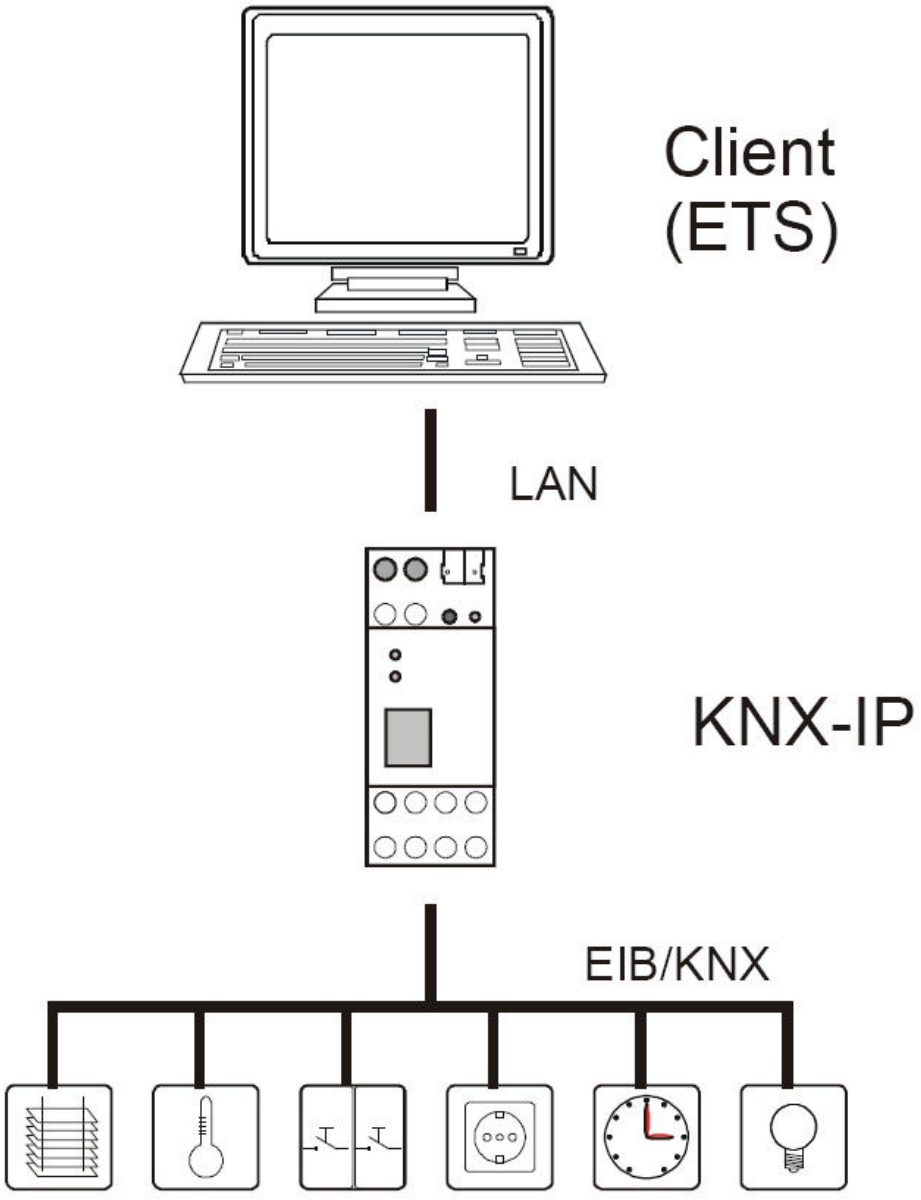
KNX IP Object Server with Power over Ethernet (PoE)

Interface and ObjectServer between LAN and EIB/KNX-Bus

(Quelle: © EIBMARKT GmbH, Reference No.: N000400)

Application area

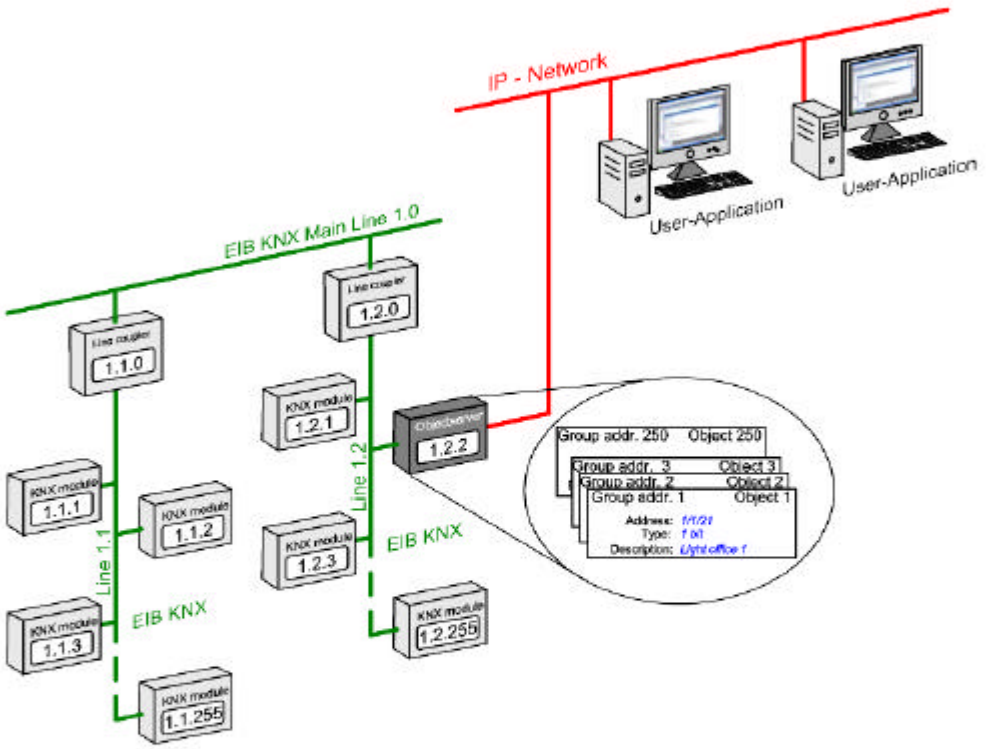
The KNX IP OBJECT SERVER is used as Interface for connecting to KNX/EIB both on telegram level (KNXnet/IP Tunneling) and on data point level (KNX Application Layer). It is possible to connect to KNX/EIB-Bus everywhere over LAN. Bus connection over the Internet with KNX IP OBJECT SERVER is also possible.
There are two ways to assign an IP-address to the KNX IP OBJECT SERVER: get the IP-address from a DHCP-server or configure it with the ETS (as ETS parameter). It requires an external 12 V to 24 V power supply (AC or DC) or can alternatively be powered via Power-over-Ethernet (IEEE 802.3af).



