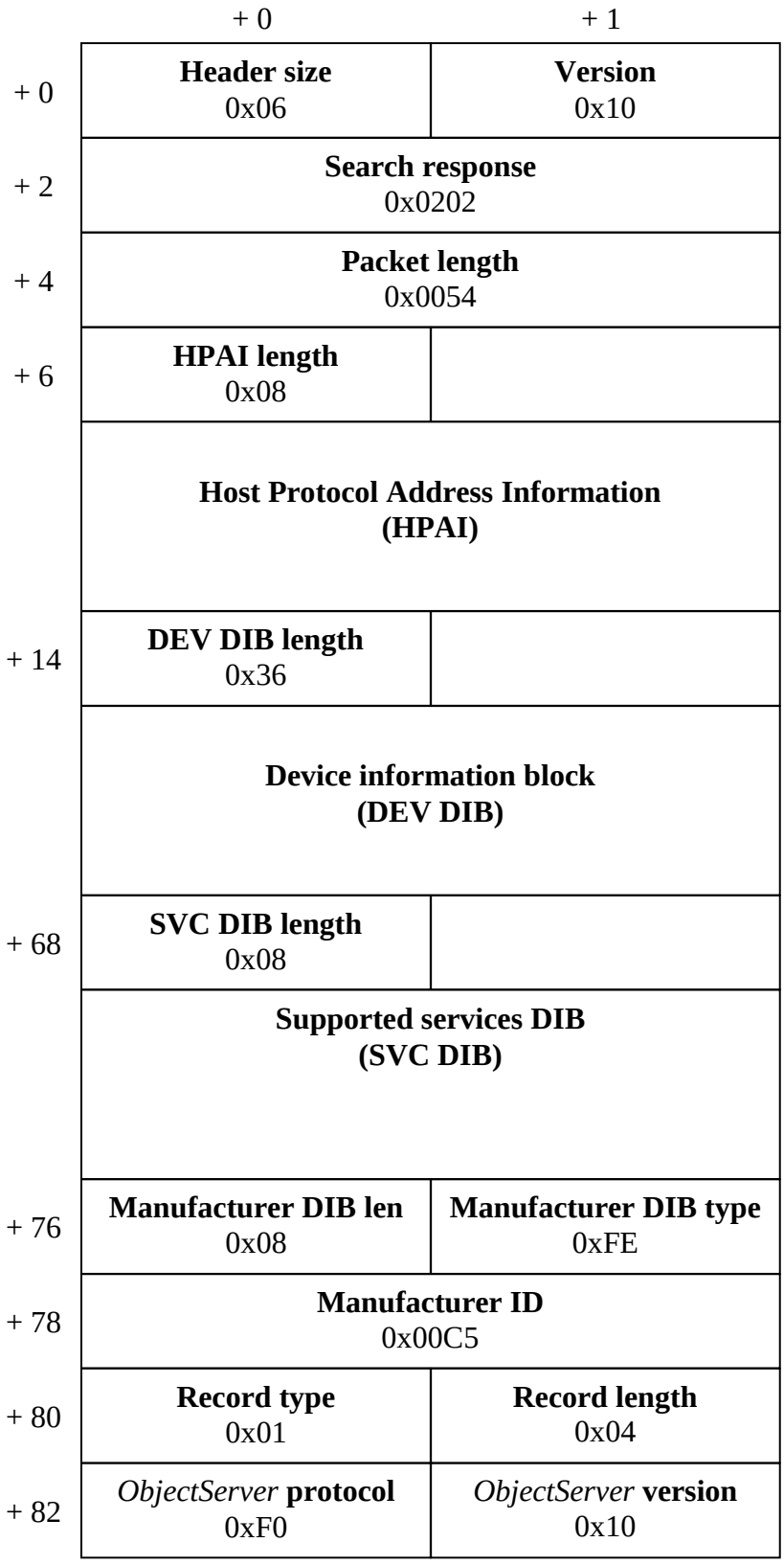

Figure 6. Structure of the Search.Res packet

Appendix A. Item IDs

ID	Item	Size in bytes	Access
1	**Hardware type** Can be used to identify the hardware type. Coding is manufacturer specific. Is mapped to property PID_HARDWARE_TYPE in device object.	6	Read only
2	**Hardware version** Version of the ObjectServer hardware Coding Ex.: 0x10 = Version 1.0	1	Read only
3	**Firmware version** Version of the ObjectServer firmware Coding Ex.: 0x10 = Version 1.0	1	Read only
4	**KNX manufacturer code DEV** KNX manufacturer code of the device, not modified by ETS. Is mapped to property PID_MANUFACTURER_ID in device object.	2	Read only
5	**KNX manufacturer code APP** KNX manufacturer code loaded by ETS. Is mapped to bytes 0 and 1 of property PID_APPLICATION_VER in application object.	2	Read only
6	**Application ID (ETS)** ID of application loaded by ETS. Is mapped to bytes 2 and 3 of property PID_APPLICATION_VER in application object.	2	Read only
7	**Application version (ETS)** Version of application loaded by ETS. Is mapped to byte 4 of property PID_APPLICATION_VER in application object.	1	Read only
8	**Serial number** Serial number of device. Is mapped to property PID_SERIAL_NUMBER in device object.	6	Read only
9	**Time since reset [ms]**	4	Read only
10	**Bus connection state**	1	Read only
