enertexbayern gmbh
simulation entwicklung consulting

Enertex® EibPC
Manual

Requirements Enertex®EibPC: Patches 3.000 or higher
Enertex®EibStudio: 3.000 or higher

Contents

Remarks

For the operation of the Enertex® EibPC you need

- An FT1.2 compatible KNX™RS232 interface **or** an Eibnet/IP interface (see page 15 for compatible interfaces)
- A DC power supply, at least 1.7 Watt output power, 20 to 30 V
- For programming a Windows® XP or a Linux® PC with LAN access.

Please note the information on commissioning on page 15 and the sequence of connecting the RS232 on page 17.

Help function

The help function is realized by means of the Acrobat Reader. Either you can directly download this document (www.eibpc.com) or open it by pressing F1 within the Enertex® EibStudio.

In this case you will have access to a context help. Mark the key word in question and press F1 (Macintosh user: CMD-?). The manual will be opened at the respective position. The Enertex® EibStudio itself contains code completion (STRG+Space) and syntax highlighting.

This PDF document uses the the Acrobat Reader's structuring in subsections, which could be also denoted as “bookmarks“. Click at the left border of the index tab to display these “bookmarks“.

This document contains links. If you read a page reference, e.g. “more on page 15“, simply click at the page number and the reader jumps to this position.

Security advice

- Installation and assembly may only be performed by an authorized electrician.
- For connecting KNX/EIB interfaces, expert knowledge gained by KNX™- trainings is assumed.
- Due to neglecting this manual damages of the device, fire or other dangers could appear.
- This manual is part of the product and has to remain at the end user.
- This device may not be used for applications with risk potential (failure, potential fault of the time switch, etc.).

License remarks

- At the purchase of the Enertex® EibPC you gain a license for using this program. The Enertex® EibStudio and all independently running components may only be used for the programming of the Enertex® EibPC.
- The manufacturer is not liable for any costs or damages incurred at the user or third parties through the use of this device, abuse or fault of the connection, fault of the device or the user equipment.
- Unauthorized changes and modifications to the equipment will void the warranty!
- The manufacturer is not liable for improper use.

Support

Standard support is via the e-mail address: eibpc@enertex.de. At http://knx-user-forum.de a separate area for support of the Enertex ® EibPC is set up. You will also find direct advice from expert users and professionals. Please send us the following information at any inquiry:

- Application program (*.epc)
- If necessary: Import addresses (*.esf)
- If necessary: Log of the bus transfer of the Enertex® EibStudio (*.log)

Via menu ”?” - "Information Pack for Support Request" one tar file with all necessary information will be generated.

Updates

At www.eibpc.com you will find updates for the Enertex® EibPC:

- Firmware updates (program with bus access)
- Patch updates (web server, internal file structure)
- Enertex® EibStudio
- On page S. 345 you will find a summary of the latest news, updates and upgrades.

Enertex® EibPC - Overview

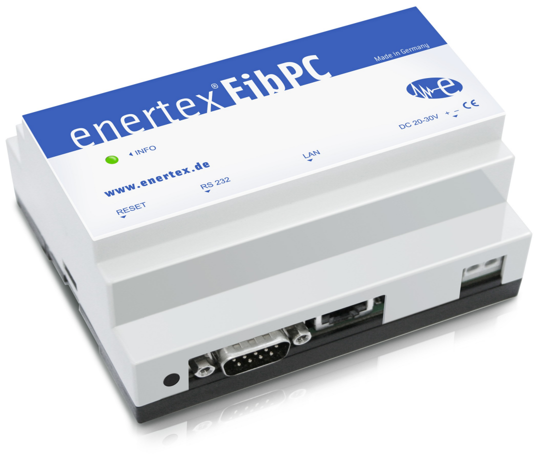


Figure 1: Enertex® EibPC

Summary

The Enertex® EibPC represents a controlling system for DIN rail mounting (6 units) for the KNX bus. With about 1.2 W power consumption, it provides an energy efficient and environmentally friendly full control over the KNX bus system.

The Enertex® EibPC is a scene actuator, a calculator, a logic center, a PLC, a time switch, a LAN and Internet connection as well as a web and e-mail client in one device. With the supplied software Enertex® EibStudio, a parametrization of the KNX bus system without ets is possible.

Ready-made function blocks

If you want to use only basic functions of the Enertex® EibPC, the use of “macros“ is sufficient. Macros are predefined function blocks which are available to users in the form of a library. The included libraries of Enertex ® Bayern GmbH provide complete automation solutions for

- Lighting controls
- Roller and blind controls (shade)
- Time switches
- Control of the conservatory

KNX-Functions

All functions of the KNX™ Bus can be addressed

Within the KNX network, the Enertex® EibPC realizes the functionality of

- Scene actuators
- Conditional instructions (if-then)
- Timers
- Time and date emitters (synchronized via LAN, KNX™ or Enertex® Eibstudio)
- Highly accurate timers (in the ms range)
- Controls with any structure
- Evaluation of mathematical expressions
- Delay elements
- Combination of KNX™ objects (gates, multiplexers, ...)
- Control of actuators (e.g. cyclic read requests)
- Storing variables in remanent memory (Patch 1.100 needed).

These functions can be used infinitely. Thus, you could define for example 65,000 scene actuators. The Enertex® EibPC handles the entire maximum possible number of objects of a KNX™ networking.

LAN-Functions

The Enertex® EibPC has a LAN interface, which realizes

- Monitoring of bus services (excluding ets [and PC])
- Sending and processing of any KNX™ telegrams (without ets)
- Synchronization of the bus time via Internet (without ets)
- Sending, receiving and processing of UDP frames (additional option NP), e.g. for the control of multimedia systems
- Sending e-mails (additional option NP)
- Integrated web server (additional option NP)
- VPN Services configurable with KNX (additional option NP)

Data logging

Logging of up to 500,000 telegrams is possible

Memory The Enertex® EibPC stores all bus telegrams. Up to 500,000 frames are held in a ring buffer, even if no PC is connected to the Enertex® EibPC. With an average bus load of three telegrams per minute this corresponds to all telegrams of the last 200 days.

Time Using time stamps, which are automatically generated by the Enertex® EibPC, the bus traffic can be analyzed at any time.

Online In addition, it is possible to view the data online and to filter by sender and group addresses.

Filter The telegrams can be already pre-filtered by the device address and group address.

Auto-log The Enertex® EibStudio allows the cyclic saving of (possibly filtered) telegrams in files.

FTP The Enertex® EibPC can store telegram data on a arbitrary FTP server. Enertex® EibStudio evaluates this binary and exports it into readable CSV text.

Software

By means of the Enertex® EibStudio as a configuration program a home automation is provided via the LAN interface of the Enertex® EibPC to a Windows®, Mac® OS X or Linux® PC. This ensures that the Enertex® EibPC can be programmed easily without the ets.

Basic The programming is carried out by a simple Basic syntax for which no time-consuming training is necessary. For the basic functionality, it is not even necessary to learn this basic. The user has a selection of available ready-made function blocks, where the user has merely to add group addresses etc.

ets The Enertex® EibStudio imports the addresses and settings of the ets. It can also be used entirely without ets import.

Commissioning

Programming

To program the Enertex® EibPC, you need a Windows® or Linux® PC and a LAN connection to the Enertex® EibPC. The Enertex® EibPC itself does not require a LAN connection for its operation, but in this case some functions such as synchronizing the system time with Internet time, etc. can not be used.

We recommend for Windows® environments the use of Windows® XP with SP3.

Power supply

The power consumption is typically about 1.5 W at 24 V DC

The Enertex® EibPC requires an external DC power supply, ranging from 20V to 30V. The power consumption is typically about 1.2 W, with LAN transfer approximately 1.7 W.

If you want to power the Enertex® EibPC by the KNX™ bus, you need to insert an appropriate choke between the bus line and the Enertex® EibPC. For an exemplary wiring diagram, see Figure 2.

Bus access

To gain access to the KNX™ bus, the Enertex® EibPC utilizes an external KNX™ interface which can be either an KNX™ RS-232 interface for the FT 1.2 protocol or an KNX™ IP interface

FT1.2 interface

Please select an adequate interface

The following FT1.2 interfaces are tested:

- EIBMarkt IF-RS232 (you need to switch to the FT1.2 mode)
- Siemens N148/04 (5WG1 148-1AB04) (you need to switch to the FT 1.2 mode)
- Gira RS232 UP 0504xx + BA 064500

Basically FT1.2-able are:

- Siemens UP 146 Z1 for FT 1.2 (5WG1 146-2AB11-Z1) + BA UP 114 (5WG1 114-2AB02)
- Berker RS-232 Data interface FT 1.2 (750601xx) + BA Up 2.0 (75040002)

The following interfaces are surely improper:

- ABB EA/S 232.5
- Siemens N148/02 (5WG1 148-1AB02)
- Berker RS-232 Data interface REG (7501 00 13)
- Berker RS-232 Data interface UP (750600 xx)

IP interface

The following EIBnet/IP-Interfaces are tested:

- Siemens N148/21
- EibMarkt EIB KNX IP Interface PoE (Art.Nr.: N000401)

Successfully tested by users are:

- ABB IPS/S 2.1
- ABB IPR/S 2.1
- Siemens N146 (5WG1146-1AB01)
- Siemens N148/22

The IP interface will be addressed and activated only after the transfer and the subsequent start of an application program.

Since the RS232-Interface does not supply a feedback over the connecting status, the EibPC sends all telegrams, even if no bus connection is developed. Thus each code can be tested also without bus connection.

Against it a defined handshake has the IP interface. The EibPC waits then for the feedback, before it sends telegrams away.

Enertex® KNXNet/IP Router

Picture 1: The Enertex® KNXNet/IP Router

More Connections

The KNXnet/IP Router (3TE) supports up to five KNXnet/IP-Tunnelconnections and can be used as line- or field coupler. Because of its integrated display you can see any time all important configuring parameters: IP-address, KNX-tool address und number of the opened tunnel connections. This allows an efficient and unproblematic initial operation.

The IP-address of the Fast Ethernet-connection can be configured manual via ETS or automatically via DHCP or Zeroconf. Via a Telnet access important operating parameters as the IP addresses of the Tunnelings are accessible. Additionally the KNXnet/IP Router has a buffered battery real time watch as well as a SNTP-Server in LAN available. In this respect the Enertex® KNXNet/IP Router is the best supplement to the Enertex® EibPC, because it can be time synchronized after an electrical power failure without internet connection.

The Enertex® KNXNet/IP Router allows furthermore the connection with a Telnet-Server, which keeps further information for the bus transport.

The KNXnet/IP Router is powered via Power over Ethernet or via an external 20-30V AC/DC power supply.

The Enertex® EibPC is able to share the power supply with the Enertex® KNXNet/IP Router, if it is ensured that this provides ca. three Watt.

Simulation

„Dry practise“

Since the RS232-Interface provides no feedback on the connection status the EibPC sends all telegrams even if no bus connection is established. Thereby you can test each code without bus connection.

In the opposite the IP Interface has a defined handshake. The EibPC is than waiting for the feedback before it sends telegrams.

Installation

Hardware

Figure 2 shows the basic structure of a KNX™ network which is connected to the Enertex® EibPC. To integrate the Enertex® EibPC into the KNX™ system, proceed as follows

Simple DIN rail mounting

1. Connect the Enertex® EibPC to a 24V - 30V DC power supply. This can be accomplished either by a separate power supply or by the KNX™ bus. If you want to connect the Enertex® EibPC to the KNX™ bus, you need to insert an appropriate choke between the bus line and the Enertex® EibPC.
2. Connect the LAN port (2) of the Enertex® EibPC to the LAN.
3. Connect the RS-232 interface (3) of the Enertex® EibPC to an KNX™ RS-232 interface (FT 1.2 protocol only) with an extension cable (“Male“ to “Female“).
4. Restart the Enertex® EibPC (via EibStudio or simply interrupt the Enertex® EibPC's power supply).

Important

Please note:

The external safety extra-low voltage is connected through the device to the ground potential of the LAN. Thus, there is no longer an isolation to the ground potential when the LAN shielding is grounded. To establish an isolation, it is recommended to use an external extra-low voltage power supply only for the Enertex® EibPC.

The Info LED simplifies the commissioning:

When the EibPC is starting, the RS-232 interface must be already connected to the switched-on bus. To ensure this, proceed as follows:

Switching on:
Info LED is on continuously

Boot operation finished:
Info LED is blinking

Shortly afterward:
The RS-232 interface is initialized :
Short impulses can be observed at the Info LED

Enertex® EibPC identifies telegram:
Short impulses can be observed at the Info LED

1. Wiring as described above.
2. Connect the interface to the switched-on bus.
3. Restart the Enertex® EibPC (by the EibStudio or simply interrupt the Enertex® EibPC's power supply).
4. **If you are using an FT1.2 RS-232 interface:** When the Enertex® EibPC is starting, the RS-232 interface must be already connected to the switched-on bus. **So do not** first switch on the Enertex® EibPC and then establish the RS-232 connection. The RS232 interface may be switched on simultaneously with the Enertex® EibPC, but not after booting the Enertex® EibPC.
5. **If you are using an IP interface:** Because an IP interface could sometimes be in a locked state, it should be reset once. The interface is accessed and activated only after the transfer and the subsequent start of an application program. The Enertex® EibPC uses the so-called tunneling mode of the IP interface. Thereby, it is exclusively occupied, i.e. you can not address the interface in this mode from the ETS. Nevertheless, in order to allow a short-term access to the interface you can separate the interface via the menu item Connect and Disconnect from Enertex® EibPC and reconnect with it, respectively.

Now the installation of the Enertex® EibPC to your KNX™ -system is finished.

Commissioning of the RS-232 interface

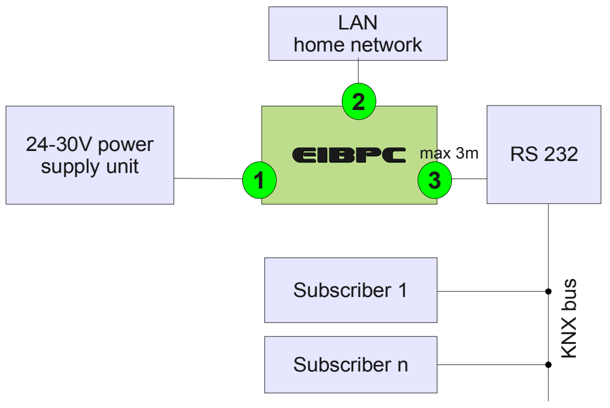


Figure 2: Connecting the Enertex® EibPC to the KNX™Bus with RS232

Commissioning of the IP interface

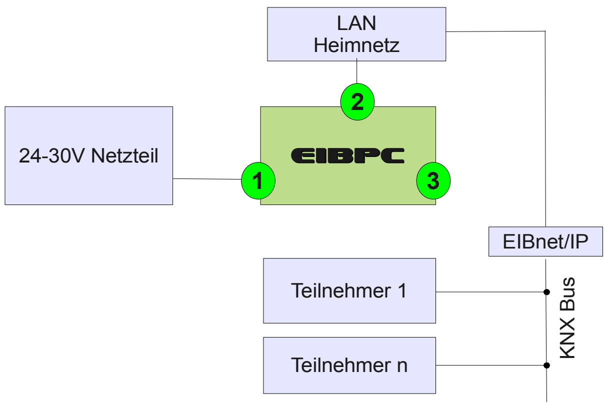


Figure 3: Connecting the Enertex® EibPC to the KNX™Bus with EIBnet/IP

Within Figure 4 the connectors of Enertex® EibPC are presented. The connectors are

1. Power supply 20-30V DC
2. LAN-interface
3. RS-232 interface
4. Info LED
5. Reset button

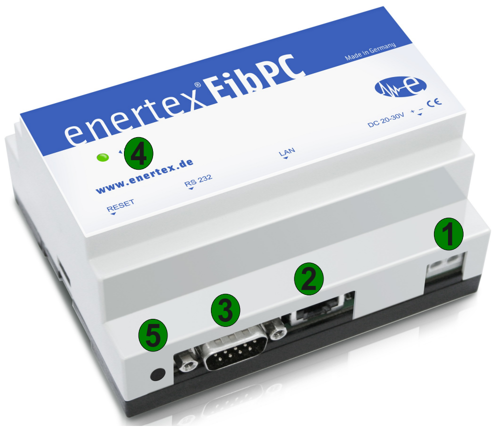


Figure 4: Enertex® EibPC connector assignment

After switching on or a software reset (via the Enertex® EibStudio) of the Enertex® EibPC you can observe the following:

1. First, the LED is continuously on (feedback on the power supply) and the boot process is started.
2. Approximately four minutes after the start, the green LED blinks every second with equal on and off times. Only during this phase, a reset is possible (see below).
3. Shortly after this phase, the RS-232 interface is initialized. A few short pulses can be seen here at the LED.
4. If a telegram is triggered on the bus (KNX switches, etc.), a short pulse sequence is visible at the LED (No user program or the like have to be loaded).

The RS232 interface and the Enertex® EibPC can be simultaneously switched on, if the connection to Enertex® EibPC already exists. It may not be connected after booting the Enertex® EibPC, because the RS-232 interface can not be properly initialized by the Enertex® EibPC.

Because this interface could be sometimes be in a locked state, it should be reset once. The interface is accessed and activated only after the transfer and the subsequent start of an application program (in case of a not duly executed separation of a connection to the ets or an access via another LAN subscriber). The Enertex® EibPC uses the so-called tunneling mode of the IP interface.

During the power up the green Info LED is on continuously. Then the Info LED is blinking every second for 3 seconds. Only during this 3s phase the reset button is active.

By pressing the reset the following is triggered:

Reset - factory settings

1. Reset to factory settings of the network configurations
2. Deletion of the application program
3. Deletion the data of the solar altitude

The pin-out of the RS-232 interface of the Enertex® EibPC corresponds to the standard EIA-232.
Use the supplied cable for the communication of the Enertex® EibPC with the KNX™ RS-232 interface.

If you have a Patch 1.101 installed Enertex® EibPC will boot in less then 40 seconds. Otherwise the boot will last for about 4 minutes.

A Patch-Updat will also boot in less then 40 seconds.

Brief instruction

Quick start - Enertex® EibStudio

For the commissioning of the Enertex® EibPC, especially the network setup, you must first configure the firewall on the router and the PC from which you access the Enertex® EibPC so that packets of the Enertex® EibPC will not be blocked.

The Enertex® EibPC must be accessible via a LAN connection via an IP address. The default setting of the Enertex® EibPC is the configuration by a DHCP server, i.e. if a DHCP enabled router is enabled, it assigns the Enertex® EibPC an IP address. Now, this address must be disclosed to the Enertex® EibStudio. In this case, this can be done automatically, i. e. the Enertex® EibStudio finds the Enertex® EibPC within the net on its own. If you want to use more than one Enertex® EibPC in the home network, you should work with fixed IP addresses, as described on page 135.

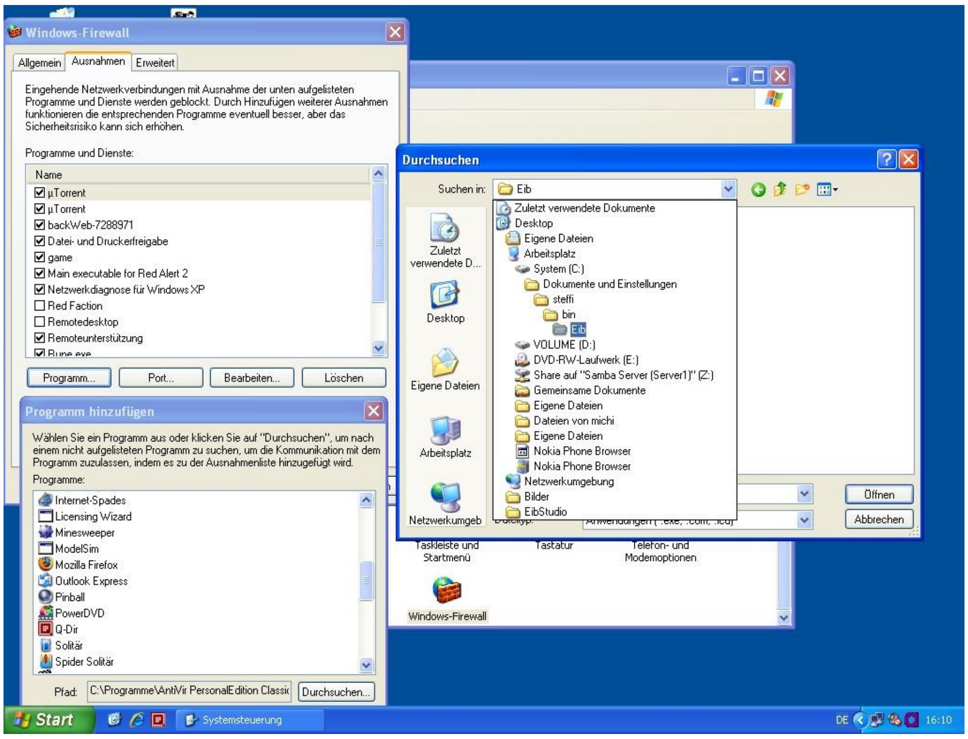
The Enertex® EibPC also finds a definite IP address without a DHCP server. In this case, when using a peer-to-peer connection, a crossover network cable is necessary!

If your router has no DHCP function, the Enertex® EibPC takes this role and adopts its own IP address, which you can retrieve from the Enertex® EibStudio (see below for more). This requires that any used firewall is configured correctly. If you want to work with a fixed IP address, please read on from page 135. With direct connection to your PC (peer-to-peer) you need a crossover cable.

1. Starting the Enertex® EibStudio

The included software Enertex® EibStudio is simply started by double-clicking. There is no installation in the conventional sense. For example, Enertex® EibStudio does not edit the Windows® registry.

2. Configuration of the firewall



Eingehende Netzwerkverbindungen mit Ausnahme der unten aufgelisteten Programme und Dienste werden geblockt. Durch Hinzufügen weiterer Ausnahmen funktionieren die entsprechenden Programme eventuell besser, aber das Sicherheitsrisiko kann sich erhöhen.

Wählen Sie ein Programm aus oder klicken Sie auf "Durchsuchen", um nach einem nicht aufgelisteten Programm zu suchen, um die Kommunikation mit dem Programm zuzulassen, indem es zu der Ausnahmenliste hinzugefügt wird.

Figure 1: Configuration of Windows®-Firewall

Important: An active firewall possibly presents the communication between the Enertex® EibStudio and the Enertex® EibPC.

When you query the IP address of your EibPC for the first time with the button *Automatic* of the dialog according to Figure 3, the Windows®-Firewall reports a message. In this message you have to allow the program nconf, which is responsible for LAN communication of Enertex® EibStudio , the access on the LAN. With it Windows® applies an automatic exception for the firewall. In this case you need not set additional settings in your Windows® firewall. So, the procedure shown in Figure 1 can be skipped.

Windows 7 and Windows 8.1

Various security software packages may prevent a program from loading into the Enertex® EibPC. Windows 8.1 users are especially affected. In addition to the firewall problems, file permissions are also relevant. The security software sometimes does not allow EibStudio to create subdirectories in the "Programs" folder. This is not possible even if you change the corresponding settings. There are two ways to get around this:

1. Copy Eibstudio to your desktop or user directory and run it there.
2. Run EibStudio from the program directory with the "-D-directory" parameter, where directory must be a path in your user directory, e.g. Download. To start a program with a parameter, you must first create a link to this file (here, EibStudioVx_xxx.exe) and then enter the parameters in the link.

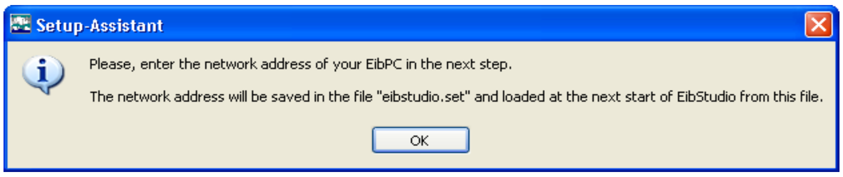
For a Windows® Firewall, you must establish an exception for the program nconf.exe (part program of the Enertex® EibStudio). For this purpose, Enertex® EibStudio has to be started at least once. You can find nconf.exe after calling Enertex® Windows® EibStudio in your user directory as shown in Figure 1.To configure this, select within the dialog “Program exceptions“ of the firewall- sub-dialog “Add program“ - (here) search and set your path corresponding to “C:\Documents and Settings\YOUR USERNAME\bin\eib\nconf.exe“. Alternatively, activate the communication port 4805 and 4806 for UDP telegrams in your firewall. If you want to send UDP telegrams, you must also open the communication port 4807.

When you first start the Enertex® EibStudio, you will be guided through a setup routine to make the necessary settings for the LAN connection between the Enertex® EibPC and the Enertex® EibStudio:

In the automatically started setup wizard (Figure 3), select the button “Automatic“. If you have a DHCP server on the network, this DHCP server assigns the Enertex® EibPC the IP address. If your Enertex® EibPC doesnot find any DHCP server, it assigns itself an IP address after booting. This can be useful if the Enertex® EibPC is directly connected to the PC. In this case, however, at least a LAN crossover cable between the PC and the Enertex® EibPC is necessary.

When you click the button “Automatic“, the address of the Enertex® EibPC appears within the window “Messages“ and will be automatically entered into the dialog of Figure 3. You can also access this dialog by OPTIONS – NETWORK SETTINGS.

Then, you save by OPTIONS – SAVE NETWORK SETTINGS your network configurations.

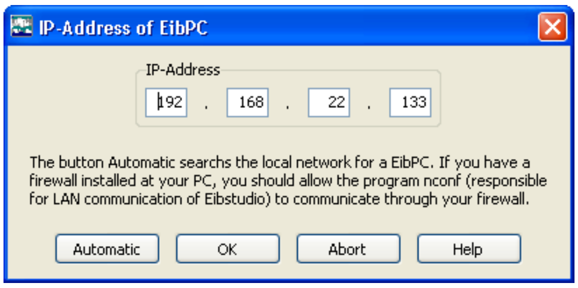

Setup-Assistant

Please, enter the network address of your EibPC in the next step.

The network address will be saved in the file "eibstudio.set" and loaded at the next start of EibStudio from this file.

Figure 2: Setup wizard 1/3

The Enertex® EibPC can be searched by the button “Automatic“


IP-Address of EibPC

IP-Address: 192 . 168 . 22 . 133

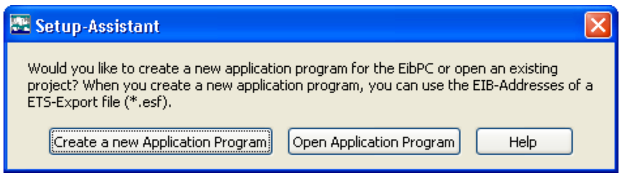
The button Automatic searchs the local network for a EibPC. If you have a firewall installed at your PC, you should allow the program nconf (responsible for LAN communication of Eibstudio) to communicate through your firewall.

Figure 3: Setup wizard 2/3

If you want to work with fixed IP addresses, more information can be found from page 135 on.

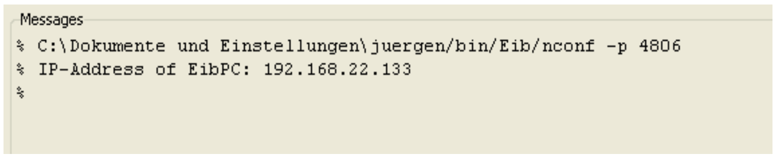
You will be prompted to create a new application program. Click the appropriate button (Figure 4), thus in the next step you can import the ESF data of your ets project into the Enertex® EibStudio. Read more about how to previously export the data from the ets (i.e. the required ESF file) on page 151

The Enertex® EibPC has been found.


Setup-Assistant

Would you like to create a new application program for the EibPC or open an existing project? When you create a new application program, you can use the EIB-Addresses of a ETS-Export file (*.esf).

Figure 4: Setup wizard 3/3

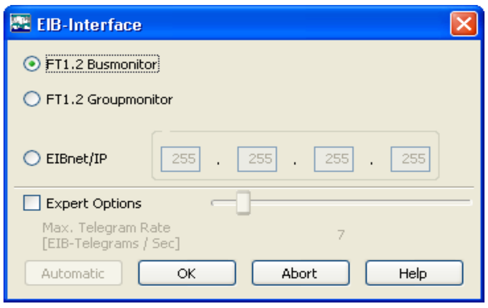

Messages

```
% C:\Dokumente und Einstellungen\juergen/bin/Eib/nconf -p 4806
% IP-Address of EibPC: 192.168.22.133
%
```

Figure 5: Message windows displays, the Enertex® EibPC has been found

Configure the Interface

The menu EIB-Interface allows to setup the KNXTM interface, which the Enertex® EibPC uses for bus access.


EIB-Interface

[✓ FT1.2 Busmonitor]

FT1.2 Groupmonitor

EIBnet/IP 255 . 255 . 255 . 255

Expert Options

Max. Telegram Rate [EIB-Telegrams / Sec] 7

Figure 6: Configure interface

- In case of an FT1.2 interface, please note the commissioning instruction on page 15 and the following You can operate the FT1.2 interface in the bus monitor mode or in the group monitor mode, the latter acknowledging all KNXTM telegrams. Basically, the group monitor mode is more tolerant if bus error occur. Please also note the information on page 139. If a FT1.2 interface has been operated in the bus monitor mode most recently, and then shall be switched to the group monitor mode, it has to be reset by a short interruption of its power supply before the application program is transmitted to the Enertex® EibPC.

- In case of an IP interface, please note the commissioning instruction on page 15 and the following and enter the relevant data of the interface into the above configuration wizard.

  The interface is accessed and activated only after the transfer and the subsequent start of an application program

  The Enertex® EibPC uses the so-called tunneling mode of the IP interface. Thereby, it is exclusively occupied, i.e. you can not address the interface in this mode from the ETS. Nevertheless, in order to allow a short-term access to the interface you can separate the interface via the menu item Connect and Disconnect from Enertex® EibPC and reconnect with it, respectively.

- With the telegram rate limitation (see Figure 6, the default value is seven telegrams per second), you can avoid overloading the KNXTM bus and guarantee the stable operation of your installation.

The further structure of the manual

Once you have configured the operational network setup, you can start to use the Enertex® EibPC within the KNX^{TM}-network.

You can use this manual in different manners: The various sections are constructed to be largely self-contained. Therefore, important details can be declared twice in different sections.

Functional block

From page 24 on, we show the application of finished (supplied) functional blocks. For this purpose you do not need any programming skills. Using an example, we show the simple usability of these functional blocks. Thus, you can create an automation within minutes. For the available functional blocks, a functional block manual exists, which explains the different applications.

If you want to exhaust all possibilities of Enertex® EibPC, we recommend the section "Step by Step" to read.

Introduction programming

From page 32 on, programming the Enertex® EibPC is presented in the form of a "step-by-step introduction. You will be able to exploit completely the possibilities of the Enertex® EibPC. The programming is simple and consistent. The manual does not require any programming knowledge.

Complete description of the Enertex® EibPC

From page 125 on, the full extent of the programming environment is systematically explained.

Self-defined functional blocks

On page 313 it is demonstrated how to define functional blocks. You can generate your own libraries with functional blocks, to be easily re-used in several different projects. By this way, you can perform repetitive tasks with almost no effort and in a short time.

Together with the Enertex® EibPC you purchase a license to use the Enertex® EibStudio. Enertex® EibStudio and all components may be used only in conjunction with the Enertex® EibPC.

Using pre-assembled functional blocks

If you want to use only basic functions of the Enertex® EibPC, it is sufficient for you to use "macros". Macros are predefined functional blocks which are available to users in the form of a so-called library.

Macros/Functional blocks

Standards

Predefined libraries

The included library of Enertex® Bayern GmbH provides you with the complete automation solutions for the following tasks:

Library	***Applications***
EnertexLight.lib	Stair lights, hallway lights with automatic timer
EnertexShadowing.lib	Shading of houses
EnertexTimingElements.lib	Timers, time control
EnertexWigaENG.lib	Winter garden control
EnertexENG.lib	Useful macros
EnertexWebENGV2.lib	Web-Interface macros (Option NP needed)
EnertexCommandFusionENG.lib	Interface to Command Fusion (iPhone or iPad) (Option NP needed)
EnertexLogic.lib	Simple Logic modules
EnertexOneWireENG.lib	Macros for using 1-Wire Sensors with additional Adapter HA7E (see www.shop.enertex.de)
EnertexCommandFusionENG.lib	Controlling a Squeezebox (Option NP needed)
EnertexTimeswitchesV2.lib	Timeswitches
EnertexFlash.lib	Lonh term storage of data in the internal flash
EnertexScenes.lib	Easy to use scene - macros
EnertexPhyMonitor.lib	Physicalliy Monitoring of KNX devices
EnertexPresence.lib	Presence simulation with the Enertex® EibPC
EnertexSqueezebox.lib	Controlling a Logitech-Sequeezebox
EnertexRussound.lib	Controlling a Multiroom Audio System „Russound“
EnertexCodingWettbewerb.lib	Different user Makros

These libraries are available in the download area of Enertex® EibPC (www.eibpc.com).

Additionally there are the following User-Libraries available:

CHGLibrary_FritzBox.lib	Fitzbox Coupling (tested User-Library)
Sonos.lib	Sonos Music player (tested User-Library)

Before the Enertex® EibPC is ready to use, you must perform the basic installation of the Enertex® EibPC:

1. Installing the Enertex® EibPC (see page 17).
2. Configuring the Enertex® EibPC and the Enertex® EibStudio (quick setup on page 20 and extended LAN setup on page 135).
3. Optional: Importing the group addresses of your ets project (see below).

Afterward, you can perform the following tasks:

4. Calling functional blocks
5. Transferring and starting the application program within the Enertex® EibPC

Staircase lighting

By means of an example, we want to explain the Enertex® EibPC: You should parametrize your customers a staircase lighting. The light duration is less 300 then seconds, the installation is depicted by in Figure 1

Exemplary installation

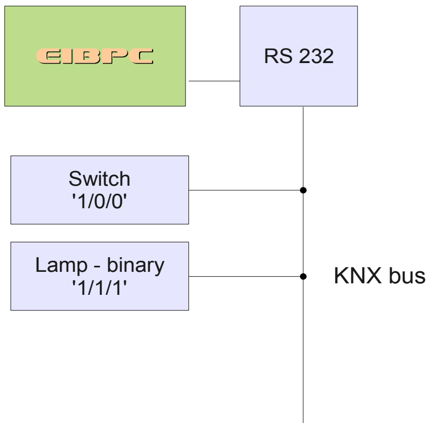
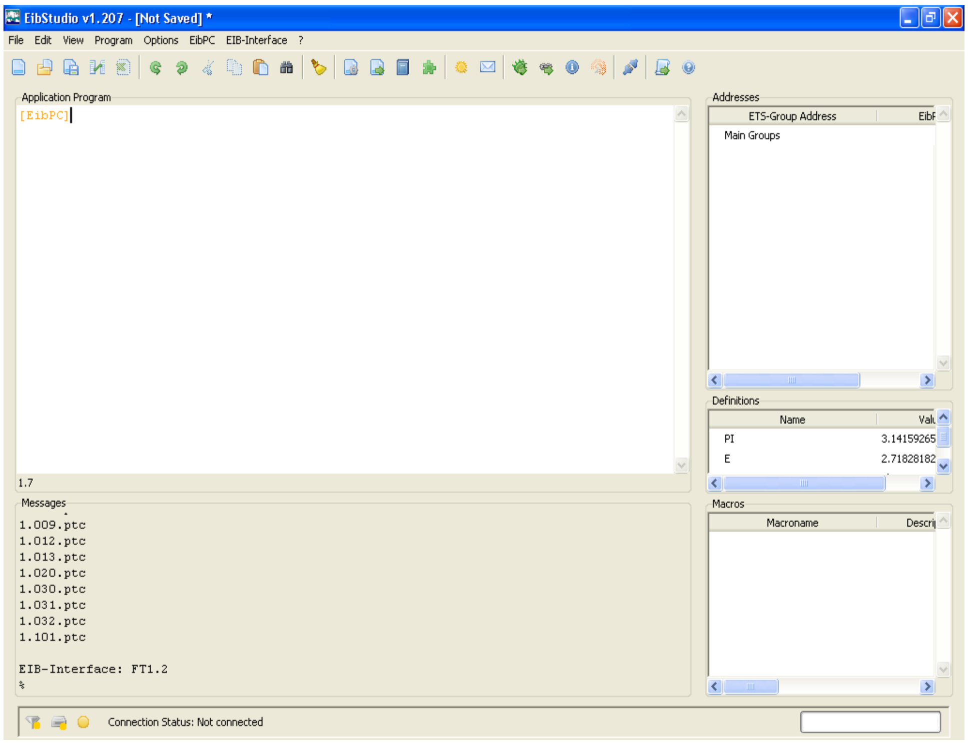


Figure 1: Block diagram staircase lighting

First, you run the Enertex® EibStudio. When you call Enertex® EibStudio for the first time, you will be guided through the setup procedure.

Then, a window appears, as shown in Figure 2.


EibStudio v1.207 - [Not Saved] *
File Edit View Program Options EibPC EIB-Interface ?
Application Program
Addresses
ETS-Group Address
Main Groups
Definitions
Name
Macros
Macroname
Messages
Connection Status: Not connected

```
[EibPC]
```

1.7

Name	Valu
PI	3.14159265
E	2.71828182

```
1.009.ptc
1.012.ptc
1.013.ptc
1.020.ptc
1.030.ptc
1.031.ptc
1.032.ptc
1.101.ptc

EIB-Interface: FT1.2
%
```

Figure 2: Enertex® EibStudio

Look for the white left and still empty region "Application Program". This region of the Enertex® EibStudio represents an integrated text editor. The necessary information for your program is stored here.

ESF-Import: Importing data from the ets

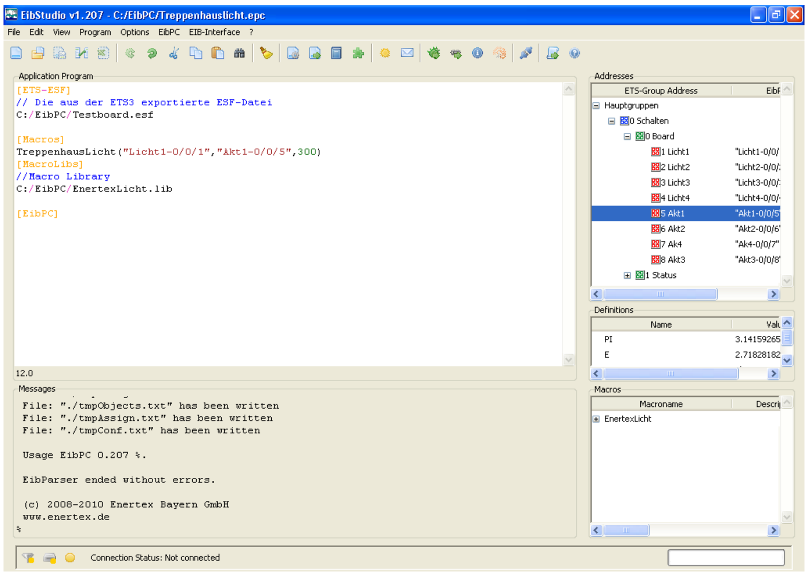
The ESF file of the project is imported by clicking on the highlighted button from Figure 2. You can recognize that the import was successful, because the addresses appear in the left window "addresses" of Figure 3 (For more details, e.g. how to create a ESF file within the ETS, see page 144).

Figure 3: ESF-Import

Then you click on PROGRAM / MACRO LIBRARY and there on the menu item ADD. Select the library "EnertexLicht.lib" and click on ADD.

Your work area should now be similar to Figure 4. Look to the right segment "addresses". Here should be your group addresses parametrized in the ETS, in our example the two addresses "switch" and "lamp", which are displayed with the physical address as an addition in the name.

Macro libraries and ets data are involved.



```
[ETS-ESF]
// Die aus der ETS3 exportierte ESF-Datei
C:/EibPC/Testboard.esf

[Macros]
TreppenhausLicht("Licht1-0/0/1","Akt1-0/0/5",300)
[MacroLibs]
//Macro Library
C:/EibPC/EnertexLicht.lib

[EibPC]
```

```
 File: "./tmpObjects.txt" has been written
 File: "./tmpAssign.txt" has been written
 File: "./tmpConf.txt" has been written

 Usage EibPC 0.207 %.

 EibParser ended without errors.

 (c) 2008-2010 Enertex Bayern GmbH
 www.enertex.de
%
```

Figure 4: Workspace Enertex® EibStudio

Now select PROGRAM / MACRO the staircase lighting. You will see a parameter dialog. Now you can fill this according to your specifications.

Your project should now be completed as shown in Figure 1. Click on the highlighted button to transfer your program to the Enertex® EibPC in order to finish your automation.

Figure 5: Your automation is done

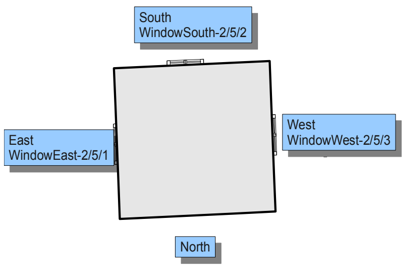
The automation is done

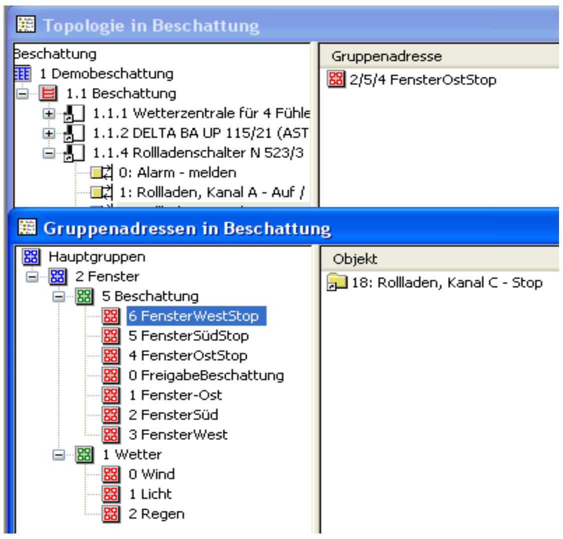
Shading of a house

You want to implement the shading of a house. To this end, you can use macros from the library "EnertexBeschattung.lib.

The orientation of the house is about in the main direction of the compass. The windows are provided with simple blinds. The building owner would now like to shade the windows by a circuit element at the group address "FreigabeBeschattung-2/5/0". The blind shall not completely go down, but remain slightly opened.

Plan and ets





Topologie in Beschattung

Beschattung
1 Demobeschattung
1.1 Beschattung
1.1.1 Wetterzentrale für 4 Fühle
1.1.2 DELTA BA UP 115/21 (AST
1.1.4 Rollladenschalter N 523/3
0: Alarm - melden
1: Rollladen, Kanal A - Auf /

Gruppenadresse
2/5/4 FensterOstStop

Gruppenadressen in Beschattung

Hauptgruppen
2 Fenster
5 Beschattung
6 FensterWestStop
5 FensterSüdStop
4 FensterOstStop
0 FreigabeBeschattung
1 Fenster-Ost
2 FensterSüd
3 FensterWest
1 Wetter
0 Wind
1 Licht
2 Regen

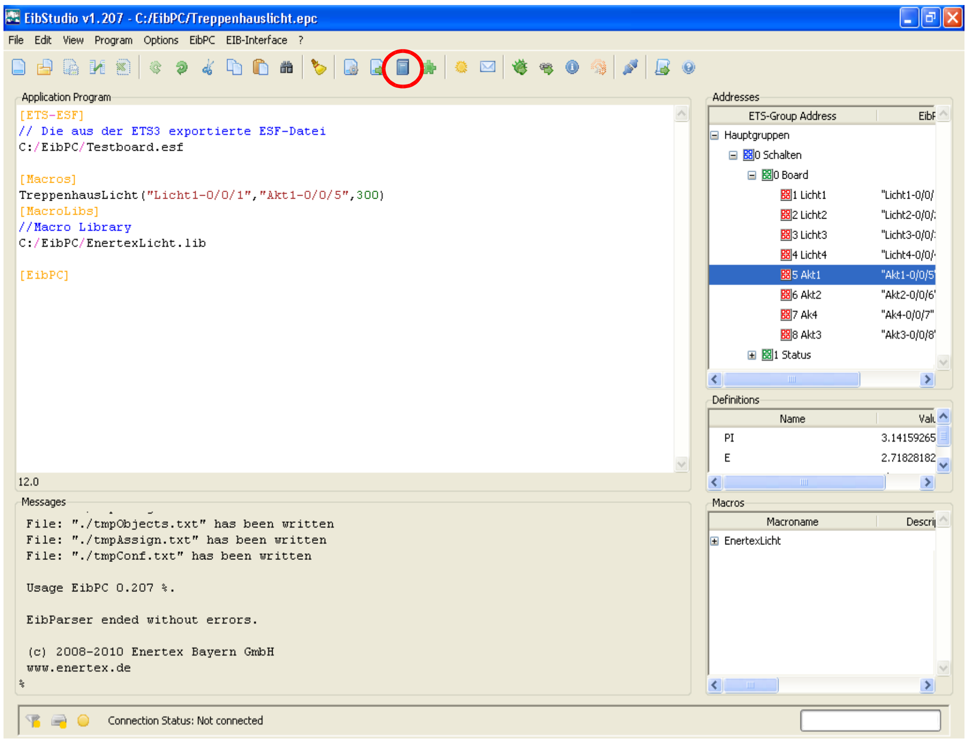
Objekt
18: Rollladen, Kanal C - Stop

Figure 6: A house with three windows – on the right the allocation of the group addresses within the ETS

Calling the macro wizard

If you have set up the project as shown in the above figure with the ets, export the group addresses out of the ets and by means of the OPC-server (file data-exchange). Then, you start the Enertex® EibStudio. When you call the Enertex® EibStudio for the first time, you will be guided through the setup procedure as already shown.

The wizard of the Enertex® EibStudio now imports the group address in a way that - as shown in Figure 7 - these group addresses are displayed in the right plane.



```
[ETS-ESF]
// Die aus der ETS3 exportierte ESF-Datei
C:/EibPC/Testboard.esf

[Macros]
TreppenhausLicht("Licht1-0/0/1","Akt1-0/0/5",300)
[MacroLibs]
//Macro Library
C:/EibPC/EnertexLicht.lib

[EibPC]
```

```
 File: "./tmpObjects.txt" has been written
 File: "./tmpAssign.txt" has been written
 File: "./tmpConf.txt" has been written

 Usage EibPC 0.207 %.

 EibParser ended without errors.

 (c) 2008-2010 Enertex Bayern GmbH
 www.enertex.de
%
```

Figure 7: Import of the group addresses into EibStudio

Now press the labeled button of Figure 7. In this case, a dialog opens and you can add libraries. Choose "EnertexBeschattung.lib" (in the download area of www.eibpc.com). Subsequently, the macros are shown, which are stored in the library (cf. Figure 9).