Item number: N000400

KNX IP Object Server (eibmarkt gmbh)

Object server
Process of up to 250 data points of KNX
up to 250 network objects



Note:

The Power over Ethernet supply is standardized acc. to Standard PoE (IEEE) and integrated via RJ45.

As an alternative, power supply can be provided by means of connecting terminals at the upper side using a separate power supply unit (12-24V DC) or transformer (12-24V AC).

Parallel power must be avoided!

Projektversion			Datum	Name	Bezeichnung: **Objektserver**	
Datum	Name	gez.:		O. Schulz	Projekt: **Anwendungen IP-Schnisttstellen**	Blattzahl: 1
		gepr.:		M. Labahn		
		EIBMARKT GmbH Friedensstrasse 54 08523 Plauen			Zeichnungs-Nr.: ipschnittstellen_en.spl	Blatt-Nr.: 1

For an increasing number of devices, such as in the areas of multimedia and security technology, the exchange of control information with the building automation is of significance. However, for certain devices it is preferable to not access the bus directly. Instead, a connection can be established to the KNX via the Ethernet. Communication via the Ethernet is particularly interesting for devices that are already equipped with a network port. If the protocol stack for TCP/UDP/IP already exists in the operating system (like Linux or Windows® CE), applications can communicate with other devices via the Ethernet with little additional effort. If you were to use KNXnet/IP Tunnelling or Routing as a solution, the devices would be able to access the KNX network but would still have to generate and interpret KNX telegrams. It is far simpler for the KNX/IP interface to take over this task. In this case, the KNX/IP interface assumes the roll of a KNX end device. This means, that the KNX communication software of the device manages data-points in order to assign received telegrams to the according communication objects. The device independently creates and sends group telegrams, too. With the ETS (Engineering Tool Software), the group objects can be configured. In the ETS, the device appears as a conventional participant of the bus. Over the parameter dialog, data types of the group objects are configured. After that, group addresses can be set as customary. A client (for example visualization) can access data-points over TCP/IP or UDP/IP without knowledge about the KNX syntax. One client addresses a data-points over the index of that object. If group addresses in the KNX network should change at any time, the Interface can be updated without problems by an ETS-Download. There is no need to update the configuration of the clients. The interface supports up to 250 group objects and for object there is separate memory reserved. Values of the communication objects will be updated automatically when they are addressed by a telegram even if there is no client connected. This is advantageous, for example for a visualization device. If it starts or wakes up, it can read the data image from the interface without stressing the KNX bus and without a noteworthy delay.

Note:

The Power over Ethernet supply is standardized acc. to Standard PoE (IEEE) and integrated via RJ45.As an alternative, power supply can be provided by means of screw connecting terminals at the upper side using a separate power supply unit (12-24V DC) or transformer (12-24V AC).

Technical data

Electrical safety
- Protection (EN 60529): IP 20
- Safety extra low voltage SELV DC 24 V

EMV requirements
- Complies EN 61000-6-2, EN 61000-6-3 and EN 50090-2-2

Environmental requirements
- Ambient temp. operating: - 5 ... + 45 °C
- Ambient temp. Non-op.: - 25 ... + 70 °C
- Rel. humidity (non-condensing): 5 % ... 93 %

Certification
- EIB / KNX

CE norm
- Complies with the EMC regulations (residential and functional buildings) and low voltage directive

Physical specifications
- Housing: Plastic
- Rail mounted device, depth 2 units
- Weight: approx. 100 g
- Fire load: approx. 1000 kJ

Operating controls
- Learning key for EIB/KNX

Indicators
- Learning-LED (red)
- Signal-LED (green) for EIB/KNX
- Signal-LED (green) for LAN

Ethernet
- 10BaseT (10Mbit/s)
- Supported internet protocols ARP, ICMP, IGMP, UDP/IP and DHCP

Power supply
- External supply 12-24V AC / 12-30V DC
- Alternative: Power over Ethernet
- Power consumption: < 800 mW

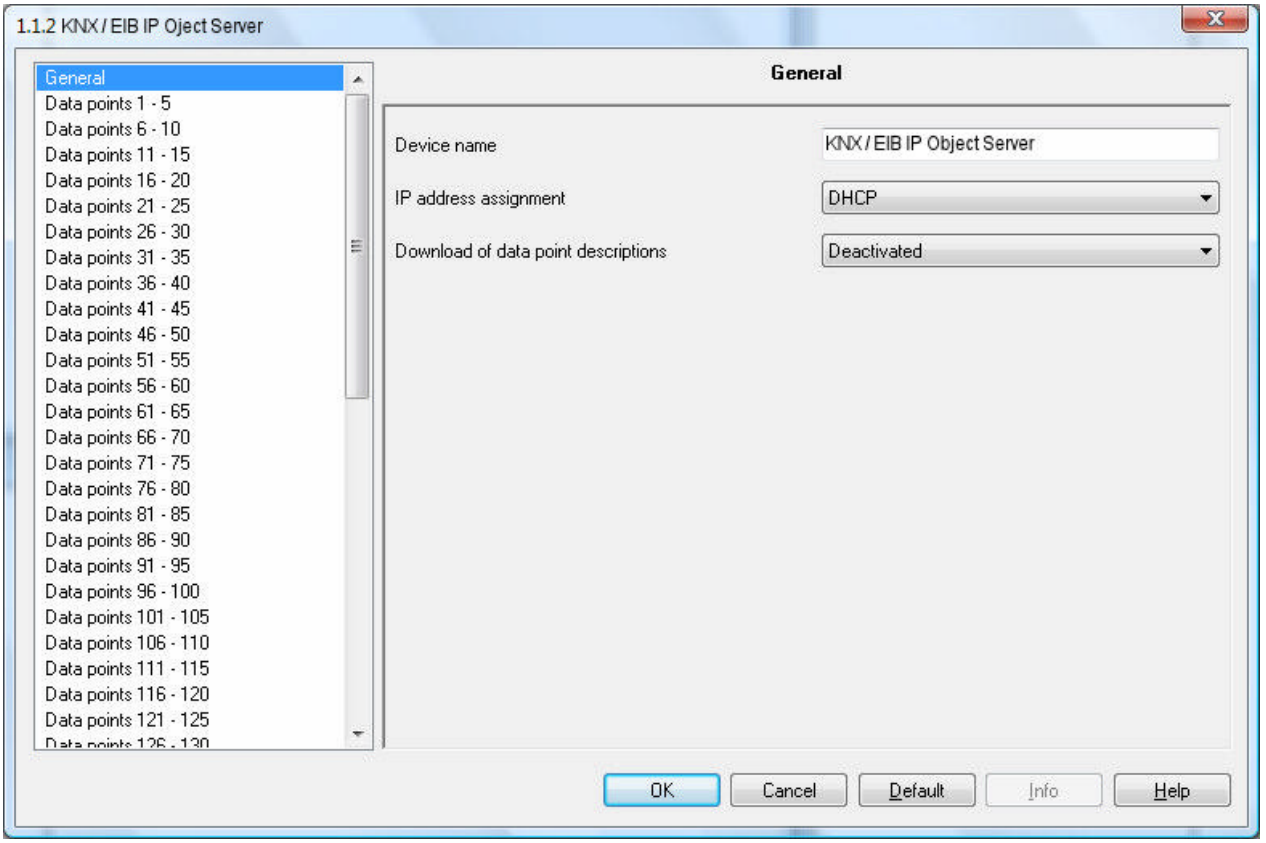
Connectors
- EIB/KNX connection terminal
- LAN RJ-45 socket
- Screw connector for power supply