11	**Maximal buffer size**	2	Read only
12	**Length of description string**	2	Read only
13	**Baudrate** Values: “0” – unknown, “1” – 19200, “2” – 115200	1	Read/Write
14	**Current buffer size**	2	Read/Write

Attention: For values, which are longer than one byte, the big-endian format is applied.

Appendix B. Error codes

Error code	Description
0	No error
1	Internal error
2	No item found
3	Buffer is too small
4	Item is not writeable
5	Service is not supported
6	Bad service parameter
7	Wrong datapoint ID
8	Bad datapoint command
9	Bad length of the datapoint value
10	Message inconsistent

Appendix C. Datapoint value types

Type code	Value size
0	1 bit
1	2 bits
2	3 bits
3	4 bits
4	5 bits
5	6 bits
6	7 bits
7	1 byte
8	2 bytes
9	3 bytes
10	4 bytes
11	6 bytes
12	8 bytes
13	10 bytes
14	14 bytes

Appendix D. FT1.2 protocol

The FT1.2 transmission protocol is based on the international standard IEC 870-5-1 and IEC 870-5-2 (DIN 19244). As the hardware interface for the transmission is the Universal Asynchronous Receiver Transmitter (UART) used. The frame format for the FT1.2 protocol is fixed to the 8 data bits, 1 stop bit and even parity bit. The default communication speed is 19200 Baud.

D.1. Communication procedure

The typical communication procedure between the host and the *ObjectServer* is shown on figure 7.

