

---

## **Open-source foundations for PC based KNX/EIB access and management**

---

KNX Scientific Conference 2005

Bernhard Erb, Georg Neugschwandtner,  
Wolfgang Kastner, Martin Kögler


**AUTOMATION  
SYSTEMS  
GROUP**

Institute of Automation  
Automation Systems Group  
Vienna University of Technology  
Vienna, Austria




[www.auto.tuwien.ac.at/knx](http://www.auto.tuwien.ac.at/knx)


## Outline



- Workstation stacks for KNX/EIB
- eibd – KNX/EIB connectivity daemon
- „KNX Live“ project
- Tweety – EIBnet/IP server
- Calimero – EIB/KNX Java APIs
- Outlook



KNX Scientific Conference 2005  
 Open-source foundations for PC based KNX/EIB access & management - 2



For Linux, several low-level drivers for the serial PEI and TP-UART host protocols have been provided. However, they do not present a unified interface to the client developer intending to build a workstation application. The platform independent EIBnet/IP protocol opens up new possibilities. Yet, appropriate hardware is still rare and another protocol layer is placed between the developer and the desired result. Three software projects are presented which address these shortcomings.

The first is a multi-user Unix daemon for KNX/EIB connectivity. The relevant parts of the EIB protocol stack to send and receive unicast, multicast and broadcast telegrams are provided. Also, the protocol state machine for the client endpoint of a reliable connection is handled. Based upon this, the daemon can also autonomously execute various device and network management procedures. Various methods for access are supported, yet their differences entirely hidden from the client.

The second is a lean EIBnet/IP server for Linux systems. It is based upon the already well-known PEI16 driver and is intended as an implementation of the minimum useful EIBnet/IP protocol subset.

Third, a set of Java packages was created to jumpstart the development of platform-independent EIBnet/IP Tunnelling clients. Encoding and decoding of interworking data formats and a simple, XML based data point database are provided as well.

All this software is open source and available on a Live Linux CD. This CD is intended as a try-out platform for both beginners and experts.

## Workstation stacks for KNX/EIB

- For visualization, configuration, development
- Methods for KNX/EIB network access
  - EIA-232 interface
    - PEI16 timing sensitive (due to RTS/CTS handshake)
      - tuwien.auto kernel driver
    - PEI10 (FT1.2): only available on BCU2
    - TP-UART based custom interface
  - EIBnet/IP
    - hardware independent
    - cEMI message format covers all media
  - USB

By its very purpose, most of the devices connected to a fieldbus are embedded nodes. Yet, there are applications whose resource requirements make it necessary to connect a general-purpose workstation (mostly a PC) to a network like KNX/EIB. This especially concerns visualization, configuration, and development tasks.

While one lower part of the network stack will always have to be implemented using special KNX/EIB hardware, a large part can rest on the PC. This provides benefits in flexibility and performance. But where to make the cut?

PCs with EIA-232 ports can directly use the PEI16/EMI1 or PEI10/EMI2 interface provided by the BCU (after level adjustment). While the timing constraints imposed by the RTS/CTS handshake of PEI16 require special handling that in most cases will have to be done by a kernel driver, the FT1.2 based protocol of the BCU2 (PEI 10) can be implemented using the plain serial driver. Although not defined in the KNX handbook, directly implementing the TP-UART host protocol offers the greatest flexibility, although it is also not without timing challenges. For the Linux operating system, low-level (kernel and user-mode) drivers for all these variants were published (tuwien.auto driver suite).

EIBnet/IP allows to communicate with an KNX/EIB installation by tunnelling over IP networks. Data exchange uses the cEMI message format, which is attractive due to its medium independent design. Beside older EMI formats, it is also used by USB interfaces available.