Operating and installation manual

ETS Database

With the ETS, following parameters can be set:

General:



Device name:
It's possible to assign any name for the KNX IP OBJECT SERVER. The device name should be significant (e.g. Data points 1st
floor), because the name is used when searching for devices.

IP-Address assignment:

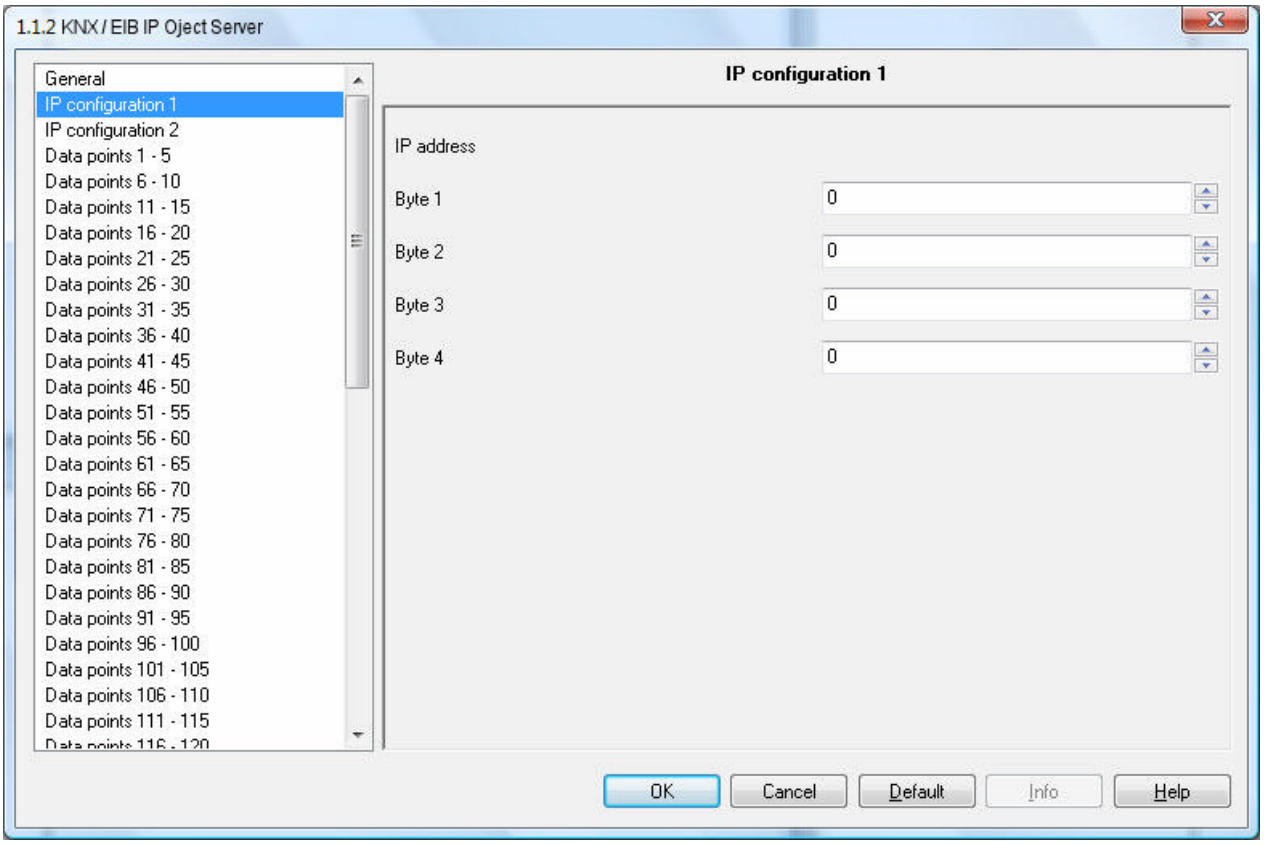
DHCP: The device can get its IP-address from a DHCP-server automatically. There must be a DHCP-server in the LAN in order to use this functionality (e.g. this can be a DSL-router with a DHCP-server integrated).

Manually: In this case, the IP-address, the subnetwork and the IP-address of the gateway have to be entered.