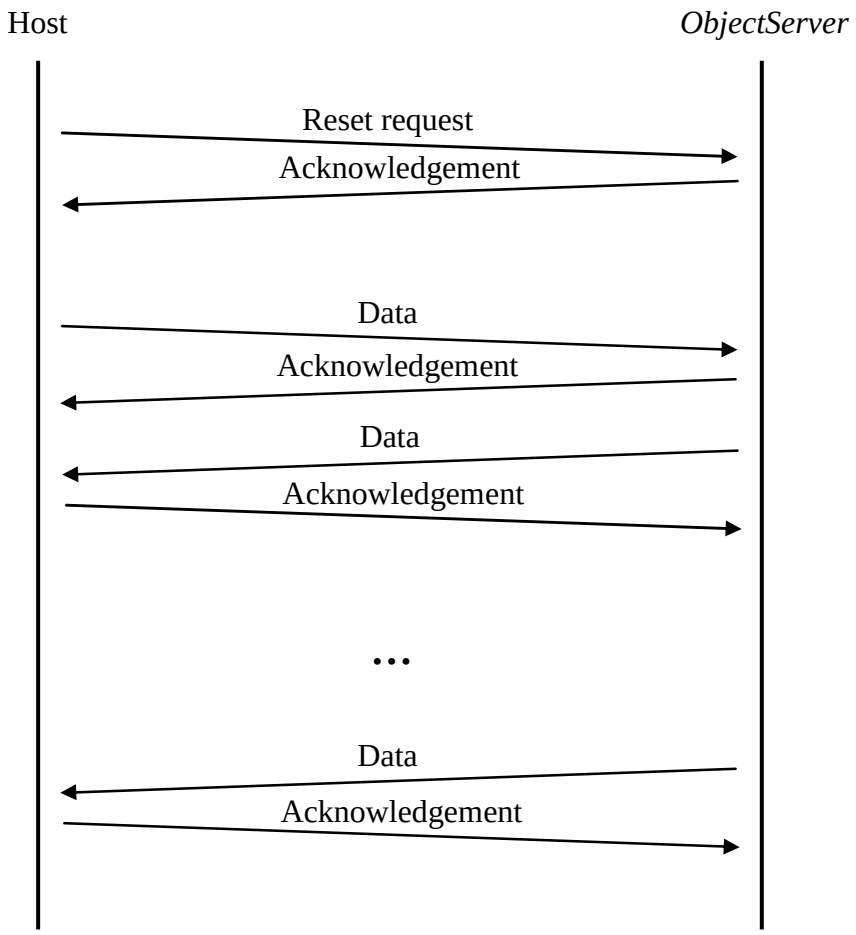

Figure 7. Typical communication procedure

In chapter D.3 is presented an example of the communication between the host and the *ObjectServer*.

D.2. Frame format

Three frame types are defined by the FT1.2 protocol .

The first one is the positiv acknowledgement frame and consists only one byte of the value 0xE5.

The second frame type is 4 bytes length and is used for the reset request and reset indication messages (Fig.8).

Reset.Req:	0x10	0x40	0x40	0x16

Reset.Ind:	0x10	0xC0	0xC0	0x16

Figure 8. Structure of the Reset.Req and Reset.Ind frames

The third frame type is variable length and used for the data messages. The frame structure is presented on figure 9.

0x68	**L**	**L**	0x68	**CR**	data	**C**	0x16

Figure 9. Structure of the data message

The both fields L contain the length of the data in this frame.

The field CR specifies the control byte of the frame. Its value is 0x73 for all odd frames after reset request sent by the host and 0x53 for the even frames. In the opposite direction (from *ObjectServer* to host) the control byte is 0xF3 for the odd frames and 0xD3 for the even frames.

The field C contains the checksum of the frame and is the arithmetic sum disregarding overflows (modulo 256) over all data and control byte.

D.3. Communication example

Host → *ObjectServer*: Reset Request

{0x10 0x40 0x40 0x16}

***ObjectServer* → Client: Acknowledgement**

{0xE5}

Host → *ObjectServer*: GetServerItem.Req (Firmware version)

{0x68 0x05 0x05 0x68 0x73 0xF0 0x01 0x03 0x01 0x68 0x16}

***ObjectServer* → Client: Acknowledgement**

{0xE5}

***ObjectServer* → Client: GetServerItem.Res (Firmare version)**

{0x68 0x08 0x08 0x68 0xF3 0xF0 0x81 0x03 0x01 0x03 0x01 0x10 0x7C 0x16}

Host → *ObjectServer*: Acknowledgement

{0xE5}

Host → *ObjectServer*: GetServerItem.Req (Serial number)

{0x68 0x05 0x05 0x68 0x53 0xF0 0x01 0x08 0x01 0x4D 0x16}

***ObjectServer* → Client: Acknowledgement**

{0xE5}

***ObjectServer* → Client: GetServerItem.Res (Serial number)**

{0x68 0x0D 0x0D 0x68 0xD3 0xF0 0x81 0x08 0x01 0x08 0x06 0x00 0xC5 0x08 0x02 0x00 0x00 0x2A 0x16}

Host → *ObjectServer*: Acknowledgement

{0xE5}