## eibd – KNX/EIB connectivity daemon

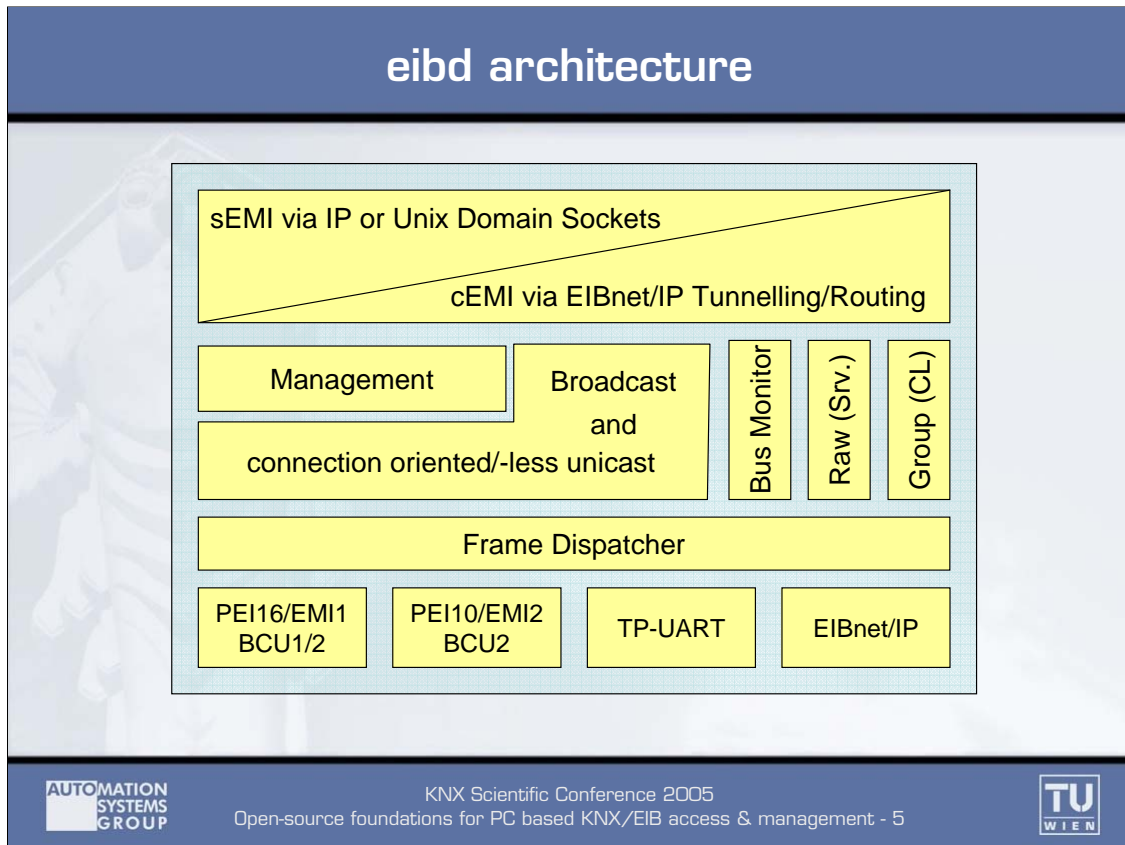
- Focus: development and management
- Flexible, powerful
- Various bus access methods
  - PEI16/EMI1, PEI10/EMI2
  - TP-UART
  - EIBnet/IP Tunnelling and Routing
- Dual client interface
  - Proprietary open message protocol (sEMI)
  - EIBnet/IP Tunnelling and Routing
- Multiple concurrent client connections
  - Best-effort bus monitor
- Handles KNX/EIB device management procedures

Within the BCU SDK project (which is covered in “Rapid Application Development for KNX/EIB BCUs”), a Unix daemon called eibd was created to cover access to the EIB/KNX TP1 network for development and management purposes.

It supports a variety of bus access methods. These are implemented by “back ends” which hide the differences behind a unified interface. While the PEI16/EMI1 relies on the auto.tuwien kernel driver, PEI10/EMI2 needs no further software. The TP-UART host protocol is implemented using kernel drivers as well as a using user mode code only. EIBnet/IP Tunnelling and Routing are supported as well.

Clients are offered both a custom message protocol tailored to the needs of the BCU SDK (sEMI – SDK EMI) as well as a limited EIBnet/IP server frontend. Clients can use sEMI to invoke various services of the EIB/KNX protocol stack. This includes sending and receiving unicast, multicast and broadcast telegrams. Also, eibd handles the protocol state machine for the client endpoint of a reliable connection. Based upon this, eibd can also autonomously execute various device and network management procedures, such as assigning individual addresses. Also, a bus monitor can be opened, which optionally can decode EIB frames. Several utility programs illustrating the use of this message format are available.

A “Raw” interface is available for clients that wish to implement the server endpoint of a reliable connection. This is necessary to enable the host eibd resides on to be managed remotely via EIB/KNX management procedures. For the BCU SDK, eibd also acts as a loader for downloading BCU applications over the network.