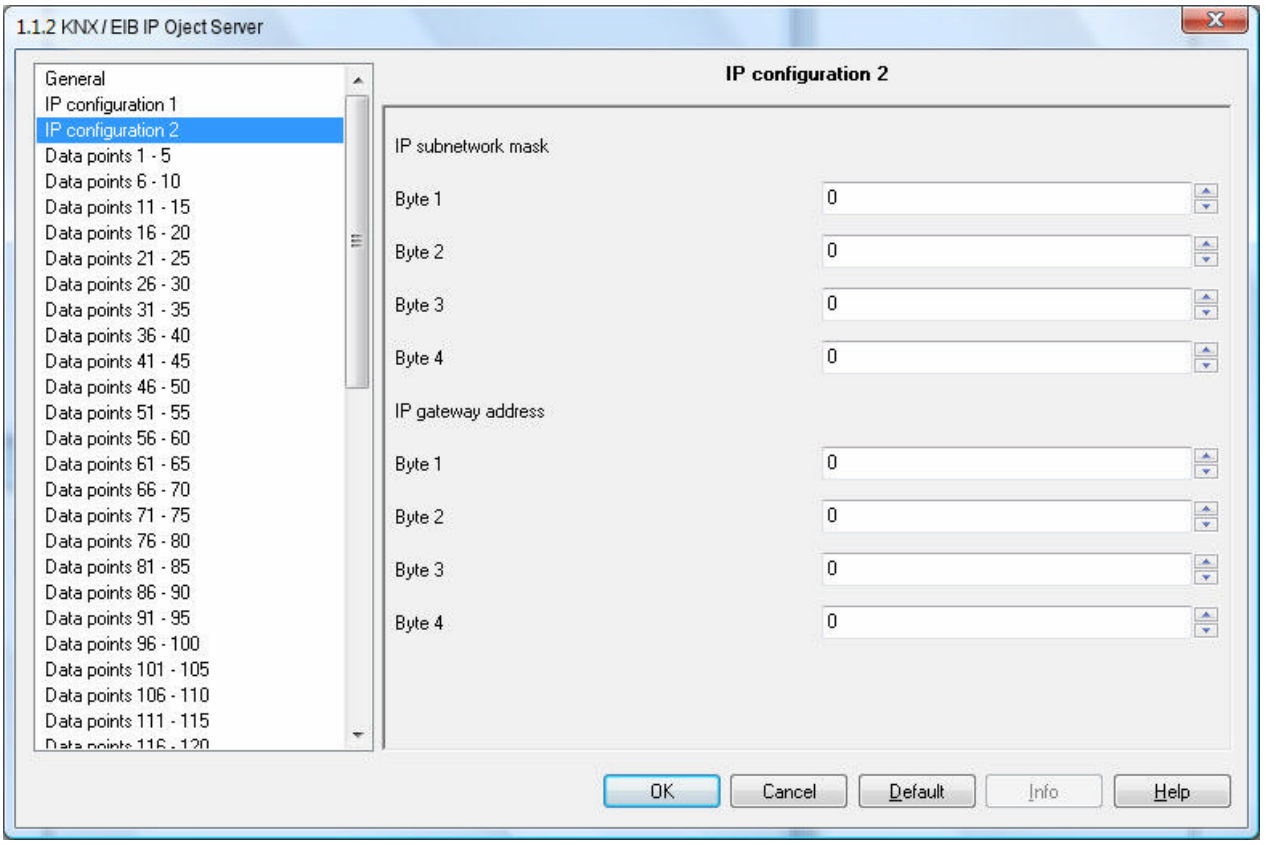
Download of the data point-descriptions:
If this parameter is activated, then descriptions will be written into the device during download.

Attention:
When changing this parameter, all datapoint-descriptions will be exchanged in the ETS.

IP-Configuration:



IP-Address:
Enter the IP-Address of the KNX IP OBJECT SERVER here.



IP-Subnetwork:
Enter the subnetwork mask here. The mask helps the device to discover, whether the communication partner is the local network. If the partner is not in the local network, then the device sends the IP telegrams not directly to the partner but to the gateway, which forwards the telegrams to the device.

IP-Gateway-Address:
Enter the IP-Address of the gateway here.
Hint: Leave 0.0.0.0 there, if the KNX IP OBJECT SERVER ought to be used only in the local LAN.

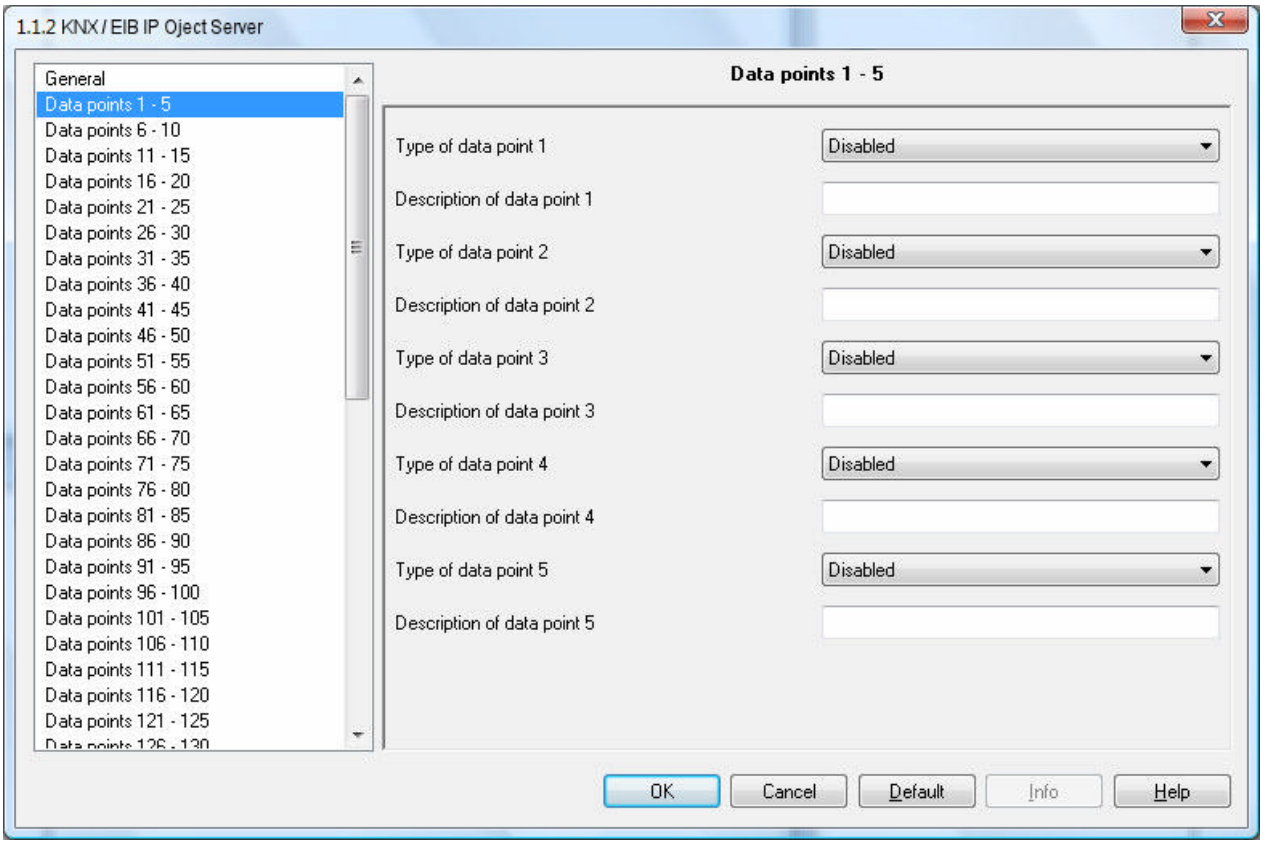
Example for IP-Address assignment:
Over a PC the KNX IP OBJECT SERVER shall be accessed.