Visualization with the iPhone

If you have an Enertex® EibPC with software option NP at your disposal, you can visualize your home automation with the Enertex® EibPC also via iPhones or similar devices. However, you need an additional tool, e.g. Command Fusion, in order to enable the iPhone to communicate.

With the aid of Command Fusion, graphically demanding visualizations for the Apple iPhone or iPod can be generated. At www.knx-user-forum.de/ you will find a link to the freely available design based on the design of the Enertex® Bayern GmbH.

Appropriate macros for the connection are pre-defined in the library EnertexCommandFusion.lib. The application of the macros is as shown recently. However, the handling of Command Fusion and the required background knowledge are not content of this manual. At www.knx-user-forum.de/ you will find further articles on this. At www.enertex.de/downloads/d-eibpc/DokuCF-1.pdf you will find a more detailed example of the implementation of the connection to the Command Fusion Tool.

EibPC and iPhone – a dream team...

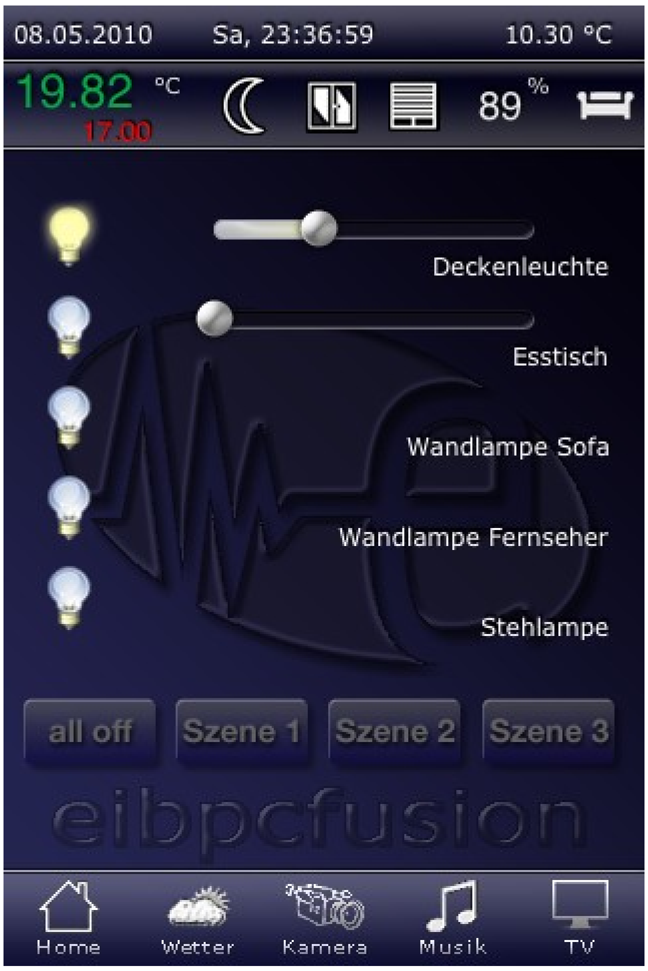
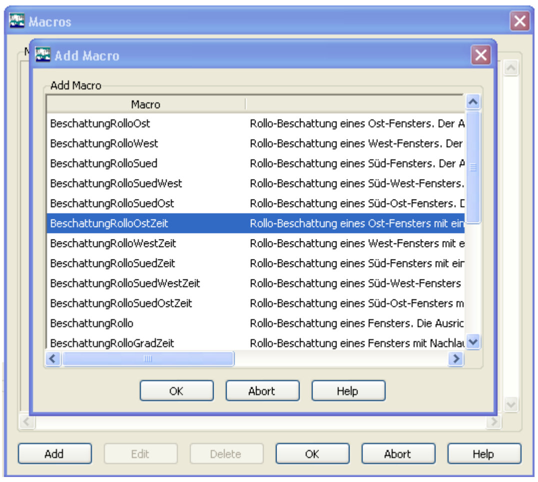


Figure 8: Visualization with the iPhone

Now you can click the macro "BeschattungRolloOstZeit" for the east window, and afterward parametrize it (see Figure 1).

Select macro



Macro	
BeschattungRolloOst	Rollo-Beschattung eines Ost-Fensters. Der A
BeschattungRolloWest	Rollo-Beschattung eines West-Fensters. Der
BeschattungRolloSued	Rollo-Beschattung eines Süd-Fensters. Der A
BeschattungRolloSuedWest	Rollo-Beschattung eines Süd-West-Fensters.
BeschattungRolloSuedOst	Rollo-Beschattung eines Süd-Ost-Fensters. D
BeschattungRolloOstZeit	Rollo-Beschattung eines Ost-Fensters mit ein
BeschattungRolloWestZeit	Rollo-Beschattung eines West-Fensters mit e
BeschattungRolloSuedZeit	Rollo-Beschattung eines Süd-Fensters mit ein
BeschattungRolloSuedWestZeit	Rollo-Beschattung eines Süd-West-Fensters
BeschattungRolloSuedOstZeit	Rollo-Beschattung eines Süd-Ost-Fensters m
BeschattungRollo	Rollo-Beschattung eines Fensters. Die Ausric
BeschattungRolloGradZeit	Rollo-Beschattung eines Fensters mit Nachlau

Figure 9: Die Macros of the library EnertexBeschattung.lib

Now this macro is parametrized for all three windows. The complete automation of the three windows is shown in Figure 10.

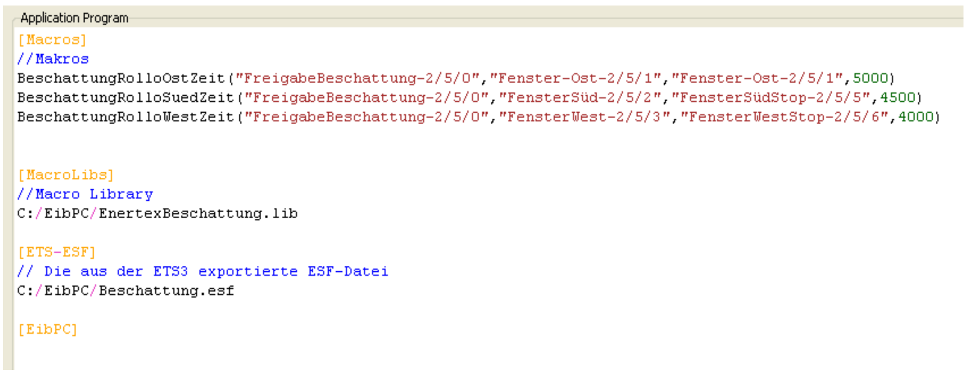
Note 1:

You can enter and modify the data directly in the text window of the user program, instead of working with the wizard.

Note 2:

The solar altitude depends on the location of the house. The defaults are set roughly corresponding to the middle of Germany. You can change these settings according to your actual location by changing the installation location inside of the menu OPTIONS – SETUP COORDINATES FOR CALCULATION OF SOLAR ALTITUDE (see page 184).

Macro editing



```
[Macros]
//Makros
BeschattungRolloOstZeit("FreigabeBeschattung-2/5/0","Fenster-Ost-2/5/1","Fenster-Ost-2/5/1",5000)
BeschattungRolloSuedZeit("FreigabeBeschattung-2/5/0","FensterSüd-2/5/2","FensterSüdStop-2/5/5",4500)
BeschattungRolloWestZeit("FreigabeBeschattung-2/5/0","FensterWest-2/5/3","FensterWestStop-2/5/6",4000)


[MacroLibs]
//Macro Library
C:/EibPC/EnertexBeschattung.lib

[ETS-ESF]
// Die aus der ETS3 exportierte ESF-Datei
C:/EibPC/Beschattung.esf

[EibPC]
```

Figure 10: Die shading of the windows – The complete automation.

The automation is done.

Programming - First steps

In the following section we will show you how to program the Enertex® EibPC. We instruct you step by step in the simple and brief programming language of the Enertex® EibPC.
Creating a program can be done in just a few steps:

Five steps to create a program

1. Installing the Enertex® EibPC (see page 17).
2. Configuring the Enertex® EibPC and Enertex® EibStudio (LAN setup, see page 17 and the following and page 135).
3. Optional: Importing the ets group addresses of your project.
4. Creating an application program in the Enertex® EibStudio.
5. Transferring and starting the application program in the Enertex® EibPC.

For the subsequent introduction to the programming of the Enertex® EibPC it is necessary to complete primarily the first two steps "Installation" and "Configuration". You can also read it just to familiarize yourself with your Enertex® EibPC. We describe the next steps within this part of the manual.

Structure of the step-by-step guide

In the following we want to introduce the user to the operation of the Enertex® EibStudio and the possibilities of the Enertex® EibPC by means of examples, step by step.

Enertex® EibStudio = Development environment for the Enertex® EibPC

The Enertex® EibStudio is an application program for Windows® and Linux® computers. It is the development environment for programming the Enertex® EibPC.

Together with the Enertex® EibPC you purchase a license to use this program.
The Enertex® EibStudio and all components may be used only in conjunction with the Enertex® EibPC

User programs are executed in the Enertex® EibPC within 1 ms cycle time only

Application programs can be conveniently created, modified and maintained by the Enertex® EibStudio. By pressing a button, the file is translated (compiled) by an integrated compiler, the so-called Enertex® EibParser, and sent to the Enertex® EibPC. The EibParser optimizes and controls the code of the user. Hence, the Enertex® EibParser ensures that even large programs can be executed within the cycle time of the Enertex® EibPC. The cycle time of Enertex® EibPC is about 1 ms (for more details see page 125).

A simple program with no group address import

Example 1: A switch, a dimmer and a switch actuator

As shown in Figure Figure 1, the EibPC is connected to the KNX™bus via an RS-232 interface with FT1.2 protocol. In addition, there is a switch which sends on the address '0/0/1' a on or off signal, a switch actuator for a lamp which receives on address '1/1/1', and a dimmer which receives a percentage value on the address '1/1/2'.

The addresses for KNX actuators and switches were assigned and programmed within the ets. The dimmer is parametrized in such a way that it sends the percentage via the address '1/1/2' immediately. The dimmer has not defined an additional switching object but rather a connected lamp lights with the specified brightness if the value is greater 0%.
We assume that initially the addresses are not exported from the ets. Therefore you need no ets for the following example You must simply be aware of the group addresses which have been assigned.

Your KNX ™ network is configured as follows:
Switch sends to '1/0/0'
Actuator (lamp) receives to '1/1/1'
Dimmer receive to '1/1/2'

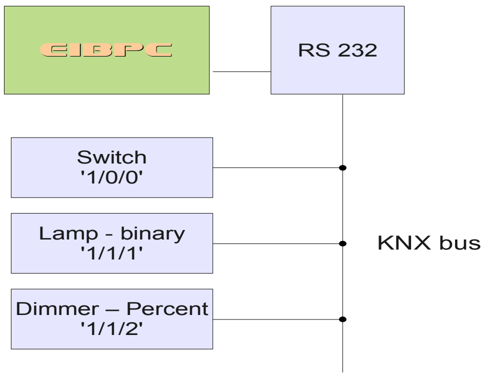


Figure 1: Block diagram Example 1

You now want to program the following functionality in your KNX™ network:

The task...

If the switch is pressed "ON", the lamp will turn on and the dimmer will go to 80%.
If it goes to "OFF", will the lights go out.

... and what is the problem?

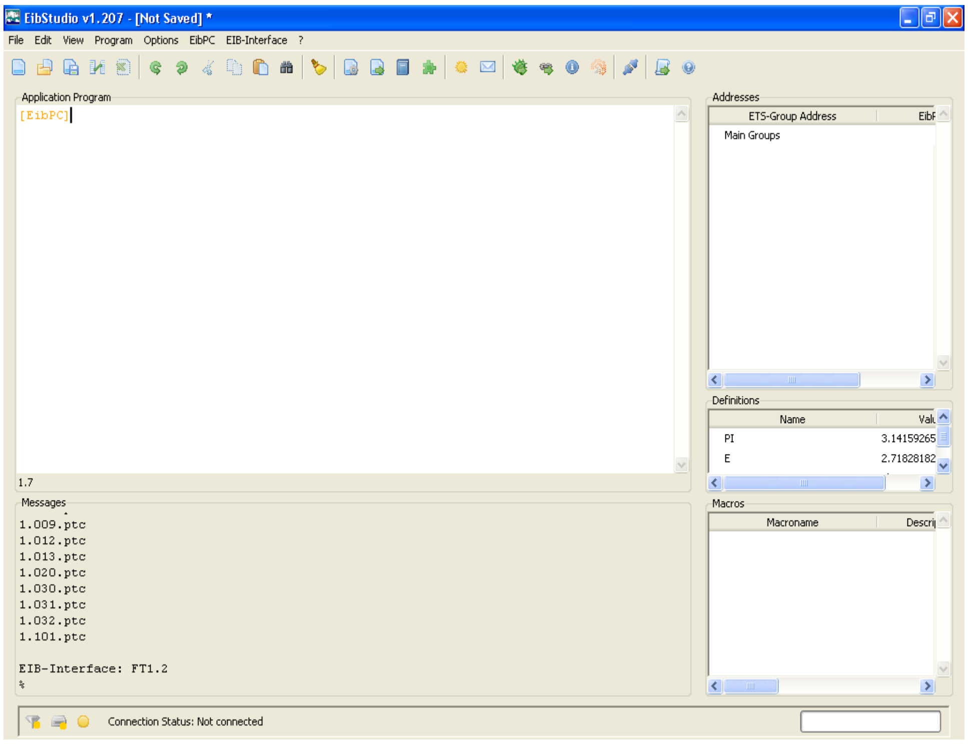
Background:
As you know, this cannot be achieved with KNX™ programming by allocating group addresses, because the switching actuator and the dimmer expects a binary value ("On or "Off"). This means, the switch has to trigger two different telegrams. But with the switch, you can at least only trigger a telegram. Most switches do not even have the option to send a percentage value to the bus. As simple a task this may seem: Without automation it is not solvable. Perhaps you think at this example also of scene actuators. We will get down to this later.

By means of the Enertex® EibPC, this task is simple to solve. We presume you have the documentation of the KNX network at hand (so you know which actuator is configured with which address), but you have no access to the ets or the original records.

First, you run the Enertex® EibStudio. Then you will see a window as shown in Figure 2.

Requirements:
- The Enertex® EibPC is installed in the LAN (see page17).
- Enertex® EibStudio is configured (LAN setup page 135).

The area "user program" is the integrated text editor for creating programs



```
[EibPC]
```

Name	Valu
PI	3.14159265
E	2.71828182

```
1.7
```

```
1.009.ptc
1.012.ptc
1.013.ptc
1.020.ptc
1.030.ptc
1.031.ptc
1.032.ptc
1.101.ptc

EIB-Interface: FT1.2
%
```

Connection Status: Not connected

Figure 2: Enertex® EibStudio

Look for the white left and still empty area "Application Program". This area of the Enertex® EibStudio represents an integrated text editor, because the application program is a simple text.

Back to the problem: One switch - two telegrams. Simply read through the following program, we explain it step by step. But you should already be able to recognize: It cannot be more simple.

This "program" must be copied or manually entered into the area "user program"

Distinguish:
0 and O
„Zero“ and letter „O“

```
[EibPC]
if ('1/0/0'b01==EIN) then write('1/1/1'b01,EIN); write('1/1/2'u08,80%) endif
if ('1/0/0'b01==AUS) then write('1/1/1'b01,AUS); write('1/1/2'u08,0%) endif
```

Now just copy the text by using the clipboard into the field "user program".

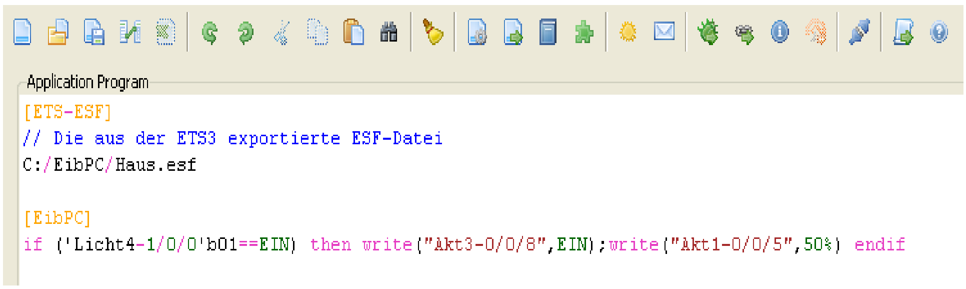
Figure 3: Compile and send to the Enertex® EibPC

This program must now be entered into the Enertex® EibStudio and subsequently transferred to the Enertex® EibPC. To do this, press the selected button (Figure 3), or click into the menu "PROGRAM" on the appropriate button or use the shortcut "CTRL-SHIFT-B".

Button or menu or shortcut

You can see the syntax highlighting in Figure 4: The Enertex® EibStudio assists you in correctly entering programs during their creation. You can find further information on syntax highlighting and code completion of keywords on page Fehler: Referenz nicht gefunden and the following .

Look for:
Syntax highlighting (page 128)
Code completion (page Fehler: Referenz nicht gefunden)



```
[ETS-ESF]
// Die aus der ETS3 exportierte ESF-Datei
C:/EibPC/Haus.esf

[EibPC]
if ('Licht4-1/0/0'b01==EIN) then write("Akt3-0/0/8",EIN);write("Akt1-0/0/5",50%) endif
```

Figure 4: Syntax highlighting in the Enertex® EibStudio

Let us look at the code more closely. Our task consists of a conditional statement ("When the switch is pressed ..."). For this purpose, the important "if" statement exists in the programming language.

if- statement

if ('1/0/0'b01==ON) then write('1/1/1'b01,ON); write('1/1/2'u08,80%) endif

The syntax of an if statement is as follows:

if (Query condition) then Statement endif

The statement block

If the query condition is met, the statement is executed once in the then branch. Thus, the if statement only reacts to changes. More about that this in this introduction.

It can also be executed multiple statements as a block. In this case, the instructions are separated by semicolons. **After the last statement there is not a semicolon**.

A query condition is considered to be fulfilled, if it takes the value 1. If it takes the value 0, then the then-branch is not executed.

What does '1/0/0'b01==EIN?

Manual group address

'1/0/0b1' is the syntax for a group address 1/0/0 with a binary datatyp. We have not imported the group address and its name from the ETS, but used it directly. In this manual, we refer to this as "manual" group entered addresses. The "generation law" for this is simply the group address surrounded by two '-marks (for more information, see on page 153). Thus, the "Generation Law" for the syntax of manually assigned addresses is defined by:

Manual address: 'group address'data type

The data type must be specified for manual group addresses and values!

Group addresses have different information depending on the number of bits in the corresponding KNX™ telegram and the interpretation of this information. We are talking about data types. If we use the manual group address and value, we must know and specify the data type.

In our example, we used two data types (more on page 153):

- The binary (value 0 and 1), characterized by the suffix 'b01' and
- the percentage (values 0.0 to 100.0), characterized by the addition of '%'.

“One“ or “One percent“?

Therefore, a value of 1 is the binary data type or data type assigned to the percentage. When we write 1b01, we are referring to the binary, when we write 1%, we mean the percentage data. Similarly, we add the data type '1/0/0'b01' to the notation of the group address. This means we want to transfer to this address an information of one bit.

A little help: Predefined values

The Enertex® EibStudio knows predefined constants to make the program more readable. The constant OFF is equivalent to 0b01 and ON to 1b01.

Because 1b01 and 1% are of different data types, you can not directly connect these to each other. Thus, the allocation *error=1b01 + 1%* to form a variable is not possible and will be reported as an error when compiling the program. We need the convert function to link different data types. We will return to this later (on page 54 and page 208 you will find more).

An overview of the predefined variables is listed on page 339. A data type is always a letter and two numerals. The only exception to this is the percentage and string types for TCP/IP telegramms. Other data types can be found on page 153 or later in this step-by-step instructions.

The data type "%" is compatible with the data type "u08" and will be internally adjusted by scaling (whereas the data type can be specified in tenths of a percent accuracy). 100% internally corresponds to a value of 255.

Now we return to expression: '1/0/0'b01==ON

And back to the example

A comparison is with „==“ programmed.

The two equal signs == are responsible for the comparison. The result of a comparison of two expressions (variables, group addresses etc.) can only take the following values:

0b01, if the two values are not equal,

1b01, if the two values are equal.

You can find further comparative links on page 172.

Now, if the value 1b01 (ON) is sent to the bus at address '1/0/0', the comparison should have the value ON. Thus, the condition of the if statement is true (equal to the value 1b01 or ON) and the then branch is executed. By the write-function (see page 164), a valid EIB-telegram with the specified value is written to the bus at the given address.

The statement

```
write('1/1/1'b01,ON)
```

writes to the group address 1/1/0 the value 1 (data types 1b01).

Finally, you should have fully understood:

```
if ('1/0/0'b01==ON) then write('1/1/1'b01,ON); write('1/1/2'u08,80%) endif
```

If not, read through this example again before you attend to the next example.

A program with group address import

Embed import file

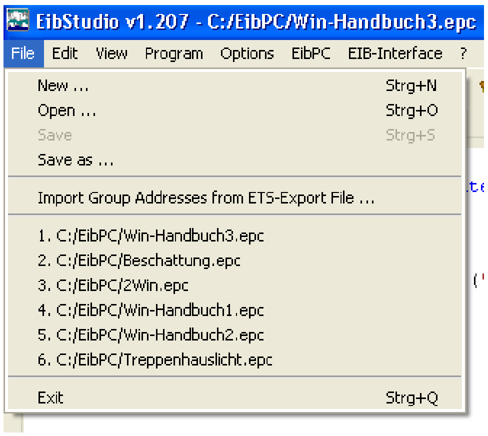
Example 2: Everything as usual - but now with import addresses

The structure remains as described in Figure Figure 1. Now, in this example we want to work with the addresses exported from the ets. You can export the addresses as described on page 151. To use these addresses, they must be imported into the Enertex® EibStudio.

Look to the dialog File in Figure 5, click Import Group Addresses from ETS-Export File (also refer to page 144).

[Sections]

Besides the program, various data of the project, such as installation location, e-mail configuration etc are also processed, there are so-called sections within the user program. These are characterized by a name in brackets. A section starts with its name and ends with the beginning of the next section or the end of the file. More information about the sections can also be found on page 125.

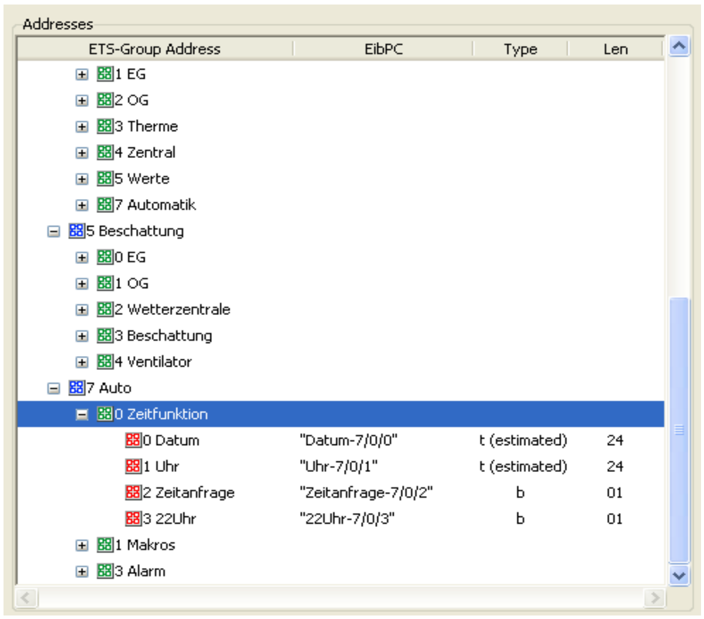


EibStudio v1.207 - C:/EibPC/Win-Handbuch3.epc
File Edit View Program Options EibPC EIB-Interface ?
New ... Strg+N
Open ... Strg+O
Save Strg+S
Save as ...
Import Group Addresses from ETS-Export File ...
1. C:/EibPC/Win-Handbuch3.epc
2. C:/EibPC/Beschattung.epc
3. C:/EibPC/2Win.epc
4. C:/EibPC/Win-Handbuch1.epc
5. C:/EibPC/Win-Handbuch2.epc
6. C:/EibPC/Treppenhauslicht.epc
Exit Strg+Q

Figure 5: Download ets Export File

The path to the file appears in the section [ETS-ESF]. Now, Enertex® EibStudio and thus the Enertex® EibPC 'knows' the group addresses from your ets programming. Now, all addresses of the ets appear in the upper right area designated "addresses". By simply drag and drop, or CTRL+C or CTRL+V you can now use the addresses in the application program (to represent the group address, see next page).

Imported Addresses



Addresses

ETS-Group Address	EibPC	Type	Len
1 EG			
2 OG			
3 Therme			
4 Zentral			
5 Werte			
7 Automatik			
5 Beschattung			
0 EG			
1 OG			
2 Wetterzentrale			
3 Beschattung			
4 Ventilator			
7 Auto			
0 Zeitfunktion			
0 Datum	"Datum-7/0/0"	t (estimated)	24
1 Uhr	"Uhr-7/0/1"	t (estimated)	24
2 Zeitanfrage	"Zeitanfrage-7/0/2"	b	01
3 22Uhr	"22Uhr-7/0/3"	b	01
1 Makros			
3 Alarm			

Figure 6: Imported addresses

Back to the problem: One switch - two telegrams, now with imported group addresses. Our implementation is the following:

```
[EibPC]
if ("Switch-1/0/0"==ON) then write("Lamp-1/1/1",EIN); write("Dimmer-1/1/2",80%) endif
if ("Switch-1/0/0"==OFF) then write("Lamp-1/1/1",AUS); write("Dimmer-1/1/2",0%) endif
```

Most of it is now expected to be known. But we highlight the group address once again.

What does "Switch-1/0/0“?

The sub-group name is used only

Within the ets, a group address consists of three parts:

1. Name of the main group
2. Name of the middle group
3. Name of the subgroup

The group address is unique only within the combination of the three names. In order not to import the full name and thus make the code unreadable, we use imported group addresses only with the help of the subgroup name. Because this is not unique, the import function adds the physical group address after a "-" character automatically. By two quotation marks the whole thing is then framed. So, the "generation law" for imported group addresses is:

Imported address: "Sub-Group Name-Group Address“{data type}

Data type is optional

Types of group addresses are indirectly determined from the usage and have to be specified by the user

On the part of the ETS software, the export is not fully implemented. Especially for data types with bit lengths of 8 or more bits, often the bit length, but not its interpretation is exported. Therefore, within the above example write("Dimmer-1/1/2"u08,80%) we can add the data type of the percentage as an attachment to the group address (u08). However, in any case, the compiler checks the data type's connectivity: In the case of the write function, this means that the first argument, the group address, **must** have the same data type as the second in the example of the percentage. Thus, the data type of the group address is indirectly specified by using the write function. We call this approach an implicit conversion (see page 153).

Background information:

Types consistently implemented

The compiler knows the data types, return values and objectives of functions, e. g. the data type of the 1st argument must be equal to the 2nd argument of the write function. Thus, if the second argument has a unique data type only, the first argument is clearly established. If the data types do not match each other, the compiler issues an error message.

Another example: The function sin (sine, page 206) computes the sine of a 32-bit floating-point number in radian. Here the compiler knows: The function sin returns a floating point number and requires a floating point number as its argument. Therefore, in a program statement "a=sin (b)" it is clearly stated that the variable a and the variable b must be of the floating point data type, 32-bit.

For this reason, you can not even program a = 2u16+4u32. Within sums, the compiler states: All summands of a sum must be of the same type. If you add up numbers of different data types, you must resort to the convert-function

The advantage of this "strict" usage of data types is the adaption of the imported group addresses. You never have to try to adapt the imported group addresses. Your program will also not provide any "surprising" results, e. g. occurring in case of involuntary conversion of 32-bit numbers in 16-bit numbers by the "decapitation" of the most significant bits.

Specifying a data type is always optional, even if the import of the group addresses is not unique. Should this cause a problem, the built-in compiler of Enertex® EibStudio points this out to you.

Back to the user program: We can further simplify, if we equip the first if function with an else branch instead of the second if-function. It is worth noting that a statement can not be interrupted by a line break. If the query condition is not met, the else-branch is executed. Thus, the syntax is:

if (Query condition) then Statement{block}1 else Statement{block}2 endif

So, the implementation of the user program is

```
[EibPC]
if ("Switch-1/0/0"==ON) then write("Lamp-1/1/1",ON); write("Dimmer-1/1/2"u08,80%) else write("Lamp-1/1/1",OFF); write("Dimmer-1/1/2"u08,0%) endif
```

You may wonder why this is written so small. The answer is:

We want to suggest: A statement can not be simply interrupted by a newline ("RETURN"). Otherwise, during the compilation by the Enertex® EibStudio there would be view a mistake. The Enertex® EibStudio always processes statements line by line. Therefore, once again urgently:

A statement must not be interrupted by a line break.

To make clear more instructions by several lines, use two backslash (\\) characters as wrapping.

Wrap

Thus we can write our program a little clearer. Furthermore we simplify by the help of the else branch:

Everything is legible

Wrapping with \\

```
[EibPC]
if ("Switch-1/0/0"==ON) then                          \\
                        write("Lamp-1/1/1",ON);        \\
                        write("Dimmer-1/1/2"u08,80%)  \\
                     else                              \\
                        write("Lamp-1/1/1",OFF);       \\
                        write("Dimmer-1/1/2"u08,0%)   \\
                     endif
```

We recommend to format the user program as just shown. This tremendously increases the readability for you and thus avoids errors.

Again as a reminder: If several statements are interpreted as a block, a semicolon has to be inserted between statements.

At if-statements, you can alternatively work with curly brackets({}):

Even nicer?

```
[EibPC]
if ("Switch-1/0/0"==ON) then {
                        write("Lamp-1/1/1",ON);
                        write("Dimmer-1/1/2"u08,80%)
                     } else {
                        write("Lamp-1/1/1",OFF);
                        write("Dimmer-1/1/2"u08,0%)
                     } endif
```

Now, we can start our program as shown in Figure 4.

Once again if

A note on the if-statement:

The query condition of the if-statement must be of data type b01, i.e. binary with length 1 bit. This is verified by the integrated compiler of the Enertex® EibStudio. Our variable "Switch-1/0/0" is of data type b01. Therefore, the query

```
if ("Switch-1/0/0"==ON) then          \\
```

is superfluous (but not wrong). In this case,

```
if ("Switch-1/0/0") then          \\
```

can be programmed. However, it would be wrong

```
if ("Switch-1/1/2") then          \\
```

because "Dimmer-1/1/2" represents an unsigned integer. The compiler notices this fact and returns an error.

Comments?

You can comment your programming. There are two possibilities:

1. Comments begin with “//“ and are at the beginning of a line.
2. **Instead** of a statement, comments can be inserted at any point (semi-colon are sensitive), which are surrounded by /* */, e. g.

```
/* This is a comment */
u=5;/* This is second in the comment */; u4=5
```

Congratulations.

Now, you have already understood the most important basics and realized how easy it is to program the Enertex® EibPC.

Below we will show you:

Timer modules, time switches, special functions, variables - The versatility of the Enertex® EibPC.

Commissioning:
The Startup

A switch button

The default setting is always zero.

Again, look at the example above: If a ON is sent to the group address "Switch-1/0/0" (or the manually entered address '1/0'0'b01), the light should be turned on. The default value of variables (more later), group addresses and so on is zero (OFF, 0, 0.0 ...). All objects in your application program are set to be valid.

But, if the light switch has been already pressed, i.e. is really ON, before the Enertex® EibPC is started? How is the program to be optimally designed? You might say: "Sometimes you can live with this fact, in such cases the user has to press ON and OFF and ON again, until something happens.

But, this could be done by the Enertex® EibPC much better.

We have designed the function:

systemstart()	Returns 1b01 at system start

The function has no argument. After the Enertex® EibPC has started the application program, the binary return value is 1b01 (for further documentation of the function, see page 213).

Before we show the application of this function, we still need:

read(*Group address*)	Read request for the group address

By the read function, we are capable to ”request“ the actuator to send its current value to the specified group address (for further details see also page 165). **Important: In order to receive the effective actuator response, the read flag has to be set within the ets.**

Now to the actual program:

When using manual group addresses:
read('1/0/0'b01)

```
[ETS-ESF]
Haus.esf
[EibPC]
if (systemstart()) then read("Switch-1/0/0") endif
if ("Switch-1/0/0"==EIN) then                            \\
                        write("Lamp-1/1/1",EIN);         \\
                        write("Dimmer-1/1/2"u08,80%)  \\
                    else
                        write("Lamp-1/1/1",AUS);         \\
                        write("Dimmer-1/1/2"u08,0%)   \\
                    endif
```

The read request will be now set at system startup. The actuator sends its value. Then, in the user program the if-statement is executed: If the switch is ON and reports this, the lights are switched on.

Some important details

If the switch sends an OFF after the read request caused by the function if systemstart(), the Enertex® EibPC does nothing. Why? We might think that in this case the else-branch is executed after the arrival of the response by the actuator. This is not the case, since at startup all objects and instructions will be accepted as valid. Objects are invalid, if they vary (except only a few special functions). And just when they vary, they in turn trigger actions. The Enertex® EibPC always "remembers" the states of its objects.

The else branch is already valid, since all objects are initialized to zero. However, if you start the application program the else-branch is not executed. Now, if the response to the read request is OFF (0b01), the else-branch is not executed, because the object "Switch-1/0/0" is already initialized with OFF. The re-arrival of the telegram does not lead to a change of states and in the else-branch the Enertex® EibPC does not trigger any telegrams.

Now we can extend the program further, so that the actuators are switched off in any case. It could be that the switch is still ON.

Then a read request should query the state of the switch and the lights are switched accordingly.

```
[ETS-ESF]
/EibPc/Haus.esf
[EibPC]
if (systemstart()) then                                \\
                    read("Switch-1/0/0") ;          \\
                    write("Lamp-1/1/1",OFF);        \\
                    write("Dimmer-1/1/2"u08,0%)   \\
                 endif

if ("Switch-1/0/0"==ON) then                                 \\
                              write("Lamp-1/1/1",ON);          \\
                              write("Dimmer-1/1/2"u08,80%)  \\
                          else
                              write("Lamp-1/1/1",OFF);         \\
                              write("Dimmer-1/1/2"u08,0%)   \\
                          endif
```

Also this program must be transferred to the EibPC.

Initialization of many group addresses

If you need to read more group addresses you can use the [InitGA] section as described at p. 166. In this case, the reads of the group addresses in this section have been read before the program is processed.

```
[ETS-ESF]
/EibPc/Haus.esf
[InitGA]
"Switch-1/0/0"
[EibPC]
if (systemstart())  then {
                    write("Lamp-1/1/1",OFF);
                    write("Dimmer-1/1/2"u08,0%)
                    } endif

if ("Switch-1/0/0"==ON) then  {
                              write("Lamp-1/1/1",ON);
                              write("Dimmer-1/1/2"u08,80%)
                          } else {
                              write("Lamp-1/1/1",OFF);
                              write("Dimmer-1/1/2"u08,0%)
                          } endif
```

Process image: Respond to changes only

The Enertex® EibPC copies the activities on the bus in its cache (from the process image). Typically, every ms the program is passed through again. The Enertex® EibPC examines at which point the cache has been changed and processes the corresponding lines once again. This means in the above example, if the switch once sends an ON at 1/0/0 (previously this was OFF), the lamp is switched ON. In the next program flow (after 1 ms) the cache of 1/0/0 is already ON and thus, as expected, an ON is not sent once again.

Separate Power On and Off

Suppose we have an installation as in Figure 7.

If the closing-switch “ON“ is pressed, the lamp should turn on and the dimmer is to go to 80%.

When the switch is OFF, the lights will go out.

A telegram for ON and a separate telegram for OFF

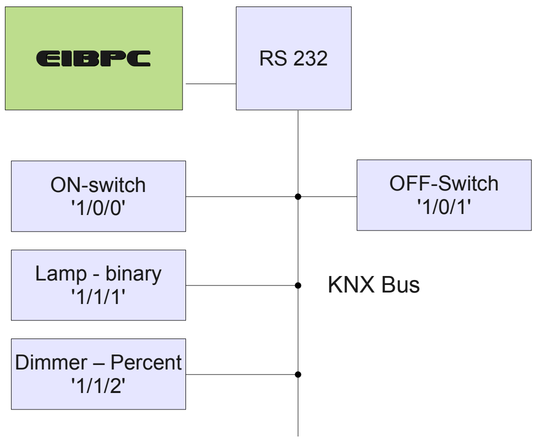


Figure 7: Block diagram of Power On and Off

If activated, our ON switch will send only ON at 1/0/0. The OFF-switch will send only OFF on 1/0/1. Because of the properties of the process image, the Enertex® EibPC responds to changes only. If we forget this, it might seem as if the following program was not running correctly - but this is not the case - and we will explain immediately why.

That will not work, as you might intend it

```
// NOTE: DEOS NOT WORK AS INTENDED
if ("ON-Switch-1/0/0"==ON) then                                   \\
                    write("Lamp-1/1/1",ON);          \\
                    write("Dimmer-1/1/2"u08,80%)  \\
                  endif
if ("OFF-Switch-1/0/1"==OFF) then                                 \\
                    write("Lamp-1/1/1",OFF);         \\
                    write("Dimmer-1/1/2"u08,0%)   \\
                  endif
```

At the first pass (after startup) the program may turn the lights on or off once again. But if the first or second if-statement is executed once again and ON will be sent to the address 1/0/0 again, nothing happens. After all, the image of the group address has already been properly evaluated and is ON. Thus, we have to know if a group address is sent again.

For such tasks, there is the function event:

event(Group address)

It is a function which indicates the receiving of a message from the bus with the given group address. It does not check whether the message has changed, what content it has or what type it is. Once a message arrives, this function goes to ON for a processing cycle of Enertex® EibPC. In Figure 8, this is shown schematically. (For further details and more event-Functions see p. 169).

*and-function: Only when a telegram arrives **and** the switch is ON, will only be switched on.*

```
// Everything is OK
if event("ON-Switch-1/0/0") and  ("ON-Switch-1/0/0"==ON) then      \\
                    write("Lamp-1/1/1",ON);                  \\
                    write("Dimmer-1/1/2"u08,80%)             \\
                  endif
if event("OFF-Switch-1/0/1") and ("OFF-Switch-1/0/1"==OFF) then   \\
                    write("Lamp-1/1/1",OFF);                 \\
                    write("Dimmer-1/1/2"u08,0%)              \\
                  endif
```

We still use the and function (AND). It links the two conditions of the if statement. More logical links can be found on page 170 or in the further course of this introduction. The "ANDing" is necessary if, for example, an OFF can also be sent at the group address 1/0/0 to switch other actuators. In this case, the event-function is also ON (1b01), as this indicates only one event (see Figure 8).

How it works event(): You recognize that responds to both flanks.

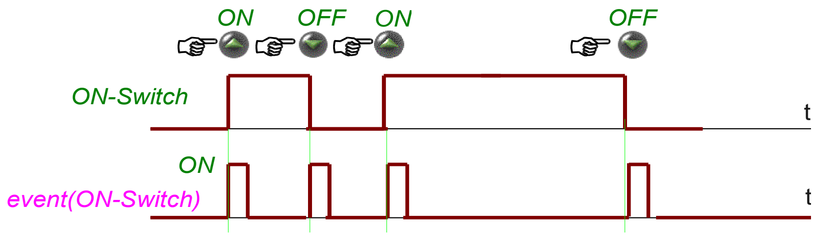


Figure 8: events on the KNX Bus

Some background knowledge

One characteristic of the event function is that this function may not be placed at if statements with else branch. For the following reason: The query condition of the if statement

```
if event("ON-Switch-1/0/0") and  ("ON-Switch-1/0/0"==ON) then \\
        write("Lamp-1/1/1",ON) else write("Lamp-1/1/1",OFF) endif
```

would to go ON on the arrival of an ON-message of the switch at 1/0/0 for one processing cycle Therefore, the if-statement (then-branch) would be executed and the lamp would be switched on. In the next cycle the event-function goes back to OFF (see Figure 8) and now the else branch is executed. The lamp immediately goes off again. To simplify these circumstances for the user, the compiler issues an error.

A hallway light control

Motion detectors and switches

Example 3: An additional motion detector

The structure is now being extended to include the motion detector with the logical address "MotionDetector-1/2/0". The dimmer will either be connected through a switch or a motion detector.

The dimmer is parametrized in such a way that it is selectively turned on and off by a switch object "Dimmer-1/1/1" (binary, i.e. data type b01). The brightness of the dimmer is transferred with the group address "DimmerValue-1/1/2. Figure 9 shows the basic structure.

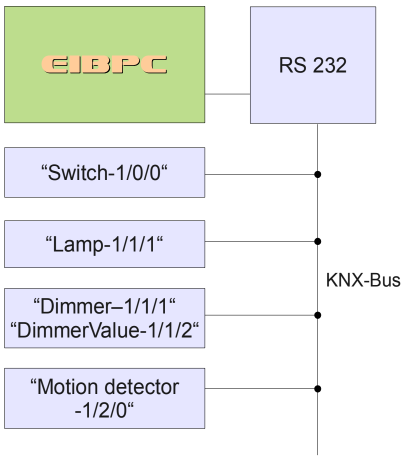


Figure 9: Block Diagram 3

Here our task

If the switch is pressed "ON", the lamp should turn on and the dimmer to go to 80%. If it goes to OFF, the lights will go out. If the switch is active, the motion is to be disabled.
If the motion detector sends an ON-telegram, the dimmer is to go to 50% of its luminosity.

Variables

Naming:
Letter + any combination of letters and numbers.
No umlauts and special characters!

Before we go further on with the implementation, we show an important element of programming: variables. Variables are placeholders for numbers, values, and telegrams. In the Enertex® EibStudio you can define a variable in several manners (see page 158). Again, one definition per line has to be used.

1. Variable=Value
2. Variable=Group address
3. Variable=function()

The data type of the variable is not added at the end as with group addresses, as it is clearly known either from the notation of the number or from the assignment to a group address. When creating the program, the Enertex® EibStudio checks whether the assignment to a certain type is consistent within the program. If you make a mistake, you will receive an error message from the Enertex® EibStudio

Thus

```
Dimmer=80%
MotionDetector="MotionDetector-1/2/0"
```

are valid definitions. The first defines the variable *Dimmer* to 80% (=204u08). The second assigns the variable *MotionDetector* to the **contents** of the telegram which is sent to the group address "MotionDetector-1/2/0.

From the Beginning to the end and from the beginning again

Like already mentioned, the Enertex® EibPC evaluates the user program continuously. Clearly spoken, when the Enertex® EibPC has reached the last instruction at the end of the application program, it starts again with the first instruction. The code is only processed, if something has changed. The states of the incoming telegrams are saved and only when the telegrams or the variables has been changed, the processing is evaluated. We named this process validation scheme.

Validation

The important topic in this example is to understand, that the statement "Set Dimmer to 80%" is only evaluated during the first run through the user program, because the value 80% will not change. On the other hand *"Motiondetector"* depends on the group address “MotionDetector-1/2/0“ and on the telegrams with this group address at the KNX™ bus.

A smalll excursus:

In the following user program the variable DimmVar is set to 80% at the first run of the program code. When the motion detector sends an ON (1b01) to the group address 1/2/0, the if-statement is fulfilled and DimmVar is set to 30%. But at the next run of the program code DimmVar is not set to 80%, because the value 80% has ***not*** *been changed. So, you can overwrite a variable or it content, respectively, with the help of the if-statement.*

Setting variables in the if statement

```
DimmVar=80%
Detector="Motiondetector-1/2/0"
if Detector  then DimmVar=30% endif
```

In the following user program the variable is set to the value of the group address 5/8/1 at the first run. When the motion detector sends ON (1b01) to the group address 1/2/0, then DimmVar is set to 30%. When the Enertex® EibPC receives the next telegram (and the value in the telegram is different from the current value), DimmVar is set to the new value containing in the telegram.

Setting variables in the if statement and with assignment of group addresses

```
DimmVar="Dimmer-5/8/1"
Detector="Motiondetector-1/2/0"
if Detector  then DimmVar=30% endif
```

You can tread the assignment in the if statement as a assignment in the 2nd level. The main level is the definition of the variable (e.g. DimmVar=80%) without leading if-statement. In the 1st (main) level each variable can only be set one times, otherwise the compiler throws an error. This means, each variable must not be located on the left hand side of a equal sign more than one times. Therefore, the next program do not run correctly:

Exactly one assignment is needed in the main level: Definition of the variable

```
//Attention invalid Code
DimmVar="Dimmer-5/8/1"
DimmVar=30%
```

We call the assignment in the main level definition of the variable. Before you can use a variable in your user program, the variable ***must be defined in the main level****. Therefore, in the next user program the variable LogicB is defined with the value OFF in the main level.*

By means of setting the variables in the if statement, you have the possibility to implement arbitrary links. Similarly, this is also possible at the definition of the variable:

```
// Variant A
LogicA="Motiondetector-1/2/0" and "Dimmer-5/8/1">50%
// Variant B (equal to A).
LogicB=OFF
if "Motiondetector-1/2/0" and "Dimmer-5/8/1">50% then LogicB=ON else LogicB=OFF endif
```

Both variables LogicA and LogicB are identical - several ways lead to Rome...