The architecture of eibd allows multiple clients to connect simultaneously. Higher-level tasks within eibd register with the frame dispatcher and state which frames (based upon addressing mode and destination address) they are prepared to process. To be able to serve multiple clients simultaneously, backends should deliver as many incoming frames as possible and leave filtering to the frame dispatcher.


In principle, this allows one client to maintain a point-to-point connection - where only frames from one single source are relevant - and another to operate in bus monitor mode. Yet, since bus monitor operation entails switching the hardware into a read-only mode, a special "best-effort" monitor mode was introduced which builds upon the fact that telegram filtering is not performed in the access hardware, but in the frame dispatcher. It forwards all frames the backend will provide in normal operation mode.

## “KNX Live” project

- Platform for exploring and understanding KNX/EIB installations
- Easy to set up and use
  - Controlled and safe environment
- Informative and illustrative
  - Freely available
  - Open source
- Focus
  - Developers: attractive protocols, new approaches
  - Technically inclined: base protocols, network access
  - End users: process data access



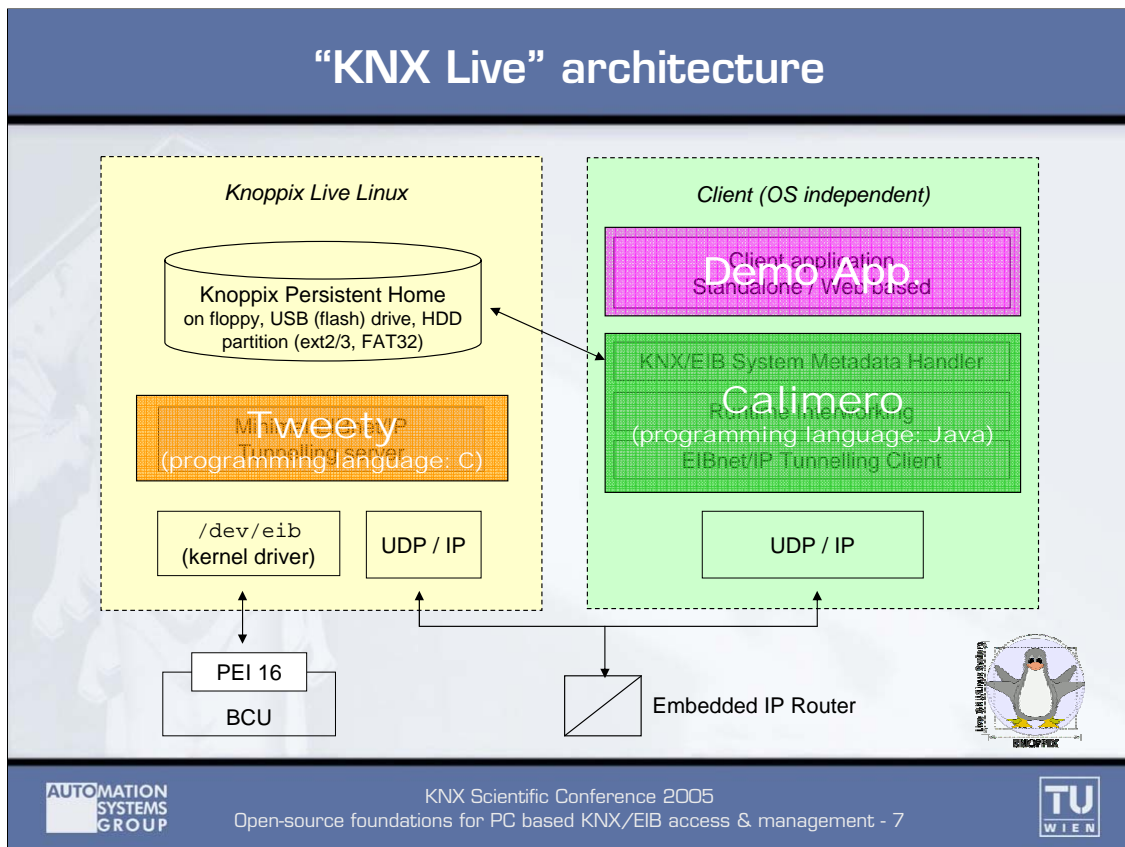
KNX Scientific Conference 2005  
 Open-source foundations for PC based KNX/EIB access & management - 6



The idea behind the “KNX Live” project is to offer a try-out platform for KNX/EIB technology. The target audience ranges from developers who want to discover KNX/EIB interfacing using Linux as well as people who are interested in the technology behind the scenes of their home KNX/EIB system. It even encompasses beginners interested in what benefits interfacing their PC with their KNX/EIB installation may bring.

