Virtual KNX/EIB devices in IP networks

1 Abstract

In home and building electronic systems the KNX/EIB bus as well as IP networks are some of the most important standards for the future. The definition of the EIBnet/IP standard was a major step in the integration of both protocols into one system.

In the standard solution, an IP network connects several KNX/IP gateways to access the KNX subsystems, enabling communication between different KNX lines via the IP backbone. But all senders and receivers of KNX telegrams are still connected to the KNX media.

Today many devices have an Ethernet connection but cannot be controlled via KNX. To integrate such a device into the KNX world it would not make sense to add an additional hardware interface to the KNX/EIB bus. It would be much more convenient to establish the KNX communication via the IP link, providing a KNX device without a connection to the KNX medium. It may be called a virtual KNX/EIB device.

For runtime communication the device must be able to act like a KNX-IP router which has only one device connected at the KNX side. The device sends and receives KNX group telegrams via multicast IP frames compliant with the EIBnet/IP routing standard.

But what about the configuration of such a device? To obtain full integration into the KNX/EIB world, the device should be programmed using the ETS software, permitting the setting of group addresses and parameters in a closed environment.

When defining new devices the most important question is the device model to be used, because the integration of a new device model into the ETS is quite difficult. The solution is to use one of the standard twisted-pair device models like the most powerful 0701 which is used, for example, by the BIM M 112. Implementing device model 0701 together with the routing
part of the EIBnet/IP standard leads to a device that can even be used with ETS2 without any changes.

The paper describes the architecture of Virtual KNX/EIB devices in IP networks. It gives an overview of the possibilities and shows a sample implementation from Weinzierl Engineering GmbH.

2 General

Today many buildings are equipped with an IP network which is used mostly as a network for personal computers. On the other hand IP networks are increasingly able to serve automation systems. So it is obvious that this technology is an interesting solution especially for building automation. Existing systems like the KNX/EIB bus can be linked to an IP network for use as a fast backbone.

![Diagram of connecting two lines through an IP network](image)

**Figure 1: Connecting two lines through an IP network**

Figure 1 shows an installation with two KNX/EIB lines connected via the IP network. To link a KNX/EIB line to the IP network a KNX/IP-Router is needed. It converts the KNX/EIB telegrams into IP frames and vice-versa according the EIBnet/IP standard. Besides, it filters the telegrams to keep the bus load low. An EIBnet/IP client can access any KNX subsystem via the IP network. Due to the routing capabilities of the EIBnet/IP servers, direct exchange of the telegrams between two servers is possible.
Figure 2 shows the theme of this paper: We replace one line consisting of the EIBnet/IP gateway and one KNX end device with a single component, obtaining a new device which may be called a **KNX/EIB device for IP media**. The former KNX/EIB device is now only present as a virtual device. It is visible in the complete network as a standard KNX/EIB device although it has no conventional connection to the KNX/EIB bus.

Instead it is linked to the KNX/EIB network via the Ethernet connection and the EIBnet/IP features of the device. When programming the virtual KNX/EIB device using the ETS, the IP network is invisible.

3 **Software**

This chapter describes the software architecture for KNX/EIB devices for IP media. It consists mainly of an UDP/IP stack, which handles the communication between the network and the virtual KNX/EIB device.

3.1 **IP-Stack**

To exchange data in an IP network, it is necessary to implement the required parts of the internet protocol. Figure 3 gives an overview of these modules.

The **physical layer** together with the **link layer** defines the hardware-specific interface. Ethernet is used as the physical medium, access to which is realized by using an Ethernet controller. It contains the analog signal interface, a memory to save telegrams, and a mechanism to calculate the checksum. As well, it examines a MAC address of the received telegram.
The **network layer** consists of the address resolution protocol (ARP) and the **internet protocol** (IP). Every device connected to the Ethernet has a unique physical address (MAC address) and a unique logical address, the IP address. The MAC address is programmed by the manufacturer, while the IP address is normally allocated by the administrator of the local network. The address resolution protocol establishes a resolution mechanism between the MAC address and the IP address.

Apart from these unique addresses the multicast addresses for MAC and IP exist to send messages to many participants simultaneously.

The main task of the internet protocol is the forwarding of telegrams to the right addresses in the corresponding part of the network.

![Overview of protocols in an IP network](image)

**Figure 3: Overview of protocols in an IP network**

The **internet control message protocol** (ICMP) is an expansion of the internet protocol. It is used for diagnostic purposes.

The **internet group management protocol** (IGMP) is also an expansion of the internet protocol. It manages the dispatching of the multicast messages.

In the **transport layer** the **transport control protocol** (TCP) and the **user datagram protocol** (UDP) are implemented. The transport control protocol guarantees reliable communication, as it uses a transport layer ACK mechanism and frame repetition. TCP is not necessary for EIBnet/IP. In our implementation it is used together with the HTTP module to obtain a small web server. Hyper-text transfer protocol (HTTP) is used to build a web server as appli-
cation. Web pages, which are stored in the program memory of the microcontroller, can be retrieved by every client in the IP network with a conventional web browser.

EIBnet/IP instead requires UDP. As UDP has no reliable data transmission, it is quite easy to implement. But the most important advantage of UDP is its ability to send multicast messages.

Above the UDP module our device has implemented the EIBnet/IP specification. EIBnet/IP defines the following groups of services:

- **Core**
- **Device management**
- **Tunnelling**
- **Routing**

The **core** group of services includes the services for discovery, self-description and management of the communication channels. Discovery is used to search for the EIBnet/IP devices in the IP network. Self-description helps obtain the capabilities list of the EIBnet/IP devices. A communication channel is a logical data endpoint in a connection between an EIBnet/IP client and an EIBnet/IP server.