Now, our task can be written with the help of variables as follows:

Switch on: Initialization of the program

systemstart() can be used arbitrarily often

or function

and function: The motion detector will turn on only the dimmer, when switch == OFF

```
[ETS-ESF]
// The exported ESF-Data from the ets
EibPC/Haus.esf
[EibPC]

// System start
if (systemstart()) then                                  \\
                         read("Switch-1/0/0") ;          \\
                         write("Lamp-1/1/1",AUS);        \\
                         write("DimmerValue-1/1/2"u08,0%)   \\
                     endif
if systemstart() then MotionDetector=OFF endif

// Variables
Switch="Switch-1/0/0"
MotionDetector="MotionDetector-1/2/0“
Dimmer=80%

// The switch
if (Switch==ON) or (MotionDetector==ON) then    \\
                                     write("Lamp-1/1/1",EIN);        \\
                                     write("Dimmer-1/1/1",EIN)       \\
                                     endif

if (Switch==ON) then write(“DimmerValue-1/1/2“u08,80%) endif

// Motion detector
if (MotionDetector==ON) and (Switch==OFF)  then \\
                                      write(“DimmerValue-1/1/2“u08,50%) \\
                                      endif

if (Switch==OFF) or (MotionDetector==OFF) then     \\
                                     write("Dimmer-1/1/1",OFF); \\
                                     write("Lamp-1/1/1",OFF)   \\
                                     endif
```

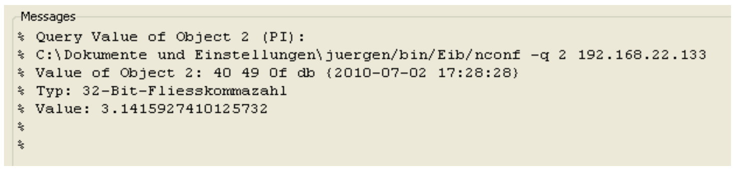
In this example, the or-function is used. It allows "ORing" logically different actuators, i.e. if the variable *Switch* is ON **or** variable *MotionSensor* is ON or both are ON, the if statement is true, and thus the then branch is executed. The and function combines the the two conditions with AND. You can compare (larger, smaller) and invert expressions. More on logical links can be found on page 170.

Troubleshooting made easy...

When programming, even the best professional makes errors. To “debug“ your program, i.e. to look for errors, you can query the value currently assigned to a variable. Figure 10 shows the debugger button (alternatively, in the menu “Query of EibPC-value of an EIB-object“). When pressed, a window with variables appears, which you can now click on. For example, Figure 11 shows the query of the constant π. Please note the message window in the Enertex® EibStudio below.

Figure 10: The debugger button

The integrated debugger



```
% Query Value of Object 2 (PI):
% C:\Dokumente und Einstellungen\juergen/bin/Eib/nconf -q 2 192.168.22.133
% Value of Object 2: 40 49 0f db (2010-07-02 17:28:28)
% Typ: 32-Bit-Fliesskommazahl
% Value: 3.1415927410125732
%
%
```

Figure 11: Value query

A hallway with time control

A time switch

Example 4: A motion detector, switches and brightness depending on the time of day

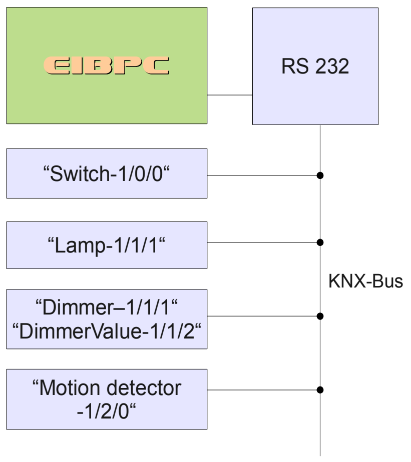


Figure 12: Block Diagram 3

We use the same example as in Figure 9 (Fig. 12). The dimmer will change its intensity time-controlled.

Here is our own task

Commissioning

If the switch is pressed "ON", the lamp should turn on and the dimmer should go to 100%. If it goes to OFF, the lights will go out. If the switch is active, the motion is to be disabled.

If the motion detector sends an ON telegram, the dimmer should go to

– 50% of its luminosity, if it is after 20:00 Clock
– 30% of its luminosity, if it is after 23:00 Clock
– 10% of its luminosity, if it is after 3:00 Clock
– 100% of its luminosity, if it is after 7:30 Clock

Switch

With our previous considerations in mind, you can do this almost without help. But you should know how to program a timeswitch. There is the time function htime:

```
htime(hour, minute, second)
```

The function returns a logical *ON* (1b01), when the day is **precisely** achieved. In the other case it is *OFF* (0b01). Using an if statement, it can now be changed at any time.

Thus our application program is:

Time control

Motion detector

```
[ETS-ESF]
// The exported ESF-Data from the ets
EibPC/Haus.esf
[EibPC]
if (systemstart()) then                                          \\
                              MotionDetector=AUS;       \\
                              read("Switch-1/0/0") ;            \\
                              write("Lamp-1/1/1",AUS);       \\
                              write("Dimmer-1/1/2"u08,0%)   \\
                       endif
// Variables
Switch="Switch-1/0/0"
MotionDetector=“MotionDetector-1/2/0“
Dimmer=100%
// The switch
if (Switch==ON)  then                                 \\
               write("Lamp-1/1/1",EIN);             \\
               write("Dimmer-1/1/1",EIN);            \\
               write("DimmerValue-1/1/2",100%) \\
               endif
if (Switch==OFF) then                               \\
               write("Lamp-1/1/1",AUS);       \\
               write("Dimmer-1/1/2"u08,0%)   \\
               endif
// Motion detector
if (htime(20,00,00)) and (Switch==OFF) then Dimmer=50% endif
if (htime(23,00,00)) and (Switch==OFF) then Dimmer=30% endif
if (htime(03,00,00)) and (Switch==OFF) then Dimmer=10% endif
if (htime(07,30,00)) and (Switch==OFF) then Dimmer=100% endif

if  (MotionDetector==EIN) and (Switch==OFF) then write("Dimmer-1/1/1",EIN); write("DimmerValue-1/1/2",Dimmer) endif
if  (MotionDetector==AUS) and (Switch==OFF) then write("Dimmer-1/1/1",AUS) endif
```

By the time function htime, time controls can be quickly and simply implemented. More time functions can be found on page 187 for minutes-, seconds-, day-, month- and year-timers.

You can use almost unlimited (65,000) switching time clocks by applying the function repeatedly. These are not related or even limited mutually. Read more about this on page 189 and the following.

Not just a time switch

For example by

```
if (stime(43)) then
```

you can execute an instruction every minute, when 43 seconds are exactly achieved.

Important special functions for time outputs are also

- delay - function,
- after - function and
- cycle - function.

You can find more in the next step of this introduction and on page 196.

Time switch and comparator time switches

When you start the Enertex® EibPC, the variable Dimmer is set to 100%. If in this case the motion detector responds, the dimming actuator will adopt this value. But, it could be, that the commissioning (transfer of the user program) takes place at e.g. 20:00:01. Since htime() performs an exact time comparison and thus the corresponding if-statement is only executed when the specified time is reached, the Dimmer is not changed. By using of the debugger (Figure 11), request the value of the variable Dimmer. With our just made assumptions, this variable would still be at 100%.

By making corresponding assumptions of the statement if systemstart()..., you could solve this problem. Alternatively, use the comparator time switches. Read more about this on page 189 and the following.

Setting the system clock

If you want to use time switches, the Enertex® EibPC has to be acquainted with the time. This is most conveniently achieved if the Enertex® EibPC can establish an Internet connection via your router or if a KNX™ device with receiver or similar device can periodically synchronize the system time. Fore more information, refer to page 149.

A release button and the validation scheme

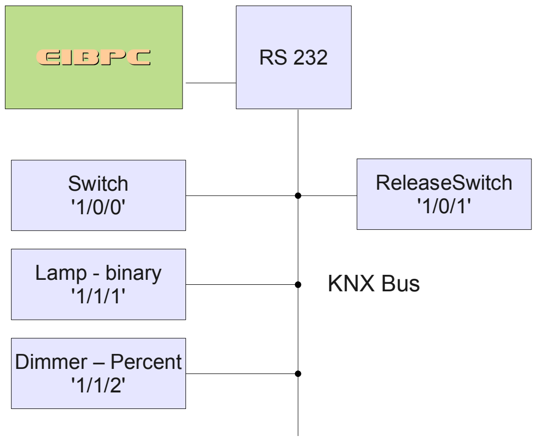


Figure 13: Block diagram of switch and release button

We have a structure presented as shown in Figure 13.

If the switch is pressed “ON“, the lamp should turn on and the dimmer has to go to 80%. If it goes to OFF, the lights will go out. If the release switch (group address 1/0/1) is set to OFF, the switch ON will be ignored.

A simple task:

Properly resolved - how it's done

```
[ETS-ESF]
EibPC/Haus.esf
[EibPC]
if (systemstart()) then read("Switch-1/0/0"); read("ReleaseSwitch-1/0/1") endif
if ("Switch-1/0/0"==EIN) and ("ReleaseSwitch-1/0/1"==ON) then       \\
                                write("Lamp-1/1/1",ON);          \\
                                write("Dimmer-1/1/2"u08,80%)  \\
                            else
                                write("Lamp-1/1/1",OFF);         \\
                                write("Dimmer-1/1/2"u08,0%)   \\
                            endif
```

Nesting of if statements

if statements can be nested. You might be of the opinion to proceed as follows:

While the syntax is valid (no error by the compiler) ...

... but does not work as intended

```
[ETS-ESF]
/EibPc/Haus.esf
[EibPC]
if (systemstart()) then read("Switch-1/0/0"); read("ReleaseSwitch-1/0/1") endif
if  ("ReleaseSwitch-1/0/1"==ON)  then                \\
           if ("Switch-1/0/0"==ON) then                 \\
                      write("Lamp-1/1/1",ON);           \\
                      write("Dimmer-1/1/2"u08,80%)  \\
                  else                                   \\
                      write("Lamp-1/1/1",OFF);          \\
                      write("Dimmer-1/1/2"u08,0%)   \\
                  endif                                  \\
         endif
```

The Enertex® EibPC is a state memory. All telegrams are internally stored and thus a process image is created.

This program is not inherently flawed, but it will work differently than you might have thought. The Enertex® EibPC processes only expressions which change. Therefore, it remembers the last state, which was sent via a group address. If an ON was sent at the group address "ReleaseSwitch-1/0/1" OFF recently exhibiting the value OFF, the inner if statement is evaluated exactly once. Thus, in this case, the lamp and the dimmer are switched on or off, whichever value was sent via the group address "Switch-1/0/0". If now - "ReleaseSwitch-1/0/1" is still ON - the group address "Switch-1/0/0" changes, no further telegram is triggered, because the condition of the outer if-statement does not change (also see comments on page 43).

The mapping of states of received telegrams is called validation scheme. It makes the operation and programming easy and intuitive, as long as the nesting of if statements is avoided. As shown in the example, this prevention is always simply possible.

We advise beginners to avoid nested if-statements.

A staircase lighting

Example 5: A staircase lighting

Figure 14 is the installation, for which a stairwell light (3 min switching time) is to be realized. The switch always returns one ON pulse (thus, it will no longer be turned off) only.

On a display element, you also want to view how often the button was pressed. You have defined the group address 1/2/2 within the ETS for the display element as a string with a maximum of 14 characters.

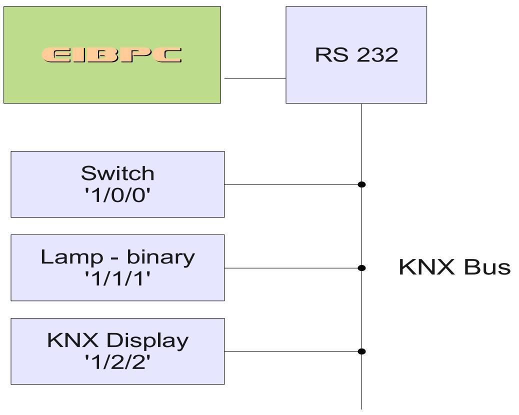


Figure 14: Staircase lighting

At system start, the light shall go out. The switch alternately provides ON and OFF telegrams. After pressing the switch (“switch position“ should be arbitrary), the light shall turn on and automatically turn off again after three minutes. The sum of the switching processes already made will be shown on KNX display element.

Two variants:
Re-trigger or not?

Variation 1: At re-pressing the switch during the 3 minutes turn-on time, the timer switch shall not restart.

Variation 2: At re-pressing the switch during the 3 minutes turn-on time, the timer switch shall restart.

First, we show the realization of ***Variant 1***. This time without the use of imported group addresses.

Variant 1: No Re-triggering

In this realization, the timer will not restart, if it is already running (no "re-triggering"). First, the program:

Definition of a KNX^{TM} string
$ Text$c14

New
event(Group address)
after(Term, Time in ms)
convert(Source, Target)

```
[EibPC]
if systemstart() then write('1/1/1'b01,OFF) endif
SwitchingOperation=OFF
Counter=0u32
EmptyString=$ Nothing yet! $c14
if event('1/0/0'b01) and (SwitchingOperation==OFF) then  \\
                                SwitchingOperation=ON;  \\
                                write('1/1/1'b01,ON);  \\
                                Counter=Counter+1u32\\
                              endif
if (after( SwitchingOperation==ON, 180000u64)) then                    \\
                                        write('1/1/1'b01,AUS);                  \\
                                        SwitchingOperation=OFF;                 \\
                                        write('1/2/2'c14,convert(Counter,EmptyString)) \\
                               endif
```

Reminder:

event(GroupAddress)

indicates, when a message is received from the bus by the given group address. It does not check whether the message has changed, what content or what type it has. Once a message arrives, it goes ON for a operational cycle of Enertex® EibPC. Thus, the condition of the if statement is true and the instruction is executed.

All possibilities to communicate with bus summarized

You have already got to know all special functions related to the communication via the KNX. Only by using these functions, you can take control the bus. A short recapitulation:

- *write(GroupAddress, value) writes a value to the group address on bus*
- *read(GroupAddress) writes a read request to the group address on bus*
- *event(GroupAddress) checks, whether a message was sent to the group address on bus*
- *Variable=group address assigns a variable to the value, which is sent in a telegram to the specified group address.*

Data type u64
64-bit, unsigned integer

We also use the after function. The delay function after, also called precision timer, operates with a resolution of 1 ms:

after(Variable {Type b01}, Time in ms {Type u64})

The precision timer expects a variable or an expression of binary type (data type b01, data types, see also page 35 and 151 and the following) as the first argument. The function after delays the input (ON and OFF), for the time specified in the second argument. The return value is also ON or OFF. This can be quite clearly represented graphically by Figure 15 . The second argument is of type integer, unsigned 64-bit. We therefore need the data type u64. This value specifies the delay time in ms.

The delay function

You can set delays for decades. If the function after is started once, it processes only one impulse at its input. The result is the dead time being equal to the delay time, see Figure 15. In the example we use a delay of

180.000ms = 3*60*1000ms = 3*60s = 3min.

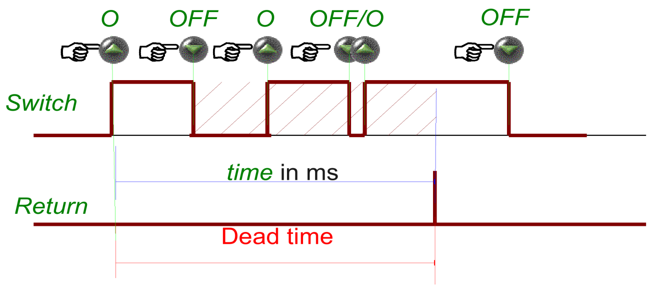


Figure 15: After-Function

The function after can not be triggered again, as shown in Figure 32 by the “dead time“. In our case (variant 1) this is desired. That is, if after has been stored once, any further changes of the input are ignored (see shading in this figure).

Arguments of functions are arbitrary expressions, constants or variables

However,in our example we write:

```
if (after(SwitchingOperation==ON, 180000u64)) then
```

In this case, the argument is not a variable, but a comparison. As already learned on page 35, the return value of a comparison is of a binary data type.

So, basically, instead of a variable also any other function can be used ('nesting'), it is only important that this function or the expression of the appropriate data type has a return value. In the above example, SwitchingOperation is already a binary data type, thus the comparison is not necessary. However, you can now use after() in this sense to delay thresholds (e.g. if after(Light>200u32,TimeValue)). The delay time must not be constant. For example, it could be entered via a telegram or a variable.

As often as you want

At this point, yet another clue: The number of high precision timer used in the program is not limited. You can always use the function multiple times. For each called after function a new object is created. Therefore, the functions do not limit each other in use. This holds always and for all other functions when programming with the Enertex®EibStudio.

Cycle controlled

An important special function for automatically timed outputs is the cycle function. Using these functions, tasks can be executed at certain time cycles. You can find more on page 196.

Still missing is the conversion function:

convert(Variable1 {Type Source}, Variable {Type Target})

This function has that value as return value, which is stored in Variable1, but is converted into the type of Variable2. The value of Variable2 is ignored, it is only used to determine the new data types. For example, a binary value can be converted into an 8-bit unsigned number by convert:

```
a='1/2/2'b01
b=convert(a,1u08)
```

In this example, we increase the variable numerator within the then branch at each switching cycle of the lamp:

Data type u32

32Bit, unsigned integer

```
Counter=Counter+1u32
```

Here, Counter is of type integer, 32-bit unsigned. It allows, to count up to 2^{32}-1 = 4,294,967,295. If the lights have actually been activated so often, this variable would get an overflow at re-pressing the switch and would begin to count at 0.

KNX™ character string

Character strings of up to 14 characters will be transmitted via group addresses of the type c14. The definition of a c14-string is framed by two $-signs.

$ Text $c14 is equivalent to the character string „ Text “ (with spaces)

This data type can be directly printed on KNX™ display elements. Besides, at the programming of the Enertex® EibPC, there are character strings with a length of up to 1400 characters. Later in this introduction, we will need these to handle LAN telegrams.

With

```
EmptyString=$ Still nothing! $c14
```

a character string variable EmptyString with content "Still nothing! "(including spaces) is defined.

Finally, within the statement

```
write('1/2/2'c14,convert(Counter,EmptyString))
```

Counter is shown on a KNX display element, converted into a character string.

EmptyString for the convert function is not evaluated. It merely serves to define the data type for the conversion. Therefore, at this positions, write('1/2/2'c14,convert(Counter,$ $)) is also a possible solution.

The following special feature in the program may still seem incomprehensible to you:

As in any other programming language: Counter=Counter+1

```
if event('1/0/0'b01) and (SwitchingOperation==OFF) then  \\
                                SwitchingOperation=ON;  \\
                                write('1/1/1'b01,ON);  \\
                                Counter=Counter+1u32\\
                              endif
```

Why is the event function linked to the variable SwitchingOperation in the if-statement? If this were not the case, a telegram would be written on bus at each event('1/0/0'b01) and the Counter is increased. In our task, we want to determine the number of cycles with activated lamp, but not the number of switching operations.

Therefore, we have to 'interlock' the event function in the if statement.

Supplement: Astro function
Day or night?

Another useful feature is the sun function: It returns the value ON (1b01) when the sun rises or OFF (0b01) if it is set (see also pages 184 and 150).

Variant 2: Press again

Turning to ***Variant 2***. For the light circuit, the timer is to be restarted again, it the light switch is pressed again.

Therefore we need:

delay(Signal {Type b01}, Time in ms {Type u64})

The delay function requires a variable or an expression of binary type (type b01, for data types, see also page 35 and 151 and the following) as the first argument. The function delay lags that ON impulse of the input for the time specified in the second argument. The type of the return value is ON or OFF. This can be quite clearly represented graphically by Figure 16. The second argument is of type integer, 64-bit unsigned, thus we need the data type u64.

An ONE-edge starts the delay timer.

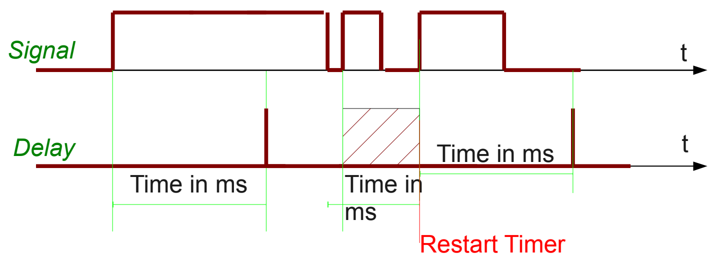


Figure 16: delay-function

However, our program has to be changed only at one point, and we have only to replace after with delay.

Variant 2:

Replace after by delay

```
[EibPC]
if systemstart() then write('1/1/1'b01,OFF) endif
SwitchingOperation=OFF
Counter=0u32
EmptyString=$ Nothing yet! $c14
if event('1/0/0'b01) and (SwitchingOperation==OFF) then  \\
                                    SwitchingOperation=ON;  \\
                                    write('1/1/1'b01,ON);  \\
                                    Counter=Counter+1u32\\
                                  endif
if (delay(SwitchingOperation==ON, 180000u64)) then                       \\
                                             write('1/1/1'b01,AUS);                       \\
                                             SwitchingOperation=OFF;                       \\
                                             write('1/2/2'c14,convert(Counter,EmptyString)) \\
                                   endif
```

At this point, please note again: The number of delay timers used in the program is not limited. This function can be used repeatedly. For each called delay function, a new object is created. Therefore, the delay-functions do not limit each other in use. This is always true and also for all other functions when programming with the Enertex® EibStudio.

Date control

Suppose you want to realize a birthday calendar by a KNX™ display element, which expects a KNX™ character string to the group address 1/2/3:

> If there is a birthday, this shall be displayed on the display element.
>
> In all other cases, the uptime of the Enertex® EibPC shall be shown.

Display birthdays

To solve this problem, the macro Online() from the library Enertex.lib is used. We need to integrate the library into our program. Click on Program / Macro-libraries and there on the menu item Add. Select the Library “Enertex.lib“ and click on Add.

We use the macro “Online“

In the second step select the macro Online() in Program / Macro. A parameter dialog is opened, which you simply confirm by clicking OK.

The macro Online() defines the variable “DisplayOnline“, a KNX™ character string (data type c14), which writes the time of last start of the user program into a string. The format is ddd.hh: mm (ddd: day from 0 to 999, hh hours, mm minutes).

Here is the program:

Insert macros

```
[Macros]
//Macros
Online()

[MacroLibs]
//Macro-Libraries
Enertex.lib

[EibPC]
Birthday=OFF
DisplayBirth=$            $c14
// Here, adding arbitrary Birthdays
if month(10,01) then DisplayBirth=$    Alex      $c14;Birthday=ON endif
if month(15,01) then DisplayBirth=$    Martin    $c14;Birthday=ON endif
if month(21,02) then DisplayBirth=$    Jürgen    $c14;Birthday=ON endif
// Reset short of switching
if htime(23,59,59) then Birthday=OFF endif

// Here, is the output of the time
if change(DisplayOnline) and !Birthday then write("HallText-0/2/1"c14,DisplayOnline) endif
if systemstart() then write("HallText-0/2/1"c14,DisplayOnline) endif

// Birthday display will blink
DisplayBlink=0
if Birthday and cycle(0,3)      then write("HallText-0/2/1"c14,DisplayBirth);DisplayBlink=1 endif
if after(DisplayBlink==1,1700u64) then write("HallText-0/2/1"c14,$Birthday$c14);DisplayBlink=2 endif
if after(DisplayBlink==2,700u64)  then write("HallText-0/2/1"c14,$    !!+++!!    $c14);DisplayBlink=0 endif
```

We have used the function month:

month(*Day*,*Month*)

The function returns a logical ONE (1b01), if the date of any year is reached. In the other case it is OFF (0b01). The switching point is exactly 00:00:00.

Every day, at 23:59:59 (statement with htime) the variable *Birthday* is switched OFF again.

You can also define periods with month function: The following variable Summer is ON during the period from 1 May to 30 September, otherwise OFF.

```
Summer=month(01,05) and !month(30,09)
```

Summer by date definition

Therefore, you are able to realize the summer as a query condition:

```
if Summer then ....
```

Shading

We use the example on page 28 again.

Now, we want to change the automation in such a way that the shading is performed via the release of the switch object, but only if it is already brighter than 5000 Luz outside at 9:00 in the morning. In the evening, all roller blinds should move to the night position (roller blinds down).

The site may be Nuremberg (49°27', 11° 5'). You can modify this data set by using the menu OPTIONS – COORDINATES FOR THE SUN POSITION CALCULATION SETTING (see page 184).

And here is the resulting program. You should already know all of the programming elements. This example illustrates only the “mixing“ at the usage of macros and programming.

Variables can also be arguments of the macros.

```
[Location]
// Longitude and Latitude of the installation location
11.083333333333334
49.45

[Macros]
//Macros
ShadingRolloEastTime(ShadeRelease,"Window-East-2/5/1","Window-East-2/5/1",5000)
ShadingRolloSouthTime(ShadePassing,"WindowSouth-2/5/2","WindowSouthStop-2/5/5",4500)
ShadingRolloWestTime(ShadePassing,"WindowWest-2/5/3",""WindowWestStop-2/5/6",4000)

[MacroLibs]
//Macro-Libraries
EnertexBeschattung.lib

[ETS-ESF]
// From the ets exported ESF-Data
Shading.esf

[EibPC]
ShadePassing=OFF
if ("PassingShading-2/5/0"==ON) and ("Light-2/1/1">5000f16) and chtime(9,00,00) then \\
		ShadePassing=ON endif

// At sunset...
if sun()==OFF then \\
write("Window-East-2/5/1",EIN); write("WindowSouth-2/5/2",ON); write("WindowWest-2/5/3",ON) endif
```

Whether the arguments of macros may be variables or group of addresses, or both, depends on the implementation of the macro. At the specified macro libraries, this is always stated within the macro-wizard.

Dew-point calculation

The following example is less about automation tasks. At this point, we want only to show how flexible calculating with variables is.

A calculator

In a winter garden, the dew point temperature (TD) dependent on the humidity (r) and room temperature (T) shall be calculated.

The necessary calculations for this purpose:

```
Denotations:
r = relative humidity
T = Temperature in °C
TD = dew-point temperature in °C
DD = vapor pressure in hPa
SDD = saturation vapor pressure in hPa
Parameters:
a = 7.5, b = 237.3 for Temperature >= 0
Formulas:
TD(r,T) = b*v / (a-v) with
v(r,T) = log10(DD(r,T)/ 6.1078
DD(r,T) = r/100 * SDD(T)
SDD(T) = 6.1078 * 10^((a*T) / (b+T))
```

The implementation is then:

Data type f32:
32-bit floating point number

log(a,b): $Log_a(b)$
pow(x,y): x^y

```
a=7.5f32
b= 237.3f32
r="SensorHumidity-1/0/2"
T="SensorTemperature-1/0/3"
SDDT=6.1078f32 * pow(10f32,(a*T)/(b+T))
DDrT= r/100f32*SDDT
v=log(10f32,(DDrT/6.1078f32))
TD=b*v/(a-v)
```

You can use your Enertex® EibPC like a calculator - other mathematical functions can be found on page 201.

Monitoring of bus services (Monitor)

The EibPC stores up to 500,000 messages in a ring buffer. These can be collected, monitored and stored in a file, e.g. to analyze these data by Microsoft® Excel or OpenOffice Calc.

When recording, the PC has **not** to be connected with the Enertex® EibPC.

Caches 500,000 messages

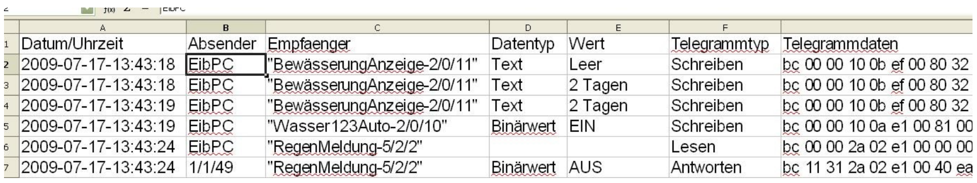
To collect the messages accumulated in the Enertex® EibPC, click in menu EibPC on Retrieve EIB-Messages. In the status bar bottom left you will see the progress in the collection of messages. The speed of the transmission of messages depends primarily on the performance of your PC.

Storing messages

Retrieve messages

If all messages are picked up or the transmission is interrupted, a dialog appears. By means of this dialog, you can save these messages into a CSV file. This CSV file is a text file that stores the data tabularly. The individual data are separated by commas. Then, you can import the data into Microsoft® Excel or OpenOffice Calc or into a spreadsheet program of your choice.

CSV: Comma-separated text file with data of your bus communication



	A	B	C	D	E	F	
1	Datum/Uhrzeit	Absender	Empfaenger	Datentyp	Wert	Telegrammtyp	Telegrammdaten
2	2009-07-17-13:43:18	EibPC	"BewässerungAnzeige-2/0/11"	Text	Leer	Schreiben	bc 00 00 10 0b ef 00 80 32
3	2009-07-17-13:43:18	EibPC	"BewässerungAnzeige-2/0/11"	Text	2 Tagen	Schreiben	bc 00 00 10 0b ef 00 80 32
4	2009-07-17-13:43:19	EibPC	"BewässerungAnzeige-2/0/11"	Text	2 Tagen	Schreiben	bc 00 00 10 0b ef 00 80 32
5	2009-07-17-13:43:19	EibPC	"Wasser123Auto-2/0/10"	Binärwert	EIN	Schreiben	bc 00 00 10 0a e1 00 81 00
6	2009-07-17-13:43:24	EibPC	"RegenMeldung-5/2/2"			Lesen	bc 00 00 2a 02 e1 00 00 00
7	2009-07-17-13:43:24	1/1/49	"RegenMeldung-5/2/2"	Binärwert	AUS	Antworten	bc 11 31 2a 02 e1 00 40 ea

Figure 17: Import of recording data in OpenOffice

The "Logging" of bus transfers facilitates the diagnosis of faults and far-reaching improvements

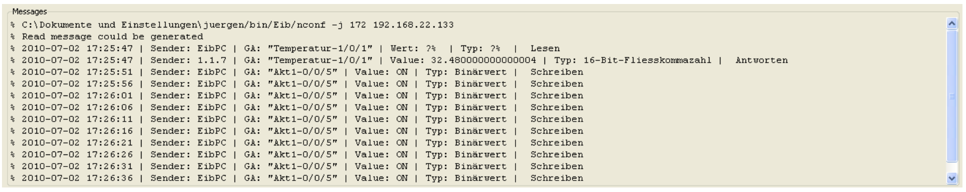
In Microsoft® Excel, select the submenu Import within the dialog Data. In the following, best choose the format described as "text" and the text label as {no}.

Save to File

Additionally, the current bus communication can be made visible. Activate the option "Retrieve EIB-Messages / Connection Status" or click on the button . If the Enertex® EibPC is connected with the Enertex® EibStudio, the button changes to .

Online telegrams

Now, in the window "Messages" all the messages are provided, which were just written on bus. Figure 18 shows a typical excerpt from the message window, which is self-explanatory.



```
% C:\Dokumente und Einstellungen\juergen/bin/Eib/nconf -j 172 192.168.22.133
% Read message could be generated
% 2010-07-02 17:25:47 | Sender: EibPC | GA: "Temperatur-1/0/1" | Wert: ?%  | Typ: ?%  |  Lesen
% 2010-07-02 17:25:47 | Sender: 1.1.7 | GA: "Temperatur-1/0/1" | Value: 32.480000000000004 | Typ: 16-Bit-Fliesskommazahl |  Antworten
% 2010-07-02 17:25:51 | Sender: EibPC | GA: "Akt1-0/0/5" | Value: ON | Typ: Binärwert |  Schreiben
% 2010-07-02 17:25:56 | Sender: EibPC | GA: "Akt1-0/0/5" | Value: ON | Typ: Binärwert |  Schreiben
% 2010-07-02 17:26:01 | Sender: EibPC | GA: "Akt1-0/0/5" | Value: ON | Typ: Binärwert |  Schreiben
% 2010-07-02 17:26:06 | Sender: EibPC | GA: "Akt1-0/0/5" | Value: ON | Typ: Binärwert |  Schreiben
% 2010-07-02 17:26:11 | Sender: EibPC | GA: "Akt1-0/0/5" | Value: ON | Typ: Binärwert |  Schreiben
% 2010-07-02 17:26:16 | Sender: EibPC | GA: "Akt1-0/0/5" | Value: ON | Typ: Binärwert |  Schreiben
% 2010-07-02 17:26:21 | Sender: EibPC | GA: "Akt1-0/0/5" | Value: ON | Typ: Binärwert |  Schreiben
% 2010-07-02 17:26:26 | Sender: EibPC | GA: "Akt1-0/0/5" | Value: ON | Typ: Binärwert |  Schreiben
% 2010-07-02 17:26:31 | Sender: EibPC | GA: "Akt1-0/0/5" | Value: ON | Typ: Binärwert |  Schreiben
% 2010-07-02 17:26:36 | Sender: EibPC | GA: "Akt1-0/0/5" | Value: ON | Typ: Binärwert |  Schreiben
```

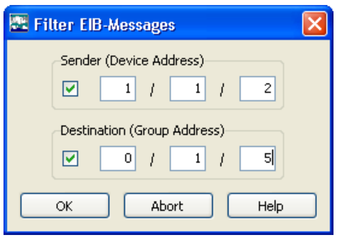
Figure 18: Recalling EIB-Messages

Since the RS232-Interface does not supply a feedback over the connecting status, the EibPC sends all telegrams, even if no bus connection is developed. Thus each code can be tested also without bus connection.

Against it a defined handshake has the IP interface. The EibPC waits then for the feedback, before it sends telegrams away.

Filter telegrams

You can set up filters so that only messages of certain group addresses or specified device addresses (senders) are displayed or stored. You must first set up the filter (see Figure 19), which you can set up within the menu item EibPC – Configure Filter.


Filter EIB-Messages

Sender (Device Address): ✓ 1 / 1 / 2

Destination (Group Address): ✓ 0 / 1 / 5

Figure 19: EIB messages filter

With the settings in Figure 19, only frames are displayed which arrive from the sender with the physical address 1/1/2 and are sent to the target group address 0/1/5.

If you like to watch all messages of a specific sender, regardless of the destination, you must disable the option target (see Figure 20).

Wildcards „?“ : Any digit, ““ Any number*

Figure 20: Filtering of all EIB messages of a sender

If the filter is activated (EibPC – Filter current EIBMessages), the status display changes as indicated Figure 21. The filter acts both to messages in the message window, as well as when saving messages to a CSV file.

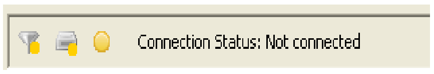


Figure 21: Filter display in the bottom right of the status display, right picture: Filter enabled

You have the option of using wildcards: If the last entry for the target is written “1/1/2**?**3“, all group addresses 1/1/2**0**3, 1/1/2**1**3 ... 1/1/2**9**3 are filtered out. You can use any number of wildcards.

To see whether the filter is active

You can configure the Enertex® EibStudio in such a manner that it regularly collects messages “on its own“. For this, the “Autolog“ dialog is available. Here, too, an active message filter sorts out messages, see page 140.

Store telegrams cyclically

You can configure Enertex® EibStudio in order to retrieve telegrams periodically and save them at your PC. To do this you have to use the "autolog"-dialog. When a filter is active, it is also applied with the autolog function. See p. 140.

Storing telegrams at a FTP-Server

Alternatively you can configure Enertex® EibStudio in order to retrieve telegrams periodically and save them at a FTP server Further information is given at p. 142.

Scenes

Scene actuator

Example 6: A scene actuator

A scene actuator is to be realized for the installation in Figure 22. The switch is configured in the ets to work as a scene switch: Calling the scene is initiated by simple pressing and saving of a preset scene by a long key press. The ETS program is such that the switch calls a scene with number 1 at address 1/0/0.

A Scene block is desired

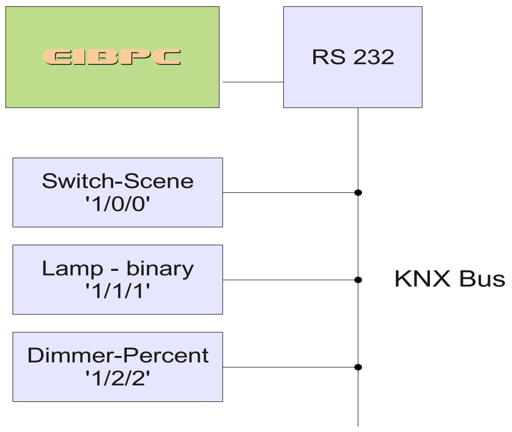


Figure 22: Scenes

Therefore, our task is:

Within the KNX network, the Enertex® EibPC should behave like a scene actuator, which has been parametrized at address 1/0/0.

A scene

To define a scene actuator, use the function

scene(*GroupAddressSceneBlock*, *GroupAddressActuator1*, *GroupAddressActuator2*, , ... *GroupAddressActuatorN*)

You can “connect“ up to 65,000 actuators to this one scene actuator. The number of arguments, i.e. the actuators assigned to a scene actuator and a scene function, respectively, is arbitrary.

The scene actuator can store 64 different scenarios (Values 0 to 63). By the *GroupAddress SceneBlock* *(8-bit unsigned)*, these scenario numbers (0u08 until 63u08) are stored and retrieved by their appropriate actuators. As usual, the actuators and switches have to be programmed in the ets.

By the scene function, this task is easily solved as follows:

The realization

```
[EibPC]
scene('1/0/0'u08, "Lamp-1/1/1“, "'Dimmer-1/2/2")
```

More is has not to be done. Now, in the Enertex® EibPC one scene actuator is created with 64 accessible scenes. Of course, you can address scenes with manual group addresses.

Power blackout?

Scene reconfiguration

Stored scenes will be preserved even at a temporary separation of the Enertex® EibPC from the power supply or at the modification of the application software (re- and changed import). If you want to delete the scenes completely from the permanent memory of the Enertex® EibPC, select the corresponding item in the menu EibPC. If you assign changed group addresses in the application program to the scene actuators stored in the Enertex® EibPC, you should do so in any case.

You can use the scene function almost arbitrary times (max. 65,000), e.g. for two actuators:

```
[EibPC]
scene('1/0/0'u08, "Lamp-1/1/1“, "'Dimmer-1/2/2")
scene('2/0/0'u08, "Lamp-1/1/1“, '1/3/2'u08, '5/8/8'f32, '4/4/56's16)
```

The Enertex® EibPC itself can also address scenes and initiate their storage.
There are two functions

Saving scenes
Calling scenes

storescene(*GroupAddresseSceneBlock*, *SceneNumber{u08}*)
callscene(*GroupAddresseSceneBlock, SceneNumber{u08}*)

The first function assigns a scene block (KNX device or scene function of the Enertex® EibPC) to save the scene with the *SceneNumber*. The second function initiates a retrieval, i.e. the scene is recalled. The scene block is activated via the group address.

Instead of the number you can also use variables, as will be shown in the next example. The group address of the scene actuator may relate to a scene actuator in Enertex® EibPC (i.e. scene function) as well as to an external scene module.

Preconfigure a scene

presetscene can be used to pre-configure a scene (p.217).

Finally, a small “goody“:

Scenes without scene button

Suppose, the switch of your KNX™ network (Figure 22) can only supply binary messages: You have programmed your system so that the OFF telegram already switches off all lights. Now, when you press ON, the scene with number 1 is to be called, i.e. a bit is now only sent to group address 1/0/0.

But you want even more:

If you press ON for a second time within the first second, the scene will be saved. You realize the “double click“ as you are accustomed from your use of the PC and mouse.

No problem:

Short for (see page 153)
Keypress=0u08
Scene=1u08

Evaluate after one second evaluating, whether switched once or twice.

```
[EibPC]
// The Scene actuator
scene('1/2/3'u08, "Lamp-1/1/1", "Dimmer-1/2/2"u08)
// Variable
Keypress=0
Scene =1

// Counter of keypress
if (("Switch-1/0/0") and event("Switch-1/0/0")) then Keypress= Keypress+1 endif

// Reset of counter
if (after(event("Switch-1/0/0"),1000u64) then Keypress=0 endif

// Saving or calling
if after(event('1/0/0'b01),1000u64) and (Keypress ==1) then callscene('1/2/3'u08,Scene) endif
if (after(event('1/0/0'b01),1000u64) and (Keypress ==2) then storescene('1/2/3'u08,Scene) endif
```

But not enough:

In the library Enertex.lib, you will find a ready function block “Doubleclick()“ for the double occupancy of buttons.

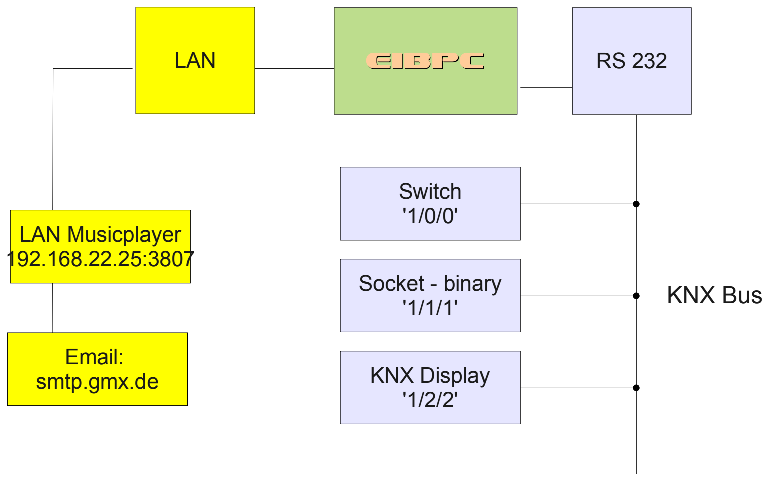
Extended LAN functions (Option NP, only)

In addition to the functions for accessing the KNX™ bus, the Enertex® EibPC can receive, send and evaluate LAN telegrams.

Multimedia control

As usual, we set ourselves a small task to present the approach and the application of LAN functions.

Enertex® EibPC must be configured for name resolution.

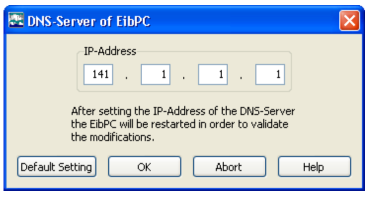


The job: A multimedia application

Figure 23: Multimedia control

- When the switch 1/0/0 is pressed, the Enertex® EibPC shall switch on the power socket controlled by the group address 1/0/0 (which supplies the music player).
- After four seconds, the Enertex® EibPC shall send the string “Start Music“ to port 3807 of the music player via a UDP telegram.
- Then, the music player sends a string “Play Music: Title“ to port 4806, in which “Title“ is the currently playing track. The Enertex® EibPC shall write the title to the KNX display.
- After playing 10 titles, the Enertex® EibPC shall send an e-mail containing the 10 played titles (concatenated, separated by space characters) to EibPC@enertex.de. Subject of the mail: “A Hello from EIBPC“.
- Then, it shall send the string “Stop Music“ via a UDP telegram to port 3807 of the music player and switch off the power socket after five seconds again.

A first problem: The e-mail access needs to be established. For this task, the Enertex® EibPC requires a DNS server, so that it handles the name resolution (see menu EibPC → DNS Server):


DNS-Server of EibPC

IP-Address

141 . 1 . 1 . 1

After setting the IP-Address of the DNS-Server the EibPC will be restarted in order to validate the modifications.

Figure 24: Setting up DNS Server

Second, the e-mail account has to be configured.

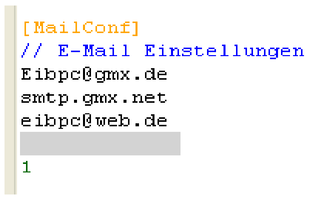
To do this, click Options → Email Settings and insert your settings for the SMTP-server (see also the dialog in Figure 26).

You must have an e-mail account with a provider.

Figure 25: E-mail configuration

Setup via dialog ...

After the “OK“, the entered data will appear in the user program as indicated in Figure 26 and can also be edited directly.



```
[MailConf]
// E-Mail Einstellungen
Eibpc@gmx.de
smtp.gmx.net
eibpc@web.de

1
```

Figure 26: E-Mail Setup

Now, the rest is fairly quickly done - you are already very familiar with the syntax of the Enertex® EibPC. What do you only need to know: What are the functions to the UDP and mail delivery? How do you handle and define larger character strings (more than 14 characters)?

Initially, the program, then the explanations:

... or by hand

```
[MailConf]
// E-mail configuration
Eibpc@gmx.de
smtp.gmx.de
eibpc@web.de
Supermollycoddle
1

[EibPC]
// Init: Saving of Title
Title=$ $
if delay(systemstart(),10000u64) then write ('1/0/0'b01,EIN) endif
if ('1/0/0'b01) then write('1/0/1'b01, EIN); Title=$ $;TitleNr=0 endif
// Transmission of Start Music
if delay('1/0/1'b01,4000u64)  then sendudp(3807u16,192.168.22.25,$ Start Music$) endif
// Strings for processing
StringPlayer= $ $
KNXTitle=$ $c14
TitleNr=0
Port=0u16;IP=0u32
Pattern=$Play Music: $
// Receive of title
if  event(readudp(Port,IP,StringPlayer)) then                                                \\
        KNXTitle= convert(split(StringPlayer,find(StringPlayer,Pattern,0u16)+size(Pattern) ,END),KNXTitle);   \\
                write('1/1/1'c14,KNXTitle);                                                  \\
                Title=Title+split(StringPlayer,find(StringPlayer,$Play Music: $,0u16),END);  \\
                TitleNr= TitleNr+1 endif
if TitleNr==10 then sendmail($EibPC@enertex.de$, $A Hello from EIBPC$,Title);          \\
                 sendudp(3807u16,192.168.22.21,$ Stop Music$);                              \\
                TitleNr=11 endif
if delay(TitleNr==11, 5000u64) then write('1/0/1'b01, OFF) endif
```

Configuration of mail

$ $: Character string with up to 1400 characters

sendudp

readudp
split: Splitting of 1400-Strings
find: Looking in 1400-Strings
size: Size of 1400-Strings

sendmail

$ $: String up to 1400 characters

First, we have defined the character string *Title*. In contrast to the data type c14, at this type the string is up to 1400 characters long. To define this data type, the character string is framed by two dollar signs and we denote this type as c1400.

$ Text $ is string “ Text “ (With spaces)

This type of data can **not** be written on KNXTM display elements and only be converted to a KNXTM type by the convert-function.

To determine the length of a c1400 string, it is created with a \0 character at the end. This must be considered in the LAN telegrams.

Next, we make use of sendudp:

sendudp: Any arguments, Not strings only!

sendudp(Port {Type u16},IP Address,Any number and types of arguments)

The sendudp function requires a variable or an expression of data type u16 as the first argument. This argument specifies the port number to which a UDP telegram is to be sent. The IP address (second argument) is entered in the usual notation and is internally compatible with the type u32. The actual data starts from the third Argument. These are arbitrary in number and type.

The Enertex® EibPC itself always sends from port 4807. This port can not be changed. When sending, the receiver's port is specified on which the recipient receives the telegram.

If your LAN device can be addressed by a name and DNS, you can also use the function

sendudp with name resolution

resolve(String{c1400})

For example in this case, you can also use an expression of the form

```
sendudp(3807u16,resolve($www.enertex.HomeMusicPlayer.de$),$ Start Music$)
```

The sendudp function can handle arbitrary data formats. Therefore, you are not restricted to the use of c1400-strings (as in this example). However, more on that later.

readudp: Port, IP and any arguments, not strings only!