Therefore, special attention was given to an easy and straightforward configuration. Even KNX/EIB beginners should be able to use the system. Moreover, experiments should not do any harm to either the PC or KNX installation. Therefore, this project was based on a Knoppix live Linux CD, which offers a stable and safe environment. This operating system is completely started from a single CD, without the need of hard disk installation. To further the goal of knowledge distribution, all additional software packages are open-source and freely available. The project also includes selected documentation regarding KNX/EIB.

Especially when considering the beginner scenario described above, the focus clearly is on process data access. Going one step further, (hobby) developers should be able to exchange data with KNX/EIB devices with the minimum amount of detail knowledge necessary.




Therefore, a client-server architecture centred around EIBnet/IP tunnelling was chosen. It allows the client to be written in an entirely platform independent manner, using Java. To keep things even simpler for the client developer, an API was created which handles the EIBnet/IP Tunnelling protocol. It is accompanied by one API for encoding and decoding of Layer 7 group communication and another for maintaining a simple list of data points. To illustrate how to make use of this API suite ("Calimero"), a demo application was created. It can easily be replaced by something more complex at any point in the future. EIBnet/IP also offers the perspective of remote access.


However, dedicated EIBnet/IP server hardware is still rare. For this reason, a minimal Linux-based EIBnet/IP Tunnelling server ("Tweety") was developed. It rests upon the tuwien.auto PEI16 kernel driver. Due to the Knoppix environment, the user need not go through the tricky compilation process. The configuration of the driver and server is achieved by a installation "Wizard". It first installs the driver, checks all serial interfaces for attached BCUs, and starts the server when successful. The server has to set the address table length of the interface BCU to zero to be able to participate properly in group communication. Although it restores it to its previous value on termination, this could be a problem when, e.g., the BCU of a light switch is used as a temporary interface and Tweety is somehow prevented from restoring the table length (the PC is halted, or the cable is disconnected prematurely). In this case, the BCU will no longer work with its previous application module until the address table length is restored. The "Wizard" script also addresses this issue.

## Tweety

- Minimal EIBnet/IP Tunnelling server
  - Search and Description
  - Tunnelling
  - Lean and robust
  - Tested with ETS (Falcon)
- Implementation
  - Linux operating system
  - PEI16/EMI1 to address existing installations
  - Uses tuwien.auto kernel driver
  - Written in C
  - Multi-threaded (pthreads)



KNX Scientific Conference 2005  
 Open-source foundations for PC based KNX/EIB access & management - 8