IP-Adress of the PC: 192.168.1.30
Subnetwork of the PC: 255.255.255.0

The KNX IP OBJECT SERVER is located in the same local LAN therefore it use the same subnetwork mask. Because of the used subnetwork the IP-address assignment is limited, only addresses with format 192.168.1.xx can be assigned the device, xx stands for the range 1-255 (without 30, because already assigned to PC). Be careful not to use one IP address more then once.

IP-adress KNX IP OBJECT SERVER: 192.168.1.31
Subnetwork KNX IP OBJECT SERVER: 255.255.255.0

Data points:
Up to 250 data points can be parameterized. Each data point gets a group address, in order to send to the bus. More then one group address can be set for one object for receiving.



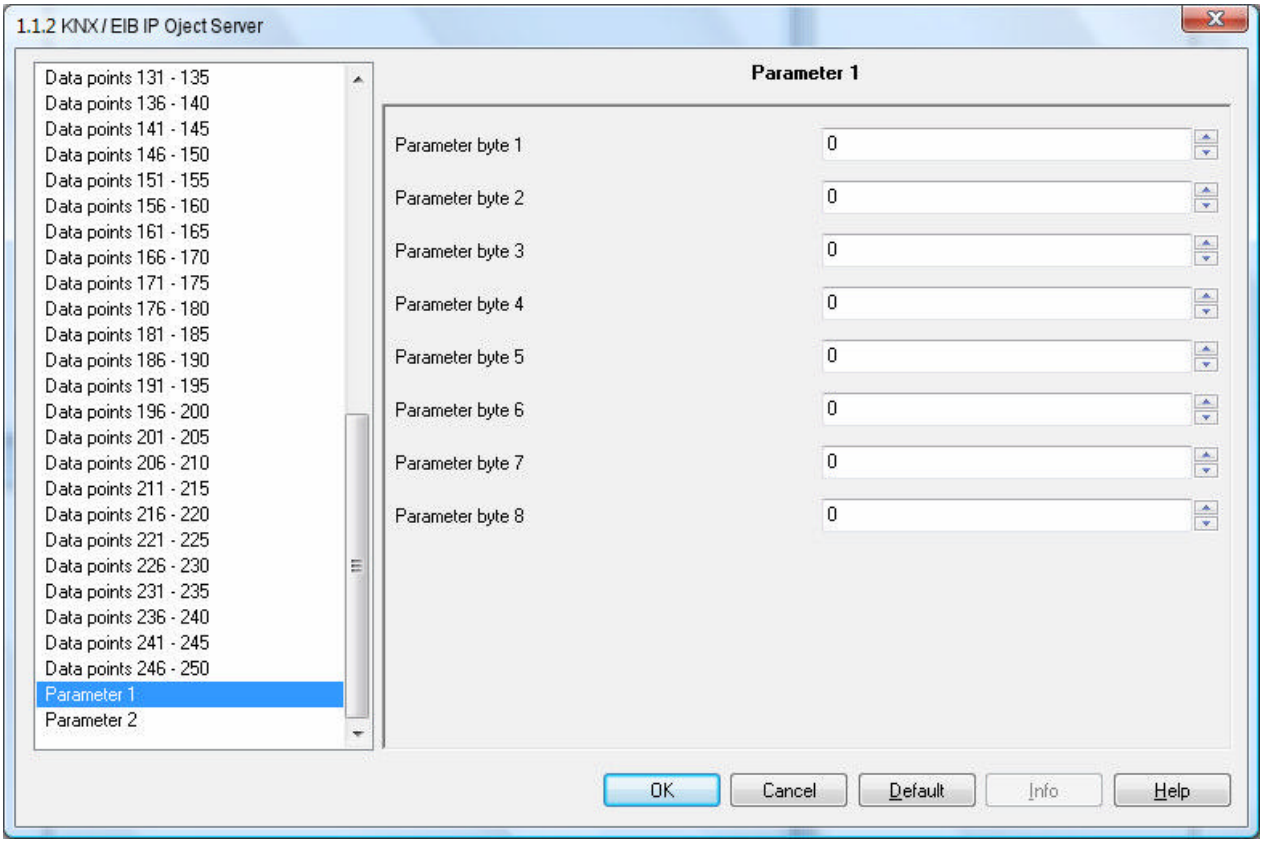
Type of the data point:
For each data point the type can be set. Following data points are available:

Bit	Byte
1	1
2	2
3	3
4	4
5	6
6	8
7	10
	14

Description of data point:
For each data point a short description (max. 30 characters) can be entered, in order to check the usage of the data point later, when device is running.

Attention:
Only if the parameter „Download of data point description“ is activated, the description will be written into the device. If you change the parameter „Download of data point description“, all data point descriptions will be exchanged in the ETS.

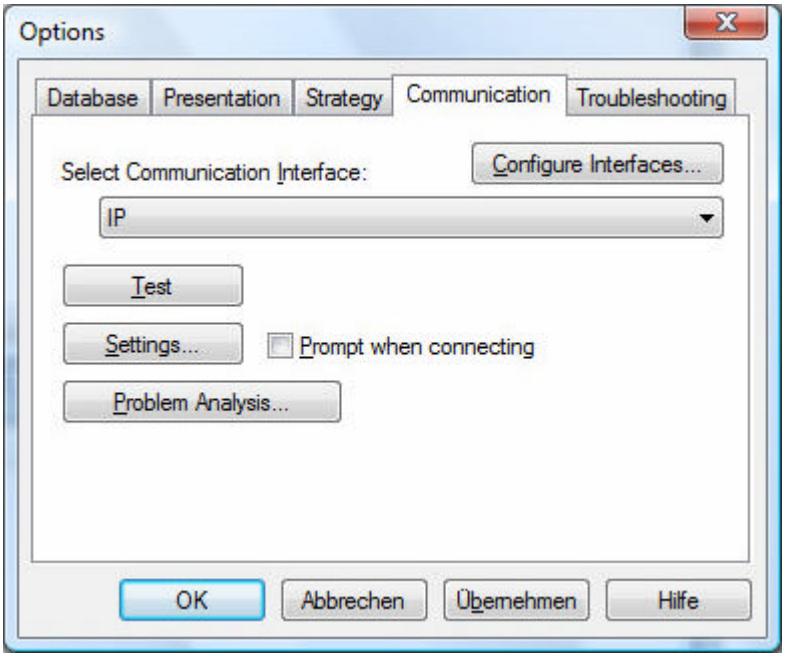
Parameter:
KNX IP OBJECT SERVER has 16 bytes of free parameters, which can be readout over Ethernet. The client has to evaluate the bytes.