The counterpart to the sending of a UDP telegram is its reception. For this purpose, the function readudp is available.

readudp(Port {Type u16},IP Address{u32}, Any arguments)

Again: The first argument is a variable or an expression of data type u16 and represents the port on which a UDP telegram was received. The IP address (second argument) is of data type u32. The actual “user data” start with the 3rd argument. Their number and data type is arbitrary.

The Enertex® EibPC always receives on port 4806. This port cannot be changed and must be considered by the sender of UDP telegrams.

The readudp-function verifies in the input buffer of the Enertex® EibPC whether there is a UDP telegram. In this case, the arguments of the readudp function are filled with data up to the length of its arguments. In any case, the variables *port* and *ip* of the function readudp are overwritten with the current data of the transmitter every time a UDP telegram is received.

Readudp and Event: A message is arrived?

A special feature is the use of event function: It indicates whether a telegram was read. In this example, we relied on the fact that the data came from the correct port and sender. In order to assure this, the following could also be written:

```
if  event(readudp(Port,IP,StringPlayer)) and (Port==808u16) and IP=192.168.22.21) then \\
```

Thus if a u32 variable integer or float f32 variable would have been defined, the statements would differently interpret the received data.

```
readudp(Port,IP,StringPlayer)
readudp(Port,IP,Integer, Float)
```

Now for string processing: split

Fragmentation of character strings

```
split(String{c1400}, PositionStart{u16}, PositionEnde{u16})
```

indicates the part of the output string between the two positions (in numbers). The major item here is 65534u16, for which the “built-in“ constant END can also be used.

You can simply concatenate character strings of type c14 and c1400 by “+“:

Concatenation of character strings

```
Title=split($ Hello $,1u16,3u16)+split($ Hello $,1u16,3u16)
```

Therefore the above term would mean that *Title* is initialized with the string “HeHe“.

We still have to perform searching in a character string with find

Finding sub-strings in character strings

```
find(Source{c1400}, SearchString{c1400}, Appearance{u16})
```

By means of this function, it can be searched in the character string *Source* for the position of the *SearchString*. By the variable *Occur*, the preceding matches could be skipped:

```
Pos=find($ Hello Welt Hello $,$Hello$,1u16)
```

Then in this example, *Pos* is reserved with the value 12u16. If the find function cannot find anything, then it returns the value 65535u16. For this value, the “built-in“ constant *EOS* (end of string) can be used.

Back to the example:
Now, the character strings of the UDP telegram can be processed:

```
KNXTitle= convert(split(StringPlayer,find(StringPlayer,$Play Music: $,0u16),EOS),KNXTitle)\\
```

As you can see, the data type of the string has finally to be converted into a KNX c14-data type. This is done by using the convert function. The conversion is done by extracting the first 14 characters from the string *StringPlayer*.

The function sendmail remains:

... and finally an e-mail will be sent

```
sendmail(ID {Type c1400}, Header {c1400}, Body{c1400})
```

The function sendmail writes a message to the e-mail server defined above (see Figure 27). It should be noted that, caused by internal system characteristics, only 255 characters of the specified character string are considered. In particular, Enertex® EibPC does not transmit more than the first 255 characters of *Body* of e-mail.

Binary telegrams

The functions sendudp and readudp can also be used to process binary data. In this case, telegrams, the structure and data layout of which are specified, but which are not character strings, are evaluated.

Therefore, we set ourselves the following problem, at which it must be taken into account that the Enertex® EibPC itself receives only from its port 4806, and that its transmitting port is always 4807.

Different definition of telegrams

The Enertex® EibPC receives from port 4806 (port of the sender 5220, sender 192.168.22.22) telegrams of the form:

Type1: Command 1000 until 1200: Arrangement of data in the telegram

```
        Command (32 bit)
        Number1 (Sign, 32 bit)
        Floating-point number (32 bit)
        Number1 (No sign 8 bit)
```

Type2: Command 2000 until 2200 : Arrangement of data in the telegram

```
        Command (32 bit)
        Floating-point number (32 bit)
        Number1 (No sign 8 bit)
```

– If command has a value of 1000, the result of the sum of the two numbers shall be sent on the same communication channel. The reply message has the same structure, and the result shall be sent at the position of the of number1. Command of the response amounts to 1100.

– If command has a value of 1001, the difference of the two numbers shall be sent on the same communication channel as a floating point number. The reply message has the same structure, and the result shall be sent at the position of the of floating point number. Command of the response amounts to 1101.

– If command has a value of 2000, the root of floating point numbers shall be calculated and sent back with command 2001. If the floating-point number is less than zero, the integer of the telegram will be set to 1, otherwise to 0 .

Now, the program is written simply: You will notice that here the nesting of if statements is also avoided (see page 51).

The two variables Command1 and Command2 realize the definition of the reading functions by readudp (processing of arguments of Type1 and Type2).

```
[EibPC]
// Variables
Command=0u32;Number1=0u32;Number2=0u32;FloatingPoint=0f32
IP=0u32;Port=0u16
// Is the telegram on the correct port?
CheckSender=(IP==192.168.22.22) and (Port==5220u16)
// The both telegram types
Command1=event(readudp(Port,IP,Command, Number1, FloatingPoint, Number2 )) and CheckSender
Command2=event(readudp(Port,IP,Command, FloatingPoint, Number1)) and CheckSender

// Sum back
if Command==1000u32 and Command1 then                                  \\
        sendudp(Port,IP, 1100u16,Number1+Zahl2, FloatingPoint, Number2 ) \\
        endif
// Difference back
if Command==1001u32 and Command1 then                                                    \\
        sendudp(Port,IP, 1100u16,Zahl1, convert(Number1, FloatingPoint)-                 \\
        convert(Number2, FloatingPoint), Number2 ) \\
        endif

// Square root
if Command==2001u32 and Command2 and FloatingPoint>0f32 then \\
        sendudp(Port,IP, 2001u16,sqrt(FloatingPoint), 0u32 )    \\
        endif
// Square root does not run, because argument is smaller than zero
if Command==2001u32 and Command2  and FloatingPoint<0f32 then  \\
        sendudp(Port,IP, 2001u16, FloatingPoint, 1u32 )         \\
        endif
```

event(readudp): Invalid at every telegram

Command1: I arrived a telegram, , IP and Port are OK.
Command2: ditto

With any incoming UDP telegram at port 5220, the function event(readudp(...)) is temporarily changing from 0b01 to 1b01.

Then, the telegram types *Command1* and *Command2* react (linked with *CheckSender*) at each telegram that 192.168.22.22 sends from its port 5220 (the destination port in Enertex® EibPC is 4806).

By linking with the variable *Command*, the desired telegram analysis is then performed. Accordingly, *Command1* and *Command2* are absolutely identical of behavior, especially *Command1* is also ON, if *Command2* goes to ON.

Therefore, *Command1* and *Command2* could be alternatively combined into one telegram realized by the variable *Telegram* as described below.

```
[EibPC]
// Variables
Command=0u32;Number1=0u32;Number2=0u32; FloatingPoint=0f32
IP=0u32;Port=0u16
// The both telegram types
CheckSender=(IP==192.168.22.22) and (Port==5220u16)
Telegram=(event(readudp(Port,IP,Command, Number1,  FloatingPoint, Number2 )) or
event(readudp(Port,IP,Command,  FloatingPoint, Number1))) and CheckSender

// Sum back
if Command==1000u32 and Telegram then                                        \\
              sendudp(Port,IP, 1100u16,Number1+Number2, FloatingPoint, Number2 ) \\
              endif

// Difference back
if Command==1001u32 and Telegram then                                                                          \\
              sendudp(Port,IP, 1100u16,Number1, convert(Number1, FloatingPoint)-                               \\
              convert(Number2, FloatingPoint), Number2 ) \\
              endif

// Square root
if Command==2001u32 and Telegram and FloatingPoint>0f32 then  \\
              sendudp(Port,IP, 2001u16,sqrt( FloatingPoint), 0u32 ) \\
              endif
// Square root
if Command==2001u32 and Telegram and FloatingPoint<0f32 then \\
              sendudp(Port,IP, 2001u16, FloatingPoint, 1u32 ) \\
              endif
```

TCP/IP server and client

If your Enertex® EibPC is equipped with software option NP, it basically works as a TCP/IP server at port 4809. Likewise, the Enertex® EibPC also provides a TCP/IP client.

The syntax for the implementation of a TCP/IP communication is exactly that of a UDP communication. Thus we can at this point simply refer to the preceding pages. The advantage for the user is that a TCP connection can be established by the same method. Therefore, code lines are simply to be re-used.

The overview in short form:

Same syntax, only „tcp" instead of „udp"

sendtcp(port {data type u16},IP address {data type u32},
arbitrary number and data types of arguments)

sendtcp expects a variable or an expression of data type u16 as the first argument. This argument indicates the port number a TCP/IP telegram shall be sent to. The IP address (second argument) has to be entered in the usual notation and is internally compatible with data type u32. The actual data start with the 3rd argument. They are arbitrary in number and data type.

The Enertex® EibPC always sends from port 4809. This port cannot be altered. For transmission, one specifies the receiver's port.

The counterpart of sending a TCP telegram is its reception. For this, the function readtcp is provided.

readtcp(port {data type u16},IP address {data type u32}, arbitrary arguments)

Here also applies: The first argument is a variable or an expression of data type u16 and provides the port of the transmission a TCP telegram has been received by. The IP address (second argument) is of data type u32. The actual “user data” start with the 3rd argument. They are arbitrary in number and data type. Again holds: If your LAN device can be addressed via DNS and a name, you can use the function resolve also here.

readtcp fills its arguments with data

readtcp looks into the input buffer of the Enertex® EibPC for a received TCP telegram. If a TCP telegram is available, the arguments of the function, but in any case *port* and *IP address*, are “filled” with data until the amount of received data (incoming TCP telegram) conforms to the data length of the arguments of the function readtcp.

To check if a message has been received, you can, like with the UDP telegrams, resort to event(readtcp()).

Event occurred?

```
if event(readtcp(Port,IP,StringPlayer)) and (Port==808u16) and (IP==192.168.22.21) then \\
```

Client and server

The Enertex® EibPC establishes connections via sendtcp and readtcp both as a client and as a server.

If you operate the Enertex® EibPC as a server, a client has to register with the Enertex® EibPC. This is performed automatically by the operating system without your interaction, because the Enertex® EibPC is always in reception mode.

Whenever the client sends a message to the Enertex® EibPC, event(readtcp()) becomes active. In the program, one has only to “look” if the IP is that of a desired peer, and then process the data accordingly. After 30 seconds of inactivity of an existing connection, the Enertex® EibPC disconnects automatically.

More details

To operate the Enertex® EibPC as a client, it has to register with the server by means of

connecttcp(port {data type u16}, address {data type u32}).

In order to disconnect duly, one needs:

closetcp(port {data type u16}, address {data type u32}).

At www.enertex.de/downloads/d-eibpc/DokuCF-1.pdf you will find a detailed example of the implementation of a more complex command processing as realized for Command Fusion.

Webserver

The Enertex® EibPC can also make with the equipment option NP a visualisation and controlling of your KNX™-installation. The visualitation is possible with the help of the Internet browers whereupon the Enertex® Bayern GmbH tested with the Mozilla Firefox.

The webserver in the EibPC is called via the input of the IP in the address line of the browser.

Visualisiation assistent

The surface can comfortable be created with the Enertex visualisation assistent.

With only six steps to your own visu:

Marvel „VisuAssi“

Step 1 – Open the assistent

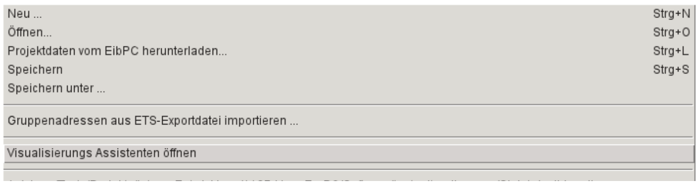

Neu ... Strg+N
Öffnen... Strg+O
Projektdaten vom EibPC herunterladen... Strg+L
Speichern Strg+S
Speichern unter ...
Gruppenadressen aus ETS-Exportdatei importieren ...
Visualisierungs Assistenten öffnen

Abbildung 28: Aufrufen des Visualisierungsassistenen

Step 2 – Install floors and rooms

Room defining

Each room gets in the visualisiation an own page.

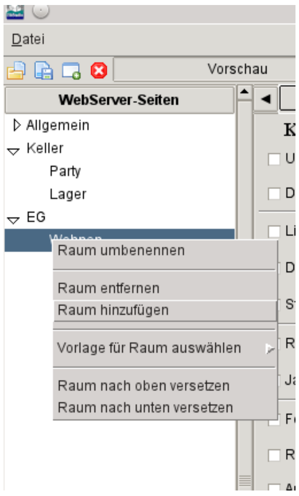


Abbildung 29: Seiten anlegen

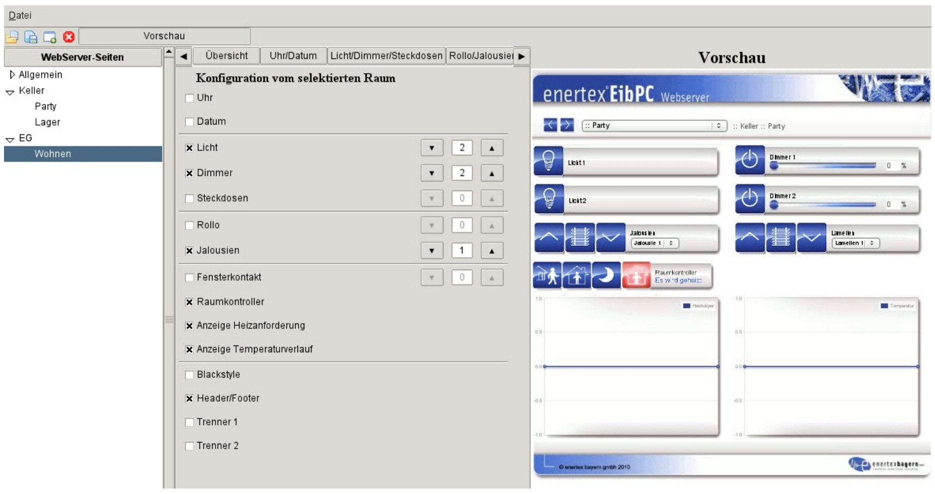
Step 3 – Creat rooms individual

Each page can be individually created

Each room can be parametrised seperate and fast.

You can choose various standard elements like lights and dimmer and jalusie and room controller.

Pretend the number of the desired elements. The preview shows symbolically the developing of the specified page.



Konfiguration vom selektierten Raum

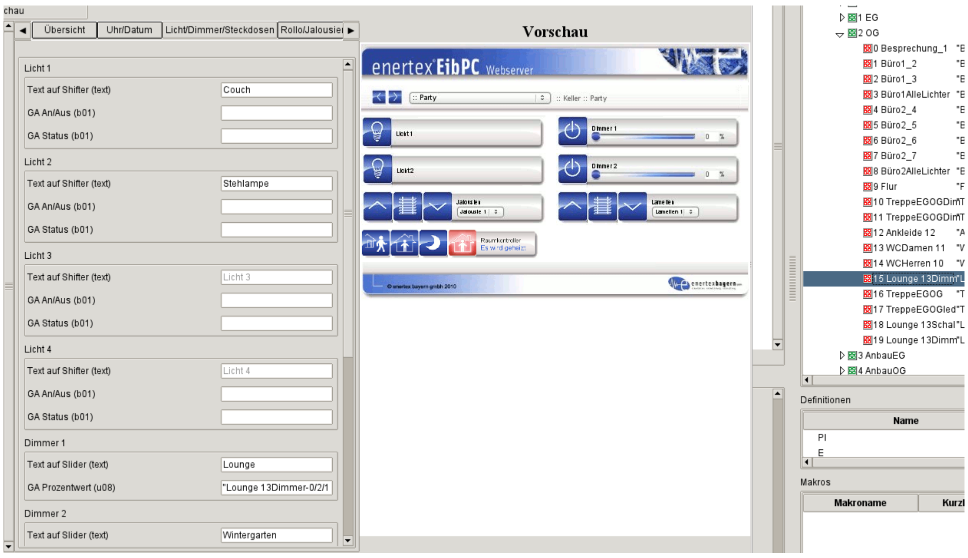
☐ Uhr		
☐ Datum		
☒ Licht	2	
☒ Dimmer	2	
☐ Steckdosen	0	
☐ Rollo	0	
☒ Jalousien	1	
☐ Fensterkontakt	0	
☒ Raumkontroller		
☒ Anzeige Heizanforderung		
☒ Anzeige Temperaturverlauf		
☐ Blackstyle		
☒ Header/Footer		
☐ Trenner 1		
☐ Trenner 2		

Picture 30: Create pages

Step 4 – Connect group addresses with elements

Connecting group addresses and elements

Choose the menu taps in menu bar, enter the display texts and Drag and Drop and assign the group addresses with elements.



Licht 1

Text auf Shifter (text)	Couch
GA An/Aus (b01)	
GA Status (b01)	

Licht 2

Text auf Shifter (text)	Stehlampe
GA An/Aus (b01)	
GA Status (b01)	

Licht 3

Text auf Shifter (text)	Licht 3
GA An/Aus (b01)	
GA Status (b01)	

Licht 4

Text auf Shifter (text)	Licht 4
GA An/Aus (b01)	
GA Status (b01)	

Dimmer 1

Text auf Slider (text)	Lounge
GA Prozentwert (u08)	"Lounge 13Dimmer-0/2/1

Dimmer 2

Text auf Slider (text)	Wintergarten

```
▷ 1 EG
▽ 2 OG
    0 Besprechung_1   "E
    1 Büro1_2         "E
    2 Büro1_3         "E
    3 Büro1AlleLichter "E
    4 Büro2_4         "E
    5 Büro2_5         "E
    6 Büro2_6         "E
    7 Büro2_7         "E
    8 Büro2AlleLichter "E
    9 Flur            "F
    10 TreppeEGOGDim"T
    11 TreppeEGOGDim"T
    12 Ankleide 12    "A
    13 WCDamen 11     "V
    14 WCHerren 10    "V
    15 Lounge 13Dimm"L
    16 TreppeEGOG     "T
    17 TreppeEGOGled"T
    18 Lounge 13Schal"L
    19 Lounge 13Dimm"L
▷ 3 AnbauEG
▷ 4 AnbauOG
```

Definitionen

Name
PI
E

Makros

Makroname	Kurzl

Picture 31: Connecting group addresses with elements

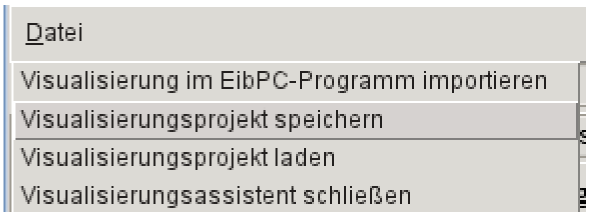
Step 5 – Saving project of visualisation and loading in the EPC-program

Saving, loading, ready

While saving the assistant proofs the existing links and warns of inconsistencies or missing group addresses.

After connecting all elements with the group address the project has to be loaded into the EibPC-program.

A dialog is leading through the necessary steps:


Datei
Visualisierung im EibPC-Programm importieren
Visualisierungsprojekt speichern
Visualisierungsprojekt laden
Visualisierungsassistent schließen

Picture 32: Saving and importing the project of visualisation

There will be three data applied and per *#include* included in the epc-program. This could be done by the assistant completely autonomous.

At the end the actual EnertexWebV*x*.lib has to be loaded in the epc-program.

How to involve macro-libraries is explainde on page 221 .

das modifizierte epc-Programm

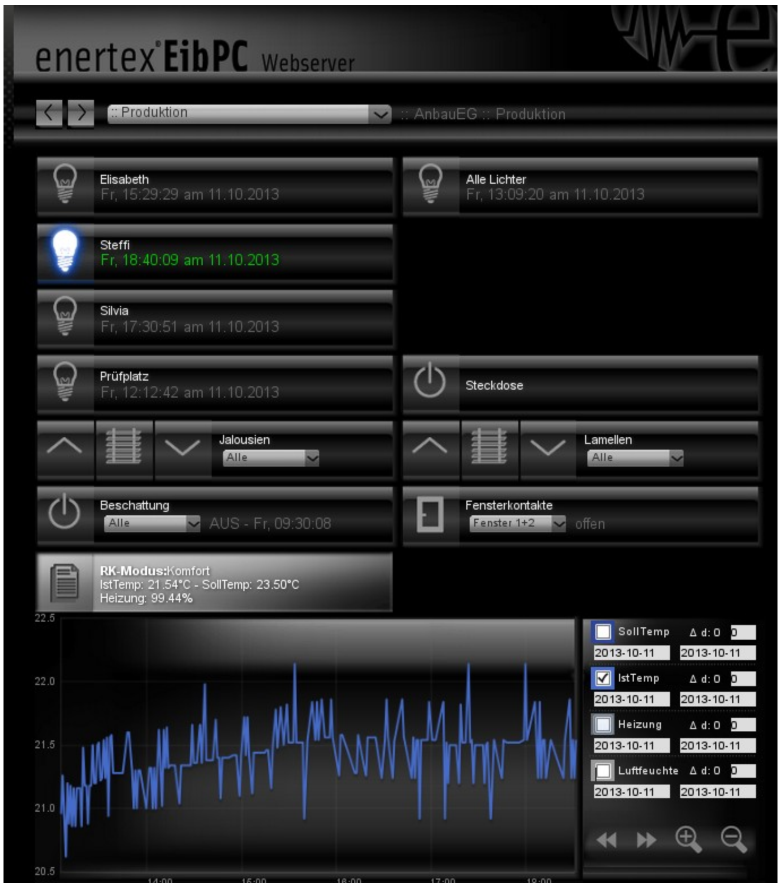
```
[MacroLibs]
//Macro-Libraries
EnertexWebV2.lib

[WebServer]
#include VA_EibPC_Webserver.epc
[Macros]
#include VA_EibPC_Macros.epc

[ETS-ESF]
// The ESF-data is exported out of ETS3
1148-building-6.esf

[EibPC]
#include VA_EibPC.epc
```

Step 6 – Transfer the program and open the browser

The program has to be transferred to the EibPC.(F3).

Should the situation arise it could come to delays after the transfer, because all statuses are requested for the webserver.

The webserver in the EibPC will be called by the input of the IP in the address line of the browser.



Picture: Webserver Example 33:

If you wish to get in deeper into the surface programming, please read the following chapter carefully.

Web server

Single-page version

With the option NP, the Enertex® EibPC may also carry out the visualization and control of your KNX™ installation. The visualization requires the usage of an Internet browser. Mozilla Firefox has been tested on all PC Plattforms.

For our example, we assume that we intend to realize

- a timer control,
- a diagram of the living room's temperature,
- the values of the weather map
- and a light switch.

At the end, the web server will look like in Figure 34. Now we show how it works. At first, we will use the single-page version only. When you enter the IP of your Enertex® EibPC in the address field of your browser, the view of the web server is displayed. It can be look like this:

Our web server's appearance

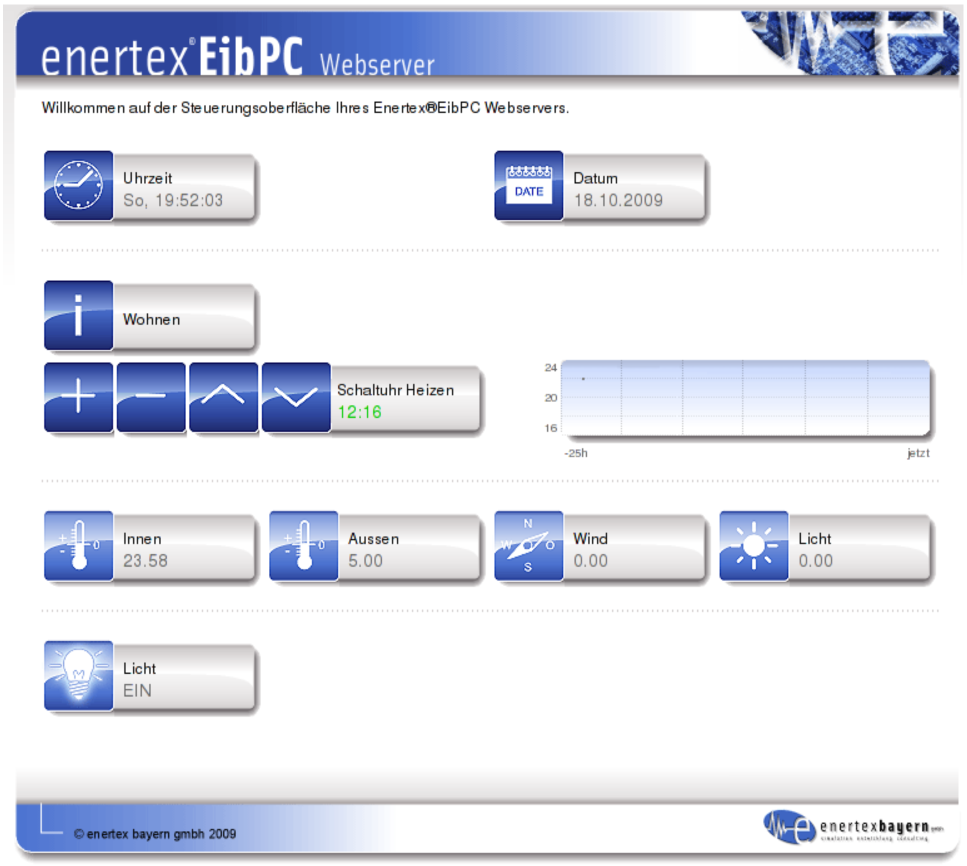


Figure 34: The web server

The minimal design:

5 rows and 4 columns

The web server arranges its elements which have either a predefined single or double length like a checkerboard pattern. There is basically the option not to use some fields and to insert dividers.

The arrangement and configuration of the web elements will be made in the section [WebServer]. Due to the fixed target of built-in icons, lengths and variables, the configuration can be purely text-based without graphical effort. Thus, in a very simple manner, a professional web interface can be created. The icons (see summary at page 291) are created in a common design.

The minimal design consists of four columns and five rows. The header and footer can be disabled. Thus, by the design of the operational elements a very ergonomic interface is created on touch panels with a resolution of 800x600. If the lines are not defined in the Section [WebServer], the area remains unfilled. Then, In the simplest case, the web server consists of a blank page.

First the elements must be configured. This is done in the user program as follows:

```
[EibPC]

[WebServer]
button(1)[CLOCK]$Time$ none button(2)[DATE]$Date$
line
button(3)[INFO]$Rooming$
shifter(4)[PLUS,MINUS,UP,DOWN]$Timer Heating$  chart(5)[$16$,$20$,$24$]
line
button(6)[TEMPERATURE]$Indoor$  button(7)[TEMPERATURE]$Outdoor$   \\
                                        button(8)[WIND]$Wind$  button(9)[WEATHER]$Light$
line
button(10)[LIGHT]$Light$
```

What you can configure everything

The following elements are available:

- *button*: single length, a graphic symbol (icon), pretending a fixed and a variable line of text, symbol dynamically changeable
- *shifter*: twofold length, one to four graphic symbols (icon), a fixed and a variable line of text, one symbol is dynamically changeable
- *chart*: twofold length, is used for display of curves (XY diagrams)
- *none*: empty element
- *line*: thin separation line
- In the overview on page 291 we find the possible symbols which can be addressed either by indices (numbers) or by the predefined constants.

Three basic rules for building the web server

Let us look again at the specification for the generation of the webserver:

1. Each element has an ID (up to 39 elements).
2. Each line of the configuration corresponds to a row of visualisation.
3. Elements are separated by one or more spaces.
4. Empty elements (*none*) will be automatically generated (e.g . the last line of the web server).
5. The webserver consists of a maximum of ten columns. Each line of the configuration corresponds to a row of visualization.

Thus we are already be able to “design“ the web server.

But now the question arises: How can we respond to the user's input in the user program?

Query button

The attentive reader will already have noticed the numbers in parentheses within the definitions of the elements. These are used for the allocation in the application program, in which the elements *button, shifter* and *chart* can be addressed by the functions webbutton, webdisplay and webchart, respectively.

The function

webbutton*(ID{u08})*

Buttons and text change

Floating-point numbers are directly converted to strings by the function. Dates, e.g. the return of settime() etc., as well.

returns for the duration of one processing pass a non-zero value, when pressing the push button on a web button (*button* or *shifter*) with the *ID* (number between 0 and 255). In case of the activation of a *button* element, the return value changes from 0 to 1 (u08) and back to 0. In case of a *shifter*-element, the return changes to 1,2,3 or 4 (u08) depending on the activated switching element of *shifter*. The *shifter* may variably show one, two, three or four icon. Here the numbers refer to the arrangement of circuit elements from left to right. If *button* or *shifter* have the same *ID*, then, in both cases of the activity, the function webbutton also reacts twice. *ID* may be randomly allocated and must not be consecutively numbered. However, only numbers between 0 and 39 are allowed.

As shown in Figure 34, each button and shifter consist of graphics and two lines of text. The first line of text has already been indicated in the above configuration at the end of the element and cannot be changed at runtime.

webdisplay(*ID{u08},Text{any data type},Icon{u08},State{u08},TextStyle{u08})*

When calling the function, the icon of the web element with *ID* (number between 0 and 255) will also be set to the symbol with the number *Icon*. Possible graphs are shown on page 159. In addition, predefined constants facilitate the selection listed in Table 2 (page 159). The argument *Text* indicates an arbitrary variable the value of which is transformed into a string is displayed at the variable text line of web element. Each icon has at least the states ACTIVE (==1), INACTIVE (==2), DARKRED (==0) and BRIGHTRED (==9).

One of these states can be passed in the argument *State*. An overview of the possible states is shown in Table 3 (page 159). The text to be displayed can be printed in the styles GREY (==0), GREEN (==1), BLINKRED(==2) and BLINKBLUE (==3).

For the display of values, we use the state DISPLAY (==3) of the icons. This condition does not exist for all icons, see page 291.

Finally, we still miss the ability to fill the XY diagrams with values. There is the function

The graph is shifted at each function call

webchart(*ID, Var{u08}, X1{c14}, X2{c14}*)

The function accesses the XY diagram *chart*. If *ID* occurs multiple times, then all elements with this *ID* are accessed. However, the *IDs* of a *chart*, *button* and *shifter*, respectively, do not interfere. At call of the function, the XY diagram of value *Var* is activated. Values from the field 1...30 can be represented. 0 means no representation, values greater than 30 are illegal and are interpreted as 0. At every call of the function, the values are displayed starting from the left. When the end is reached after 47 calls, the values are shifted to the left. The x-axis label is specified by the arguments *X1, X2* (data type c14) and can be adjusted at runtime.

Thus, our application program reads as follows:

```
[EibPC]
// Webserver
if stime(0) then webdisplay(21,settime(),CLOCK,INACTIVE,GREY) endif
if stime(0) then webdisplay(22,setdate(),CLOCK,INACTIVE,GREY) endif
// Display of the living room temperature
if cycle(30,0) then webchart(10,convert(8.5*("RkLivingroomTemp-3/1/28"-16.0),0), $-25h$c14,$now$c14)
endif
// Set timeswitch
TimeswitchH=0
TimeswitchM=0
if webbutton(6)==1 then TimerH=TimerH+1 endif
if webbutton(6)==2 then TimerH=TimerH-1 endif
if webbutton(6)==3 then TimerM=TimerM+1 endif
if webbutton(6)==4 then TimerM=TimerM-1 endif
if TimerH>23 then TimerH=0 endif
if TimerM>59 then TimerM=0 endif
if change(TimerM) or change(TimerH) then \\
        webdisplay(0,convert(TimerH,$$c14)+$:$c14+convert(TimerM,\\
        $$c14),DOWN,INACTIVE,GREEN) endif

// Weather station
if stime(0) then webdisplay(2,"RkLivingroomTemp-3/1/28",TEMPERATURE,DISPLAY,GREY) endif
if stime(0) then webdisplay(3,"ThermeExternalTemperature-3/3/13",TEMPERATURE,DISPLAY,GREY) endif
if stime(0) then webdisplay(4,"Wind-3/5/1",WIND,DISPLAY,GREY) endif
if stime(0) then webdisplay(5,"Light-3/5/2",WEATHER,DISPLAY,GREY) endif

// Light switch
if webbutton(30)==1 then if "WorkroomLight-1/1/16" then write("WorkroomLight-1/1/16",AUS) endif; \\
                if !"WorkroomLight-1/1/16"      then write("WorkroomLight-1/1/16",EIN) endif \\
                endif
if "WorkroomLight-1/1/16" then webdisplay(30,$ON$c14,LIGHT,ACTIVE,GREY) else \\
webdisplay(30,$OFF$c14,LIGHT,INACTIVE,GREY) endif
```

Scale within a chart

The webchart can represent 47 integer values on the x-axis and 30 integer values (lines) on the y-axis. In our example, the x-axis corresponds to the period and the y-axis to the measured value. Because we visualize room temperatures, a scale of 16 to 24 (if no frost monitoring is made) is appropriate.

By cycle, we display the time interval (x-axis). The total period shown in the webchart is calculated from 47 times time interval, i.e. in the example 30x47 minutes = 23.5 hours.

The calculation of the scale (y-axis) is as follows: 8 degrees (16 to 24) divided into 30 steps. Therefore, we select 30/8 = 3.75 as scaling for the temperature values.

Accessing the definitions of variables for the IDs of the configuration

Because the web elements in the application program have the same IDs as in the configuration of the web server, we can use u08 variable definitions in the section [EibPC] for generating constants for the access:

```
[WebServer]
button(ClockWebID)[CLOCK]$Time$ none button(DateWebID)[DATE]$Datum$
[EibPC]
ClockWebID=21
DateWebID=22
// Webserver
if stime(0) then webdisplay(ClockWebID,settime(),CLOCK,INACTIVE,GREY) endif
if stime(0) then webdisplay(DateWebID,setdate(),CLOCK,INACTIVE,GREY) endif
```

Please note that here the web server evaluates the variable statically. So, if the variable *ClockWebID* changes at runtime, the index still refers to the initial value 221.

The control concept of the web server

What happens when you click on the buttons?

The integrated web server is designed in such a way that it immediately responds to pressing the buttons in the web browser and sends corresponding information to the processing loop. Moreover, when pressing a button, the icon of a symbol group always changes to the state ACTIVE immediately, which is characterized by a lighting effect. This aims at facilitating the detection of the activity.

If the application program reacts to this keypress, e.g. by changing the display state via webdisplay or webchart, the HTML file of the web server is modified. Approximately 1.5 seconds after actuation, the web server sends an update command to the browser, which implies the call of the new HTML file. This response time depends on the utilization of the Enertex® EibPC and may increase slightly. In addition, the web server generates this update command every 30 seconds without user input on the web browser. Therefore, the web interface provides a current state.

Multiple-page version

Multipage version blue-Design

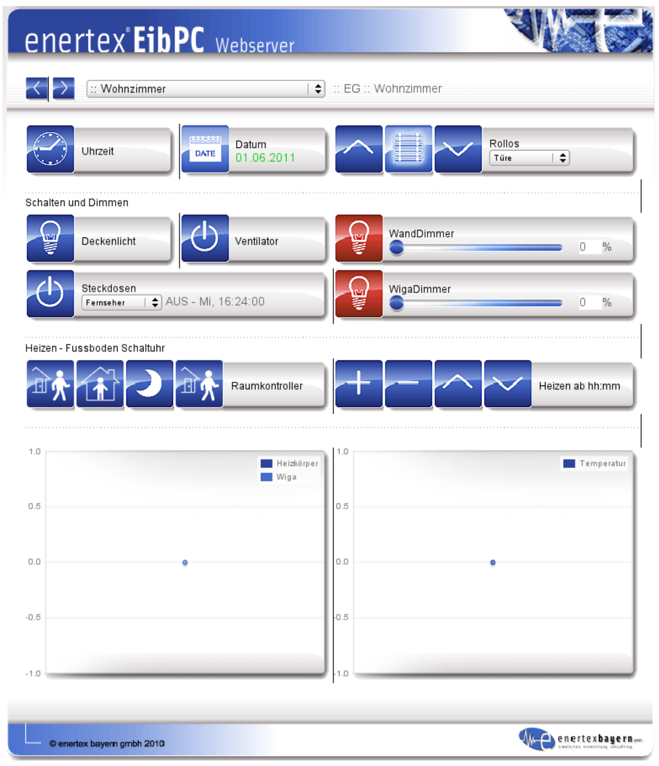


Figure 35: The web server – multipage-version, blue-Design

Multipage version black-Design

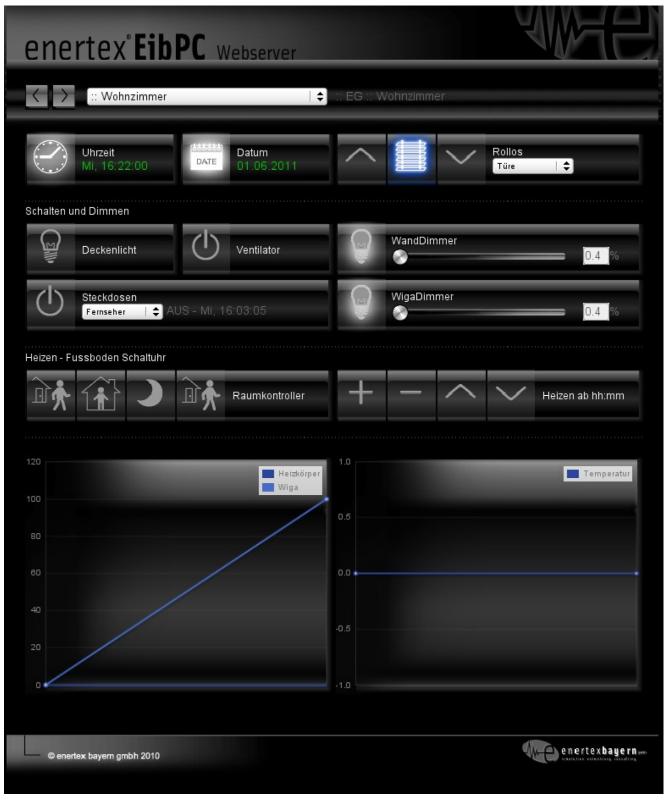


Figure 36: The web server – multipage-version, black-Design

In figures 35 and 36 the two current desgin variations are shown.
By default, the blue version is used.

In the following, we will show the application with multiple pages.

When using multiple pages, it makes sense to divide this pages thematically into groups, e.g. basement. Each page will have its own heading, e.g. central heating room. The web server displays this assignment in the listbox of the page navigation (cf. Figure 37).

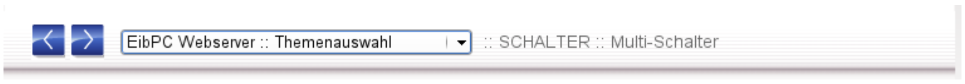

EibPC Webserver :: Themenauswahl
:: SCHALTER :: Multi-Schalter

Figure 37: Page navigation

To generate a page-oriented visualization, we need at first the page element in section [WebServer].

Defining a page

Working with IDs makes it easier later on

```
[WebServer]
page(CentralHeatingRoomPageID)[$Basement$,$Heating$]
[EibPC]
CentralHeatingRoomPageID=1
```

A page is, therefore, generated by

page(ID)[$GroupName$,$PageName$

This command generates the page assignment automatically. If a page has been assigned to a group by this command, in the above example to “Basement”, it appears in this order in the selection list box. The page itself has the name “Heating“. The quick selection (forward and back button, respectively, in Figure 37) is given by the order of the definition.

By *header* and *footer,* you can switch off the internal header and footer, respectively. Besides, you have the possibility to embed a graphic from an external source. The design of the the header and the footer is identical on every page.

Local element:
The element is bound to the page.
Global element:
The element is displayed identically on all pages.

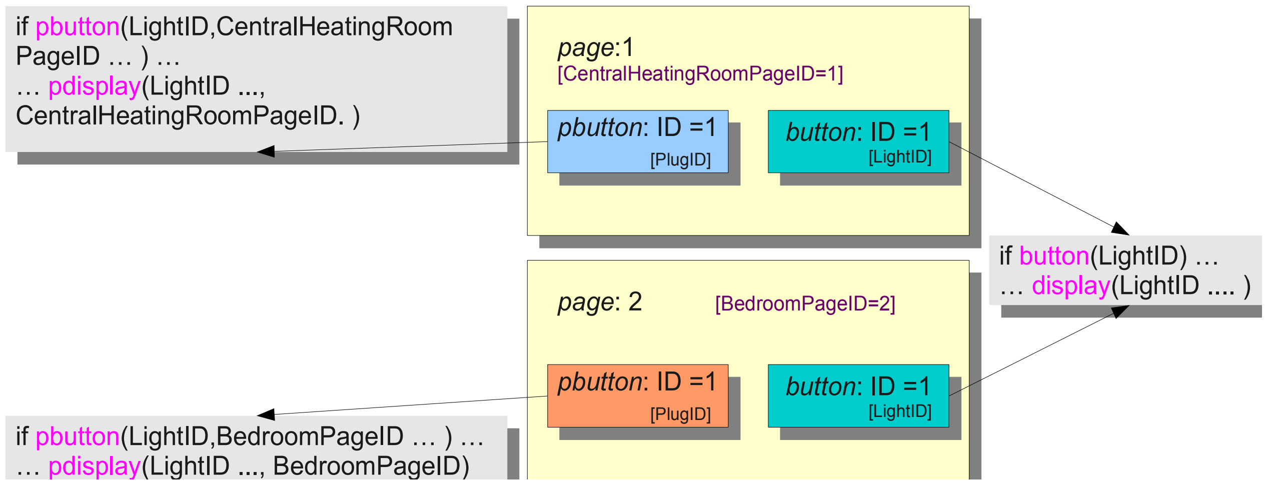
There exist global and local elements for the visualization. Global means that an element can be used on different pages, but has been instanced only once. Accessing or changing the element via the application program is therefore identical for all pages. However, a local element can be changed and accessed only on one page. Local and global elements have an identical design. Local elements are differ to global ones by the prefix “p” (page). The page numbering starts with 1. To exemplify this, we extend the above code:

Global IDs may match local ones. However, separate objects are addressed.

```
[WebServer]
// Page 1
page(CentralHeatingRoomPageID)[$Basement$,$Heating$]
button(LightID)[LIGHT]$All Lights$ none pbutton(PlugID)[LIGHT]$Sockets$
// Page 2
page(BedroomPageID)[$2nd floor$,$Bedroom$]
button(LightID)[LIGHT]$All Lights$ none pbutton(PlugID)[LIGHT]$Sockets$

[EibPC]
CentralHeatingRoomPageID=1
BedroomPageID=2
LightID=1
PlugID=1
if '1/2/4'b01 then display(LightID,$ON$,LIGHT,ACTIVE,BLINKBLUE) else \\
                    display(LightID,$OFF$,LIGHT,INACTIVE,GREY) endif
if '1/2/5'b01 then display(PlugID,$ON$,LIGHT,ACTIVE,BLINKBLUE,CentralHeatingRoomPageID)  else \\
                    display(PlugID,$OFF$,LIGHT,INACTIVE,GREY,CentralHeatingRoomPageID) endif
if button(LightID)==1 then write('1/2/4'b01, OFF) endif
if pbutton(PlugID,CentralHeatingRoomPageID)==1 then write('1/2/5'b01,OFF) endif
```

The *button* with *LightID* is global, it is identical on every page. Therefore, if you change its status icon or status text line by display or webdisplay, respectively, it changes on all pages. This is different with the local element *pbutton* with *PlugID*. On change of the group address ('1/2/5'b01), only the element with *PlugID* and the page ID *CentralHeatingRoomPageID* (i.e. the element with *PlugID* on this page) is changed. To achieve a change of the graphic or the text, respectively, here one has to use the function pdisplay. We point it out again by way of the following figure:

Figure 38: Global and local (page-oriented) elements



if pbutton(LightID,CentralHeatingRoom PageID …) … … pdisplay(LightID ..., CentralHeatingRoomPageID.)
page:1
[CentralHeatingRoomPageID=1]
pbutton: ID =1
[PlugID]
button: ID =1
[LightID]
if button(LightID) … … display(LightID ....)
page: 2
[BedroomPageID=2]
pbutton: ID =1
[PlugID]
button: ID =1
[LightID]
if pbutton(LightID,BedroomPageID …) … … pdisplay(LightID ..., BedroomPageID)

When requesting the status by means of the functions button and pbutton, respectively, it has to be differentiated between global and page-oriented elements (prefix „p“). The former function refers to the state of a global button, the second to a page-oriented (=local) one.

The fundamental advantage of a global element is its property of being displayed identically on all pages. This way, alarm messages and status displays (e.g. time) can be generated that are to be utilized for all pages in the same manner. Solely by their appearance, global and local elements cannot be distinguished.

By means of global elements, alarm messages or time displays for every page can be realized.

This is true in the same way for all further elements the names of which differ in the prefix *p*.

The *mbutton* constitutes another interesting element (Figure 38).

To begin with, we look at its “global” variant. This element has two control levels.

1. The button with symbol display on the left.
2. A configurable listbox with up to 255 elements.

Control element and selection

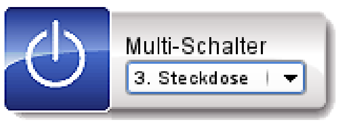


Figure 39:mbutton

The concept presents itself to the operator as follows: By pressing on the listbox, the control element can be configured or selected, respectively. If the web page e.g. visualizes a room with four switchable sockets, the socket to be switched when the button on the left is being pressed can be selected here. By pressing the button, the requested action will be performed.

Therefore, the following task presents itself to the programmer:

This is our task

A) Find out which element of the listbox is active.
This takes place with the aid of mbutton() (see below)

B) Switching the control element by means of display()

C) Find out if the button has been activated.
Again, this takes place with the aid of mbutton() (see below)

D) Updating the display via display()

The function

mbutton(*ID{u08},selection{u08}*)

is designed for the evaluation of the listbox.
If the listbox is changed in such a way that the element with the index *selection* is changed, the function returns 255, and if the button is pressed, the value index of the *selection*. We want to exemplify this. At first, we have to add the element *mbutton* (for its syntax, see page 275) to our web server.
To this end, we complete our configuration:

A global Mbutton with 3 elements

```
[WebServer]
// Page 1
page(CentralHeatingRoomPageID)[$Basement$,$Heating$]
button(LightID)[LIGHT]$All lights$ button(PlugID)[LIGHT]$Sockets$
mbutton(mPlugID)[SWITCH][$1.Socket$,$2.Socket$,$3.Socket$]$MultiSwitch$
// Page 2
page(2)[$2nd floor$,$Bedroom$]
button(LightID)[LIGHT]$All lights$ button(PlugID)[LIGHT]$Sockets$
```

In this manner, we have added the global web element *mbutton*. Similarly, we could have defined a local element (*mpbutton*).