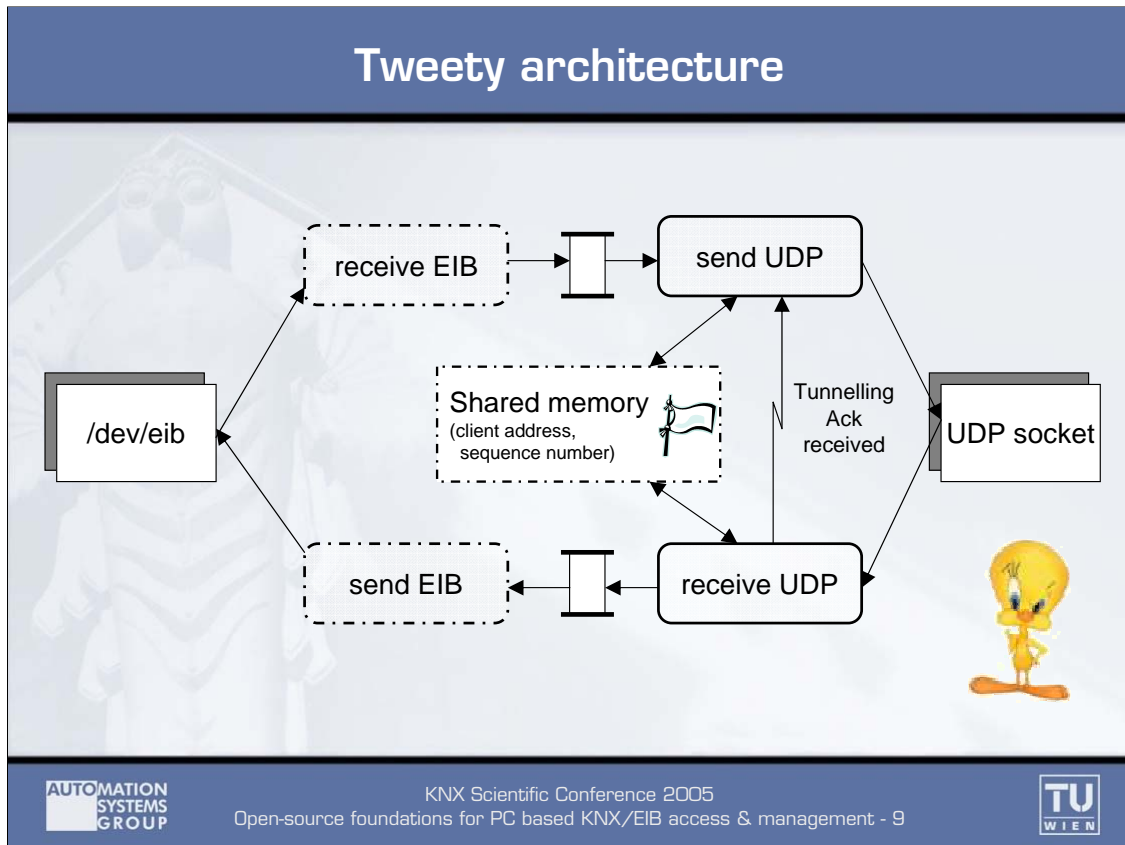


While eibd is designed to be powerful and flexible, which necessarily entails a certain complexity, Tweety is focussed on a much smaller task. It is intended to cover a minimal useful subset of the EIBnet/IP server protocol. In its current version, Tweety only implements the Core and Tunnelling parts. Similar to eibd, the Device Management service protocol was left out, since changing the port number or friendly name is easier achieved on a PC-based implementation using command line parameters. Additional options can be set via definitions in the Makefile.

Tweety builds upon PEI16/EMI1. This choice allows it to offer both BCU1 and BCU2 support with a single code base. EIBnet/IP clients connect to Tweety using UDP. Only Tunnelling on Link Layer and one single user at a time are supported. Nevertheless Tweety is a fully functional implementation, which has been successfully tested with ETS (Falcon). These deliberate restrictions have lead to a small and robust architecture, which is implemented in C with use of the pthreads library.

Tweety is divided into four concurrently executing entities. Since the PEI16 kernel driver only allows a single process to connect, the receive and send units on the KNX/EIB TP1 side (*receive EIB*, *send EIB*) are executed as pthreads, whereas their UDP counterparts (*send UDP*, *receive UDP*) are executed as processes. Interprocess communication is handled via named pipes and a shared memory object, which is protected by a semaphore. The shared memory holds information about the connected client and the connection sequence numbers.