The **device management** services are used to configure and manage the gateway using the ETS or other client program.

**Tunnelling**, along with routing, is one of the two ways to communicate. When using tunnelling, a point-to-point connection over a communication channel is established between the EIBnet/IP server and client. Tunnelling is used, for example, when downloading an application or to access a specific KNX device for configuration and diagnostic purposes.

**Routing** is responsible for the exchange of KNX/EIB telegrams between different EIBnet/IP servers. An EIBnet/IP server forwards only those telegrams that correspond to the filter criteria. Routing uses multicast telegrams.
3.2 KNX-Stack

The KNX communication stack is implemented according to the current KNX/EIB specification. As our stack is not connected to one of the KNX/EIB media, the physical layer and the data link layer are not present. So only the hardware-independent, so-called KNX common kernel is left. The source code for the KNX/EIB stack is based on our stack implementation for twisted pair (device model 0701), which is certified by EIBA and the Konnex Association.

The structure of the firmware can be seen in figure 4. The most important modules will be explained in the following paragraphs.

**Figure 4: The KNX/EIB stack**

As the network layer is already medium independent, it and all upper modules can be used without changes.

The most important part of the transport layer is the handling of connection-oriented communication. According to the KNX profiles we use styles that correspond to the device model to be emulated. For device model 0701 it is style 3.

The application layer is the most complex one, because a large variety of services, identified via their APCI-code (APCI = Application layer Protocol Control Information), must be handled. Therefore we have divided it into several modules.

- **Broadcast Communication**

  Broadcast communication is used to assign an individual address to a KNX/EIB device. Our KnxStack supports the requests PhysicalAddressRead and PhysicalAddressWrite and handles
the learning mode. In addition it supports the corresponding requests using the serial number of the device.

- **Group Communication**

The group communication of the application layer is realized via the module $\text{KnxAl}$. It handles all $\text{GroupValueRead}$ and $\text{GroupValueWrite}$ requests received from the transport layer. The object descriptors are read from the loaded virtual EEPROM memory space. The object values can be stored as well in EEPROM memory but mostly are located in the virtual RAM area. The RAM flags are stored in the heap memory and are not accessible via the bus.

As in the KNX/EIB specification no presentation layer is provided, the data format (byte ordering, etc.) at application layer must be according to the EIB interworking standards EIS.

- **Interface Objects (Properties)**

In device model 0701 the so-called interface objects have been defined to access device data without the usage of fixed memory locations. Interface objects are used for the configuration of the system but can also be used for the application.

- **Memory Access**

The memory access via bus is directed to the virtual memory of the device. Supported are the requests $\text{MemoryRead}$ and $\text{MemoryWrite}$. In device model 0701 memory access is protected through load state machines and access protection.

- **Access Protection**

Seen from the ISO/OSI reference model, access protection is a task of the session layer. As in KNX/EIB no session layer is specified, and access protection has been assigned to the application layer. In our KNX stack it is realized in a separate module. According to device model 0701 it handles 16 access levels and 15 keys. The lowest access level has no key. The current access level can be set via the request $\text{Authorize}$, and the keys can be modified by sending a $\text{KeyWrite}$ service to the device.

### 3.3 Application

The application may be any kind of device for home and building automation. It may be a sensor or an actuator or a network device which offers a KNX interface via the EIBnet/IP protocol.
3.4 The complete firmware

The architecture of the complete device is evident when all presented firmware modules are assembled. Figure 5 shows the block diagram containing the virtual KNX/EIB device.

![Block diagram](image)

**Figure 5: The complete firmware architecture**

In spite of the fact that several complex protocols (IP, UDP, EIBnet/IP and KNX) must be realized, it is possible to implement the complete firmware on an 8-bit mikrocontroller. The following paragraph gives an overview of the used hardware platform.
4 Hardware design

Figure 6 shows the block diagram of the virtual KNX/EIB device:

![Block diagram of the virtual KNX/EIB device]

The core of the KNX/EIB device for Ethernet is the microcontroller of the ATmega family from Atmel. The advantage of this microcontroller is an external address bus that can be used to connect the peripheral ICs. By connecting this way, data exchange can be accelerated. A 32 kByte memory chip and the Ethernet controller are connected to the external address bus using an address latch decoder. To the Ethernet, the device uses a standard Ethernet controller. The physical connection to the Ethernet is via an RJ45 connector. To guarantee a galvanic separation between the Ethernet and the virtual KNX/EIB device a transformer chip must be used.

Figure 7 shows the realisation of the virtual KNX/EIB device. The Atmega microcontroller can be seen in the middle of the virtual KNX/EIB device. The Ethernet controller is placed right over it.

In the right upper edge a KNX/EIB interface is realized, as well as a USB interface on the left upper edge (though neither are used in this sample implementation).

A number of control buttons and LEDs enable the user to provide input and to view different states of the virtual KNX/EIB device.
5 Conclusion

The virtual KNX/EIB device for IP described in this document combines the benefits of Ethernet networks with the well-defined KNX protocol for building automation. The management and configuration capabilities of the KNX standard in particular, together with available tools such as the ETS program, open new perspectives for building control over IP.

Our virtual KNX/EIB device architecture enables the implementation of automation devices without connection to a KNX medium but completely integrated into the KNX world. It opens the possibility for development of new KNX devices with extended communication capabilities over IP networks or for simple and cost-effective integration of existing network devices into the KNX environment.

6 Bibliography

- Konnex Association: KNX standard (Version 1.1), Brussels, February 2004; CD-ROM
- Konnex Association: EIBnet/IP system specifications, Konnex Association 2003
  (available on www.weinzierl.de)