These variables simplify programming

How do we address the element? To this, we add the following lines to the program:

State 0 of mbutton() means: The user has operated the listbox.

Adjusting the display: If any information changes, call display(). One call for all changes.

```
mPlugID=1
mPlug1=1
mPlug2=2
mPlug3=3
mText=$AUS$
mICON=SWITCH
mState=ACTIVE
mDeco=GREY
mChoice=1
// Querying the listbox
if mbutton(mPlugID,mPlug1)==255 then {
		mChoice=1
} endif
if mbutton(mPlugID,mPlug2)==255 then {
		mChoice=2
} endif
if mbutton(mPlugID,mPlug3)==255 then {
		mChoice=3
} endif
// Adjusting the display
if change(mText) or change(mICON) or change(mState) or change(mDeco) or change(mChoice) then {
		display(mPlugID,mText,mICON,mState,mDeco,mChoice)
} endif
```

Now the user already can operate the listbox, and the Enertex® EibPC „memorizes“ this setting. What is yet missing is task D) and E) of the explanations above.

It is conspicuous that display passes a text, although page 267 states:

No text can be displayed in the second line.

In the present case indeed no text is displayed, but the element is not “bothered” by display() supplying another argument. An mshifter element which basically exhibits the same appearance but has double width could display a dynamic text in the second line. Therefore, a further adaptation to this element in the application program is not necessary.

To distinguish the button, we have to use the function mbutton again, and we complete our program as follows:

1. Socket is at '1/5/1'b01

2. Socket is at '1/5/2'b01

3. Socket is at '1/5/3'b01

```
if mbutton(mPlugID,mPlug1)== 1 then {
        write('1/5/1'b01,!'1/5/1'b01)
} endif
if mbutton(mPlugID,mPlug2)== 1 then {
        write('1/5/2'b01,!'1/5/2'b01)
} endif
if mbutton(mPlugID,mPlug3)== 1 then {
        write('1/5/3'b01,!'1/5/3'b01)
} endif
```

What is still missing is the design of the button which shall represent the state of the switching actuator: This is achieved as follows:

```
if '1/5/1'b01 and mChoice== mPlug1 then {
        mText=$ON$;
        mDeco=GREEN;
        mState=DISPLAY
} endif
if !'1/5/1'b01 and mChoice== mPlug1 then {
        mText=$OFF$;
        mDeco=GREY;
        mState=INACTIVE
} endif
if '1/5/2'b01 and mChoice== mPlug2 then {
        mText=$ON$;
        mDeco=GREEN;
        mState=DISPLAY
} endif
if !'1/5/2'b01 and mChoice== mPlug2 then {
        mText=$OFF$;
        mDeco=GREY;
        mState=INACTIVE
} endif
if '1/5/3'b01 and mChoice== mPlug3 then {
        mText=$ON$;
        mDeco=GREEN;
        mState=DISPLAY
} endif
if !'1/5/3'b01 and mChoice== mPlug3 then {
        mText=$OFF$;
        mDeco=GREY;
        mState=INACTIVE
} endif
```

This points out why use variables: These make it now possible to design the program in a simple way: If it is switched from outside or the variable *mChoice* is altered, the information for the visualization is modified via display() accordingly, and this change itself initiates the output process to the web element.

You can work in the same way with the elements *mshifter* and the page-oriented versions *mpshifter* and *mpbutton*, respectively.

With *mbutton*, the user has a very compact control element for multiple tasks and the operation of a multiple selection, respectively. The programmer has now the task to process this possibilities accordingly. In other words: Albeit the user sees only one button with e.g. three choices, the programmer of the application program faces the same effort of processing three buttons.

Compact to the user

However, the above code can be wrapped easily in a macro. You will find in the library EnertexWeb.lib the according Mbutton macros. Hence, our code can be realized simply by calling

```
[Macros]
Mbutton3(Plug,mPlugID,'1/5/1'b01,'1/5/2'b01,'1/5/2'b01)
[MacroLibs]
EnertexWeb.lib.
```

For the variations of the page-oriented elements and for the elements with double, triple and quadruple choice, respective macros can be found in the library EnertexWeb.lib.

Another element for web design is the *pslider* as a local page-oriented slider and the *slider* as the global variation thereon.

The current state is queried via getslider and set via setslider

```
[WebServer]
// Page 3
page(SliderPageID)[$2nd floor$,$LivingRoom$]
slider(SliderID)[LIGHT]$All dimmers$ pslider(pSliderID)[LIGHT]$TV illumination$
[EibPC]
SliderPageID=3
SliderID=4
pSliderID=5
MySlider=getslider(SliderID)
if change(MySlider) then setslider(SliderID,MySlider,LIGHT,INACTIVE) endif
MypSlider=getpslider(pSliderID,SliderPageID)
if change(MypSlider) then {
		setpslider(pSliderID,MypSlider,LIGHT,INACTIVE,SliderPageID)
} endif
```

Here, too, the user has to perform the programming of the s*lider* and its position, respectively. On release of the s*lider*, the web server transmits the state of the *slider* as a return value of data type u08 via the function

Global....

getslider(*ID{u08})*

or

....local

getpslider(ID{u08},PageID{u08}).

To retain its position, the *slider* has to be set to the requested value by means of

setslider(*ID{u08},Value{u08},Icon{u08}, Style{u08})*

or

setpslider(ID{u08},Value{u08},Icon{u08}, Style{u08}, PageID{u08}),

respectively.

If you want to jump to other pages or embed other web pages in your web pages, work with the elements *plink*, *link*, *frame* and *dframe* that are to be used as follows - **at this, keep in mind the safety risk of your web client when using external links***:*

Linking

```
[Webserver]
page(5)[$Miscellaneous$,$Links$]
plink(1)[MAIL][1]$Go to page 1$        link(2)[MAIL][$http://www.shop.enertex.de$]$Shopping at Enertex$
frame [$http://www.enertex.de$]
dframe [$http://www.enertex.de$]
```

The configuration of the elements plink and link, respectively, is handled equal to the button element. By means of the function pdisplay the Icon and the text can be changed.

mchart and picture

With it, we reach two further important web elements for visualization with the Enertex® EibPC, the *mchart*. This element can be utilized for embedding external camera pictures or weather forecasts as well as for displaying up to four different graphs for the visualization of arbitrary runs.

Embedding external sources is achieved as follows:

Embedding external graphics

```
page(WeatherID)[$Miscellaneous$,$Weather$]
mchart[DOUBLE,CENTERGRAF]($The weather$, \\
        $http://de.weather.yahoo.com/images/eur_germany_outlook_DE_DE_440_dmy_y.jpg$) \\
mchart[DOUBLE,CENTERGRAF]($Satellite picture$ ,\\
        $http://de.weather.yahoo.com/images/eur_satintl_440_dmy_y.jpg$)
```

You will find the exact syntax of *mchart* and *mpchart*, respectively, particularly the visualization options, on page 278. At this, the design of an XY chart is to be configured as follows:

```
mpchart[DOUBLE,XY]($Temperature$,TempID,LINE,$Flow$,FlowID,LINEDOTS,\\
$OutdoorTemperature$,OutdoorID,COLUMN)  \\
mpchart[DOUBLE,XY]($Temperature$,TempID,LINE)
```

The first statement defines a *chart* which records up to three graphs, whereas at the second statement only one graph is recorded.

To fill these in the application program with data every 15 minutes, we proceed as follows:

XY charts

```
[EibPC]
TempID=0
FlowID=1
OutdoorID=2

//Time display
Time=convert(hour(),0.0)+convert(minute(),0.0)/60.0
if mtime(15,0) then mpchart(TempID,Time,'2/3/5'f16,WeatherID) endif
if mtime(15,0) then mpchart(FlowID,Time,'2/3/4'f16,WeatherID) endif
if mtime(15,0) then mpchart(OutdoorID,Time,'2/3/4'f16,WeatherID) endif
```

Congratulation. You have now reached the end of this comprehensive introduction and should now be able to work with the Enertex® EibPC on your own.

Visualization wizard

From EibStudio® V2.200, you can also create web pages using the visualization wizard.

Detailed web server example

The following is a complete visualization of a building. This visualization is active in the foyer of the Enertex building on a 27-inch HP - Android panel.

This visualization has been developed within the scope of an internship and explains the essential points in the programming and creation of the web pages.

In the entrance area at the Enertex® Bayern GmbH

Figure 40:HP Android based visualization

The aim is to visualize status display with a central switching possibility of the house control as well as weather information on several pages.

Let's start with the main page, which is shown as follows:

Main page of visualization

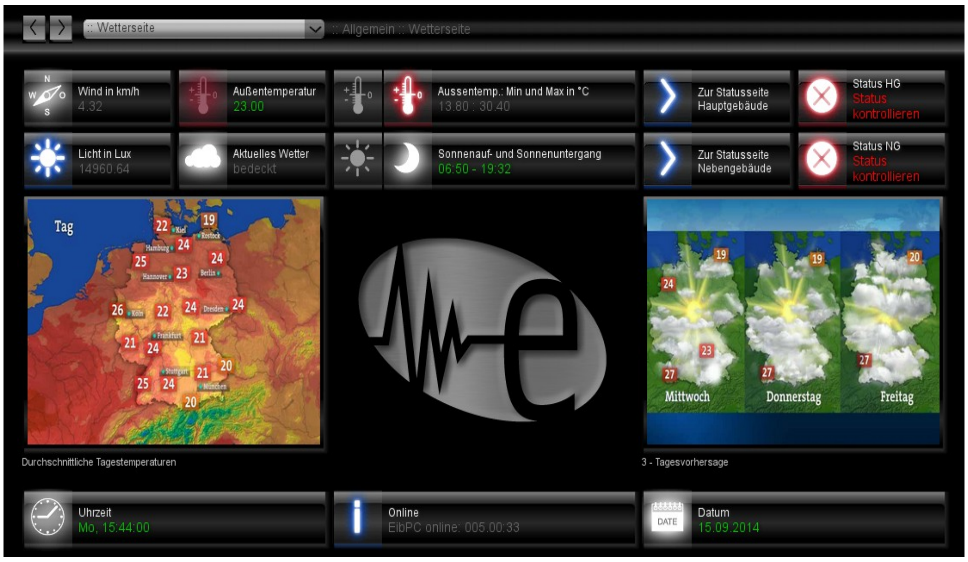


Figure 41:Main page of visualization

At the beginning, we want to allocate our structure to the programming in the EibStudio, this classification is necessary for the functionality of the instructions. Basically there are different sections marked with "[]". In these sections, we will introduce sub-points where step-by-step (//) descriptions, variables are defined, web server pages are added, or statements are issued.

```
[MacroLibs]
/Libraries/InternEnertexWebV3.lib
[ETS-ESF]
//
[WebServer]
//
[Macros]
//
[EibPC]
//// Declarations
//// Functions
```

To create our first page in the EibPC, a page ID is required. If possible, this should reflect the display component of the page and is written to the [EibPC] section.

```
[EibPC]
//// Declarations
// Initializing the PageIDs for the Visualization Wizard
GeneralWeatherPageID = 1
```

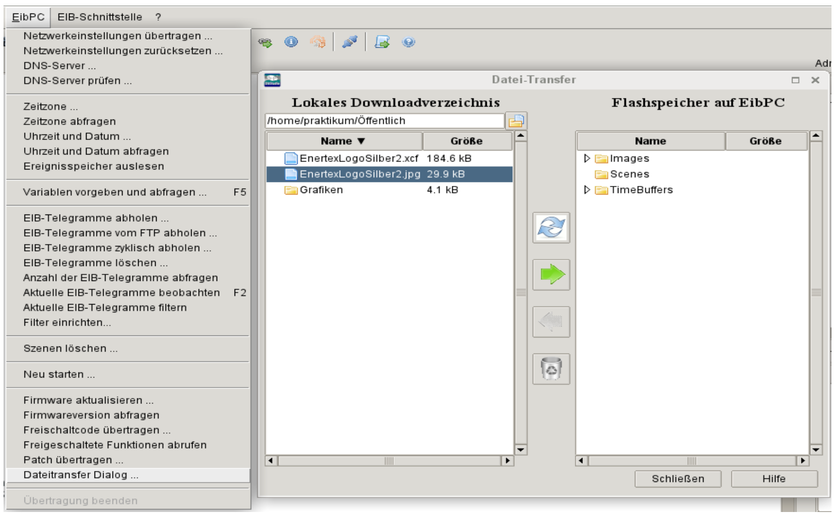
Initially you have to be clear about in addition to names (weather page) and category (general) of the main page, as well as in which design direction the web server should go. We opted for the design variant black and as an optical rounding we implemented a background graphic, which represents the company logo (see Figure 62). Attention: Background graphics are not scaled! An element has a size of 180x65 pixels. In the width, a further 43 pixels have to be added once. For an arrangement of 6 x 8 elements, the perfect size for a background graphic is 1123x520 pixels. The graphic must be arranged in such a way that the end of the visu in the middle of the page is marked with the desired company logo.

If the background image has been edited so far, you can include it in the [WebServer] section:

```
[WebServer]
//Web elements for the  weather page in the category General:
page(GeneralWeatherPageID)[$General$,$WeatherPage$]
design $black$[$/upload/EnertexLogoSilber2.jpg$]

[EibPC]
//// Declarations
//  Initializing the PageIDs for the Visalization Wizard
GeneralWeatherPageID = 1
```

For the EibPC to find the background graphic also under the specified path, it must be exported. This is done with the aid of the *EibStudio* under the tab *EibPC* and the menu item *Data transfer dialog*.

Upload pictures


EibPC
EIB-Schnittstelle
?
Netzwerkeinstellungen übertragen ...
Netzwerkeinstellungen zurücksetzen ...
DNS-Server ...
DNS-Server prüfen ...
Zeitzone ...
Zeitzone abfragen
Uhrzeit und Datum ...
Uhrzeit und Datum abfragen
Ereignisspeicher auslesen
Variablen vorgeben und abfragen ... F5
EIB-Telegramme abholen ...
EIB-Telegramme vom FTP abholen ...
EIB-Telegramme zyklisch abholen ...
EIB-Telegramme löschen ...
Anzahl der EIB-Telegramme abfragen
Aktuelle EIB-Telegramme beobachten F2
Aktuelle EIB-Telegramme filtern
Filter einrichten...
Szenen löschen ...
Neu starten ...
Firmware aktualisieren ...
Firmwareversion abfragen
Freischaltcode übertragen ...
Freigeschaltete Funktionen abrufen
Patch übertragen ...
Dateitransfer Dialog ...
Übertragung beenden
Datei-Transfer
Lokales Downloadverzeichnis
Flashspeicher auf EibPC
Schließen
Hilfe

/home/praktikum/Öffentlich

Name ▼	Größe
EnertexLogoSilber2.xcf	184.6 kB
EnertexLogoSilber2.jpg	29.9 kB
Grafiken	4.1 kB

Name	Größe
Images	
Scenes	
TimeBuffers	

Figure 42:Data transfer dialog

With the possibility to load your own images into the Enertex EibPC, you can also use them to create your own display for the header and footer graphics. First of all, you have to upload the graphics in advance to the Enertex® EibPC.

In our example, we set the header and the footer to (0) respectively, thus omitting the header and footer, as well as the possibility to use our own pictures.

After the design work has been completed so far, the implementation of the display buttons takes place. These can be provided with group addresses and icons (see page 291) as required. Many of the buttons work with pre-built macros, which are present in the Enertex libraries.

Basically you distinguish between **local** and **global** elements in the display. Global means that the element can be used on different pages, but it was only generated once. Therefore, an access or modification of the element by the application program is the same for all pages. On the other hand, a local element can only be changed in one page. Local and global elements are identical to the design, the former being indicated by the addition "p" (page).

Some of the elements (eg shifter (ClockID) [CLOCK] $ Time $ and shifter (DateID) [DATE] $ Date $) were initialized as global on our main page since these should serve as a complete information on all sides. For these buttons, web element IDs must be initialized, which are also written to the [EibPC] section.

```
[EibPC]
//// Declarations
//  Initializing the PageIDs for the Visualization Wizard
GeneralWeatherPageID = 1
//Initialization of the web element IDs
WindID = 1
WolkenID = 2
ClockID = 3
DateID = 4
```

In order to be able to work faster in the future, it is recommended to select function-oriented ID designations.

The local elements are created directly in the web server configuration with IDs from (0 to 39), but are just like the global elements for the time being without functionality.

The difference between button and shifter is just the size of the elements. While a button has a simple width, the shifter is displayed with a double width.

These elements are configured as follows:

shifter(ID)[Graphics1, Graphics2, Graphics3, Graphics4]$Text$	button(ID) [Graphics] $Text$
pshifter(ID)[Graphics1, Graphics2, Graphics3, Graphics4]$Text$	pbutton(ID) [Graphics] $Text$

Graphic 2 to graphic 4 are optional

Now we know how the elements are integrated into the [WebServer] section of our site. They can be arranged in various ways as chessboard patterns. The fixed configuration of built-in icons, lengths and variables means that the configuration can be done purely text-based without the need for any graphics.

```
[WebServer]
//Web elements for the weather page in the category General:
page(GeneralWeatherPageID)[$General$,$WeatherPage$]
design $black$[$/upload/EnertexLogoSilber2.jpg$]
header(0)
footer(0)
// First Line
button(WINDID)[WIND]$Wind in km/h$ \\
  pbutton(31)[TEMPERATURE]$Outdoor temperature$\\
  pshifter(17)[TEMPERATURE,TEMPERATURE]$Outer temp.: Min and Max in °C$\\
  pbutton(5)[OKCIRCLE]$Status HG$
// Second Line
pbutton(15)[WEATHER]$Light in Lux$ \\
  button(WolkenID)[WEATHER]$current weather$\\
  pshifter(14)[WEATHER,NIGHT]$Sunrise- and Sunset$\\
  pbutton(6)[OKCIRCLE]$Status NG$
// Third Line
shifter(ClockID)[CLOCK] $Time$\\
  pshifter(13)[INFO]$Online$\\
  shifter(DateID)[DATE]$Datum$
```

If we compile our previous program, the following representation is shown in the web server.

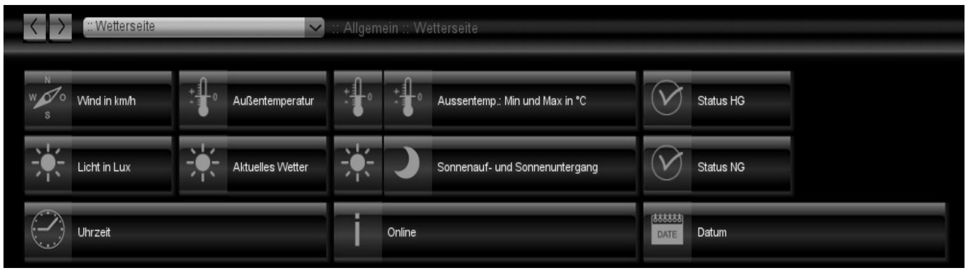


Figure 43:Elements in chessboard pattern

The arrangement of the elements is shown as desired. However, our logo can not be seen on the background graphics because it is partly hidden by the elements, but we also scaled it for a representation of 6x8 fields in the middle.

To create something for the eye, but also to get more information, we embed two graphical weather forecasts. These should frame our background artwork on the side and stand above the created global footer of time and date. For the integration we use the following function:

picture(ID)[Height,Type]($Description$,$www-LINK$)

For the selected links, care should be taken that these are independent of time and date. Thus the pages are updated independently and it does not need any other code to generate a refresh.

Examples:

Good:

$http://wetter.tagesschau.de/import/wetter-cms/vorhersagen/img/de-vs-tt_webL.jpg$

Bad:

$http://www.wetteronline.de/?daytime=day&diagram=true&fcdatstr=**20140916**&iid=DL&pid=p_city_local&sid=Pictogram$

After selecting the two links, we must also integrate our picture functions in the section [WebServer]. In order to achieve the framing of the logo on the sides, two **none** elements must be used between the picture functions. These are used as placeholders in the web server and are not displayed in any way

```
[WebServer]
//Web elements for the weather page in the category General:
page(GeneralWeatherPageID)[$General$,$WeatherPage$]
design $black$[$/upload/EnertexLogoSilber2.jpg$]
header(0)
footer(0)
// First Line
button(WINDID)[WIND]$Wind in km/h$ \\
  pbutton(31)[TEMPERATURE]$Outdoor temperature$\\
  pshifter(17)[TEMPERATURE,TEMPERATURE]$Outer temp.: Min and Max in °C$\\
  pbutton(5)[OKCIRCLE]$Status HG$
// Second Line
pbutton(15)[WEATHER]$Light in Lux$ \\
  button(WolkenID)[WEATHER]$current weather$\\
  pshifter(14)[WEATHER,NIGHT]$Sunrise- and Sunset$\\
  pbutton(6)[OKCIRCLE]$Status NG$
// Third Line
picture(7)[DOUBLE,CENTERGRAF]($Average daily temperatures$,
$http://wetter.tagesschau.de/import/wetter-cms/vorhersagen/img/de-vs-tt_webL.jpg$) \\
  none \\
  none \\
  picture(8)[DOUBLE,CENTERGRAF]($3 – Day prediction$,$http://wetter.tagesschau.de/import/wetter-
cms/vorhersagen/img/de-vs-3t_webL.jpg$)
// Fourth Line
shifter(ClockID)[CLOCK] $Time$\\
  pshifter(9)[INFO]$Online$\\
  shifter(DateID)[DATE]$Date$
```

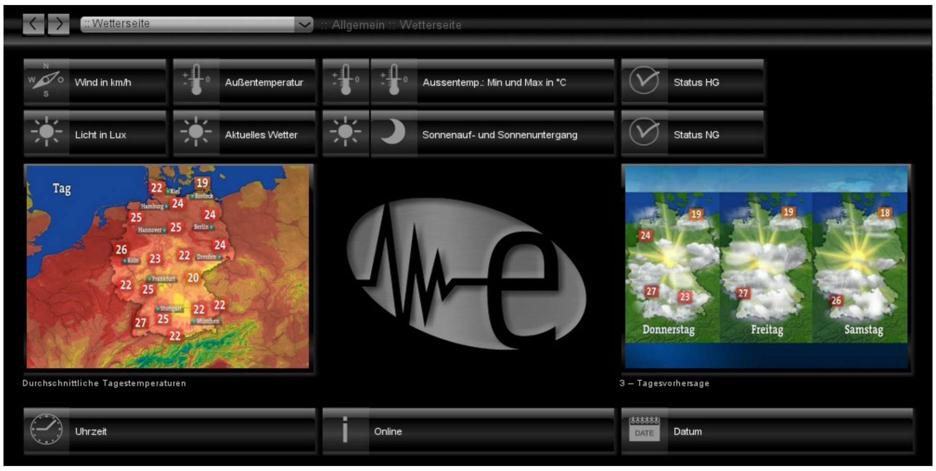
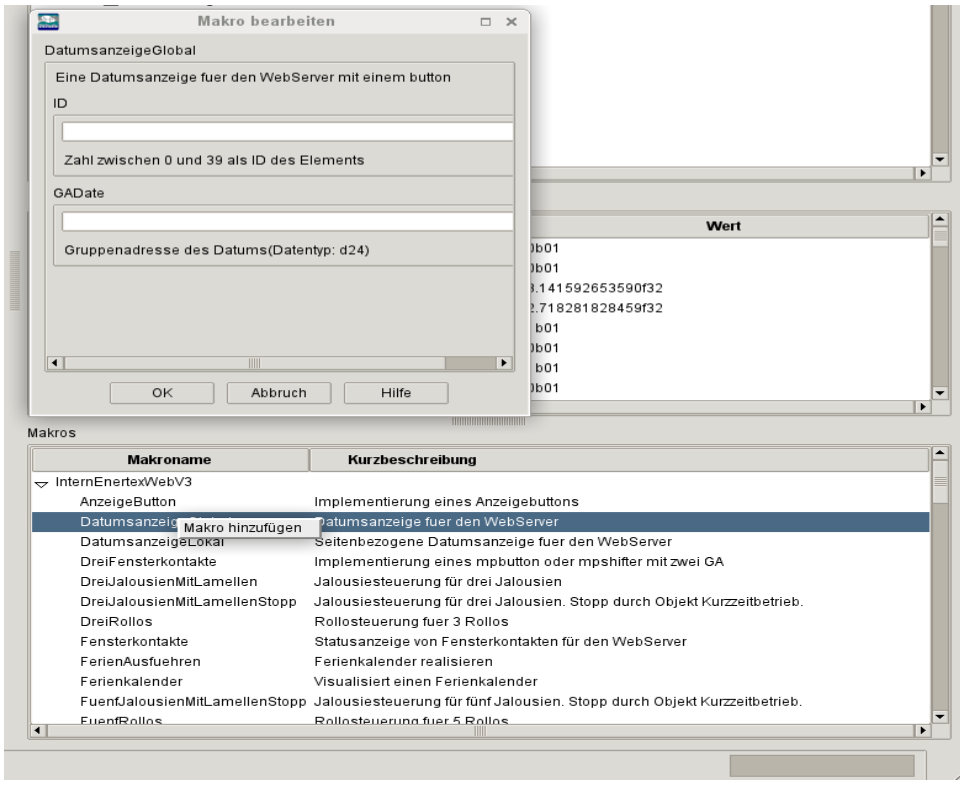


Figure 44:The configuration of the buttons

The layout of our Advertisement Button is now complete and we can devote ourselves to the functionality. To do this, we have to go to the [Macros] section of the code. With the help of the **macro library (InternEnertexWebV3)** the appropriate functionality can be found for each button / shifter.

Example date display global:



DatumsanzeigeGlobal

Eine Datumsanzeige fuer den WebServer mit einem button

ID

Zahl zwischen 0 und 39 als ID des Elements

GADate

Gruppenadresse des Datums(Datentyp: d24)

Makroname	Kurzbeschreibung
InternEnertexWebV3	
AnzeigeButton	Implementierung eines Anzeigebuttons
DatumsanzeigeGlobal	Datumsanzeige fuer den WebServer
DatumsanzeigeLokal	Seitenbezogene Datumsanzeige fuer den WebServer
DreiFensterkontakte	Implementierung eines mpbutton oder mpshifter mit zwei GA
DreiJalousienMitLamellen	Jalousiesteuerung für drei Jalousien
DreiJalousienMitLamellenStopp	Jalousiesteuerung für drei Jalousien. Stopp durch Objekt Kurzzeitbetrieb.
DreiRollos	Rollosteuerung fuer 3 Rollos
Fensterkontakte	Statusanzeige von Fensterkontakten für den WebServer
FerienAusfuehren	Ferienkalender realisieren
Ferienkalender	Visualisiert einen Ferienkalender
FuenfJalousienMitLamellenStopp	Jalousiesteuerung für fünf Jalousien. Stopp durch Objekt Kurzzeitbetrieb.
FuenfRollos	Rollosteuerung fuer 5 Rollos

Figure 45:Macro selection

If the correct macro is applied for each Button / Shifter and the corresponding fields are filled in, the following lines are created in the [Macros] section

```
[Macros]
// Macros for the weather page in the category General:
DateDisplayGlobal(DateID,"Date-7/0/0")
TimeDisplayGlobal(ClockID,"Time-7/0/1")
WindDisplayButtonGlobal(WINDID,"Wind-14/6/0")
MinMaxTemperatureDisplayButtonLokal(2,GeneralWeatherPageID,"Outdoor temperature-14/6/3")
LightDisplayButtonLocal(4,GeneralWeatherPageID,"Light-14/6/4")
SunRiseSetDisplayShifterLokal(5, GeneralWeatherPageID)
OnlineDisplayButtonLocal(9,GeneralWeatherPageID)
```

Now all display buttons are assigned a function, up to

```
pbutton(1)[TEMPERATURE]$Outdoor temperature$
pbutton(3)[OKCIRCLE]$Status HG$
pbutton(6)[OKCIRCLE]$Status NG$
```

The ID's (3) and (6) will be used later. The pbutton (1), however, is to be given a dynamic icon, which is colored according to the outside temperature and also indicates this. So blue is said to be cold, gray for moderate, dark red for pleasant warm and red for hot.

This is achieved by the following function:

pdisplay(ID, Text, Icon, State, Text Style, PageID, [Mbutton])

This is used to call pbutton or pshifter

First the division of the different states must be created. We opted for:

>= 27 °C = hot
< 27 °C und >= 18 °C = pleasant warm
< 18 °C und >= 5 °C = moderate
< 5 °C = cold

The return value of the sensor may vary depending on the outside temperature sensor. Our weather station returns the value f16. Thus: 0f16 = 0°C

The status statements must now be packaged in if statements in the [EibPC] sector and transferred with pdisplay.

```
[EibPC]
//// Functions
/// Temperature display dynamic
if change("Outdoor temperature-14/6/3") and "Outdoorr temperature-14/6/3">=27f16 then pdisplay(1,"Outdoor temperature-14/6/3",TEMPERATURE,BRIGHTRED,BLINKRED,GeneralWeatherPageID) endif

if change("Outdoor temperature-14/6/3") and "Outdoor temperature-14/6/3">=18f16 and "Outdoor temperature-14/6/3" <27f16  then pdisplay(1,"Outdoor temperature-14/6/3",TEMPERATURE,DARKRED,GREEN,GeneralWeatherPageID) endif

if change("Outdoor temperature-14/6/3") and "Outdoor temperature-14/6/3">=5f16 and "Outdoor temperature-14/6/3" <18f16  then pdisplay(1,"Outdoor temperature-14/6/3",TEMPERATURE,DISPLAY,GREY,GeneralWeatherPageID) endif

if change("Outdoor temperature-14/6/3") and "Outdoor temperature-14/6/3" < 5f16  then pdisplay(1,"Outdoor temperature-14/6/3",TEMPERATURE,ACTIVE,BLINKBLUE,GeneralWeatherPageID) endif
```

Now, the pbutton (1) "knows" how to display its icon at the corresponding outside temperature.

To give the last display a functionality the following button is missing:

```
button(WolkenID)[WEATHER]$Current weather$
```

Here we want to implement a sun exposure dependent (lux value) display of the current cloudiness. It is differentiated in summer time (month May to month September) and winter time (month October to month March), since the sun elevation angle (elevation) and thus the intensity of the irradiation varies. If it rains, a rain report is issued.

Here, too, a division of the various states must be applied. We chose the following:

Summer time:

Irradiation <= 20000 Lux and this longer than 1 minute, then covered

20000 lux <irradiance <= 45000 Lux and this is longer than 1 minute, then cloudy

45000 lux <irradiance <= 70000 Lux and this longer than 1 minute, then sunny

70000 Lux <irradiance and this for more than 1 minute, then strong sunshine

Winter time:

Irradiation <= 3500 lux and this longer than 1 minute, then covered

3500 lux <irradiance <= 9000 Lux and this longer than 1 minute, then cloudy

9000 lux <irradiation <= 15000 Lux and this longer than 1 minute, then sunny

15000 lux <irradiation and this longer than 1 minute, then strong sunshine

Caution:

Summer time and winter time result in two conditions in the if statement. Thus, summer time and winter time must be defined as variables. Since the variables can either be true or false or ON or OFF, we define a time period using the month (dd, mm) function. This is also done in the section [EibPC] below the web element ID's.

```
[EibPC]
//// Declarations
// Initializing the PageIDs for the Visualization Wizard
GeneralWeatherPageID = 1
//Initialization of the web element IDs
WINDID = 1
WolkenID = 2
ClockID = 3
DateID = 4
//Variables
Summer time=month(01,05) and !month(30,09)
Winter time=month(01,10) and !month(30,04)
```

Now we realize the display with the functions:

webdisplay(ID,Text,Icon,Status,TextStyle)
delay(Signal, Time)

If we enter our conditions as an if statement in the section [EibPC], the following code is generated:

```
[EibPC]
//// Functions
/// Weather
//Summer time
if ( Summer time == ON and "Rain message-14/6/1"== ON) then webdisplay(WolkenID, $It is
raining$,WEATHER,STATE6,GREY) endif

if( Summer time == ON and "Rain message-14/6/1"==OFF and delay("Light-14/6/4" <= 20000f16,60000u64))
then webdisplay(WolkenID,$covered$,WEATHER,STATE5,GREY) endif

if ( Summer time == ON and "Rain message-14/6/1"==OFF and "Light-14/6/4" > 20000f16 and delay("Light-
14/6/4" <= 45000f16,60000u64)) then webdisplay(WolkenID, $cloudy$,WEATHER,STATE4,GREY) endif

if ( Summer time == ON and "Rain message-14/6/1"==OFF and "Light-14/6/4" > 45000f16 and delay("Light-
14/6/4" <= 70000f16,60000u64)) then webdisplay(WolkenID, $sunny$,WEATHER,DISPLAY,GREY) endif

if ( Summer time == ON and "Rain message-14/6/1"==OFF  and  delay("Light-14/6/4" > 70000f16,60000u64))
then webdisplay(WolkenID, $strong sunshine$,WEATHER,BRIGHTRED,BLINKRED) endif
//Winter time
if (Winter time == ON and "Outdoor temperature-14/6/3" < 0.0 and  "Rain message-14/6/1"==ON) then
webdisplay(WolkenID, $It is snowing$,ICE,ACTIVE,BLINKBLUE) endif

if (Winter time == ON and "Rain message-14/6/1"== OFF and delay("Light-14/6/4" <= 3500f16,60000u64))
then webdisplay(WolkenID,$covered$,WEATHER,STATE5,GREY) endif

if (Winter time == ON and "Rain message-14/6/1"== OFF and "Light-14/6/4" > 3500f16 and delay("Light-
14/6/4" <= 9000f16,60000u64)) then webdisplay(WolkenID, $cloudy$,WEATHER,STATE4,GREY) endif

if (Winter time == ON and "Rain message-14/6/1"== OFF and "Light-14/6/4" > 9000f16 and delay("Light-
14/6/4" <= 15000f16,60000u64)) then webdisplay(WolkenID, $sunny$,WEATHER,DISPLAY,GREY) endif

if (Winter time == ON and "Rain message-14/6/1"== OFF and delay("Light-14/6/4" >15000f16,60000u64))
then webdisplay(WolkenID, $strung sunshine$,WEATHER,BRIGHTRED,BLINKRED) endif
```

In order to complete the page, we need our two additional pages of status main buildings and status secondary buildings. These pages should serve as central switching possibilities and indicate whether and if yes on which floor still windows are open.There are forwarding to the corresponding floor built in.

These forwardings are also missing on the main page. In the case of an open window or a luminous lamp, in any room in the building, the respective status button (status HG) should change to red and make an X from the √.

We therefore include two links in the form of plink (ID) [Graphics] [PageSprungIndex] $ Text $ in the Web server, which are to pass on (status HG) and (status NG).

```
[WebServer]
//Web elements for the weather page in the category General:
page(GeneralWeatherPageID)[$General$,$WeatherPage$]
design $black$[$/upload/EnertexLogoSilber2.jpg$]
header(0)
footer(0)
// First Line
button(WINDID)[WIND]$Wind in km/h$ \\
   pbutton(1)[TEMPERATURE]$Outdoor temperature$\\
   pshifter(2)[TEMPERATURE,TEMPERATURE]$Outer temp.: Min and Max in °C$\\
   plink(10)[RIGHT][StatusHGID]$To status page main building$\\
   pbutton(3)[OKCIRCLE]$Status HG$
// Second Line
pbutton(4)[WEATHER]$Light in Lux$ \\
   button(WolkenID)[WEATHER]$Current Weather$\\
   pshifter(5)[WEATHER,NIGHT]$Sunrise and Sunset$\\
   plink(11)[RIGHT][StatusNGID]$To status page Secondary building$\\
   pbutton(6)[OKCIRCLE]$Status NG$
// Third Line
picture(7)[DOUBLE,CENTERGRAF]($Average daily temperatures$,
$http://wetter.tagesschau.de/import/wetter-cms/vorhersagen/img/de-vs-tt_webL.jpg$) \\
   none \\
   none \\
   picture(8)[DOUBLE,CENTERGRAF]($3 – Day prediction$,$http://wetter.tagesschau.de/import/wetter-
cms/vorhersagen/img/de-vs-3t_webL.jpg$)
// Fourth Line
shifter(ClockID)[CLOCK] $Time$\\
   pshifter(9)[INFO]$Online$\\
   shifter(DateID)[DATE]$Date$

page(StatusHGID)[$General$,$Status page main building$]
design $black$
header(0)
footer(0)

page(StatusNGID)[$General$,$Status page secondary building$]
design $black$
header(0)
footer(0)

[EibPC]
//// Declarations
// Initialization of the PageIDs for the Visualization wizard
GeneralWeatherSideID = 1
StatusHGID = 2
StatusNGID = 3
```

The linking of the new pages now works. In order to improve this visually, we use the function in the [EibPC] section:

plink (ID, text, icon, icon state, PageID, PageSprungIndex)

This can change the status of the icon by plink.

```
[EibPC]

////Functions
// plinks main page
plink(10,$To status page main building$,RIGHT,ACTIVE,GeneralWeatherPageID,StatusHGID)
plink(11,$To status page secondary building$,RIGHT,ACTIVE,GeneralWeatherPageID,StatusNGID)
```

After everything has been written in the different sections and we successfully compiled the code, our web server main page is as shown in Figure 41.

The status page of the main building is now being built. It should include switch-off facilities for all the lights on a floor as well as the entire main building and indicate whether a window is open on the respective floor.

Status page of the main building:

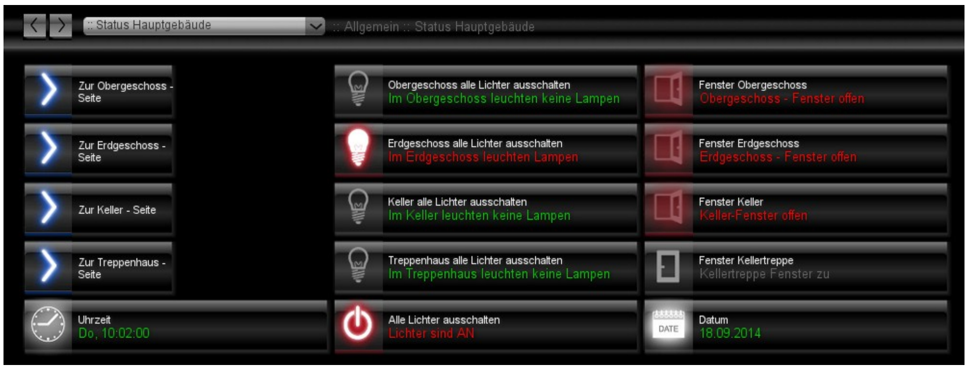

:: Status Hauptgebäude
:: Allgemein :: Status Hauptgebäude
Zur Obergeschoss - Seite
Zur Erdgeschoss - Seite
Zur Keller - Seite
Zur Treppenhaus - Seite
Uhrzeit
Do, 10:02:00
Obergeschoss alle Lichter ausschalten
Im Obergeschoss leuchten keine Lampen
Erdgeschoss alle Lichter ausschalten
Im Erdgeschoss leuchten Lampen
Keller alle Lichter ausschalten
Im Keller leuchten keine Lampen
Treppenhaus alle Lichter ausschalten
Im Treppenhaus leuchten keine Lampen
Alle Lichter ausschalten
Lichter sind AN
Fenster Obergeschoss
Obergeschoss - Fenster offen
Fenster Erdgeschoss
Erdgeschoss - Fenster offen
Fenster Keller
Keller-Fenster offen
Fenster Kellertreppe
Kellertreppe Fenster zu
Datum
18.09.2014

Figure 46:Status display with jump targets

In order to get back into the individual floors faster, we include plinks again. Now the individual rooms are listed and can therefore be controlled separately. As soon as a lamp is lit in a room, a lamp symbol will glow red on the status page of the main building and the corresponding text will appear in the display. If you now operate the shifter, all lamps on the floor will be switched off. If you want to do this with several or all floors with just one click, this can be realized by pressing the main switch in the bottom line, between the time and date display.

The page is now built step by step again and starts as our main page with the integration of the shifter and plinks. This will give them their functionality and design flare. In order to ensure the functionality, however, you should **before** build up your various **floor pages** with all the lights and windows. This leads to a better overview of the group addresses and facilitates the error search, if something goes wrong with the if statements.

Example floor side:

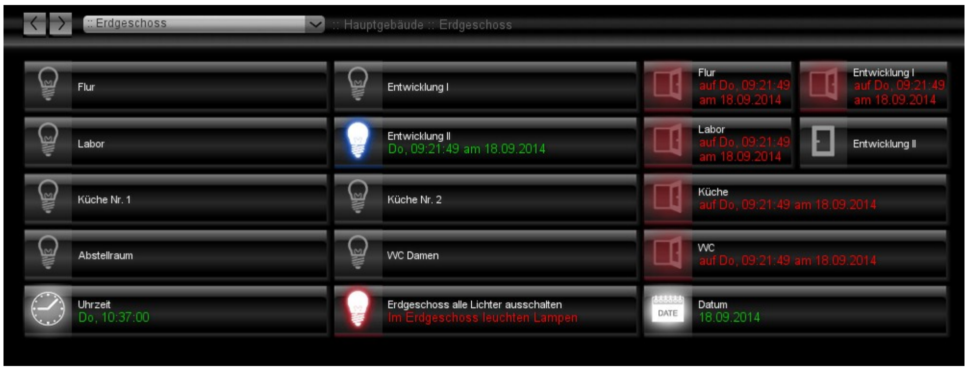

:: Erdgeschoss
:: Hauptgebäude :: Erdgeschoss
Flur
Labor
Küche Nr. 1
Abstellraum
Uhrzeit
Do, 10:37:00
Entwicklung I
Entwicklung II
Do, 09:21:49 am 18.09.2014
Küche Nr. 2
WC Damen
Erdgeschoss alle Lichter ausschalten
Im Erdgeschoss leuchten Lampen
Flur
auf Do, 09:21:49 am 18.09.2014
Entwicklung I
auf Do, 09:21:49 am 18.09.2014
Labor
auf Do, 09:21:49 am 18.09.2014
Entwicklung II
Küche
auf Do, 09:21:49 am 18.09.2014
WC
auf Do, 09:21:49 am 18.09.2014
Datum
18.09.2014

Figure 47:Detailed displays

Once you have finished, you can go back to the Status page.

As already mentioned, we first create our basic framework with button and shifter. We have implemented the floor switches as global because we integrate them on the static side and the floor side. The example deals with only the main switch of the ground floor, the others can be associated one by one.

```
[Web server]
//Web elements for Status main building
page(StatusID)[$General$,$Status main building$]
design $black$
header(0)
footer(0)
// First Line
plink(5)[RIGHT][FirstFloorID]$To First floor – Page$
  none
  shifter(MainSwitchFirstFloor)[LIGHT]$First floor turn off all lights$
  pshifter(11)[WINDOW]$Window First floor$
// Second Line
plink(6)[RIGHT][GroundFloorID]$To Ground floor-Page$
  none
  shifter(MainSwitchGroundFloor)[LIGHT]$Ground floor turn off all lights$
  pshifter(10)[WINDOW]$Window Ground floor$
// Third Line
plink(7)[RIGHT][BasementID]$To basement – Page$
  none shifter(MainSwitchBasement)[LIGHT]$Basement turn off all lights$ pshifter(9)[WINDOW]$Window
basement$
// Fourth Line
plink(15)[RIGHT][StaircaseID]$To staircase-Page$
  none
  shifter(MainSwitchStaircase)[SWITCH]$Staircase turn off all lights$
  pshifter(31)[WINDOW]$Window Cellar staircase$
//Fifth Line
shifter(ClockID)[CLOCK] $Time$
  pshifter(20)[SWITCH]$Turn off all lights$
  shifter(DateID)[DATE]$Date$
[EibPC]
//// Declarations
// Initialization of the PageIDs for the visualization wizard
GeneralWeatherPageID = 1
//Initialization of the web element IDs
WINDID = 1
WolkenID = 2
ClockID = 3
DateID = 4
MainSwitchGroundFloor = 5
```

To the function of the floor switches:

In principle, each floor switch consists of a variable which can be true or false (type b01). The variable is written by the status group addresses of the lamps and is true as soon as a light is on in a room. For example:

If (Status_Room1 == OFF or Status_Room2 == OFF or Status_Room 3 == OFF) then the variable Licht_Erdgeschoss should be set to ON, otherwise set to ON.

In the EibStudio, this is then implemented for all desired floors as follows:

```
[EibPC]
//// Declarations
//Variables
EGAllLights = 0b01

//// Functions
//Ground floor all lights
if("Lab_1_Status-0/1/18" == OFF and "Office_2_Status-0/1/19" == OFF and "Construction_3_Status-0/1/20"
== OFF and "Construction_4_Status-0/1/21" == OFF and "Floor_5_Status-0/1/22" == OFF and
"Kitchen_6_Status-0/1/23" == OFF and "Kitchen_7_Status-0/1/24" == OFF and "Storeroom 8_Status-0/1/25"
== OFF and "WCLadies_9_Status-0/1/26" == OFF) then EGAllLights = OFF else EGAllLights = ON endif
```

So in our web server the display also fits to the value of the variable, we adjust our shifter using the function webdisplay (ID, Text, Icon, State, TextStyle, [Mbutton]).

```
[EibPC]
////Functions
/// Web display Lights global Variables
// EG
if EGAllLights == ON then web display (MainSwitchGroundFloor,$On the ground floor lamps light
up$,LIGHT,BRIGHTRED,BLINKRED) else web display(MainSwitchGroundFloor,$There are no lamps on the
ground floor$,LIGHT,INACTIVE,GREEN) endif
```

In order for the shifter to perform the desired effect, which means to turn off all lights on the floor, our shifter must be assigned with a finished macro. The macro can be found under the following name:

UmschaltShifterAUSGlobal

```
[Macros]
UmschaltShifterAUSGlobal(MainSwitchGroundFloor,"EGallLights-0/5/1",EGAllLights,LIGHT)
```

If the macro is integrated, all the lights on the desired floor will turn off and the shifter's icon display will change from red to gray with green status fonts.