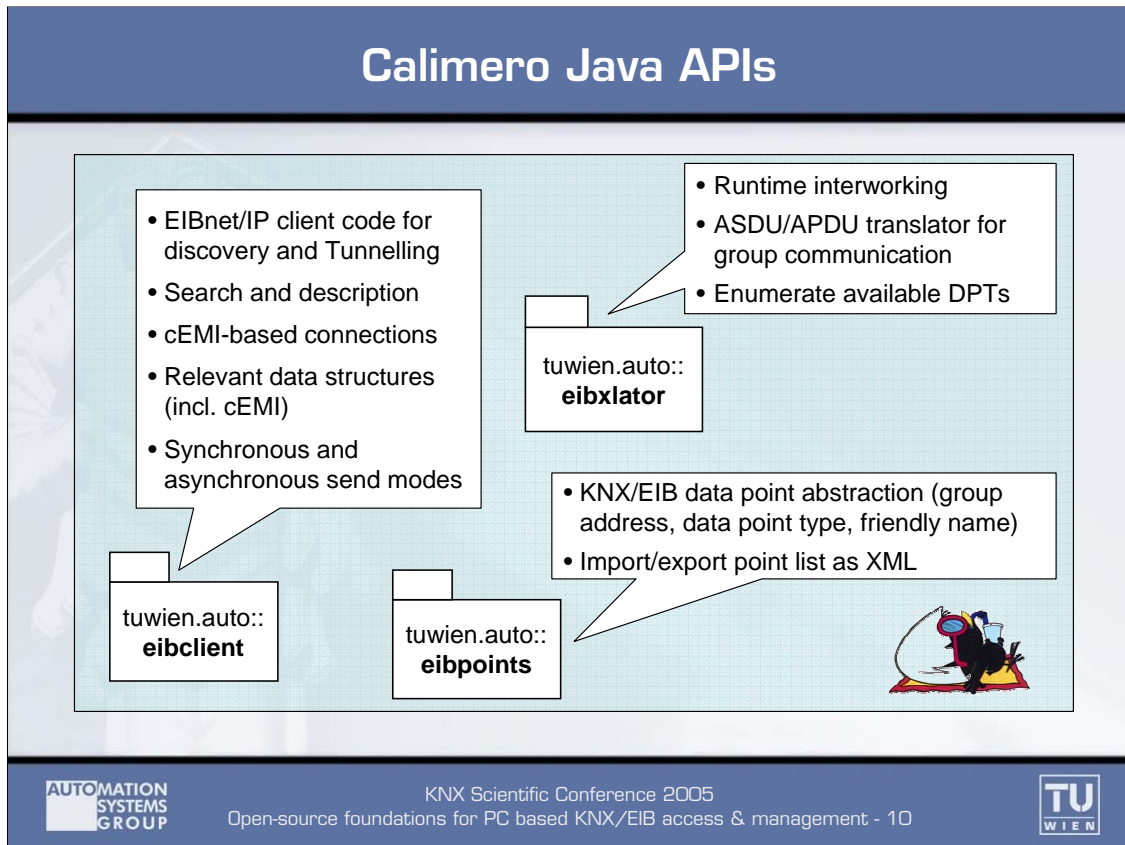


*Receive EIB:* This thread listens to the device node (e.g., /dev/eib), and blocks until a new frame arrives. On reception the frame is passed to the pipe, and the thread returns to the blocking state.

*Send EIB:* This thread waits until a new cEMI message is put into the pipe by the Receive UDP process. After reading it, the cEMI message is converted into EMI1 format and passed to the device driver. Tweety sends frames to the device driver in synchronous mode. If the message cannot be dispatched after a fixed time, the server is halted.

*Send UDP:* This process waits until a new message is in the pipe. After reading it, it performs the EMI1 to cEMI conversion and sends the frame to the EIBnet/IP client. If no client is connected the frame is discarded. The client address and the sequence numbers are taken from the shared memory. After sending the process waits for the signal that the receive UDP process sends whenever it has received a Tunnelling acknowledgement. After that, the send sequence number is incremented and the next frame is processed. If no Tunnelling Ack message is received (and thus no signal caught) within one second after transmission, the server tries to resend the frame. This procedure is repeated only once; after the second timeout, the server closes the connection.

*Receive UDP:* This process listens on the socket for incoming UDP messages. Every UDP request is handled here, including Tunnelling Acks. If a Tunnelling request is received the process places the message body into the pipe, and waits until the next request arrives.



Calimero is a collection of Java APIs that together form a foundation for further EIB/KNX high level applications (including remote access and control). Still, these APIs can be used largely independently.

`Eibclient` is a EIBnet/IP client library that, in its current version, supports Tunnelling connections as well as search and description. The aim of this API's is to allow the establishment of EIBNet/IP connections with minimal effort. The entire EIBnet/IP message exchange including heart beating is hidden from the client programmer, who merely is faced with an API to send and receive cEMI messages. In addition, `eibclient` is able to parse cEMI messages to extract the relevant data as well as assemble them. The current implementation is limited to TP1 standard frames.

`Eibxlator` is a collection of encoders /decoders for Application layer protocol data units (APDUs) relevant for exchanging and interpreting group values. In this version only a subset of KNX DPT's are implemented, but the package was designed to make the addition of further ones easy.

`Eibpoints` offers the ability to maintain a list of the data points in the KNX/EIB system and their relevant data, including group address, friendly name, and DPT type. The data point list abstraction offers lookup facilities by name and address. The entire configuration can be exported and imported as an XML file or stream.