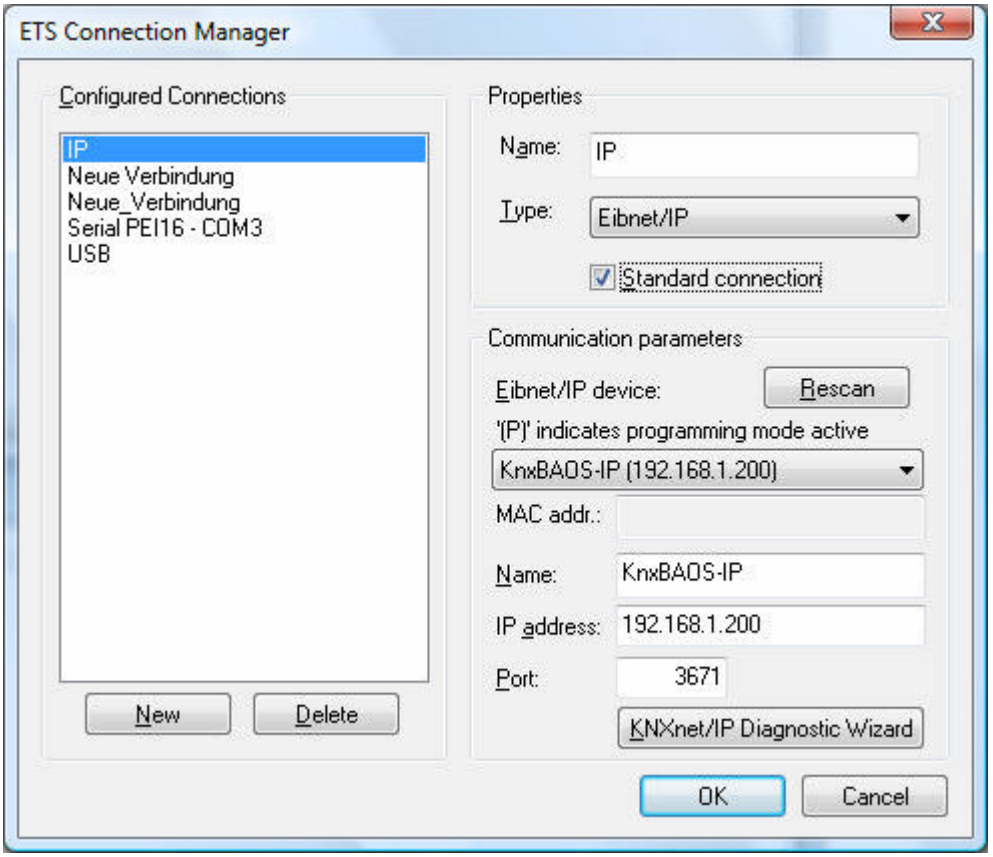
Parameter Byte: For each of the 16 bytes a value between 0 and 255 can be entered.

ETS Connection Manager

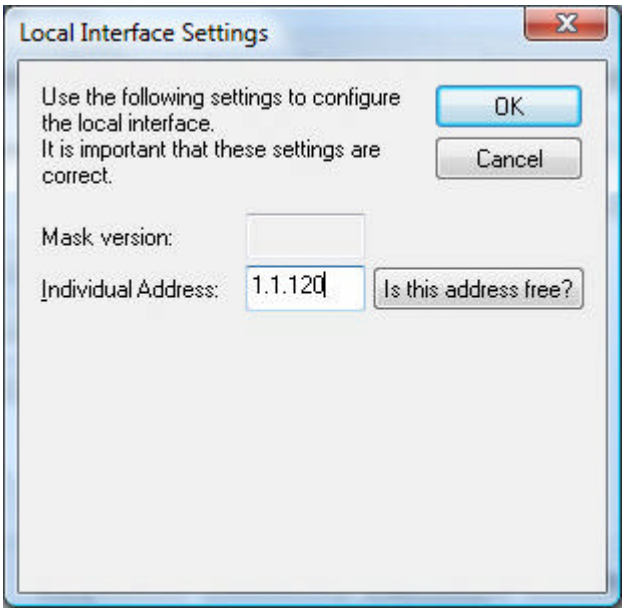
If the IP-configuration of the KNX IP OBJECT SERVER is valid, then this device can act as an interface to KNX/EIB. In order to use this function, go to the ETS (version 3.0c or higher), take Extras/Options then the tab communication:



Click on *Configure Interfaces…* to open the ETS connection manager.
Create a new connection of type EIBnet/IP. The ETS automatically starts searching for KNXnet/IP devices. All detected devices should be shown. The preferred device has to be selected.



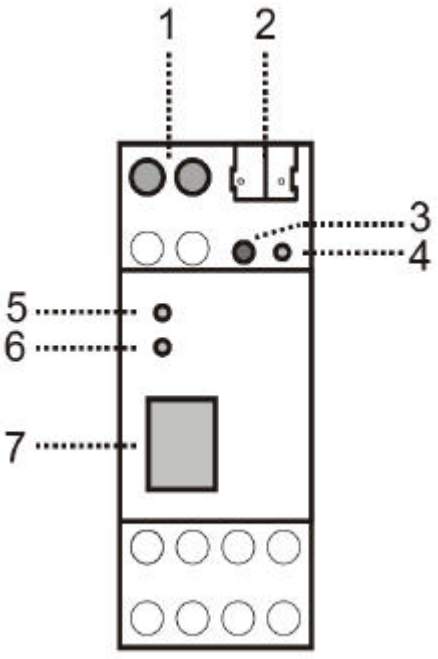
In order to access the KNX/EIB, the KNX IP OBJECT SERVER needs a second physical address. The second physical address is only used for bus access and can be adjusted separately:
In ETS menu go to Extras/Options and choose Communication. Then choose the already created connection. After a click on *Settings…* the following dialog is opened:



Enter here an unused physical address. The address shall be valid in the line where the interface is installed. A dummy device may be created in the ETS-project to reserve this address.