This procedure is carried out with new variables as well for the window contacts. The difference to the lights consists in the execution as local and in the functionality of the shifters. If you do not have windows which can be closed by a servomotor, the inclusion of a macro at the end is no longer necessary. The shifter thus serves only as a display.

```
[EibPC]
//// Declarations
//Variables
EG_WindowOpen = 0b01
////Functions
//Ground floor window
if "Office2+3WindowContact-4/1/0" or"Construction4+5WindowContact-4/1/2" or "Kitchen7+8WindowContact-4/1/4" or "WCLadies10WindowContact-4/1/6" or "Lab1WindowContact-4/1/7" or "Floor6+9WindowContact-4/1/8" then EG_WindowOpen = 1b01 endif
/// pdisplay window local variables
//EG
if EG_WindowOpen then pdisplay(10,$Ground floor - Window open$,WINDOW,BRIGHTRED,BLINKRED,StatusID) else pdisplay(10,$Window close$,WINDOW,INACTIVE,GREY,StatusID) endif
```

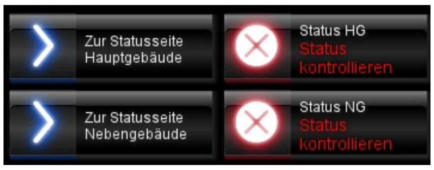
In order to be able to switch all floors or the complete main building, a group address must be created in the ETS, which switches all desired lamps. This is then inserted into the macro *ShiftShifterAUSLocal*. Now you can switch the building with one click. To adjust the icon we use *pdisplay* and represent it depending on the condition.

```
[EibPC]
//// Declarations
//Variables
AllLightsHG = 0b01
////Functions
/// pdisplay Lights local variables
//Main switch
if AllLightsHG == ON  then pdisplay(20,$Lights are ON$,SWITCH,BRIGHTRED,BLINKRED,StatusID) else pdisplay(20, $Lights are OFF$,SWITCH,INACTIVE,GREEN,StatusID)endif
```

In order to make sure that the status of lamps and windows is OK, we are linking the variables of the individual floors (EgAllLights, EG_WindowOpen, ...) with a new status variable StatusHG

```
[EibPC]
//Variablen
StatusHG = 0b01
// if-Status
if ( Stairs == ON or OGAllLights== ON or EGAlleLichter == ON or KGAllLights == ON or EG_WindowOpen == ON or KG_WindowOpen ==ON or OG_WindowOpen == ON ) then StatusHG = ON else StatusHG = OFF endif

if StatusHG == ON then pdisplay(5,$Status monitoring$,CROSSCIRCLE,BRIGHTRED,BLINKRED,GeneralWeatherPageID) endif
if StatusHG == OFF then pdisplay(3,$Status OK$,OKCIRCLE,ACTIVE,GREEN,GeneralWeatherPageID) endif
```

If this is also done for the secondary building, we will have the following representation on our main page, each with a luminous lamp of the two buildings:



So our control center would be ready. The status of our lights is displayed on the main page and we have the possibility to go to the central page with just one click. At a glance we get information about the outside temperature, cloudiness, 3-day weather trend, wind speed, date, time as well as sunrise and sunset.

What is missing now is a real look into the future. We create an additional weather page that will tell us all about the coming hours and days.

```
[Webserver]
page(WeatherPageID)[$General$,$Weather page 2$]
design $black$
header(0)
footer(0)
```

For this we need a large database of weather data. To get to this a user account is required on Weather Wunderground.

If you have successfully completed this, you must generate a key under the Key Settings tab. Here, for Developer the free version is sufficient. However, you must take into account that the free version has a limited number of inquiries. This amounts to a maximum of 10 inquiries per day and 500 per minute.

If you have generated your key, you must install it in our Macro Weather Forecast.

```
[Macros]
WetterForecast(Key,WetterStation,Germany,after(systemstart(),1300u64) or cycle(10,00))
```

For the weather station you have to register your city or search for a suitable station for your area.

If the data is all entered, you will see a list of new variables that is provided by Weather Underground after system startup in the variable query (F5). Check if your weather station also provides the data you want to implement. If everything fits, the data can now be displayed using a button or shifter.

Example:

```
[Webserver]
page(WeatherPageID)[$General$,$Weather page 2$]
design $black$
header(0)
footer(0)

button(WINDID)[WIND]$Wind in km/h$ pbutton(14)[WEATHER]$Weather$ pbutton(25)[WEATHER]$Rain
probability$ pbutton(24)[RAIN]$humidity$  pbutton(12)[TEMPERATURE]$Outer temp.: Min$ pbutton(13)
[TEMPERATURE]$Outer temp.:Max$

[EibPC]
///Wounderworld Weather
//Weather forecast today
if after(systemstart(),3000u64) or  change(WeatherForecast_Today_Wind_Max_Direction) then
pdisplay(11,$aus$ + convert(WeatherForecast_Today_Wind_Max_Direction,$
$),WIND,ACTIVE,BLINKBLUE,WeatherPageID) endif

if after(systemstart(),3000u64) or change(WeatherForecast_Today_Temperatur_Min) then
pdisplay(12,convert(WeatherForecast_Today_Temperatur_Min,$$)+$
°C$,TEMPERATURE,ACTIVE,BLINKBLUE,WeatherPageID)endif

if after(systemstart(),3000u64) or change(WeatherForecast_Today_Temperatur_Max) then
pdisplay(13,convert(WeatherForecast_Today_Temperatur_Max,$$)+$
°C$,TEMPERATURE,BRIGHTRED,BLINKRED,WeatherPageID)endif

if after(systemstart(),3000u64) or change(WeatherForecast_Today_Weather) then
pdisplay(14,convert(WeatherForecast_Today_Weather,$$),WEATHER,STATE4,GREY,WeatherPageID)endif

if after(systemstart(),3000u64) or change(WeatherForecast_Today_RainProbability) then
pdisplay(25,convert(WeatherForecast_Today_RainProbability,$
$),WEATHER,STATE6,BLINKBLUE,WeatherPageID)endif

if after(systemstart(),3000u64) or change(WeatherForecast_Today_Humidity) then
pdisplay(24,convert(WeatherForecast_Today_Humidity,$$),RAIN,ACTIVE,BLINKBLUE,WeatherPageID)endif
```

With the picture function, we add a wind indicator which shows the wind directions in Germany.

```
[Webserver]
picture(9)[DOUBLE,ZOOMGRAF]($Wind directions$,$http://wwwdyn.zdf.de/ext/weather/wind-brd-0.jpg$)
```

We also place a temperature profile of the last 24 hours with the timechart function.

```
[Webserver]
mtimechart(8)[QUAD,NOAUTOSCALE,48,-10,45]( $Outdoor temperature$,LEFTGRAF, ChartBuffer1)
```

In order for the timechart to know which values it should represent, we must make the following configurations:

timebufferconfig(ChartBufferID, MemTyp, Laenge, DataTyp)

MemTyp indicates whether the memory is written in the ring (0) or linear (1). The length of the max. Recording of the time series is indicated with length (0u16 to 65565u16). DataType represents a representative number of the time series, e.g. 0f16 for 16-bit numbers or 3% for u08 values.

Now the time series have to be "filled" with data. The function

timebufferadd(ChartBufferID, Daten)

does this task.

After the time series has been recorded for some time in the Enertex® EibPC, it must be ensured that these values are not lost even when the program is restarted or rebooted. The functions

timebufferstore(ChartBufferID)

timebufferread(ChartBufferID)

are created for this task. timebufferstore permanently stores the values of the timebuffer with the ChartBufferID into the flash memory of the Enertex® EibPC, timebufferread reads back a stored buffer.

```
button(ELEVATION)[WEATHER]$Height angle in °$ button(SunAltitude)[UP]$Altitude of sun$
button(AZIMUTH)[WEATHER]$AZIMUTH$ pbutton(23)[CLOCK]$Highest altitude of sun:$
[EibPC]
//mtimechart
Len=35040u16
Datatyp=3.3f16
MemTyp=0
timebufferconfig(ChartBuffer1, MemTyp, Len,"Outdoor temperature-14/6/3" )

if mtime(0,0) or mtime(15,0) or mtime(30,0) or mtime(45,0) then
{timebufferadd(ChartBuffer1,"Outdoor temperature-14/6/3")
} endif

if chtime(01,00,00) then {
timebufferstore(ChartBuffer1)
} endif

if systemstart() then {
timebufferread(ChartBuffer1)
} endif
```

Thus, the temperature profile would be displayed correctly. By using the timechart even better, we add the sun's course. For this we need the azimuth angle and the elevation angle of the sun. We get these two data through the functions

azimuth()

Height of elevation: elevation()

The Enertex® EibPC must know the geographic location and latitude of the location. These can be entered in the Enertex® EibStudio (see page 150). To get to your coordinates and further information, this site could help you here.

Now expand the timechart and the configuration as follows:

```
[Webserver]
mtimechart(8)[QUAD,NOAUTOSCALE,48,-10,45,-10,65]( $Outdoor temperature$,LEFTGRAF,
ChartBuffer1,$Sun altitude$,RIGHTGRAF,ChartBuffer2
[EibPC]
Len=35040u16
Datatyp=3.3f16
MemTyp=0
timebufferconfig(ChartBuffer1, MemTyp, Len,"Outdoor temperature-14/6/3" )
timebufferconfig(ChartBuffer2, MemTyp, Len,elevation() )
if mtime(0,0) or mtime(15,0) or mtime(30,0) or mtime(45,0) then {timebufferadd(ChartBuffer1,"Outdoor
temperature-14/6/3");
timebufferadd(ChartBuffer2,elevation() )
} endif
```

The two angle displays, as well as a status button for the sun's course, which indicates whether the sun is rising or falling, can now also be inserted. For this you need the azimuth angle.

The solar altitude is always at azimuth angle = 180 degrees, before the sun rise. To know when the sun was high, let us spend the time at azimuth angle 180 °. This results in the following code.

```
[Webserver]
button(ELEVATION)[WEATHER]$Height angle in °$ button(SunAltitude)[UP]$Altitude of sun$
button(AZIMUTH)[WEATHER]$AZIMUTH$ pbutton(23)[CLOCK]$Highest altitude of sun:$
[EibPC]
if change(azimuth()) then webdisplay(AZIMUTH,(convert(convert(azimuth(), 0.0 ), $$c14 ) + $ Grad $c14),
WIND, ACTIVE, GREEN) endif
if change(elevation()) then webdisplay(ELEVATION,(convert(convert(elevation(), 0.0 ), $$c14) + $ Grad$c14),
WIND, ACTIVE, GREEN) endif

if (change(elevation()) and azimuth()<175f32 and elevation()>0f32 ) then webdisplay(Altitude of sun, $is
rising$, UP, BRIGHTRED, BLINKRED) endif

if (change(elevation()) and azimuth()>175f32 and elevation()<185f32 )then webdisplay(Altitude of sun,$Sun in
zenith$, WEATHER, BRIGHTRED, BLINKRED) endif

if(change(elevation()) and azimuth()>185f32 and elevation()>0f32 )then webdisplay(Altitude of sun,$is
sinking$ , DOWN, ACTIVE, BLINKBLUE) endif

if (change(elevation()) and azimuth()>185f32 and elevation()<0f32 )then webdisplay(Altitude of sun,$Night$ ,
NIGHT, DISPLAY, GREY) endif

if azimuth()>=185f32  then pdisplay(23,settime(),CLOCK,ACTIVE,GREEN,WeatherPageID) endif
```

Next, we add a rain radar as a picture. We opted for this version, which is available for all countries. First you have to display the radar as a graphic, which gives you this view. A new image is recorded every five minutes, which we store as a variable and then displayed in the web server. Overall we need 12 variables. The configuration looks as follows:

```
[Webserver]
 picture(4)[DOUBLE,CENTERGRAF]($RainRadar$,$www.google.de$)
[EibPC]
URL1 = $$
URL2 = $$
URL3 = $$
URL4 = $$
URL5 = $$
URL6 = $$
URL7 = $$
URL8 = $$
URL9 = $$
URL10 = $$
URL11 = $$
URL12 = $$
if mtime(01,05) then picture(4,$RainRadar$,WeatherPageID,URL1) endif
if mtime(06,05) then picture(4,$RainRadar$,WeatherPageID,URL2) endif
if mtime(11,05) then picture(4,$RainRadar$,WeatherPageID,URL3) endif
if mtime(16,05) then picture(4,$RainRadar$,WeatherPageID,URL4) endif
if mtime(21,05) then picture(4,$RainRadar$,WeatherPageID,URL5) endif
if mtime(26,05) then picture(4,$RainRadar$,WeatherPageID,URL6) endif
if mtime(31,05) then picture(4,$RainRadar$,WeatherPageID,URL7) endif
if mtime(36,05) then picture(4,$RainRadar$,WeatherPageID,URL8) endif
if mtime(41,05) then picture(4,$RainRadar$,WeatherPageID,URL9) endif
if mtime(46,05) then picture(4,$RainRadar$,WeatherPageID,URL10) endif
if mtime(51,05) then picture(4,$RainRadar$,WeatherPageID,URL11) endif
if mtime(56,05) then picture(4,$RainRadar$,WeatherPageID,URL12) endif
```

With this code every five minutes the radar image is refreshed, but the URLs are still missing. For this we need the URL of the graphic, which is returned at 09.45 as follows:

The numbers marked in bold are responsible for the date, italicized and underlined indicates the time.

```
http://www.wetteronline.de/?
pid=p_radar_map&ireq=true&src=radar/vermarktung/p_radar_map/wom/2014/09/30/Intensity/
BAY/grey_flat/201409300745_BAY_Intensity.gif#1412074528673
```

Programming for experts

Performance

Cycletime

In the following paragraph program code and examples will be explained, which get deeper in the programming of the Enertex® EibPCs. By the way interesting code fragments from the macro libraries as well as the background of the procedure are explained.

Background

One of the most asked questions of the user is: How much time does the *Enertex® EibPC in fact need for the processing? In principal it depends on the size of program respectively the kind of programming and occurring events. By “validation” (p.* 121) of the program, only those parts of the program are activated per cycle that actually change. Therefore in the normal case the processing is done in less than 1 ms in more complex programs in a few ms. The time of cycle depending of the program will fluctuate. Therefore the minimal and maximal processing time is interesting while using the Enertex® EibPCs.

About the Enertex® EibStudio can after each cycle the operating system of EibPCs a break of up to 250 ms are given, e.g. to send emails to process webserver requests etc. The minimal time slices of the Linux system are about 1 ms.

To calculate the processing time of the Enertex® EibPCs, the function afterc can be used:

A countdown – counter: remaining time is counted down by Max

```
afterc(variable {Typ b01}, max{Typ u64}, remaining time {Typ u64})
```

This function is triggered as the after-function with a change of *variable* (1. argument) from OFF to on: The return value is after the specified time *max* (2. argument in ms) for one processing cycle to ON. In each cycle from the beginning of the trigger pulse of variable while the remaining time *variable* while the *remaining time* (3. argument) is updated as countdown timer. The initial value of *variable* is *max*. The change of *remaining time* is always at exactly the time at which the processing is active in one cycle. The chance of *remaining time* is thus the sum of the aforementioned deadtime plus the processing time of the preceding cycle. This allows the cycle time calculated by using systemstart triggers a afterc -timer and starts the countdown of *remaining time* e.g.

```
Max=1000000000000000u64
if afterc(systemstart(), max, remaining time) then { ..... } endif
```

Max is here chosen as large as possible to ensure that the end of the countdown is reached not possible.
With the code

```
MaxZyklusZeit=max(StoppZeit-Restzeit-PerformanceZeit,MaxZyklusZeit);
MinZyklusZeit=min(StoppZeit-Restzeit -PerformanceZeit,MinZyklusZeit);
```

Calculating of cycle times

can thus be calculated with an accuracy of about ± 1ms (time slice Linux system time) the minimum and maximum cycle time.

A special case is still taken into account: During the initialisation of the very first program run all parts of the program must be run through, then the basis of the validation later “only when neccessary” are evaluated. Therefore the first processing loop may well need serveral hundred ms, when the program reaches a memory usage of about 30. The start of the countdown counter must therefore be delayed if you do not want to take into account the initialisation of the program as a special case in the measurement of cycle times.

Therefore delaying the pulse of systemstart at startup with another timer after timer by a nesting:

```
if afterc(after(systemstart(),10000u64), Max, Restzeit) then { ... } endif
```

In total the calculation of the cycle time as follows:

And the entire code: MaxCycle is the maximum duration of a processing MinCycleTime the minimum duration.

```
// Berechnet die minimale und maximale Zyklusdauer
// der Verarbeitung. Dabei ist die Performance-Angabe im EibStudio immer
// als Offset dabei.

Max=1000000000000000u64
Restzeit=0u64
StoppZeit=Max
MaxZyklusZeit=0u64
MinZyklusZeit=Max
// Im EibStudio ggf. geändert, Defaultwert ist 20ms
PerformanceZeit=20u64

// Die erste Zyklus kann etwas länger dauern ...
if afterc(after(systemstart(),10000u64), Max, Restzeit) then {
        StoppZeit=0u64;
} endif

MaxZyklusZeit=max(StoppZeit-Restzeit-PerformanceZeit,MaxZyklusZeit);
MinZyklusZeit=min(StoppZeit-Restzeit -PerformanceZeit,MinZyklusZeit);
```

Timerstop

The timer uses the argument afterc remaining time (s.a.) for storing the elapsed time timer. The user must therefore ensure that various afterc timer use different variables to this store:

Each timer needs its “own” variable of remaining time

```
// Zähler 1
RestZeit1=0u64
RestZeit2=0u64

if afterc(systemstart(),10000u64, Restzeit1) then {
        write('1/2/3'c14,$Timer1$c14)
} endif
if afterc(systemstart(),13000u64, Restzeit2) then {
        write('1/2/3'c14,$Timer2$c14)
} endif
```

The same applies to the function

delayc(TriggerVariable {Typ b01}, Max{Typ u64}, RemaingTimest {Typ u64})

whose timer – just like delay – through every change of the TriggerVariable (1. argument) from OFF to ON is triggered again. Again that for the rest of time each with its own variable must be used otherwise disrupt the timer each other.

When the timer expires the value of 3. arguments (remaining time) to 0u64, upon triggering of the timer it is set to the value of Max. If the remaining time is changed during an active phase by the user so the expiration time of the timer ist changed.

The variable remaining time can be modified by the program and thus influenses the course of the timer

```
RestZeit1=0u64
 if afterc(systemstart(),10000u64, remainingtime1) then {
        write('1/2/3'c14,$Timer1$c14)
} endif
if remainingtime1>1000u64 then remainingtime1=500u64 endif
remainingtime2=0u64
if delayc(systemstart(),13000u64, remainingtime2) then {
        write('1/2/3'c14,$Timer2$c14)
} endif
```

In the above example only the afterc timer is changed the rest of the time variable delayc timer remains unchanged.

With this a timer can now be stopped if there is no longer need for e.g. the end of the process and the associated action of the if-statement.

Timer

termination conditions=>

```
MyTrigger=OFF
remainingtime1=0u64
 if afterc(MyTrigger,10000u64, remainingtime1) then {
      write('1/2/3'c14,$Timer1$c14)
} endif
if MyTrigger== OFF then remainingtime1=0u64 endif
```

If in the example *MyTrigger* switches to ON the timer is started, if *MyTrigger* switches to OFF before the expiry of the time, the timer is stopped by setting *remainingtime1=0u64 .* The then-branch is not executed.

If you want to stop the timer before but running the then-branch it must *RestZeit1=1u64* be set. In this case the execution is performed in the next processing cycle.

Queue

The event-based processing in EibPC requires the programming of socalled “state machine”. The (abstract) basic principle of a “state machine” is that programming is not performed sequentially but that the software assumes a certain state depending on events.

When exchanging data with another device e.g. via TCP/IP telegrams, you can define the following states:

1. Receive data from the other participants
2. Send data to the other participants
3. Cache data of the other participants
4. Evaluate the data of the other participants
5. Perform various KNX actions on the bus

Each of these conditions is at least in principle independently of the other i.e. the EibPC has to accept data while e.g. KNX telegrams arrive. In addition various states can “triggering” each other respectively the arrival of a KNX telegram encourage the data processing.

On http://knx-user-forum.de/special/Enertex EibPC and Command Fusion is a complete description of a “state machine” located, which processed a queue.

Command Fusion

In the supportforum following request originated. The users want to switch to the macro `At_Sunset_Capped_withRelease` at sunset respectively at the latest at a certain time in a macro group address.

In the same way the macro is used: At`_Sunset_Capped_withRelease` at sunset.

Presence state machine

```
Bei_Sonnenuntergang_Gedeckelt_mitFreigabe(Sued,FreigabeVar,"Licht Wohnen-2/2/3",AUS,22060000,22,31,00)
Bei_Sonnenaufgang_Gedeckelt_mitFreigabe(Sonnenaufgang1,FreigabeVar,"Rolläden Ost-5/2/0",RAUF,7200000,07,28,00)
```

The macros are parameterized with the release-variable *FreigabeVar* .

For this purpose the release is divided into the following observation periods:

- Day mode: Sunrise to sunset
- Early mode: Period after 0:00 clock and before sunrise
- Late mode: After sunset and not after 0:00 clock

The user presses a group address "Presence-8/1/1" (Typ b01, ON==present).

The release-variable *FreigabeVar* should be switched dependent on the following states.

State 1:

Description:

States of realisation

Early mode

Target:

It should not be run through a macro regardless of whether "Presence-8/1/1" is ON or OFF.

FreigabeVar **has to be set to OFF respectively has to remain in the (OFF-)-condition.**

State 2:

Description:

Day mode

Target:

If "Presence-8/1/1" is set to ON, *FreigabeVar* has to be set to ON, the macros will be activated, if "Presence-8/1/1" is set to OFF. *FreigabeVar* should set to OFF the macros will be deactivated.

If the group address "Presence-8/1/1" is changed (bus telegram/user) should the *FreigabeVar* immediately accept its value.

State 3:

Description:

Late mode

Target:

If"Anwesenheit-8/1/1" is set to ON, *ReleaseVar* should be set to ON, the macros so are activated, if "Presence-8/1/1" is set to OFF. *FreigabeVar* should be set to OFF the macros will deactivated.

This can now directly be converted into a program:

Direct picture of the “state machine” in EibPC

```
FreigabeVar=AUS
TFrueh=chtime(00,00,01) and !chtime(12,00,00)
// Zustand 1: Frühmodus
if TFrueh and !sun() then FreigabeVar=AUS endif
// Zustand 2: TagModus
if sun() and change("Anwesenheit-8/1/1") then FreigabeVar="Anwesenheit-8/1/1" endif
// Zustand3   Spätmodus
if !TFrueh and !sun() then FreigabeVar="Anwesenheit-8/1/1" endif
```

Especially here is the use of variable *TFrueh.* This is realized via a link from one timer at midnight and a second at noon. This is ensures that *TFrueh* is set at 0:00 clock to ON and from the afternoon to OFF.

TFrueh : morning (from 0:00 clock)

Presence simulation

The macro collection includes macros for presence simulation. The basis concept of these macros is to be explained in the following.

With a presence simulation two states can be differentiated.

1. Record

During this phase selected group addresses are recorded before. Group telegrams are often triggered by residents e.g. upon actuation of switches. The recording is usually performed over a 2-week interval in which the recording continuously overwrites the old values.

2. Play

If the resident of a property e.g. goes on vacation the group telegrams will now be triggered by the Enertex® EibPC so that outsiders will have the impression of presence of the residents. There the play has to take place same day and time, so that e.g. the recording of Saturday is played on a Saturday again too.

As above mentioned conditions the following is needed:

- Determination of raw data of the telegrams
- Determination of sending group address
- Determination of telegrams arrival time
- Recording of data
- Sending of raw data time-shifted to the bus

Determination of sending group address

For this task you need the function readrawknx:

Raw information from the bus

readrawknx(*Sim_Control {u08}, Sim_Sender{u08}, Sim_GA{u08}, Sim_IsGA{b01}, Sim_RoutingCnt {u08}, Sim_Len{u08}, Sim_Data{c1400}*)

If any KNX telegram is observed on the bus the function readrawknx updated its arguments. In this case the arguments of the function are "filled" with data. The received user data are then copied to the argument *Sim_Data ,* the amount of data (bit length) can be queried with the variable *Sim_Len .*

Upon receipt of a telegram the argument *Sim_IsGA* is set accordingly, i.e. is it an ordinary group telegram so this argument is set by readrawknx to ON and *Sim_GA* contains the address itself. The function readrawknx can be linked to event in order to process the arrival of a telegram

With the selected definitions

```
Sim_GA=0u16
Sim_IsGa=OFF
Sim_RoutingCnt=0
Sim_Len=0
Sim_Data=$$c4000
Recorder=$$c4000
Timestamp=$$c4000
```

you can now process the arrival of a telegram as follows:

```
if event(readrawknx(Sim_Kontroll,Sim_Sender,Sim_GA,Sim_IsGa,Sim_RoutingCnt,Sim_Len,Sim_Data))  then ....
```

Identy group addresses

It should be noted that the group address *Sim_GA* is calculated as 16-bit value. In order to compare this address with the usual spelling is the function getaddress at your disposal. In the following example

```
MeinGA=getaddress("Licht-1/2/3")
```

there is now MeinGA the 16-bit value which represents the group address and how this is also copied *Sim_GA.* Now it is determined out of which group address the arrived telegram has been sent.

With the help of variables

```
Sim_GA=OFF
```

should the recording of an incoming message be triggered as follows. For each recorded group address are if-queries deposited. *Sim_GA* is determined as above mentioned by readrawknx .

Code-part 1

For each group address that is to be recorded an if-query

```
if Sim_GA==getaddress("Heizvorlauf-0/0/1") then Sim_MyGA=ON else Sim_MyGA=OFF endif
if Sim_GA==getaddress("Temperatur-3/5/0") then Sim_MyGA=ON else Sim_MyGA=OFF endif
if Sim_GA==getaddress("Licht-1/0/29"u16) then Sim_MyGA=ON else Sim_MyGA=OFF endif
```

The both modes Record/Play are realised via

```
Sim_Play=OFF
```

At *Sim_Play* = ON the existing recording should be played and at OFF the recording should be started.

Determination of raw data of the telegrams

Now it is necessary how the raw data of the telegrams on the bus can be determined. For this purpose

Code-part 2

```
if event(readrawknx(Sim_Kontroll,Sim_Sender,Sim_GA,Sim_IsGa,Sim_RoutingCnt,Sim_Len,Sim_Data)) and
Sim_Len!=0 and Sim_IsGa and !Sim_Play then {
  if !Sim_MyGA then Sim_Next=OFF endif;
  if Sim_MyGA then {
     if Sim_Len==1 then Sim_RawData=convert(stringcast(Sim_Data,0u08,1u16) and 0x7F,0u32) endif;
     if Sim_Len==2 then Sim_RawData=convert(stringcast(Sim_Data,0u08,2u16),0u32) endif;
     // Byte Order has to be considered
      if Sim_Len==3 then Sim_RawData=convert(stringcast(Sim_Data,0u08,2u16),0u32)*256u32
+convert(stringcast(Sim_Data,0u08,3u16),0u32) endif;
     if Sim_Len==5 then Sim_RawData=convert(stringcast(Sim_Data,0u08,2u16),0u32)*16777216u32
+convert(stringcast(Sim_Data,0u08,3u16),0u32)*65536u32+convert(stringcast(Sim_Data,0u08,4u16),0u32)*2
56u32+convert(stringcast(Sim_Data,0u08,5u16),0u32)  endif;
     Sim_Next=ON;
  } endif;
}endif
```

Sim_RawData are raw data in u32 format. If only one bit has been sent, so 31 bits are "unused". Die incoming data are written from readrawknx in *Sim_Data* string variable. These are basically regarded as raw data and then be converted into u32 bit value.The arrangement of data in 4 bytes (32bits) unifies the saving of the telegrams data and simplifies the method (how to show yet).

For processing these raw data on string *Sim_RawData* now the single bytes have to be interpreted as 1-byte integer values. This happens with the help of function stringcast.

X=stringcast(*src{cxxxx}, dest, Pos{u16}*)

This function start to look at the bytes on string *src* from the byte-position *Pos. dest* on there gives the target data type conversion on, which specifies the number of bytes and defines the conversion to the result *X*. Based on Picture 1 it is explained: The graphic shows the string as byte arrangement. At position 3{u16} the value is hexadecimal 0x74.

String as bytearray

			3u16	4u16	5u16	6u16				
0xXX	0xXX	0xXX	0x74	0xA0	0xE1	0x01	0xXX	0xXX	0xXX	0xX

Picture 1: String src as arrayfield.

A statement Z1=stringcast(*src,0,3u16*) will define a variable Z1 from the data type u08 (argument „0") . The value is obtained from *src* (Picture 1) on position 3{u16} and is thus in this case 0x74 (dezcmal 116). A statement Z2=stringcast(*src,10u32,3u16*) however defines die number 0x74a0e101 (decimal 1956700417). This number of bytes, which are extracted from the string is obtained by the argument 10u32: The data type u32 is 32 bits long and consists of 4 bytes. The value 10 of „10u32" itself is ignored, here. The order of bytes remains unchanged in the stringcast function.

Back to the example: *Sim_RawData* contains the data of the incoming telegrams in the first 4 bytes. The order of the bytes on the bus is different to the order of the bytes of the Linuxsystem of the Enertex® EibPCs. In order to use these data the byte order has to be reversed i.e. the last bit has to be in the first place etc. This rearrangement is realised by the help of multiplication by 256 and 65536 and 16777216.

The present processing of raw data is limited to max. 32 bit telegrams. Longer data telegrams can not be recorded, on the other hand bytes will be surely wasted by recording 1 bit elements, because all telegrams are treated equally. Nevertheless this approach to some extent an optimal compromise because the processing is easier later.

The code-part 2 calculates now the data of the u32 – variable *Sim_RawData* .

Determination of telegrams arrival time

The points of transmission time of the telegrams have to be determined relative, because a previously recorded simulation relative (time-shifted) to the starting point of the simulation have to take place.

Code-part 3

The relative timestamp...

```
// Die Uhr wird gestartet (Countdowntimer)
if Sim_Start then {
            Position=0u16;
            Sim_MyGA=OFF;
            if !Sim_Play then {
                  stringset(TimeStamp,convert(Interval,0u32),Position);
            } endif;
} endif

// Die Uhr wird gestoppt nach dem Intervall
if afterc(Sim_Start,Interval,Timer) then {
            Position=0u16;
} endif
```

... is saved only in 32 bits wide

When changing from *Sim_Start* to ON the first if-statement initialises the string timestamp. In addtion a afterc-timer (a.m.) is started. *Interval* determines how the duration of the recording is, e.g. 1 day = 86400000ms. This function updates at each loop run as a countdown-timer die variable *Timer*. This function relatively counts down from the starting point the elapsed time in ms. In the string *Timestamp* the start is written on position zero but in order to simplify the maximum recording duration is limited on 32 bit (49 days).

Recording of data

if with code-part 1 is set, that the incoming GA is to be recorded (*Sim_MyGA* at ON), thus the data in the string *Data* and die group address in the string *Recoder* are saved. As the group addresses are only 16 bits wide, the bit length can saved in the same array at the same time. For storing the raw data in one string stringset is used.

Abpeichern
- Data
- Recoder
- Timestamp

stringset(*dest{cxxxx}, src, pos{u16}*)

This function writes into the target string *dest* on its position of location *Pos* the (binary) contents of *src*.

Code-part 4

```
if  !Sim_Play and Sim_Next then {
            stringset(TimeStamp,convert(Timer,0u32),Postion);
            //ggf. alten Zeitstempel löschen
            stringset(TimeStamp,convert(Timer,0u32),Postion+4u16);
            // GA abspeichern
            stringset(Recorder,Sim_GA,Postion);
            // Die Länge speichern
            stringset(Recorder,Sim_GA,Postion+2u16);
            // Den Wert speichern
            stringset(Data,Sim_RawData,Postion);
            Sim_MyGA=OFF;
            Sim_Next=OFF;
            Sim_GA=65365u16;
            Postion=Postion+4u16;
            // Überlauf?
             if Postion>capacity(TimeStamp) then Sim_Start=OFF endif;
} endif
```

The fact that the timestamp, data- and group addresses are stored 32 bits wide, the position of a telegram is equal in thess strings, which simplifies the processing. In a c1400 string are recorded up to 350 telegrams. With the help of 65k strings are recorded up to 16341 telegrams. In the present case the telegram memory was with c4000 determinated by 1000. The function capacity shows how many bytes the string can maximum save.

After the preset time the recoding will restart in code-part 3. The first stored values are overwritten , the old values are preserved, which can disturb. Therefore in the above code-part 4 a possibly existing timestamp out of a previous recording is deleted.

Playing of a recording

The playing of a recording is relatively simple. For this purpose there are only the group address and the raw data is "loaded" (strings) and these are written to the bus. In this case the timer from code-part 3 has to be restarted. The present countdown time on *Timer* is compared with the timestamp in *Timestamp* and initialize a letter when falling below of the time:

Code-part 5

Playing

```
if Sim_Play and Timer<convert(stringcast(TimeStamp,1u32,Position),0u64) then {
    SimGA_Out=stringcast(Recorder,0u16,Position);
    SimGA_Len=stringcast(Recorder,0,Position+2u16);
    SimGA_Val=stringcast(Data,0u32,Position);
    if SimGA_Len==1 then write(address(SimGA_Out),convert(SimGA_Val,EIN)) endif;
    if SimGA_Len==2 then write(address(SimGA_Out),convert(SimGA_Val,0)) endif;
    if SimGA_Len==3 then write(address(SimGA_Out),convert(SimGA_Val,0u16)) endif;
    if SimGA_Len==4 then write(address(SimGA_Out),SimGA_Val) endif;
    Position=Position+4u16;
} endif
```

The data types due to the use of the raw data need not be observed. Only the length of telegrams is to be evaluated so that they correspond to those of the recording.

The macro-library EnertexPresence.lib is realised in this manner.

In the library the recording will be broken down into smaller day intervals and assembled later when playing. The recording then starts each to the next day interval.

Useful

Encoding at C14

The KNX™ standard requires that devices with 14-byte messages („c14" types) have to implement only the ASCII code. This is allowed in an extension optional ISO8859-1, which itself exists only from 1-byte character (comp. http://de.wikipedia.org/wiki/ISO_8859-1).

The specification of the string in the Enertex® EibStudio follows the coding of the operating system, so that e.g. Windows XP, German version, the ° character (MASCULINE ORDINAL INDICATOR) is handed over as a one-byte 0xBA to the parser.

If you work under OSX, 10.6.8, with the Enertex® EibStudio, so is the system encoding UTF-8. In this case the ° character is a 2-bytes character (0xC2BA) and the EibParser throws out an error message that this is not allowed.

Accordingly the EibParser compiled in the WinXP version of Enertex® EibStudio

```
st=$10 °C$c14
```

without problems, but not on OSX. In the first case the °-character it is a permissible 1-byte character, in the second case an impermissible 2-bytes character. The offset 0xC2 is not constant: e.g. "ä" appears in 8859-15 the 8-bit value 0xE4 on the other hand in UTF-8 the 16-bit value 0xc3a4.

In general it can be stated: Windows-XP and Win7 users (in the German version) can use that °-character and Umlauts easily, OSX or Linux user not respectively is here crucial coding system for the application.

Windows, OSX or Linux

Will you still use an extended ASCII set of IS8859-15, must be used for UTF-8 systems with the following construct.

```
string=convert(encode($Hällo °C$,$utf-8$c14,$iso8859-15$c14),$$c14)
```

The problem of encoding occurs only in the use of special characters in the c14 string, because this has to be represented as a 1-byte ASCII. All other strings (cX with X from 1 to 65534) are automatically converted from Enertex® EibStudio into UTF-8 encoding and platform independent.

Therefore the instruction time is not platform independent e.g. would certainly lead under Windows XP at an invalid character on the bus.

To create a code for c14 strings platform independent, must therefore be resorted to the compiler directive #ifdef and the predefined constants OSX, WIN and LINUX:

```
#ifdef OSX
string=convert(encode($Hällo °C$,$utf-8$c14,$iso8859-15$c14),$$c14)
#endif
#ifdef WIN
string=$Hällo °C$c14
#endif
```

Now EibStudio is not able to display the telegram e.g. in OSX because "ä" in UTF-8 would be an another character code as on the KNX™ bus. On bus the Enertex® EibPC sends the correct code, which can easily be shown based on the raw data.

String concatenation with different length

In string processing is often resorted to the concatenation i.e. the “concateantion” of strings.

Thus e.g. in the code

Concatenation of same size strings

```
s1=$Hallo $c1000
s2=$Welt$c1000
s3=s1+s2
```

the string *s3* will have the content *Hello World.* The data type control in the EibParser ensures that *s3* is of type c1000. The EibParser ensures that the concatenation can record the size of the longest string , in the present case are for *s1+s2* 1000 Bytes. *s3* are assigned as a result of the concatenation *s1+s2* 1000 Bytes.

If 950 bytes of data already available in *s2* and in *s1* in turn is 90 bytes then 40 bytes are in the concatenation “lost” because only *s3* can max. hold 1000 Bytes.

The following code is to be sonsidered as well:

```
s1=$Hallo $c1000
s2=$Welt$c1000
s3=$$c2000
if htime(10,00,00) then s3=s1+s2 endif
```

Concatenation of two strings and allocation on a bigger string

Again the concatenation is *s1+s2* the length of 1000 Bytes, as they are composed out of two 1000 byte-strings. The assignment to the 2000 bytes long *s3* ovvurs only after the concatenation. However as already the concatenation operation has limited the length up to 1000 bytes here bytes can get “lost”.

This is in the following code different:

Concatenation of two strings and allocation to a smaller string

```
s1=$Hallo $c1000
s2=$Welt$c1000
s3=$$c200
if htime(10,00,00) then s3=s1+s2 endif
```

Again the concatenation is *s1+s2* the length of 1000 bytes, as they are composed out of two 1000 byte strings. The assignment of the 200 bytes long *s3* occurs only as a result of the concatenation: First the concatenation operation *s1+s2* limited the length up to 1000 bytes, allocating limited to *s3* its length to 200 bytes, so assuming, where 800 bytes of data „lost”.

If the concatenation *s1+s2* in no case lose data, a dummy variable has to be introduced:

Prevent data loss:

```
s1=$Hallo $c1000
s2=$Welt$c1000
s3=$$c2000
dummy=$$c2000
if htime(10,00,00) then s3=s1+s2+dummy endif
```

This ensures that *s1+s2+dummy* 2000 bytes can hold as a result. Therefore the concatenation will deliver 2000 bytes to *s3* as a result.

FTP Data streams

Four data streams

With the help of configurable FTP transfers any ASCII ("plaintext") files can be written to an external FTP server. The maximum file size is 64 kB.

For this purpose, four different handles (= ID number of transfers) are created, which - by itself buffered queue - create these files on the server. The files are via timeout earlier (and then fewer bytes if necessary) written or initiated by flushftp () by the user. The file names are assigned automatically by the firmware by date and time.

In the following, the procedure must be described in detail when creating and applying these FTP outsourcing.

First, the stream and its handle must be defined in the program. For this purpose, the function

ftpconfig(server,user,password,path,timeout)

is needed (P. 283). A handle refers to a unique number (ID) for a transfer and is about tantamount to a name.

Configuration of the transfer

The first three arguments are used to configure the Tranfers: IP address, user name and password, then follows the target directory on the server and a timeout parameter. Use this statement to reserve a 64 Kbyte buffer in Enertex ® EibPC. The transfer of the buffer occurs when either the buffer was completely filled (more on this below) or the number *timeout* seconds have elapsed since the last transfer.

```
// ServerDaten
server=$ftp.enertex.de$
user=$enertex$
password=$enertex$
path=$KNX/Telegramme$

// Timeout in Sekunden
timeout=900u32

// FTP Queue anlegen
// Wenn Handle ungleich Null, dann ist das fehlerfrei gelungen
Handle=ftpConfig(server,user,password,path,timeout)
```

Several strings are summarised in a line of text

During operation, the data must now be written into the buffer. Therefore

sendftp(handle,data1,[data2],[...])

is needed. The function allows arbitrary strings as arguments, because the target file is also just a text file.Any data in the form of numerical values must be converted using the Convert function. In this case an LF CR (newline suitable for Windows) is inserted at the end of the data transmission of sendftp. All call to sendftp can pass more than one substring, but no more than 1400 bytes assume total. Accordingly, the maximum length is 1400 bytes:

```
// Daten in die Queue schreiben
Data1=$Daten Nr. $
Data2=$ des internen Zählers - $
Nr=0u16
status=3
// minütlich werden die Daten Data1 in den internen Buffer geschrieben
// nach 15 Minuten (timeout) werden die Daten am FTP-Server ausgelagert
if stime(60) then {
  status=sendftp(Handle, Data1,convert(Nr,$$),Data2,convert(settime(),$$));
  Nr=Nr+1u16;
} endif
```

If the variable *status* to 1, writing to the buffer of the transfer was successful. However, this has nothing to do with the fact that the data have arrived on the FTP server.

For this, the status of the FTP data stream must be queried.

Therefore is

ftpstate(handle)

available.

With

```
ftpstatus=ftpstate(Handle)
if ftpstatus==5 then write('1/2/3'c14,$FTP Overflow$c14) endif
```

the following status can be obtained:

- Configures / error-free = 0
- the last transmission was error-free = 1
- the FTP server was not reachable = 2
- the password / user is not allowed = 3
- The target directory does not exist and it could not be created = 4
- The queue has an overflow (= 5), this can only occur if the transmission was not successful before.
- Handle is not defined = 6

If it is for the processing of importance to determine the level of the stream buffer, this can be learned with the aid of

ftpbuffer(handle)

ftptimeout(handle).

The first function returns the number of unused bytes in the buffer, the second function describes the elapsed time since the last transfer.

```
if mtime(0,0) then {
        //Füllstand des FTP Buffers
        buffer=ftpbuffer(Handle)+1u16
        //Bereits verstrichene Zeit seit dem letzten Transfer in Sekunden.
        timeout=ftptimeout(Handle)
} endif
```

In addition to the automatic writing of the data to the FTP server, the buffer can also be manually emptied (“flushed”) with the use of the function

flushftp(handle)

while you are uploading the data to the FTP server “manually”.

```
// Daten "manuell" flushen (nur dann wird die Übertragung aktiv)
// täglich um 00:00:00 Uhr
if htime(00,00,00) then {
  status=flushftp(Handle);
} endif
```

If no manual flushing or writing is done, the Enertex® EibPC is going to initiate the transfer independently. The transfer takes place when the buffer is full or the configured timeout elapsed (in seconds) since the last transfer.

Use of own Html code and graphics on the Web server

Weboutput

With the weboutput field of the web server, the user can show his own HTML code on the visu (vgl. S. Fehler: Referenz nicht gefunden / 285). In the output field a simple text can be represented, but it is also possible to represent dynamically a complex HTML code.

Incorrect or invalid HTML code in weboutput may interfere with the page layout. Such errors are not corrected by the free support. Please work here with tools as shown on the link http://www.quackit.com/html/online-html-editor/ to test the HTML code.

Thereto you have to define the output field in the web server:

Defintion of two output fields

Weboutputs are always global

```
[WebServer]
page (2) [$Haus$,$Energie$]
weboutput(Out1)[QUAD,ICON] weboutput(Out2)[QUAD,NOICON]
[EibPC]
Out1=2
Out2=3
```

You can note that the weboutput field can only be set globally. The element can be displayed with or without an icon (ICON or NOICON). The width is set to 2 unit width, the height can be set single (SINGLE), double (DOUBLE) or quadruple (QUAD).

The restriction to global elements arises from the possibility that the Weboutput-box can absorb 65 Kbytes of data. For 40 global elements, which make 2 MB, you have to keep free space in RAM for these items.

With the function

weboutput(ID,Data)

Maximal 65000 Signs

is the data written of the field. In this case is Data a string with a maximum length of 65534 bytes (type c65534). A special feature is that this string can be a valid html code. This makes it possible to dynamic formatting and display.

We are going to describe the both at the outset specified fields so that a website as in Abbildung 2 is created:

Abbildung 2: Gewünschte Ausgabe

For the creation of the actual HTML code, please refer to http://de.selfhtml.org. The Html code can be preset using the website as the following:

```
if systemstart() then {
        weboutput(Out1,$<h4>Berechnung der <i>Energieeffizienz</i></h4> <ul style="list-style-type:disc"> <li>Obergescho&szlig;:  10 kWh </li> <li> Untergescho&szlig;: 10.3 kWh </li> <li> Erdscho&szlig;:      2.3 kWh </li> </ul> ________________________ <br>        Summe: 22.6 kWh  $c10000)
} endif
```

You can note, that the code inside the $-Sign can't be wrapped. In the development it's recommended to create and test the HTML code seperately.

With the help of an other dependency as the if systemstart() the text and the formatting can be changed the whole time even during the term of the program.

The second weboutput field should also have its own graphic. At first a PNG or JPG data has to be uploaded at the Enertex® EibPC. This could be done ex ante with the Enertex® EibStudio, which is described on P. 200. The path of the graphic for the weboutput is `/upload/` + data name. Thereby the graph and some text and the HTML formatting will be initialize with the following statement:

HTML Code...

can also be adjusted dynamically

```
if systemstart() then {
   weboutput(Out2,$  <table border="1"><tr> <td class="oben"> <img src="/upload/effb.jpg" alt="Bild fehl"></td> <td class="mittig"><b>Das ist ziemlich wenig! </b><br> Super Sache, wenn wenig Energie im <br> Haus verbraucht wird. <br>Freut sich der Geldbeutel und der <br> <b> Hausbesitzer</b>!</td></tr></table>$)
} endif
```

The output can be made depended of current values e.g. meter readings of an KNX device, which is shown in the following.

An engery meter sends via the GA "Energy-2/3/5“,"Energy-2/3/6“ "Energy-2/3/7“ of type u32 the consumption in Wh. We first define the variables in kWh as a string (c.1400).