## Dim the light with Calimero

```

try {
    CEMI_Connection tunnel= new CEMI_Connection(
        new InetAddress("localhost",
            EIBnetIP_Constants.EIBNETIP_PORT_NUMBER));

    PointPDUxlator dimVal = PDUxlatorList.getPointPDUxlator(
        PDUxlatorList.TYPE_8BIT_UNSIGNED,
        PDUxlator_8BitUnsigned.DPT_SCALING[0]);

    dimVal.setServiceType(PointPDUxlator.GROUP_VALUE_WRITE);

    dimVal.setValue(",75");

    CEMI_L_DATA message = new CEMI_L_DATA(
        CEMI_L_DATA.MC_L_DATAREQ,
        new EIB_Address(),
        new EIB_Address("0/0/1"),
        dimVal.getAPDU);

    tunnel.sendFrame(message, CEMI_Connection.WAIT_FOR_CONFIRM);
}

catch (EIBClientException ex) { } // connection error

```

This code example shows how easily a dimmer can be set to another value using the Calimero APIs. First, a new cEMI connection is instantiated, giving the IP address of the server and the default port number for EIBnet/IP. The constructor tries to establish a tunnelling connection with the server. If the operation succeeds and the Connect\_Response frame has been received heartbeating is started and the constructor returns the cEMI connection object. If something goes wrong an Exception is thrown including an appropriate error message. The cEMI connection object not only allows to send and receive cEMI messages but also handles the heart beating and disconnection. For receiving, EventHandlers can be registered that are called on every incoming frame.

The next step is to construct a PDU translator. For this purpose the static method `getPointPDUxlator` is called with the required major and minor type. If the requested DPT type is implemented we get a instance of `PointPDUxlator` class, which is the base class for each PDUxlator implementation. If the major or minor type can not be found an Exception is thrown.

Then the service type is set, in our case a group value write request. The byte representation of the Layer 7 message can be retrieved using `getAPDU()`. Now we set the PDU translator value. Note that each `PointPDUxlator` class implements the abstract method `setValue(String value)` so that the values can be always set through this method, regardless of the specific type. If the parsing does not succeed an Exception is thrown.

All that is left to do is to set up a new cEMI L\_Data request with the appropriate destination address and send it to the server. The source address can be left blank as the EIBNet/IP server will place its own address into it. We set the APDU bytes extracted from the translator as message body and send the frame using the tunnel instance.

In the example the send method `WAIT_FOR_CONFIRM` is used, in which the request `sendFrame()` blocks until the Tunnelling Ack and even the cEMI\_L\_Data.con message have been received. The other send mode, `IMMEDIATE_SEND`, returns immediately. The message status must in this case be checked by the programmer itself using the `getStatus()` method.

## Outlook and future work

- Re-examine auto.tuwien PEI16 kernel driver
  - More robust timing (BCU2)
- Merge back end codebases for eibd and Tweety
  - Keep simple, but e.g. add USB
- Extend Calimero
  - Management connections (eibclient)
    - Straightforward extension
  - USB connections (eibclient)
    - Also cEMI based
    - Via javax.usb – just standardized
  - Implement further DPTs (eibxlator)
  - Grouping of data points (eibpoints)
- Project homepage
  - <http://www.auto.tuwien.ac.at/projects/knxlive>

Two open KNX/EIB workstation stack implementations were presented. One („eibd“) is written in C++, supports various bus access methods and is geared more towards device management, while the other („Calimero“) is written in Java, builds upon EIBnet/IP and exclusively targets group communication. Eibd can also act as EIBnet/IP server. In addition, a lean EIBnet/IP Tunnelling server is available („Tweety“). All components are placed under the GPL. A bootable Live Linux CD with these components plus a PEI16 driver preinstalled can be downloaded from the project homepage.

Further work is necessary on the PEI16 driver, which currently exhibits problems when used with a BCU2. Also, eibd and Tweety should use the same access hardware abstraction to ease the inclusion of further hardware interfaces like USB.

Regarding Calimero, possible next steps include support for management connections, further data point types and the grouping of data points in the data base (e.g., to represent a dimmer with its subfunctions or access points by location). USB support is also of interest on this level. Since KNX/EIB USB interfaces also use cEMI, such an extension should be able to re-use considerable amounts of the existing implementation.