Installation and Connection

The KNX IP OBJECT SERVER is designed for installation on DIN rail with a width of 2 units (36mm). It has the following display and control elements:



1: Connector for external power supply
(12 V to 24 V AC or 12 V to 30V DC)

2: Connector for KNX/EIB with a bus terminal

3: Learn key

4: Learn LED (red)

5: LED (green):
- Lights up to indicate bus voltage on KNX/EIB
- Flashes to indicate telegram traffic

6: LED (green):
- Lights up to indicate Ethernet connection
- Flashes to indicate telegram traffic

7: RJ 45 socket for connecting an Ethernet patch cable

An external power supply only needs to be connected if the switch in use does not support Power-over-Ethernet.

KNX *ObjectServer* protocol
Version 1.0

Contents

1. WHAT IS AN OBJECTSERVER? 13

2. COMMUNICATION PROTOCOL 13

2.1. GETSERVERITEM.REQ 14

2.2. GETSERVERITEM.RES 14

2.3. SETSERVERITEM.REQ 15

2.4. SETSERVERITEM.RES 15

2.5. GETDATAPOINTDESCRIPTION.REQ 16

2.6. GETDATAPOINTDESCRIPTION.RES 16

2.7. GETDESCRIPTIONSTRING.REQ 17

2.8. GETDESCRIPTIONSTRING.RES 17

2.9. GETDATAPOINTVALUE.REQ 18

2.10. GETDATAPOINTVALUE.RES 18

2.11. DATAPOINTVALUE.IND 20

2.12. SETDATAPOINTVALUE.REQ 20

2.13. SETDATAPOINTVALUE.RES 21

2.14. GETPARAMETERBYTE.REQ 22

2.15. GETPARAMETERBYTE.RES 22

3. ENCAPSULATING OF THE OBJECTSERVER PROTOCOL 23

3.1. FT1.2 23

3.2. KNXNET/IP 23

3.3. TCP/IP 24

4. DISCOVERY PROCEDURE 24

4.1. KNXNET/IP DISCOVERY ALGORITHM 24

APPENDIX A. ITEM IDS 27

APPENDIX B. ERROR CODES 28

APPENDIX C. DATAPOINT VALUE TYPES 28

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.).

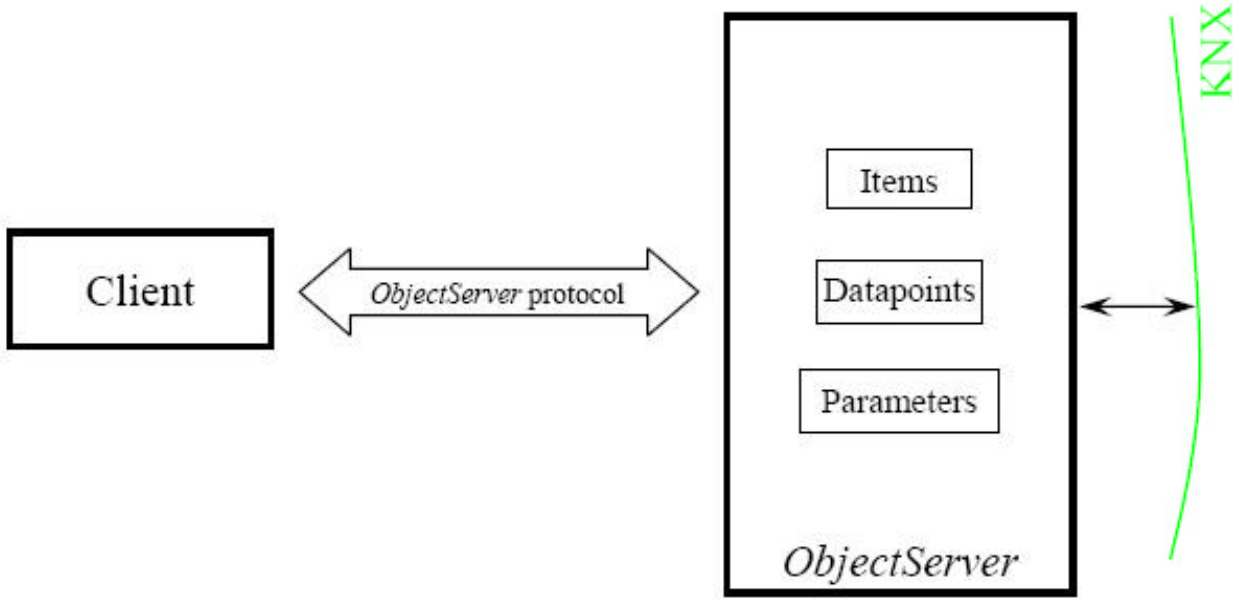


Figure 1. Communication between ObjectServer and Client

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

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 freedefined 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.

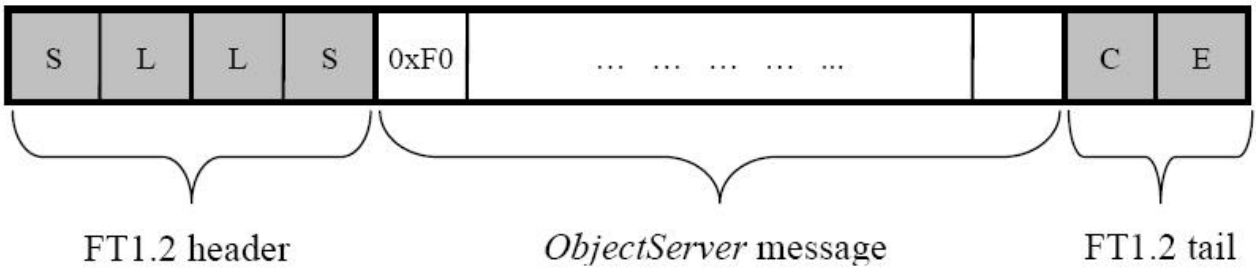


Figure 2. Integration of the ObjectServer message into the FT1.2 frame

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).