Convert the consumption in kWh

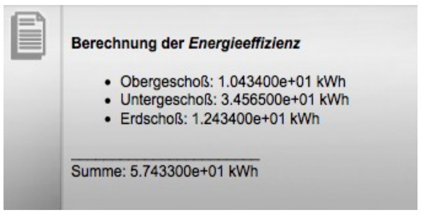
```
ConsumptionOG_kWh=convert(convert("Energy-2/3/5",1f32)/1000f32,$$)
ConsumptionEG_kWh=convert(convert("Energy-2/3/6"1f32)/1000f32,$$)
ConsumptionUG_kWh=convert(convert("Energy-2/3/7",1f32)/1000f32,$$)
Sum_kWh= convert(convert("Energy-2/3/7“+"Energy-2/3/6"+"Energy-2/3/5“,1f32)/1000f32,$$)
```

At twelve o'clock the values should be displayed daily:

and on the webserver as a string – link (note: the “+” sign)

```
if htime(12,0,0)  then {
            weboutput(Out1,$<h4>Berechnung der <i>Energieeffizienz</i></h4> <ul style="list-style-type:disc"> <li>Obergescho&szlig;: $+ VerbrauchOG_kWh +$ kWh</li> <li> Untergescho&szlig;: $+VerbrauchUG_kWh+$ kWh </li> <li> Erdscho&szlig;: $ +VerbrauchEG_kWh+$ kWh </li> </ul> _______________________ <br>       Summe:$ +Summe_kWh+$ kWh  $c10000)
} endif
```

Depending on the actual transmitted values, the display will be on the web server (compare with Abbildung 3):



Berechnung der *Energieeffizienz*

- Obergeschoß: 1.043400e+01 kWh
- Untergeschoß: 3.456500e+01 kWh
- Erdschoß: 1.243400e+01 kWh

____
Summe: 5.743300e+01 kWh

Abbildung 3: Dynamische Ausgabe

In the code section the HTML string is made of substrings by the use of concatenation (“+”-Signs). It is important to ensure, that the concatention produces the matching string length. The function weboutput can transfer up to 65564 bytes to weboutput-element. The concatenation consists only of $$ (=c1400) and one c10000 string. The string concatenation reserves for the result the number of bytes, such as the “longest” string-argument is predented.In this case it makes 10.000 bytes, which are given through the one c10000 string in the code (shown above).

At this point it should be said, that special signs could be composed of multiple bytes, as already described on P. 108. The concatenation could bring theoretically more than 10000 bytes as a result, if the strings exhaust the full length of their definition. In this case the “overlaying” signs cannot respect the concatenation function and accordingly the concatenation function is going to cut the signs of the string before copying into the result. It is up to the User if he respects it. (compare with p. 109).

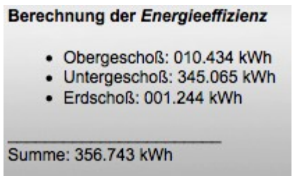
For reasons of clarity the HTML code can although be outsourced in a own data. To this you work with the #include-directive (compare with p. 128).

Back to the example:

The most users don't like the output representation of the exponential floating-point representation. Therefore the representation of values should be more readable with the function stringformat. This function changes a number into a string - whereupon leading zeros and the indicated accuracy and floating-point representation can be parameterized.

Arguments:
1. Value (her f32)
2. Conversion of F32 in floating-point represenation: 4
3. Representaion with leading zeros: 4
4. Maximum length: 8
5. Accuracy: 1 point

```
VerbrauchOG_kWh=stringformat(convert("Energie-2/3/5",1f32)/1000f32,4,4,8,1)
VerbrauchEG_kWh=stringformat(convert("Energie-2/3/6",1f32)/1000f32,4,4,8,1)
VerbrauchUG_kWh=stringformat(convert("Energie-2/3/7" ,1f32)/1000f32, 4,4,8,1)
Summe_kWh=stringformat(convert("Energie-2/3/5"+"Energie-2/3/5"+"Energie-
2/3/5",1f32)/1000f32,4,4,8,1)
```


Berechnung der *Energieeffizienz*

- Obergeschoß: 010.434 kWh
- Untergeschoß: 345.065 kWh
- Erdschoß: 001.244 kWh

Summe: 356.743 kWh

Abbildung 4: Formatierte Ausgabe

Picture, Header und Footer

You can use the possibility of uploading own pictures onto the Enertex® EibP for your own creation of your ad for header, footer, picture graphics. At first you have to upload the graphics (as described on p. 200) onto the Enertex® EibPC. In the following this has been happened with *untergang.png and aufgang.png* and *sun.png*.

Picture 5: Use your own graphics

Instead of the external links in header and footer (comp. p 287) you can now have a reference to the praphics that were saved in the Enertex® EibPC link. Instead of an external link, enter the graphic stored as a path `/upload/` + filename in the configuration:

```
[WebServer]
page (2) [$Haus$,$Sonnenstand$]
header(2) $/upload/untergang.png$
weboutput(Out1)[DOUBLE,NOICON] picture(0)[DOUBLE,CENTERGRAF] ($Mein Bild$,
$/upload/sun.png$)
footer(2) $/upload/aufgang.png$

[EibPC]
Out1=2
Sonnenaufgang=stringformat(sunrisehour(),0,3,2,2)+$:$+stringformat(sunriseminute(),0,3,2,2)
Sonnenuntergang=stringformat(sunsethour(),0,3,2,2)+$:$+stringformat(sunsetminute(),0,3,2,2)

if systemstart() or htime(0,0,0) then weboutput(Out1,$<h4><i>Sonne heute: $
+convert(setdate(),$$)+$ </i></h4> <ul style="list-style-type:disc"> <li> Sonnenaufgang :$
+Sonnenaufgang+$</li> <li>Sonnenuntergang $+Sonnenuntergang+$</li>$ ) endif
```

Sunrise and sunset formatted appropriately as as string

Now you can design your own website as shown in the above Picture 5 .

Visualisation of time series

With the EibPC (from V3.000)time series can be easily added, pemanently stored and visualised. For this purpose a (global) diagram element art (p. 279) is availabe on the webserver.

Advanced Mtimechart (EXT)

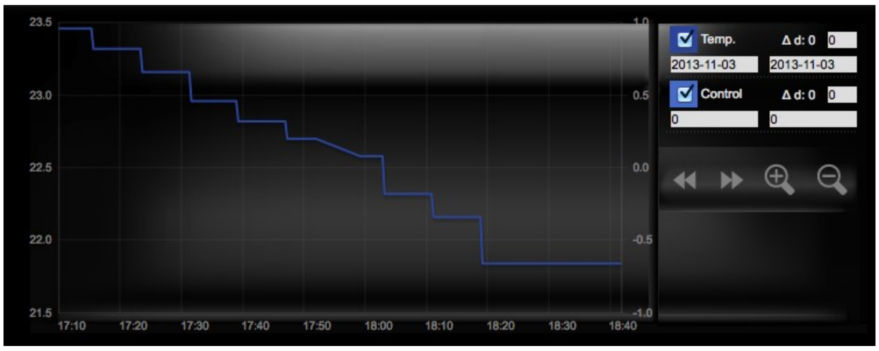
As given in the definition on p. 279, diagrams can be labeled with two axes and the scaling can be chosen automatically or by value range specification. There are diagrams in sizes DOUBLE (2,2), TRIPLE (3,2), QUAD (4,2),LONG (4,4) specified with the unit heights and widths as shown in the parentheses.

In addition there are the EXT-diagrams EXTDOUBLE, EXTRIPLE, EXTLONG, they are enable the advanced functionality with a simpler parametrization. With these we want to start. We first define a webelement mtimechart with sixfold width, where both axes are displayed and are scaled automatically.

```
[WebServer]
page(1)[$Log$,$Room1$]
mtimechart(R1_ID)[EXTLONG,NOAUTOSCALE,192,17,30,0,100] ( $Temp$,LEFTGRAF, ChartBuffer0,
$Control$,RIGHTGRAF,ChartBuffer1 )
```

In the definition is set beside the specification, that no automatic scaling of the the axes is desired, that the diagram max. 192 pairs of value (value/time) represents. In this case 256 pairs are possible. The first y-axis represents values in the range from 17 to 30, the second from 0 to 100. The first buffer (buffer0) is assigned to the left axis and the second buffer (buffer1) to right axis.

Mtimechart diagram with extended applications for moving, outfading , zooming from graphes



Picture 6: Timechart webelement EXTLONG

Comparing time series: Field Δd

Picture 6 shows this mtimechart-diagram, which displays two of the four possible time series. The user can scroll left and right in the time series (buttons << respectively >>), as well as zooming. Both operations are applied to all graphs of a mtimechart. With the field Δd can be an offset in days for the displayed time series individually adjusted, in order e.g. to compare the consumption data from two time series during the year. These control functions are part the EXT-diagrams itself so no further expenditure occurs by programming the webelement. Only the time series (timebuffer) have to be configured and recorded.

Now consider the following definition (comp. 254):

timebufferconfig(*ChartBufferID, MemTyp, Length, DataTyp*)

Adjust the memory size to the needs

This function allows up to 256 (ID 0 to 255) various buffers for recording time series. *MemTyp* indicates whether the memory in the ring (0) or linear (1) is decribed (more on this below). The length of the max. recording of time series is specified with *Length* (0u16 to 65565u16). Per stored value (see below) time series requires 12 bytes regardless of the stored *DataTyp*. It is recommendable to adjust the size of the memory to the real needs: A time series with the max. length occupies 780 kB RAM.

DataTyp displays a representative number of time series e.g. 0f16 for 16-bits numbers or 3% for u08 values. The number itself is not further processed and serves the compiler to win only the type information. We use the timebuffer with ID 0 for recording the temperature group address 1/2/3 (type f16) and the ID 1 for the adjusting size of the heat-controller 1/2/4 (u08).

```
[EibPC]
R1_ID=1
// Timebuffer IDs vergeben:
ChartBuffer1=1
ChartBuffer1=2
// timebufferconfig: Einen Zeitbuffer konfigurieren
 MemTyp=0
Len=35040u16
Datatyp=3.3f16
timebufferconfig(ChartBuffer1, MemTyp, Len, "Temperature-1/2/3")
timebufferconfig(ChartBuffer2, MemTyp, Len, "Controll-1/2/4")
```

The readability of the code is increased, if we specify in the above example as the last argument the to be stored variable or group address. This is not absolutely necessary e.g. timebufferconfig(ChartBuffer1, MemTyp, Len, 2.2f16) or timebufferconfig(ChartBuffer2, MemTyp, Len,2) would also configure the timebuffer correctly.

With the configuration of the timebuffer to the webelement mtimechart the memory of the time series (timebuffer) is submitted for presentation by configuring their ID (=handle, acces of number). In this case the webelement accesses always out the last valid data in the memory.

Now the time series must be “filled” with data. The function

Write a value and the timestamp in the memory of time series

timebufferadd(*ChartBufferID, Daten*)

completes this task. The function writes the current value of the variable or group address (*data*) as well as the timestamp, which is derived from system time of the Enertex® EibPC, in the memory of the selected time series. So there a time series exists exactly out of a combination value-timestamp. Values can be up to 4 bytes long. Timestamps internally nedd 8 bytes.

Picture 7:Building of time series (timebuffer)

As Picture 7 should suggest, it is not necessarily so that the values in the timebuffer in the same interval have to be included, although this can often be the case when logging of energy data. The webelement mtimechart evaluates correctly the timestamp.

Writing a new value automatically updates the linked webelement

If the argument *MemTyp* from timebufferconfig was defined as a ring[store] so after reaching the last value the memory will be filled again from the beginning. i.e. the oldest value is replaces with the latest. Is *MemTyp* defined as linear[memory] then the recording stops if the memory is full

With a timeseries of linked diagram are automatically updated in the visualization i.e. it can be represented basically the same time series in different diagrams. For example writing every 15 minutes a value in the buffer and indicating the most recent 192 values in our diagram, you only need the following code:

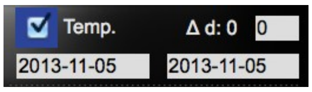
```
// Werte in den Buffer schreiben
if mtime(0,0) or mtime(15,0) or mtime(30,0) or mtime(45,0) then {
        timebufferadd(ChartBuffer0,"Temperature-1/2/3");
        timebufferadd(ChartBuffer1,"Controll-1/2/4");
} endif
```

With

timebuffersize(*ChartBufferID*)

the level of buffer can be accessed at any time.

The mtimechart webelement now displays 192 values, which is equivalent to a period of 2 days. Our buffer has space for 35040 values, which corresponds to ¼ hours values one year recording time. Picture 8 shows the option for the user to represent the past values: It an be given a start- and end date. If more than the configured number of values in the web element are stored in the same period in the time series as the diagram adjusts the display so that it hides intermediate values.



0
2013-11-05 2013-11-05

Picture 8: Timechart moving

Values from the past show ...

Example: The user sets a period of four days (e.g. 2013-07-11 bis 2013-09-13). In the here given configuration in the time buffer (ID 0 und 1) 384 values are stored. The diagram can only display 192 values and shows therefore in presentation each second value, effectively ½ hour values over 4 days will be displayed. Values fluctuations that are present in ¼ hour intervals, are no longer displayed. Th time axis is scaled or adjusted to the time specified. If the user configures the date fields in different time intervals the axis is scaled so that the stored values are displayed from oldest to the newest date.

... or compare with the past

With the field for Δd an offset in days for the presentation interval of the time buffer is set. The time axis is not scaled, but stops at the set date range. The values of the time series of this day back shifted are read from the memory. In this way curves of different time series are overlaid and compared.

It is important to note: If the user moves or scales a diagram, he disconnect the diagram from the real-time web server, i.e. further changing of values, which are written in the time series (time buffer) are no longer visible on the web server until a page refresh (usually F5) of the browser is running. This does not affect the other elements of the website.

After the time series was taken over some time in the Enertex® EibPC it has to be ensured that these are not los even if reloading of program or restarting the values. The functions

timebufferstore(*ChartBufferID*)

timebufferread(*ChartBufferID*)

are created for this task (comp. p. 256).

timebufferstore sets the values of the timebuffer with the *ChartBufferID* permanently into the flash memory of the Enertex® EibPC , timebufferread reads a stored buffer back. In addition the values with Enertex® EibStudio as described on page 200 to an external device to ensure data can be downloaded and uploaded.

Thus we store our buffer every 24 h in the following way:

Saving in the flash

```
// Wert im Flash speichern
if chtime(01,00,00) then {
        timebufferstore(ChartBuffer0);
        timebufferstore(ChartBuffer1);
} endif
```

The values we save back at startup as follows:

Loading from the flash

```
if systemstart() then {
        timebufferread(ChartBuffer0);
        timebufferread(ChartBuffer1);
} endif
```

Is a art declared as EXTDOUBLE so eliminates the fields of Picture 8, in particular the moving and superimposing of different time buffer no longer possible.

Simple mtimechart

Less „ease of operation“, especially in the application with a touch panel but more space for the representation provide the “simple” i.e. not EXT-format types of mtimecharts (comp. Picture 9). In this form the diagram is reminiscent of the mcharts respectively mpcharts (comp. p. Fehler: Referenz nicht gefunden and p. 252), where also the time axis is automatically scaled and taken out of the time buffer.

Picture 9: Standard-art webelement

The definition of the element in the web server is identical to the definition of the EXT-types, except that instead of EXTLONG is standing as format LONG.

```
[WebServer]
page(1)[$Log$,$Room1$]
mtimechart(R1_ID)[EXTLONG,NOAUTOSCALE,192,17,30,0,100] ( $Temp$,LEFT, ChartBuffer0,
$Control$,RIGHT,ChartBuffer1 )
```

Also here: If the user moves or scales a diagram he disconnects the diagram from the real time webserver, i.e. further changing of values, which are written in the time series (time buffer) are no longer visible on the web server until a page refresh (usually F5) of the browser is running. This does not affect the other elements of the website.

The time series is basically defined the same as in the code sections on p. 115 and p. 116.

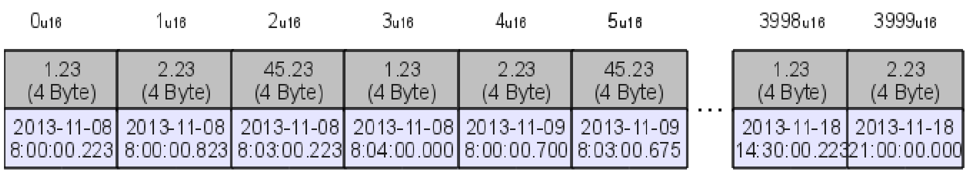
With the function

mtimechartpos(TimeChartID,ChartIdx,ChartBuffer,StartPos,EndPos)

mtimechart(TimeChartID,ChartIdx,ChartBuffer,StartZeit,EndZeit)

(comp. p.) can also be changed by the application program the display similar to the direct value input in Picture 8 of the display area of the art-webelements.

mtimechartpos requires additionally to the ID and the graph index mtimechart the position of the value range of the data in the buffer to which the value is fixed. As indicated in Figure 10 “numbers” the Enertex® EibPC every space from 0 up to max. configured value n-1. In this case, n is the configured buffer length. Figure 10 shows a buffer with length 4000, start position 0 and end position 3999. With the help of mtimechartpos one can fall back to the specified position in the time buffer where position 0 is always the oldest value in the buffer and position n-1 (in the example, the 3999) is the most recent value in the buffer.



0_{u16}	1_{u16}	2_{u16}	3_{u16}	4_{u16}	5_{u16}		3998_{u16}	3999_{u16}
1.23 (4 Byte)	2.23 (4 Byte)	45.23 (4 Byte)	1.23 (4 Byte)	2.23 (4 Byte)	45.23 (4 Byte)	…	1.23 (4 Byte)	2.23 (4 Byte)
2013-11-08 8:00:00.223	2013-11-08 8:00:00.823	2013-11-08 8:03:00.223	2013-11-08 8:04:00.000	2013-11-09 8:00:00.700	2013-11-09 8:03:00.675		2013-11-18 14:30:00.223	2013-11-18 21:00:00.000

Figure 10:Structure of the timebuffer with index

Selecting the values on the position in the timebuffer with the function mtimechartpos() or the time with mitimechart()

mtimechart does not evaluate the index of the graph but the value of the timestamp itself. Here have to be specified the time statements StartTime,EndTime in the argument as utc-millisecond format. In order to simplify this for the user, you can fall back to the function

utc(*Zeit*)

(comp. 180). This converts a string specifying of the form $2013-01-30 14:00:00$ into the utc-millisecond format.

```
if systemstart() {
      mtimechart(1,0,ChartBuffer0,utc($2013-01-30-14-00-00$),utc($2013-01-30 14:00:00$))
} enduf
```

Change of the displayed buffer of a mtimechart

Of interest is the possibility to “separate” the pre-configured linking in the web element from time series to the graph and to display the graph in another buffer.

Here is another example: As shown in Abbildung 4 should be taken a selection via a mpshifter-webelement, which is displayed in the recorded timebuffer.

Picture 11:Change of the presentation during running time

In the webserver the three elements shown are defined in which the pshifter is only used to display the current time. At the start of the application program the webelement ist linked to the timebuffer with ID chartbuffer3.

Using the same diagram for different timebuffer: For this purpose the year will be chosen with the selection box at the bottom left. The application program sets up the connection of diagram graph to the destinated timebuffer.

```
[WebServer]
page(PageID)[$Log$,$Room5$]
design $black$
mtimechart(TimeChartID)[LONG,2,255,30,17,256,0] ( $Room1$,LEFTGRAF, ChartBuffer3)
mpshifter(SelectID)[$2011$,$2012$,$2013$][DATE]$Room1$ pshifter(ClockID)[CLOCK]
$Aktuelle Uhrzeit$
```

We define three time series (time buffer),

```
MemTyp=1
Len=30640u16
Datatyp=3.3f16
timebufferconfig(ChartBuffer0, MemTyp, Len, "RkWohnzimmerTemp-3/1/28")
timebufferconfig(ChartBuffer1, MemTyp, Len, "RkWohnzimmerTemp-3/1/28")
timebufferconfig(ChartBuffer2, MemTyp, Len, "RkWohnzimmerTemp-3/1/28")
```

which now record data for every 1 year in ¼ time:

```
Y2011=date(1,1,11) and !date(1,1,12)
Y2012=date(1,1,12) and !date(1,1,13)
Y2013=date(1,1,13) and !date(1,1,15)
if (mtime(45,00) or mtime(45,00) or mtime(15,00) or mtime(00,00) ) and Y2011 then {
        timebufferadd(ChartBuffer0,"RkWohnzimmerTemp-3/1/28");
} endif
if (mtime(45,00) or mtime(45,00) or mtime(15,00) or mtime(00,00) ) and Y2012 then {
        timebufferadd(ChartBuffer1,"RkWohnzimmerTemp-3/1/28");
} endif
if (mtime(45,00) or mtime(45,00) or mtime(15,00) or mtime(00,00) ) and Y2013 then {
        timebufferadd(ChartBuffer2,"RkWohnzimmerTemp-3/1/28");
} endif
```

For each year a timebuffer, comp. p. 199 for backing up data to an external PC.

The „sampeln“ (storage) of the values on the buffer.

If the user now changes the selection box the corresponding time buffer should be displayed:

Evaluate selection box

```
if mpbutton(SelectID,1,PageID)==255 then {
        mtimechartpos(TimeChartID,0,ChartBuffer0,0u16,30639u16);
        pdisplay(SelectID,$Es wird 2011 dargestellt$,DATE,DISPLAY,GREY,PageID,1)
} endif
if mpbutton(SelectID,2,PageID)==255 then {
        mtimechartpos(TimeChartID,0,ChartBuffer1,0u16,30639u16);
        pdisplay(SelectID,$Es wird 2012 dargestellt$,DATE,DISPLAY,GREY,PageID,2)
} endif
if mpbutton(SelectID,3,PageID)==255 then {
        mtimechartpos(TimeChartID,0,ChartBuffer2,0u16,30639u16);
        pdisplay(SelectID,$Es wird 2013 dargestellt$,DATE,DISPLAY,GREY,PageID,3)
} endif
```

It can be seen how the graph with index 0 of the mtimechart is “diverted” to the different time buffer via ID. We fall back to the function mtimechartpos, which links the year chart buffer each with the graph 0.

Even a small addition to the clock display: This is now shown in the exact seconds in visualization, because the real-time web server adjusts every change of the “second hand”.

Validating scheme

For advanced programming using the validation scheme the following chapter includes the back ground.

Backgorund

The main task of an automation (control) system is to go through the user program so, that both event-oriented message processing, such as the arrival of KNX telegrams, as well as cycle-oriented processing, such as timer, timer switches etc. are processed. No need telegram may stuck in a queue for processing. In the Enertex ® EibPC always all tasks shall be processed in same priority.

This demand is the first design criteria for the firmware:

- *Any statement in the application program must be treated as a process, processed within a cycle time. Each statement is referred to the cycle-time processed in parallel.*

As a consequence we can state:

- *In principle, the application program is processed cyclycally.*
  The program starts reading the "inputs" (messages) from timers, switches, etc, then it is processing the user programm, then will switch the "outputs" and finally the whole thing starts over again. So this is different to a "normal" program. That will be started only once and depending on the user inputs subroutine calls occur and will be executed until the processing or other user initiates an other input.

Many KNX devices which can e.g. in addition to a touch panel operation mode, process simple logic, have a cycle time starting from 100ms. For the pure KNX solution with perhaps 10 AND and OR combinations this will be enough and due to the cycle time of 100ms performance problems might not occur. However, this logic is not very flexible or powerful. The Enertex ® EibPC is not limited in this manner. In addition, it has the capability to work with TCP, UDP and any RS232 telegrams and also to carry out extensive calculations. Therefore, a simple increase in cycle time is not enough.

Finally:

- *The user shall have no problems with the cycle time and the programming shall be made as easy as possible. A program shall be possible as follows*
  if Switch==ON then Write("Light-1/3/5“,ON) endif
  We have already noted, that the program has to by processed cyclically. If the program is processed in the style of a classic programming, initially it will be recognized in a cycle, the variable is in ON state and the Light has to be switched ON by initiating a KNX telegram. In next cycle the same holds true and then another telegram would be initiated. This is not intended, of course.

The tasks are

1. *Real time*
   The whole program shall be processed in one cycle
2. *Multithreading*
   Each statement shall be processed in one cycle
3. *Easy programming*
   The first steps when programming the user shall rapidly find solutions: „If a switch is pressed, send a telegram to the actor“

From these requirements, the validation concept has its origins. Each variable, function or expression used knows

- *whether it has changed,*
- *whether it has remained constant,*
- *whether an event occurred,*
- *whether dependent expressions are changing.*

Concept

If a change has occurred in this sense, the object to the status "invalid" and initiates the evaluation of the object and its dependencies again. If the object is then processed, it is valid. An "evaluation" of the function "write" for example, is equivalent to writing on the bus.

The entire program is compiled in that way, so it will implement this dependency of the objects in an object table.

Variables

```
[EibPC]
x=2
y="SaunaDimmer-1/0/1"+3%+x
z='1/2/3'b01 or '1/2/4'b01
```

With this definition, variables are defined and initialized. After the first pass, x is set to 2 and y to the value of the group address plus 3% plus x. In the next and all subsequent cycles changes x does not change any more, because 2 is constant. The things are different for y: If the group address is incoming and the transferred value has changed, y is re-calculated. y is dependent on an expression that has changed, and get this information "told". y “knows” therefore, that it must be re-calculated. It was "invalidated" by the group address. Where y is only invalid, if the value of the group address has actually changed. Similarly, y would recalculate when x would change. The invalidation of the depending objects is made “bottom up" up to the level at which the altered dependencies no longer have the effect of changes. If initially 1/2/3 is ON, the OR expression is ON and “tells” to z, if the OR was OFF before. Now if 1/2/4 is ON, the OR wlil be invalidated and shall be calculated again. Since OR is already ON, z is not invalidated, and therefore need not be recalculated.

Optimum performance is therefore a priori guaranteed, since at the KNX bus max. 1 telegram will arrive each 40ms. This telegramm certainly will not invalidate all objects, because only a certain -in a spophistacted program a vanishing- part will depend on this message. Therefore the EibPC reaches a cycle time of micro seconds.

Processing functions

```
[EibPC]
x=sin(3.14f32)
tan(2.0f32)
y=cos("Temperature-1/0/1")
z=event("Temperature-1/0/1")
```

Functions that process expressions are evaluated as follows: If there is a change of the argument (or more arguments), the function or its "outputs" will become invalid and will be re-calculated. In the example the sine and tangent are calculated only once during the runtime of the program. The cosine is calculated if the temperature changes, as well as z with the arrival of a telegram on 1/0/1.

Output functions

```
[EibPC]
write("Temperature-1/2/1",22.3)
write("Switch-1/2/10",!"Switch-1/2/10")
read("Temperature-1/2/1")
```

Functions which generate an output e.g. writing on the KNX bus or TCP/UDP/RS232 telegrams will never be invalidated by their arguments. Therefore, the above program will never write anything to the the KNX Bus. If no validation scheme would be active, the second statement would write to the KNX Bus in the process-cycle time of the EibPC.

Time controlling funtions

```
[EibPC]
o=stime(19)
```

Timing functions are not invalidated by their arguments, either. These are set by the timer (internal clock) of the Enertex® EibPC and are invalidated by it depending on the set time. Here, o alternates to ON for a processing cycle each time after 19 seconds of the system timer.

Nesting of if-statements

The if statement acts like a function whose argument is the query condition. If the query condition is invalid, it is evaluated. This evaluation is independent of whether the query condition changes its result. Since the query condition can only be invalid if its value changes, an invalid query condition is the same as a modified query condition (for non nesting if-statements).

```
[EibPC]
a=stime(33)
if stime(33) then write("Temperatur-1/2/1",22.3) endif
if '1/2/3'b01 then write("Temperatur-1/2/1",22.3) endif
```

In the example, stime and depending on it the "a" will be invalid each 33 seconds after the last full minute. The query condition of the if statement is therefore invalid and therefore it will be re-evaluated. When query condition changing and is equal to ON, the if statement invalidates once the functions of its then-branch. Therefore the write function becomes "invalid" and is evaluated. Evaluating means in this case, write the desired telegram on the bus.
'1/2/3'b01 becomes invalid each time a telegram is sent on this group address. This invalid signal is only forwarded to the if statement, if the incoming message (to this group address) has a different value than the last.

```
[EibPC]
a=1
if  '1/2/3'b01 then a=3 endif
```

After initialization a is 1. If '1/2/3'b01 changes from OFF to ON (message arrives), a is set to 3. a is constant at this value and is not changed (reset) in the next cycle to 1, because "1" is a constant and is never changing and therefore valid. Due to this it will not send an invalid to a.

Nested if

```
[EibPC]
a=1
b='1/2/4'b01
z=0
if  '1/2/3'b01 then {
     if b==ON then  a=3 endif;
     z=cos(1);
    write('1/3/4'b01,OFF)
} endif
```

If the inner if statement would only be invalidated by its query condition (b==ON), the outer if would not be considered. Therefore, the compiler builds the construct as follows: The inner if statement is invalidated by the outer if (and not by its own query condition). The evaluation of the inner query is triggered by this invalidate and therefore in any case, the then-branch is evaluated not depending on the change of the query condition but its state (b==ON). The outer If statement invalidates all objects of its then-branch in the order as listed in the program. Then the invalid expressions are evaluated. Consider the following example:

```
a=OFF
b=1
if change(a) then {
   if b==1 then write('1/2/3'b01,OFF);b=2 endif
   if b==2 then write('1/2/3'b01,ON) endif
} endif
```

After a changes the outer if statement becomes invalid. This invalidates all statements in the then-branch. Then the processing begins: The first inner if statement is active, sets b to 2 and writes OFF to 1/2/3. The second inner if statement receives an "invalid" signal from the outer if statement before the first if-statement is active, but it is not yet processed. In the meantime b is set to 2. The second inner if statement is now processed and the condition is ON. Therefore, ON is written to 1/2/3. For all other additional changes of a, the outer if statement becomes active and then only the second inner if statement (as b is not equal to 1).

We change the example as follows:

```
a=OFF
b=1
if change(a) then {
   if b==2 then write('1/2/3'b01,ON) endif
   if b==1 then write('1/2/3'b01,OFF);b=2 endif
} endif
```

Now after the first change of a the first inner if statement becomes invalid, too. But the query condition is 0 (OFF), since the variable b is still 1. The second inner if statement is invalidating its then-branch and “executes” this branch. The next (and all other) only the first if statement is executed.

As with the first example (referring to variable y) the following if statement will invalidate the then-branch only, if the OR operation itself is "invalid":

```
if  '1/2/3'b01 or '1/2/4'b01 then { ... } endif
```

time controls in the branch

Time functions are evaluated only in a nested if-statement if the if-statement sends an "invalid" to its then-branch. Please see the following example:

```
key='1/2/3'b01
a=OFF
if  key  then {
   if htime(12,00,00) then {
        a=ON
   } endif
} endif
```

a is here only set to ON, if the key is pressed exactly at 12:00:00 (htime only generates at this time an ON-pulse), what might not be intentionally. A solution in this case, would be the use of chtime. Then a is set to ON if the key is pressed (sending an ON telegram at '1/2/3'b01) after 12:00 and before midnight.

The else-branch

The else branch of an if function is like a second stand-alone function with if-negated query condition:

```
[EibPC]
a=OM
b=1
c='1/2/3'b01
if systemstart() then {
   if b==1 then {
                  write('1/2/3'b01,OFF);b=2
          }  else {
                  write('1/2/3'b01,ON)
          } endif
} endif
// is same as
//    if systemstart() then {
//         if b==1 then write('1/2/3'b01,OFF);b=2 endif
//         if  !(b==1) then write('1/2/3'b01,ON) endif
//    } endif
```

Writing to internal queues

If the internal queues are used ,e.g. if writing to the bus, the validation scheme is handling these quees at the end of a complete data processing for the desired output:

Write a telegram

```
[EibPC]
c='1/2/3'u08
b=1
if systemstart() then {
   write('1/2/3'u08,b);
   b=2
} endif
```

In this example write is executed . The data is processed in the queue after all other objects are valid. This means, b equals to 2. Therefore the value 2 is sent to 1/2/3.

The same holds true for a queue to IP telegrams (UDP, TCP/IP) and RS232 telegrams:

Write to RS232 or UDP

```
[EibPC]
b=1
s=$Hello$
if systemstart() then {
   if b==1 then {
               sendudp(4809u16,192.168.22.1,s);
               s=$World$;
               b=2
   }  else {
              sendudp(4809u16,192.168.22.1,s)
     } endif
} endif
```

*In this case (see the examples before) the string **World** is sent twice. Again, the queue is processed after all objects are valid.*

The same holds true for reading and writing to the flash and writng to strings with stringset.

Write to the flash

```
[EibPC]
b=1
s=$$
if systemstart() then {
  stringset(s,b,0u16);
  writeflash(b,0u16);
  b=b+1
} endif
```

First, b is incremented by 1, and therefore both s and the writeflash-queue both work with b equals 2.

Asynchronous processing

For some function calls (such as creating a TCP connection) it can not be guaranteed that they will update their return value within a processing loop of the EibPC. Therefore, these functions were implemented as separate threads within the firmware. These threads update their return values according to their own processing and is asynchronous to the main development loop. Consider the following example:

```
[EibPC]
// TCP off == 5
TCP=5
if after(systemstart(),2000u64) then {
  TCP=connecttcp(233u16,192.168.2.100)
} endif
```

2 seconds after start-up, connectcp is called. Since the processing now starts the appropriate thread, the return value is initially 0 (connection is set up) and leaves the if-Statement. Some time after the connection initializing, the connecttcp will update the variable TCP - regardless of the if statement - to 1 (connected).
Similarly, the return values of functions resolve, readflash, ping, writeflash and sendmail are asynchronous processing functions.

Enertex® EibStudio

Basics

Easy and convenient user interface

The Enertex® EibStudio is an application program for Windows® and Linux® computers. It is the development environment for programming the Enertex® EibPC. With the acquisition of Enertex® EibPC you purchase a license to use this program. The Enertex® EibStudio may only be used in conjunction with the Enertex® EibPC. By Enertex® EibStudio you can create, modify and maintain user programs easily.

Installation

No setup required!

The Enertex® EibStudio is an executable file. All necessary program parts are integrated parts of this file. Therefore, it requires no installation. You can directly run this program from any location. In particular, your Windows® registry or your system is not modified by the Enertex® EibStudio

The programming language

The Enertex® EibPC is programmed in a simple, specially designed and user-friendly language. For the Enertex® EibPC, the program needs to be translated (compiled), because in this way the code is adapted in an optimal manner for the Enertex® EibPC and no performance problems may occur. The necessary compiler called EibParser is an integral part of the Enertex® EibStudio. An editor is also part of the Enertex® EibStudio. It knows all the key words of the instruction set of the programming language and has syntax highlighting and autocompletion.

Compiler

The application program is entered in the form of a text file. To process different data in an appropriate way, there are so-called sections within the user program. These are characterized by a name in brackets. A section starts with its name and ends at the beginning of the next section or the end of the file.

Sections

The following sections subdivide the text

● [EibPC]	Your programming
● [ETS-ESF]	Import of ets Data (page 151)
● [Location]	Geographical data of the installation location (page 184 and page 150)
● [MacroLibs]	Predefined function libraries to be included page 24 and the following and page 313)
● [Macros]	Use of function libraries
● [RS232]	Configuration of the RS232 interface, if it is not used by FT1.2 (S. 228).
● [FTP]	Storing EIB telegrams at a FTP server (S. 142).
● [Performance]	Setting for Performance of EibPC (S. 127).
● [MailConf]	Configuration of the mail delivery (Option NP) p. 137
● [WebServer]	Configuration of the web server (Option NP) p. 289
● [VPN]	VPN Configuration (p. 244)
● [InitGA]	Initialize group adresses

User program

The user program (section [EibPC]) is your program for home automation. If you no longer want to get into the programming, a comprehensive library is available, which you can operate without any programming knowledge. Refer to page 24.

The application program (section [EibPC]) consists of several statements (see below). The processing is always done line by line, i.e. a statement should not to be interrupted by a newline (“RETURN“). If you want a line break for improved readability, the line must be completed by a double backslash “\\“ sign and can subsequently be continued in the next line.

Comments

The comments begin with “//“ and are at the beginning of a line. Besides, there are comments which appear instead of a statement (semicolon) surrounded by /* */, e. g. `/* This is the comment. */;`

Statements

Statements are:

- Definition of variables, see page 158,
- Function calls, see page 170,
- if statements, see page 161.

Comments, enclosed by /* and */ character (C-syntax)

Possible comments:

```
[EibPC]
/* Super, this comment  */ c=$$
z= 1 // a variable was defined

/* Super, a comment also with semicolon  */; c2=$$

// comment again

/* my first  if-statement */ if /* caution condition is 1b01 */ EIN  /* and now the then-path */ then
{
    c=elog()
   /* a comment without semicolon */
   /* also without semicolon */
 }  endif

if  AUS then {
    //  also possible
    z=4;
    // finished
} endif
```

Statement block

If two or more statements are executed as a block, they can be combined into a block of statements. These instructions are to be separated by a semicolon.

There is no semicolon after the last statement. This is the same for a statement block:

Line break

It may not interrupted by a newline ("RETURN"). If you want a line break for improved readability, the line must be finished by a double backslash "\\" and can subsequently continued in the next line of the statement block. In if-statements, you can alternatively work with curly brackets ({}; see example on page 38).

Performance

The application program is executed according to a so-called "validation scheme". This means, a statement is only evaluated if a change occurs to its dependency. For further explanation see page 162. The application program is run through all 1 ms, i.e. incoming telegrams are processed in 1 ms intervals.

Afterwards the main routine in the application program is evaluated. You can set the time spent for this process between 1 and 50ms. The more the time is set, the more time is spent for the web server to to handle telegrams. The refresh time of the web server can also be changed to increase or decrease the response time of the graphical user interface (vgl. Figure 11).

With the menu Optionen – Performance Settings you can set both time intervals to control the performance. The dialog creates the following section, which is inserted in the application program and could also be manipulated directly in the application program code:

Refresh time after pressing a button is currently not supported by the firmware, i.e. this setting is ignored.

```
[Performance]
// Performance setting: cycle time, refresh after pressing a button, automatic refresh
15
1500
10
```

Interaction between processes in the firmware

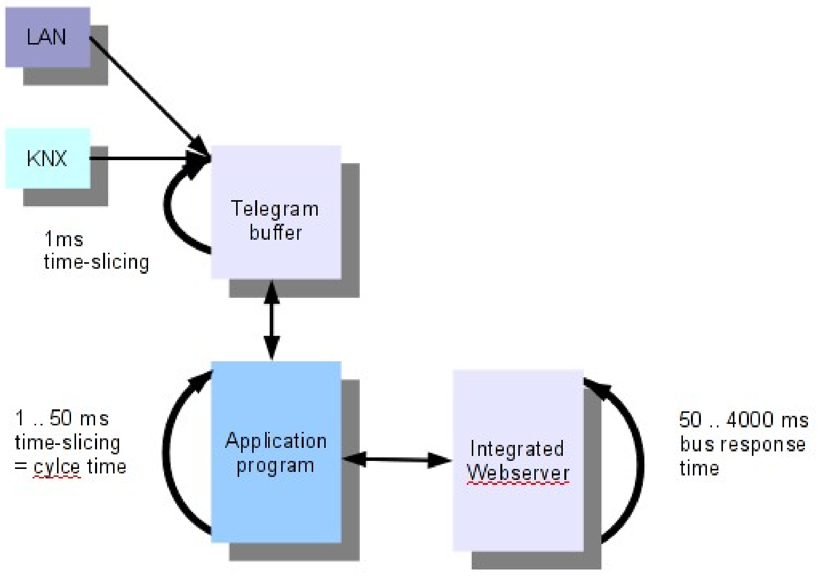


Abbildung 1: Processing application program

Autocompletion

Using the keyboard shortcut CTRL+Space (Space), the autocompletion of terms presents after typing the first letter a list of matching functions and definitions such as shown in Figure 2.

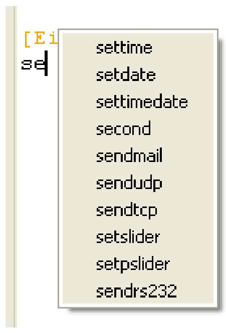


Figure 2: Auto completion after entering "s" and Ctrl-Space ("Space")

Syntax highlighting

In addition, the built-in editor is able to highlight the syntax of the notation. When saving the program, this always happens if this function is not disabled in the menu VIEW.

As you enter a new user program the syntax highlighting can be easily initiated by pressing CTRL+SHIFT+D.

Alternatively, after typing a character the syntax highlighting can be take place automatically within the menu Edit.

Directives

#include

Directive

- #include *file*

Effect

- Includes a file (path and filename) at the position in the program
- In an included File, another #include can be used recursively.

#break_if_older_version

Directive

- #break_if_older_version *number*

Effect

- The compiler "EibParser" stops, if its version number is smaller than the given *number*

#addto

Directive

- #addto [EibPC]
- #addto [Macros]
- #addto [WebServer]

Effect

- The compiler (EibParser) inserts the following lines of the section whose end is either defined by a new section or another #addto directive.

Example 1

```
[EibPC]
a=1
[Macros]
mymacro(a)
#addto [EibPC]
b=a+1
```

Example 2

```
[WebServer]
page(1) [$EG$,$Kitchen$]
button(2) [LIGHT] $Text$
[EibPC]
a=1
#addto [WebServer]
page(3) [$EG$,$Basement$]
button(2) [LIGHT] $Text$
```

#define

Directive

- #define *String*

Effect

- A symbol (string) is defined for the preprocessor

Note: The Compiler processes this symbol only with #ifdef directives

#undef

Directive

- #undef *String*

Effect

- Un-define a symbol (string) for the Praeprocessor

#ifdef

Directive

- #ifdef *String*

Effect

- The following code is compiled until a *#endif* occurs, if *String* is defined.

#ifndef

Directive

- #ifndef *String*

Effect

- The following code is compiled until a *#endif* occurs, if *String* is not defined.