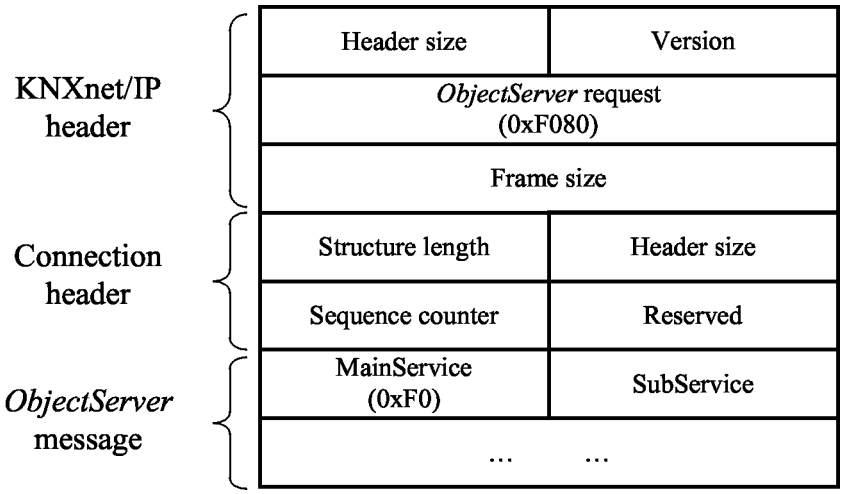


Figure 3. Integration of the ObjectServer message into the KNXnet/IP frame

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.

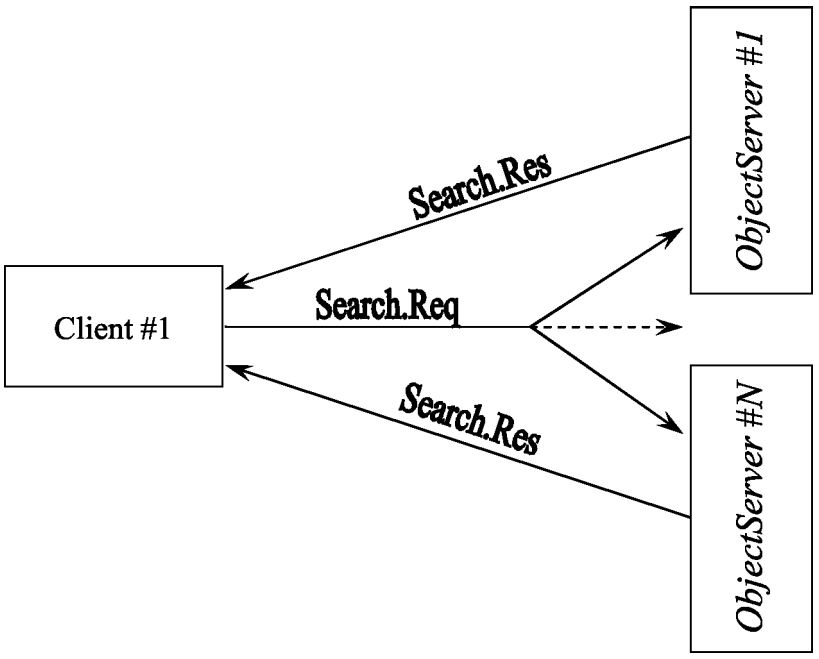


Figure 4. KNXnet/IP discovery

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.

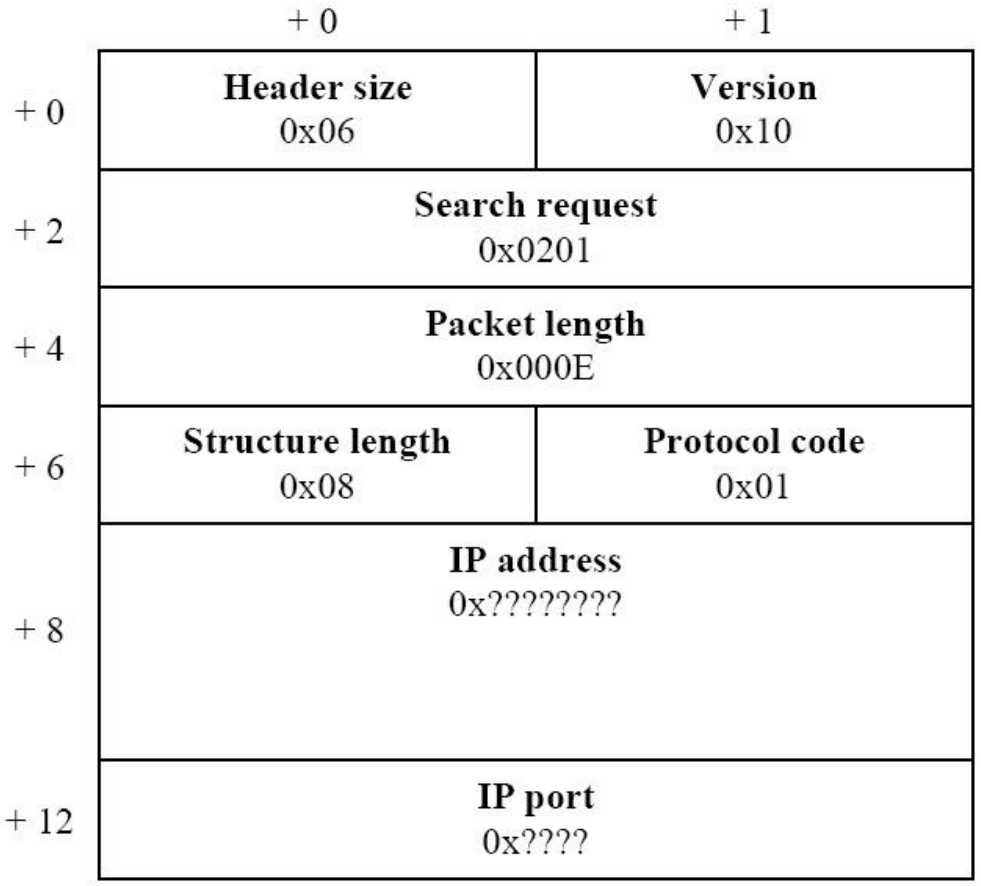


Figure 5. Structure of the Search.Req packet

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.

	+ 0	+ 1
+ 0	**Header size** 0x06	**Version** 0x10
+ 2	**Search response** 0x0202	
+ 4	**Packet length** 0x0054	
+ 6	**HPAI length** 0x08	
	Host Protocol Address Information (HPAI)	
+ 14	**DEV DIB length** 0x36	
	Device information block (DEV DIB)	
+ 68	**SVC DIB length** 0x08	
	Supported services DIB (SVC DIB)	
+ 76	**Manufacturer DIB len** 0x08	**Manufacturer DIB type** 0xFE
+ 78	**Manufacturer ID** 0x00C5	
+ 80	**Record type** 0x01	**Record length** 0x04
+ 82	*ObjectServer* **protocol** 0xF0	*ObjectServer* **version** 0x10

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**	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

EIBMARKT GmbH
Friedensstraße 54
08523 Plauen
Germany