#endif

Directive

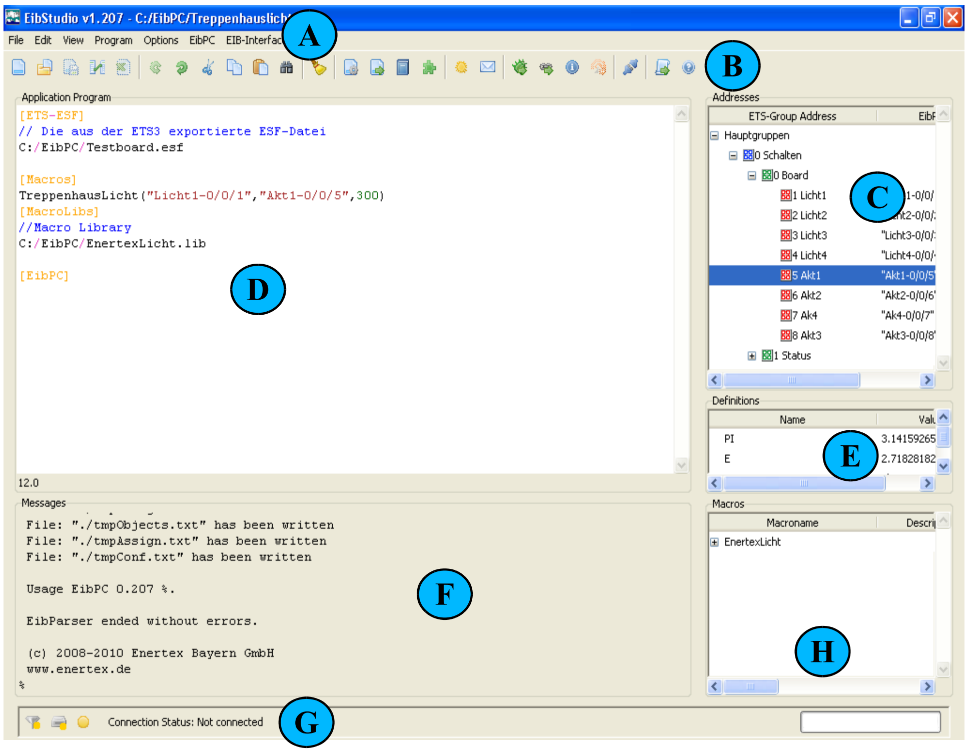
- #endif

Effect

- ends an #ifdef directive

Desktop

At a glance
– clear and structured



```
[ETS-ESF]
// Die aus der ETS3 exportierte ESF-Datei
C:/EibPC/Testboard.esf

[Macros]
TreppenhausLicht("Licht1-0/0/1","Akt1-0/0/5",300)
[MacroLibs]
//Macro Library
C:/EibPC/EnertexLicht.lib

[EibPC]
```

```
File: "./tmpObjects.txt" has been written
File: "./tmpAssign.txt" has been written
File: "./tmpConf.txt" has been written

Usage EibPC 0.207 %.

EibParser ended without errors.

(c) 2008-2010 Enertex Bayern GmbH
www.enertex.de
%
```

Name	Value
PI	3.14159265
E	2.71828182

Figure 3: Desktop Enertex® EibStudio for Windows XP®

Legend for Figure 2:

A) Menu bar - access to all functions
B) Toolbar - quick access to important functions
C) Pane to import addresses - they can be inserted via copy and paste into your project
D) Editor - with syntax highlighting and autocompletion
E) Pane for predefined constants - these are inserted via copy and paste
F) Pane for messages generated by the EibParser when compiling
G) Current connection status
H) Macros of loaded libraries

Menubar

Menubar	Short-Cut (Windows)	Description
File -		
New ...	Ctrl+N	Create a new application program and request an optional ETS export file
Open ...	Ctrl+O	Open an existing user program
Save	Ctrl+S	Overwrite the previously saved or opened file
Save as ...		Save the user program to a given file
Import group addresses from ETS-Export file		Import the addresses of an ETS export file
visualization assistant		Open the visualization assistant
Exit	Ctrl+Q	Close the Enertex® EibStudio
Edit -		
Undo	Ctrl+Z	Undo the last action
Restore	Ctrl+Shift+Z	Restore the last step
Cut	Ctrl+X	Cut a selected passage
Copy	Ctrl+C	Copy a selected text passage
Paste	Ctrl+V	Insert a text passage from the clipboard
Comment selection	Ctrl+D	Comment the current selection of the application program
Uncomment selection	Ctrl+Shift+D	Uncomment the current selection of the application program, if applicable
Copy addresses		Copy the selected address in the address window
Copy definition		Copy the selected variable in the definitions window
Add Macro		Insert a macro
Search	Ctrl+F	Search an expression
Replace	Ctrl+G	Replace an expression
View -		
Clear Message Console		Remove the text in the message window
Automatic Syntax Highlight		Terms are highlighted while editing (syntax highlighting)
Highlight Syntax	Ctrl+Shift+D	If automatic highlighting is disabled, activate syntax highlighting for the already existing text
Font Size 8		Change the font in the editor to size 8
Font Size 9		Change the font in the editor to size 9
Font Size 10		Change the font in the editor to size 10

Font Size 11		Change the font in the editor to size 11
Font Size 12		Change the font in the editor to size 12
Font Size 13		Change the font in the editor to size 13
Font Size 14		Change the font in the editor to size 14
Font Size 15		Change the font in the editor to size 15
Font Size 16		Change the font in the editor to size 16
Program -		
Compile Program	F7	The user program is compiled by the Enertex® EibParser; in the message window a status report appears.
Compile Program, send it to EibPC and run it	F3	The user program is compiled by the Enertex® EIBParser, sent to the Enertex® EibPC and run; in the message window a status report appears.
Reset Program of EibPC		The present program is deleted and replaced with an empty program
Macro Library ...		Add macro library
Macros ...		Add macro
Options -		
IP address of EibPC ...		The IP address of the Enertex® EibPC can be entered
Load EibStudio Settings ...		Load a *.set file that contains the settings of the Enertex® EibStudio
Save EibStudio Settings		Overwrite the previously opened or saved *.set file
Save EibStudio Settings as ...		Save the current settings of the Enertex® EibStudio in a given file
Setup Coordinates for Calculation of Solar Altitude		Edit the coordinates used for calculating the sunset / sunrise / azimuth / elevation
English		Activate German language
Deutsch		Activate English language
Email settings ...		Data for the e-mail configuration
VPN configuration		Data for VPN configuration
Webserver password protection configuration		Https configuration
Performance settings ...		Data for Performance settings
RS232 settings ...		Configuration of RS232 interface, if it is not used as EIB interface
NTP time synchronization		NTP configuration
FTP settings ...		Configuration for storing EIB telegrams on FTP server
Check for updates ...		Connects to the internet and checks for software updates
Check automaticly for updates		Option, which checks at ech start of Eibstudio for updates
Enertex® EibPC -		

Transfer Network Settings		The network settings will be sent to the Enertex® EibPC
Reset Network Settings		The network settings will be reset to the factory settings
DNS-Server ...		Specify a DNS server in the IPv4 notation
Test DNS-Server ...		The specified DNS server is checked and an appropriate message is displayed
Time Zone ...		Change the time zone for the Enertex® EibPC
Query Time Zone		The currently selected time zone of the Enertex® EibPC is displayed in the message window
Time of Day and Date ...		The time and date for the Enertex® EibPC can be changed
Query Time and Date		The present time and date are displayed in the message window
Read Event Memory		Read the event memory
Set and query Variables ...		The current value of an EIB-object or a variable is queried and displayed in the message window The value of an object or a variable can be changed and written to the KNX™ bus
Retrieve EIB-Messages ...		EIB telegrams are retrieved from EibPC and exported as CSV file
Retrieve EIB-Messages from FTP ...		Binary files with stored EIB telegrams are retrieved from FTP server and exported as CSV file
Retrieve EIB-Messages cyclically ...		All the stored EIB telegrams will be deleted
Delete EIB-Messages ...		Delete all EIB telegrams stored in the Enertex® EibPC
Query number of EIB-Messages		Request the number of the Enertex® EibPC saved from telegrams
Monitor current EIB-Messages	F2	Print the current telegram in the message window
Filter current EIB-Messages		Current EIB telegrams can be filtered before output
Configure Filter ...		Filter for EIB-telegrams set
Delete scenes ...		All scenes stored in the Enertex® EibPC are deleted
Restart ...		The Enertex® EibPC is restarted
Upload Firmware ...		A firmware update is sent to the Enertex® EibPC
Installed Firmware Version		The current firmware version, serial number and applied patches are shown
Transfer Activation Code		An activation code is transmitted to the Enertex® EibPC
Activated Features		The available features are shown.
Transfer patch ...		A patch update is sent to the Enertex® EibPC
Finish Transfer		The transfer to the Enertex® EibPC is manually terminated
EIB-Interface		
Choose ...		Choose the way the Enertex® EibPC connects to the bus (FT1.2 or EIBnet/IP)
Open connection		(Re)connect the Enertex® EibPC to an EIBnet/IP interface
Close connection		Close an existing connection of the Enertex® EibPC to an EIBnet/IP interface

Connection state		Query the status of the connection of the Enertex® EibPC to an EIBnet/IP interface
? -		
Help	F1	Open the manual
Pack information for support request		Creates a compressed file containing all the necessary firmware and software data for efficent support
Info		Version information of the Enertex® EibStudio·, Enertex® EibPC

Network configuration

Switching on

After connecting the power supply and the network cable, the Enertex® EibPC is operational in about 2-3 minutes and responds to network requests. For the installation and the status indicator (built-in LED) see also page 17 and the following

DHCP

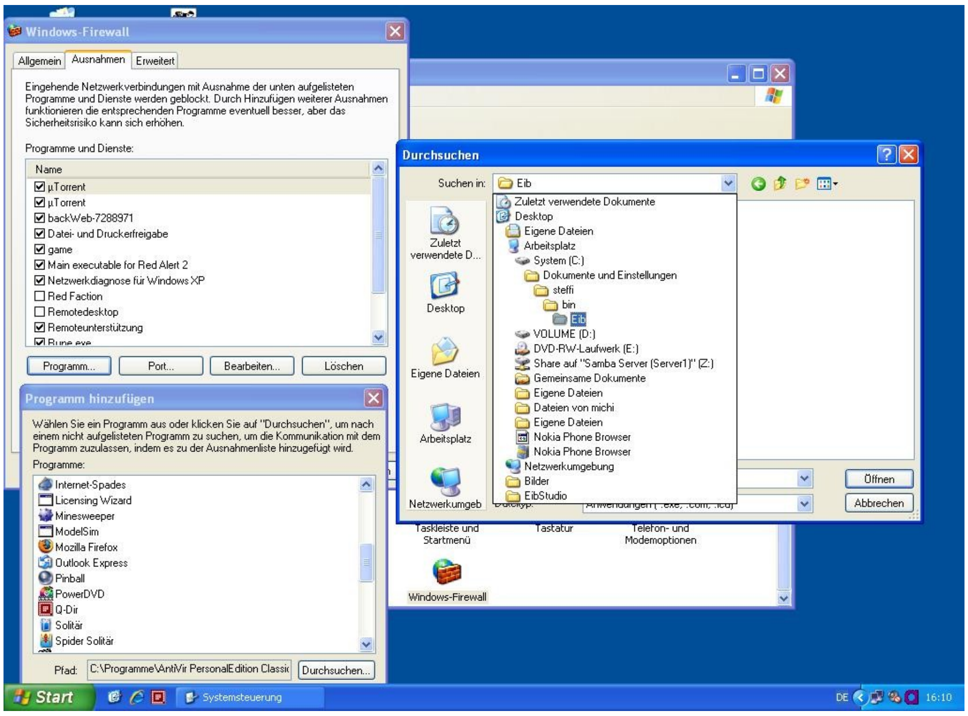
You have the option to assign the Enertex® EibPC to a fixed IP address within your home network or to work with a DHCP server. When delivered, the Enertex® EibPC is set to DHCP. If the Enertex® EibPC finds a DHCP server, it assigns itself to a free IP address. For this, the Enertex® EibPC must be connected with a cross-over cable to a PC.

If the firewall is appropriately set up, you can query the IP address in the menu Options → Network settings as shown in Figure 5 by clicking the push button "Automatic". Now, in the window "messages", you will see the response of Enertex® EibPC. Then, you can set up the Enertex® EibPC with a fixed IP address (see below).

Unlock the firewall

Important: An active firewall avoids the communication between the Enertex® EibStudio and the Enertex® EibPC and has to be configured accordingly. For details see the instructions of your firewall. Activate the communication port 4805 and 4806 for UDP telegrams in your firewall. If you want to send UDP telegrams (see page 235), you have to open the port 4807 additionally.

Alternative: For example, in a Windows® firewall you can also directly establish an exception for the program nconf.exe (part program of the Enertex® EibStudio). For this, the Enertex® EibStudio has to be started at least once. After calling Enertex® EibStudio, you can find nconf.exe within your Windows® user directory, as shown in Figure 4. To configure this, select within the dialog "Program exceptions" of the firewall- sub-dialog "Add program" - (here) search and set your path corresponding to "C:\Documents and Settings\YOUR USERNAME\bin\eib\nconf.exe". Alternatively, activate the communication port 4805 and 4806 for UDP telegrams in your firewall. If you want to send UDP telegrams, you must also open the communication port 4807.



Eingehende Netzwerkverbindungen mit Ausnahme der unten aufgelisteten Programme und Dienste werden geblockt. Durch Hinzufügen weiterer Ausnahmen funktionieren die entsprechenden Programme eventuell besser, aber das Sicherheitsrisiko kann sich erhöhen.

Programme und Dienste:

Name
☑ µTorrent
☑ µTorrent
☑ backWeb-7288971
☑ Datei- und Druckerfreigabe
☑ game
☑ Main executable for Red Alert 2
☑ Netzwerkdiagnose für Windows XP
☐ Red Faction
☐ Remotedesktop
☑ Remoteunterstützung
☑ Rune.exe

Wählen Sie ein Programm aus oder klicken Sie auf "Durchsuchen", um nach einem nicht aufgelisteten Programm zu suchen, um die Kommunikation mit dem Programm zuzulassen, indem es zu der Ausnahmenliste hinzugefügt wird.

Programme:

Internet-Spades
Licensing Wizard
Minesweeper
ModelSim
Mozilla Firefox
Outlook Express
Pinball
PowerDVD
Q-Dir
Solitär
Spider Solitär

Pfad: C:\Programme\AntiVir PersonalEdition Classic

Figure 4: Configuration of Windows®-Firewall

Problems with firewalls

Kaspersky with Windows 7

- When activated the Kaspersky firewall and Windows firewall is disabled, can not send the Eibstudio
- If the Kaspersky firewall and Windows firewall is disabled, can not send the Eibstudio
- If the Kaspersky firewall and disabled the Windows firewall is enabled, send the Eibstudio

The same behavior was described with Windows Vista and Norton 360.

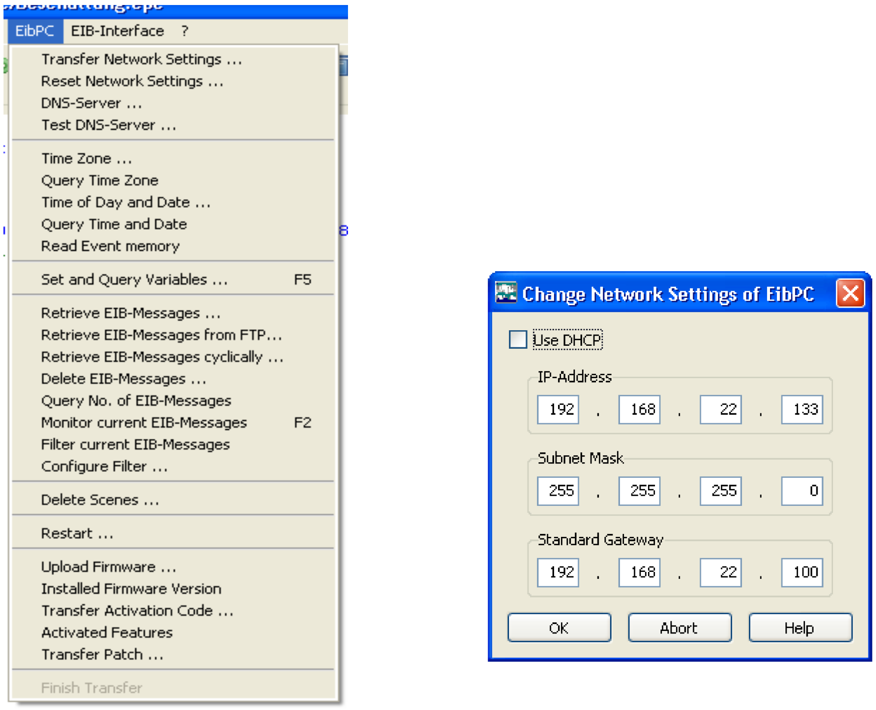
Built-in DHCP Replacement

If you do not have an active DHCP server in the network, the Enertex® EibPC is looking for a free network address after booting. This address can be queried in menu Options → Network settings by clicking on the button “Automatic“, as shown in Figure 5. In the window “messages“ you will see the response of the Enertex® EibPC which is entered directly into the dialog Figure 5. If you see an error message, please follow the instructions.

Figure 5: Enter network settings

If you want to configure the Enertex® EibPC to have a fixed IP address, you must first address the Enertex® EibPC (see Options → Network settings dialog Figure 5 and button “Automatic“) either via DHCP or the built-in DHCP replacement (e.g. a direct connection to your PC). Then, if you are already connected to the Enertex® EibPC, the network settings of the Enertex® EibPC can be easily changed. Click on menu EibPC → Change Network Settings of EibPC, as shown in Figure 6. The dialog on the right should open with the default settings. Now enter your desired network settings.

Fixed IP


EibPC
EIB-Interface
?
Transfer Network Settings ...
Reset Network Settings ...
DNS-Server ...
Test DNS-Server ...
Time Zone ...
Query Time Zone
Time of Day and Date ...
Query Time and Date
Read Event memory
Set and Query Variables ... F5
Retrieve EIB-Messages ...
Retrieve EIB-Messages from FTP...
Retrieve EIB-Messages cyclically ...
Delete EIB-Messages ...
Query No. of EIB-Messages
Monitor current EIB-Messages F2
Filter current EIB-Messages
Configure Filter ...
Delete Scenes ...
Restart ...
Upload Firmware ...
Installed Firmware Version
Transfer Activation Code ...
Activated Features
Transfer Patch ...
Finish Transfer
Change Network Settings of EibPC
OK
Abort
Help

☐ Use DHCP

IP-Address: 192 . 168 . 22 . 133

Subnet Mask: 255 . 255 . 255 . 0

Standard Gateway: 192 . 168 . 22 . 100

Figure 6: Enter network settings

To actually take over the network settings, the data have to be transmitted to the Enertex® EibPC. Then, the Enertex® EibPC will restart (2-3 min restart process). If you have assigned a new address to the Enertex® EibPC, this address must be disclosed to the Enertex® EibStudio. Here, in Options → Network settings dialog Figure 5, you have to enter the new data.

IP address	= Address of the Enertex® EibPC, must be an address of the new home network
Netmask	= Your netmask, mostly 255.255.255.0
Default Gateway	= Network address of the router

If the Enertex® EibPC has taken over the data, it must be re-booted.

Save and open the network settings

A new network configuration for the Enertex® EibStudio should be saved in a file with the extension “set“. For this, there is the dialog OPTIONS → SAVE NETWORK SETTINGS available.

An existing network configuration is opened by the corresponding dialog in the FILE menu. The Enertex® EibStudio stores this file by default with the extension “set“, as shown in Figure 7.

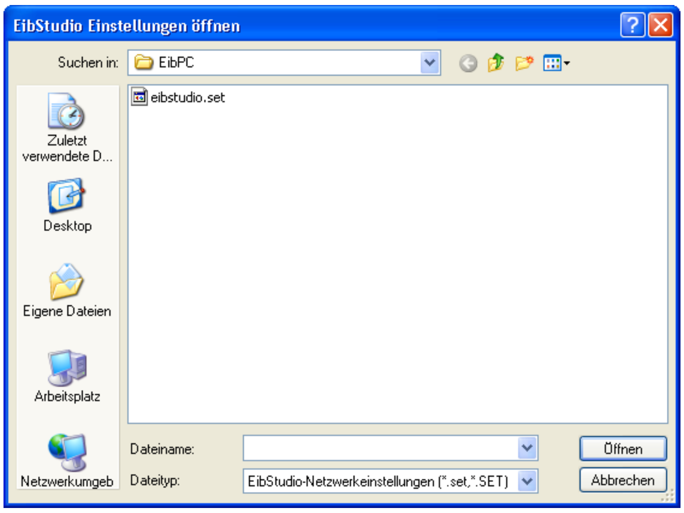


Figure 7: Open network preferences

DNS server

If the Enertex® EibPC has to establish an Internet connection, then the Enertex® EibPC connects itself to a DNS server for the purpose of name resolution. You can change the default connection. For this, the dialog Figure 8 is available.

Figure 8: Setting DNS Server

If you want to check whether the Enertex® EibPC has taken over the data and whether the DNS server is reachable, just click EIBPC → TEST DNS SERVER AND PAY ATTENTION TO THE WINDOW “MESSAGES“.

Factory settings

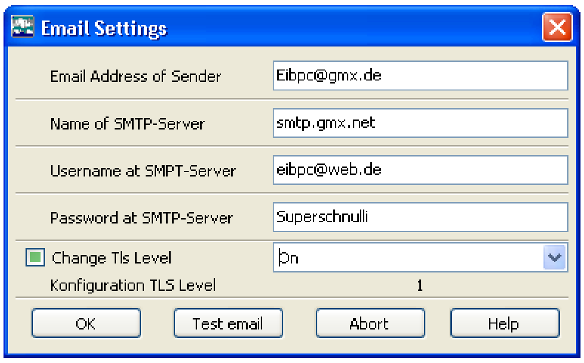
The factory settings can be reset using the reset button (see p. 19 for usage). In this case, you may also delete any stored scenes and the user program.

The network settings can also be restored by the function EIBPC → RESET NETWORK SETTINGS. For this feature, the Enertex® EibPC must be restarted by interrupting of the power supply. Take note of the appropriate dialog message.

E-Mail settings (Option NP)

If you run the additional Enertex® EibPC software package “Option NP”, you can also send e-mails with the Enertex® EibPC. For this, the activation codes have to be imported (see page 235) and a SMTP access has to be set up. Also, you should have set up the DNS server (see above) and the Enertex® EibPC should have an online connection to the Internet and a valid SMTP Server.

As shown in Figure 9, the e-mail settings can be configured in the dialog OPTIONS → EMAIL SETTINGS.


Email Settings

Email Address of Sender	Eibpc@gmx.de
Name of SMTP-Server	smtp.gmx.net
Username at SMPT-Server	eibpc@web.de
Password at SMTP-Server	Superschnulli
Change Tls Level	On
Konfiguration TLS Level	1

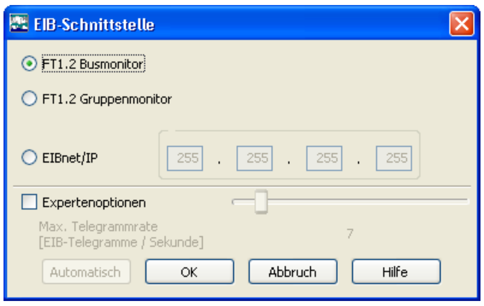
Figure 9: E-Mail setup

After clicking the "OK" button, the input data appears in the user program (see Figure 10) and can also be edited directly.

Figure 10: E-Mail setup

Via the menu EIB INTERFACE, THE KNXTM INTERFACE THROUGH WHICH THE ENERTEX® EIBPC ACCESSES THE KNXTM BUS MAY BE SELECTED.

Configuration of the KNX™ Interface


EIB-Schnittstelle

(●) FT1.2 Busmonitor
( ) FT1.2 Gruppenmonitor
( ) EIBnet/IP 255 . 255 . 255 . 255
[ ] Expertenoptionen
Max. Telegrammrate [EIB-Telegramme / Sekunde] 7

Figure 11: Configure Interface

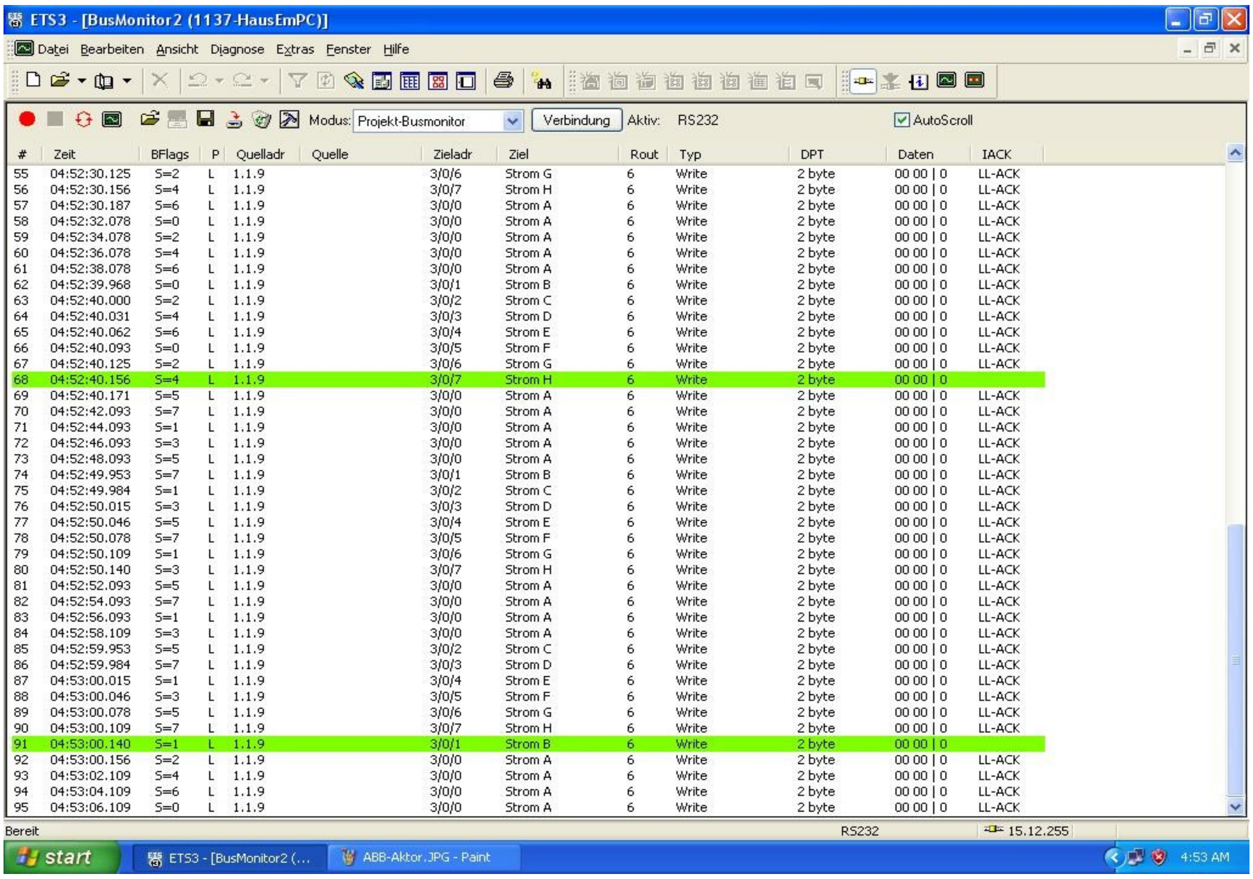
- In case of an FT1.2 interface, please note the commissioning instruction on page 15 and the following. You can operate the FT1.2 interface in the bus monitor mode or in the group monitor mode, the latter will acknowledge all KNX™ telegrams. Basically, the group monitor mode is more tolerant with bus errors. Please also note the information on page 139. If an FT1.2 interface has been recently operated in the bus monitor mode, and shall be switched to the group monitor mode, it has to be reset by a short interruption of its power supply before the application program is transmitted to the Enertex® EibPC.

- In case of an IP interface, please note the commissioning instruction on page 15 and the following. and enter the relevant data of the interface into the above configuration wizard. The interface is accessed and activated only after the transfer and the subsequent start of an application program

  The Enertex® EibPC uses the so-called tunneling mode of the IP interface. Most IP Interfaces provide only one tunnel. Thereby, it is exclusively occupied, i.e. you can not address the interface in this mode from the ETS. Nevertheless, in order to allow a short-term access to the interface you can separate the interface via the menu item CONNECT and DISCONNECT from Enertex® EibPC and reconnect with it, respectively.
- By means of the telegram rate limitation, the default value of which is seven telegrams per second, you can avoid overloading the KNX™ bus and guarantee the stable operation of your installation.

KNX^{TM} bus errors

Basically, the reliability of an FT1.2 interface can be verified by the bus monitor of the ETS. If the FT1.2 interface works properly, it can be concluded that this will be the same with the Enertex® EibPC.


ETS3 - [BusMonitor2 (1137-HausEmPC)]
Datei Bearbeiten Ansicht Diagnose Extras Fenster Hilfe
Modus: Projekt-Busmonitor
Verbindung
Aktiv: RS232
AutoScroll
start
ETS3 - [BusMonitor2 (...
ABB-Aktor.JPG - Paint
4:53 AM

#	Zeit	BFlags	P	Quelladr	Quelle	Zieladr	Ziel	Rout	Typ	DPT	Daten	IACK
55	04:52:30.125	S=2	L	1.1.9		3/0/6	Strom G	6	Write	2 byte	00 00 | 0	LL-ACK
56	04:52:30.156	S=4	L	1.1.9		3/0/7	Strom H	6	Write	2 byte	00 00 | 0	LL-ACK
57	04:52:30.187	S=6	L	1.1.9		3/0/0	Strom A	6	Write	2 byte	00 00 | 0	LL-ACK
58	04:52:32.078	S=0	L	1.1.9		3/0/0	Strom A	6	Write	2 byte	00 00 | 0	LL-ACK
59	04:52:34.078	S=2	L	1.1.9		3/0/0	Strom A	6	Write	2 byte	00 00 | 0	LL-ACK
60	04:52:36.078	S=4	L	1.1.9		3/0/0	Strom A	6	Write	2 byte	00 00 | 0	LL-ACK
61	04:52:38.078	S=6	L	1.1.9		3/0/0	Strom A	6	Write	2 byte	00 00 | 0	LL-ACK
62	04:52:39.968	S=0	L	1.1.9		3/0/1	Strom B	6	Write	2 byte	00 00 | 0	LL-ACK
63	04:52:40.000	S=2	L	1.1.9		3/0/2	Strom C	6	Write	2 byte	00 00 | 0	LL-ACK
64	04:52:40.031	S=4	L	1.1.9		3/0/3	Strom D	6	Write	2 byte	00 00 | 0	LL-ACK
65	04:52:40.062	S=6	L	1.1.9		3/0/4	Strom E	6	Write	2 byte	00 00 | 0	LL-ACK
66	04:52:40.093	S=0	L	1.1.9		3/0/5	Strom F	6	Write	2 byte	00 00 | 0	LL-ACK
67	04:52:40.125	S=2	L	1.1.9		3/0/6	Strom G	6	Write	2 byte	00 00 | 0	LL-ACK
68	04:52:40.156	S=4	L	1.1.9		3/0/7	Strom H	6	Write	2 byte	00 00 | 0	
69	04:52:40.171	S=5	L	1.1.9		3/0/0	Strom A	6	Write	2 byte	00 00 | 0	LL-ACK
70	04:52:42.093	S=7	L	1.1.9		3/0/0	Strom A	6	Write	2 byte	00 00 | 0	LL-ACK
71	04:52:44.093	S=1	L	1.1.9		3/0/0	Strom A	6	Write	2 byte	00 00 | 0	LL-ACK
72	04:52:46.093	S=3	L	1.1.9		3/0/0	Strom A	6	Write	2 byte	00 00 | 0	LL-ACK
73	04:52:48.093	S=5	L	1.1.9		3/0/0	Strom A	6	Write	2 byte	00 00 | 0	LL-ACK
74	04:52:49.953	S=7	L	1.1.9		3/0/1	Strom B	6	Write	2 byte	00 00 | 0	LL-ACK
75	04:52:49.984	S=1	L	1.1.9		3/0/2	Strom C	6	Write	2 byte	00 00 | 0	LL-ACK
76	04:52:50.015	S=3	L	1.1.9		3/0/3	Strom D	6	Write	2 byte	00 00 | 0	LL-ACK
77	04:52:50.046	S=5	L	1.1.9		3/0/4	Strom E	6	Write	2 byte	00 00 | 0	LL-ACK
78	04:52:50.078	S=7	L	1.1.9		3/0/5	Strom F	6	Write	2 byte	00 00 | 0	LL-ACK
79	04:52:50.109	S=1	L	1.1.9		3/0/6	Strom G	6	Write	2 byte	00 00 | 0	LL-ACK
80	04:52:50.140	S=3	L	1.1.9		3/0/7	Strom H	6	Write	2 byte	00 00 | 0	LL-ACK
81	04:52:52.093	S=5	L	1.1.9		3/0/0	Strom A	6	Write	2 byte	00 00 | 0	LL-ACK
82	04:52:54.093	S=7	L	1.1.9		3/0/0	Strom A	6	Write	2 byte	00 00 | 0	LL-ACK
83	04:52:56.093	S=1	L	1.1.9		3/0/0	Strom A	6	Write	2 byte	00 00 | 0	LL-ACK
84	04:52:58.109	S=3	L	1.1.9		3/0/0	Strom A	6	Write	2 byte	00 00 | 0	LL-ACK
85	04:52:59.953	S=5	L	1.1.9		3/0/2	Strom C	6	Write	2 byte	00 00 | 0	LL-ACK
86	04:52:59.984	S=7	L	1.1.9		3/0/3	Strom D	6	Write	2 byte	00 00 | 0	LL-ACK
87	04:53:00.015	S=1	L	1.1.9		3/0/4	Strom E	6	Write	2 byte	00 00 | 0	LL-ACK
88	04:53:00.046	S=3	L	1.1.9		3/0/5	Strom F	6	Write	2 byte	00 00 | 0	LL-ACK
89	04:53:00.078	S=5	L	1.1.9		3/0/6	Strom G	6	Write	2 byte	00 00 | 0	LL-ACK
90	04:53:00.109	S=7	L	1.1.9		3/0/7	Strom H	6	Write	2 byte	00 00 | 0	LL-ACK
91	04:53:00.140	S=1	L	1.1.9		3/0/1	Strom B	6	Write	2 byte	00 00 | 0	
92	04:53:00.156	S=2	L	1.1.9		3/0/0	Strom A	6	Write	2 byte	00 00 | 0	LL-ACK
93	04:53:02.109	S=4	L	1.1.9		3/0/0	Strom A	6	Write	2 byte	00 00 | 0	LL-ACK
94	04:53:04.109	S=6	L	1.1.9		3/0/0	Strom A	6	Write	2 byte	00 00 | 0	LL-ACK
95	04:53:06.109	S=0	L	1.1.9		3/0/0	Strom A	6	Write	2 byte	00 00 | 0	LL-ACK

Bereit RS232 15.12.255

Figure 12: Problems caused by an excessive rate of telegrams when using an FT1.2 interface

To check if your project generates such a rate of telegrams that messages are already lost at the hardware level, proceed as follows:

1. First, connect the FT1.2 interface to the ets.
2. Open the project in the ets and start the **bus monitor**. Be careful **not to use the group monitor** of the ets.
3. If the bus monitor of the ets displays telegrams with a green background (Figure 12), the installation is not free of problems. In this case, reduce the frequency of the telegrams by parametrizing the respective actuator with telegram rate limitation.
4. If the previously noticed faulty telegrams do not appear any more, you have achieved an error-free installation relating to the traffic of the KNX^{TM} bus and are now able to connect the FT1.2 interface to the Enertex® EibPC.

The bus monitor of the Enertex® EibPC

Connection status feedback

By the menu EibPC → Retrieve EIB-Messages you can check the status of the network connection. If this option is set, the current connection status is displayed on the left lower message pane. Alternatively, a click to this push button will suffice. If the Enertex® EibPC is connected with the Enertex® EibStudio, the button will change to and the status display will change from the status "?" to "Connected to 192.168.22.133" (if the latter is the specified network address of the Enertex® EibPC).

Connection indicator and status

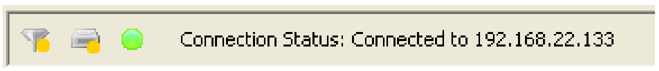

Connection Status: Connected to 192.168.22.133

Figure 13: Connection status

Two bus monitors

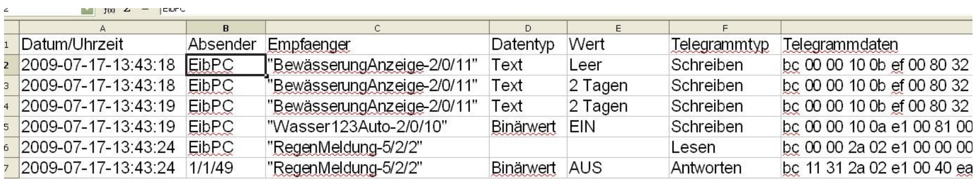
The Enertex® EibPC has two ring buffers. The larger of the two stores up to 500,000 messages and is designed for long-term studies. No PC needs to be connected while recording with the Enertex® EibPC is performed. The second ring buffer for 10 telegrams is intended for the observation of the current bus traffic. Here you have the ability to filter for specific sender addresses (devices) and group address and to query only relevant data.

Save bus data

To collect the telegrams accumulated in the Enertex® EibPC, click in menu EibPC on Retrieve EIB-Messages. In the bottom left of the status bar you will see the progress in the collection of telegrams. The speed of the transmission of telegrams depends primarily on the performance of your PC.

If all messages are picked up or the transmission is interrupted, a dialog appears. By this dialog, you can save these telegrams into a CSV file. This CSV file is a text file storing the data tabularly. The individual data are separated by commas. Then, you can import the data into Microsoft® Excel or OpenOffice Calc or into a spreadsheet program of your choice.

Typical bus data



Datum/Uhrzeit	Absender	Empfaenger	Datentyp	Wert	Telegrammtyp	Telegrammdaten
2009-07-17-13:43:18	EibPC	"BewässerungAnzeige-2/0/11"	Text	Leer	Schreiben	bc 00 00 10 0b ef 00 80 32
2009-07-17-13:43:18	EibPC	"BewässerungAnzeige-2/0/11"	Text	2 Tagen	Schreiben	bc 00 00 10 0b ef 00 80 32
2009-07-17-13:43:19	EibPC	"BewässerungAnzeige-2/0/11"	Text	2 Tagen	Schreiben	bc 00 00 10 0b ef 00 80 32
2009-07-17-13:43:19	EibPC	"Wasser123Auto-2/0/10"	Binärwert	EIN	Schreiben	bc 00 00 10 0a e1 00 81 00
2009-07-17-13:43:24	EibPC	"RegenMeldung-5/2/2"			Lesen	bc 00 00 2a 02 e1 00 00 00
2009-07-17-13:43:24	1/1/49	"RegenMeldung-5/2/2"	Binärwert	AUS	Antworten	bc 11 31 2a 02 e1 00 40 ea

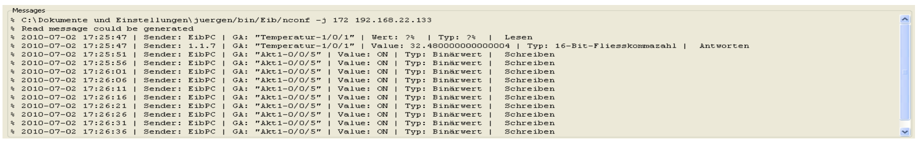
Figure 14: Import of recording data in OpenOffice

In Microsoft® Excel, select the sub menu Import within the dialog Data. In the following, best choose the format described as "text" and the text label as {no}. Similarly, you proceed at OpenOffice Calc®. In this manner, you get data in the format of Figure 14.

Current bus communication

Additionally, the current bus communication can be made visible. Activate the option "pick EIB telegrams / Connection Status" or click on the button . If the Enertex® EibPC is connected with the Enertex® EibStudio, the button changes to .

Now, in the window "Messages" all the telegrams are provided which were just written on the bus. Figure 15 shows a typical excerpt from the message window, which is self-explanatory.



Messages

```
% C:\Dokumente und Einstellungen\juergen/bin/Eib/nconf -j 172 192.168.22.133
% Read message could be generated
% 2010-07-02 17:25:47 | Sender: EibPC | GA: "Temperatur-1/0/1" | Wert: ?%  | Typ: ?%  |  Lesen
% 2010-07-02 17:25:47 | Sender: 1.1.7 | GA: "Temperatur-1/0/1" | Value: 32.480000000000004 | Typ: 16-Bit-Fliesskommazahl |  Antworten
% 2010-07-02 17:25:51 | Sender: EibPC | GA: "Akt1-0/0/5" | Value: ON | Typ: Binärwert |  Schreiben
% 2010-07-02 17:25:56 | Sender: EibPC | GA: "Akt1-0/0/5" | Value: ON | Typ: Binärwert |  Schreiben
% 2010-07-02 17:26:01 | Sender: EibPC | GA: "Akt1-0/0/5" | Value: ON | Typ: Binärwert |  Schreiben
% 2010-07-02 17:26:06 | Sender: EibPC | GA: "Akt1-0/0/5" | Value: ON | Typ: Binärwert |  Schreiben
% 2010-07-02 17:26:11 | Sender: EibPC | GA: "Akt1-0/0/5" | Value: ON | Typ: Binärwert |  Schreiben
% 2010-07-02 17:26:16 | Sender: EibPC | GA: "Akt1-0/0/5" | Value: ON | Typ: Binärwert |  Schreiben
% 2010-07-02 17:26:21 | Sender: EibPC | GA: "Akt1-0/0/5" | Value: ON | Typ: Binärwert |  Schreiben
% 2010-07-02 17:26:26 | Sender: EibPC | GA: "Akt1-0/0/5" | Value: ON | Typ: Binärwert |  Schreiben
% 2010-07-02 17:26:31 | Sender: EibPC | GA: "Akt1-0/0/5" | Value: ON | Typ: Binärwert |  Schreiben
% 2010-07-02 17:26:36 | Sender: EibPC | GA: "Akt1-0/0/5" | Value: ON | Typ: Binärwert |  Schreiben
```

Figure 15: Viewing EIB telegrams

Autolog

If the Enertex® EibStudio has been started, the bus messages can also be cyclically stored in a file. The cycle time can be 99 days at most, i.e. the data will be stored in a csv-file all 99 days. In this case the file name is fixed.

It is constructed according to the pattern "autologous-2009-11-20-13-53-45.csv", thus in this example: Autolog, from 20.11.2009, 13:53:45 clock.

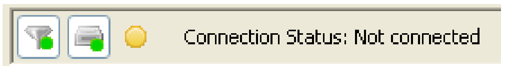


Figure 16: Autolog (2. Symbol) with filtering enabled

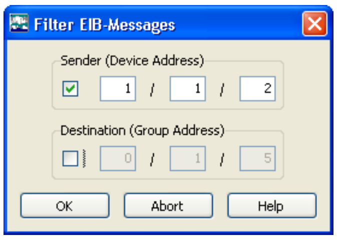
Filter with wildcards

You can set up filters so that only telegrams of certain group addresses or specified device addresses (senders) are displayed or stored. You must first set up the filter (see Figure 17), which you can set up within the menu item EibPC – Configure Filter.

Figure 17: Filter EIB telegrams

With the settings in Figure 17, only frames which arrive from the sender with the physical address 1/1/2 and are sent to the target group address 0/1/5 are displayed,.

If you like to watch all telegrams of a specific sender, regardless of the destination, you must disable the option target (see Figure 18).



Filter EIB-Messages

Sender (Device Address)
[✓] 1 / 1 / 2

Destination (Group Address)
[ ] 0 / 1 / 5

Figure 18: Filtering of all EIB telegrams of a sender

If the filter is activated (EibPC – Filter current EIB-Messages), the status display changes as indicated in Figure 19. The filter acts both to telegrams in the message window, as well as when saving to a CSV file like the Autolog function.

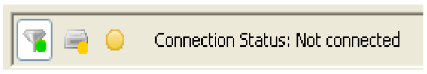

Connection Status: Not connected

Figure 19: Filter display in the bottom right of the status display, right picture: Filter enabled

You have the option of using wildcards:

The question mark “?“ stands for an arbitrary digit, the asterisk “*“ for any number.

If the last entry for the target is written “1/1/2**?**3“, all group addresses 1/1/2**0**3, 1/1/2**1**3 ... 1/1/2**9**3 are filtered out. You can use any number of wildcards. Write 1/*/203, and all frames having the major group 1 and subgroup 203 (and any middle group) are going to be processed .

Enertex® EibPC can also store telegrams on a FTP server. In this case the telegrams will be stored in binary form at the FTP server. You can set the configuration of the FTP service in the menu OptionS – FTP-Transfer.

Directly Storing on a FTP server

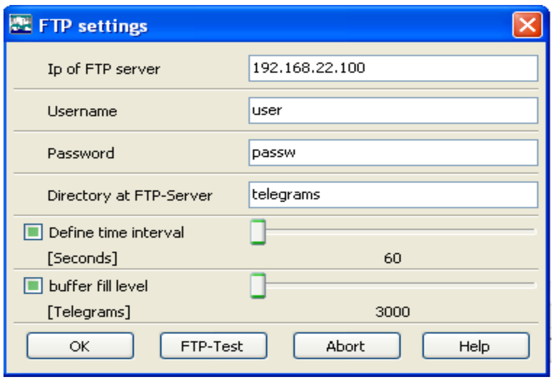
Alternatively you also can specify the settings direct in the application program by creating the section [FTP] like this:

```
[FTP]
// FTP server
192.179.169.67
//Username
roman
//password
praktikum
//Remote FTP-Directory
telegrams/MyLog
//Timeout
60
//Max. buffer
3000
```

Similarly, you can set the configuration via Options – FTP settings (Fig. 20).

The data is swapped cyclically. The interval is defined by either the time or the buffer fill level.

Dialog (Fig. 20) swapped either in a time interval of 60 seconds or when a buffer level of 3000 messages is reached


FTP settings

Ip of FTP server	192.168.22.100
Username	user
Password	passw
Directory at FTP-Server	telegrams
Define time interval [Seconds]	60
buffer fill level [Telegrams]	3000

Figure 20: Configuration dialog FTP settings

Binary data stored on the FTP-Server can be evaluated and exported into a CSV file by Enertex® EibStudio. Therefore use the Menu EibPC – Retrieve EIB Messages from FTP.

Remark

For a correct conversion of the binary files on the FTP into a readable format the current loaded application program must be the same as the program, which has stored the binary telegram data on the FTP.

Compile and transfer the user program

A user program must be compiled, transfered to the Enertex® EibPC and started. As depicted in Figure 21, you can separately perform these steps in the menu PROGRAM, press the Ctrl+Shift+B shortcut or the button . The program will be transferred and started. The status display (see Figure 13) of the transfer process is shown. Dependent on the program size, the transfer and running and start of the user program takes 1 to 5 seconds.

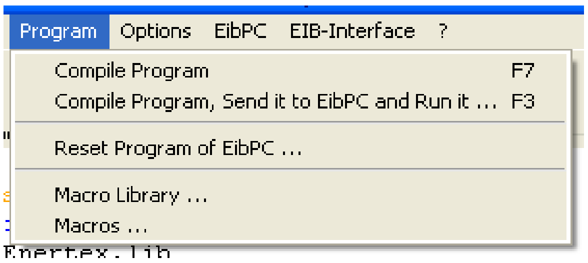
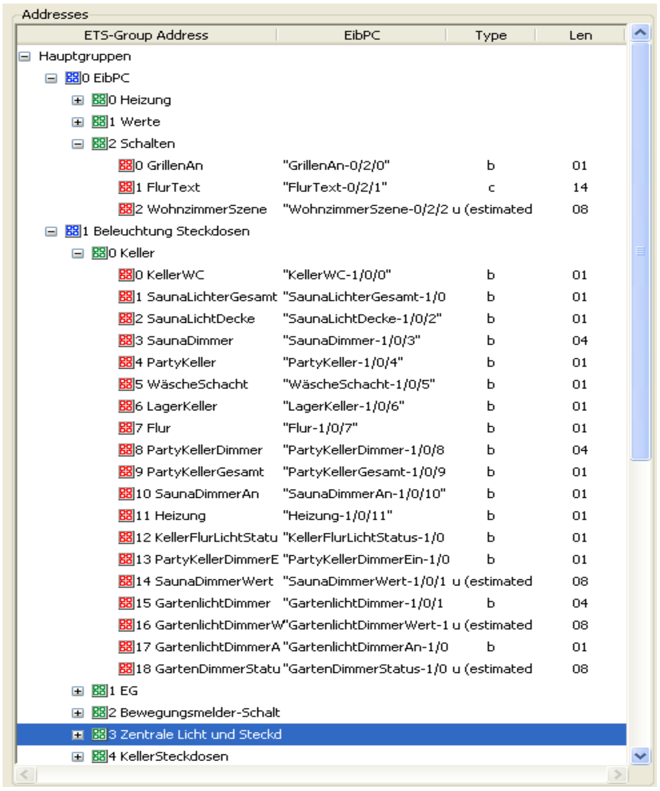


Figure 21: Compiling user program, send to EibPC and start

ETS addresses

The Enertex® EibStudio is directly able to import addresses from the current ets project. For more details, see the description on page 151. Imported addresses are displayed in the window Address, as shown in Figure 22. By a drag and drop mechanism or by copying the selection (CTRL+C), you can also simply copy the addresses across the keyboard and then pasted in the application program.

Importing the exported ets addresses


Addresses

ETS-Group Address	EibPC	Type	Len
Hauptgruppen			
0 EibPC			
0 Heizung			
1 Werte			
2 Schalten			
0 GrillenAn	"GrillenAn-0/2/0"	b	01
1 FlurText	"FlurText-0/2/1"	c	14
2 WohnzimmerSzene	"WohnzimmerSzene-0/2/2	u (estimated	08
1 Beleuchtung Steckdosen			
0 Keller			
0 KellerWC	"KellerWC-1/0/0"	b	01
1 SaunaLichterGesamt	"SaunaLichterGesamt-1/0	b	01
2 SaunaLichtDecke	"SaunaLichtDecke-1/0/2"	b	01
3 SaunaDimmer	"SaunaDimmer-1/0/3"	b	04
4 PartyKeller	"PartyKeller-1/0/4"	b	01
5 WäscheSchacht	"WäscheSchacht-1/0/5"	b	01
6 LagerKeller	"LagerKeller-1/0/6"	b	01
7 Flur	"Flur-1/0/7"	b	01
8 PartyKellerDimmer	"PartyKellerDimmer-1/0/8	b	04
9 PartyKellerGesamt	"PartyKellerGesamt-1/0/9	b	01
10 SaunaDimmerAn	"SaunaDimmerAn-1/0/10"	b	01
11 Heizung	"Heizung-1/0/11"	b	01
12 KellerFlurLichtStatu	"KellerFlurLichtStatus-1/0	b	01
13 PartyKellerDimmerE	"PartyKellerDimmerEin-1/0	b	01
14 SaunaDimmerWert	"SaunaDimmerWert-1/0/1	u (estimated	08
15 GartenlichtDimmer	"GartenlichtDimmer-1/0/1	b	04
16 GartenlichtDimmerW	"GartenlichtDimmerWert-1	u (estimated	08
17 GartenlichtDimmerA	"GartenlichtDimmerAn-1/0	b	01
18 GartenDimmerStatu	"GartenDimmerStatus-1/0	u (estimated	08
1 EG			
2 Bewegungsmelder-Schalt			
3 Zentrale Licht und Steckd			
4 KellerSteckdosen			

Figure 22: Addresses

Debugger

To “debug“ your program, i.e. to look for errors, you can query

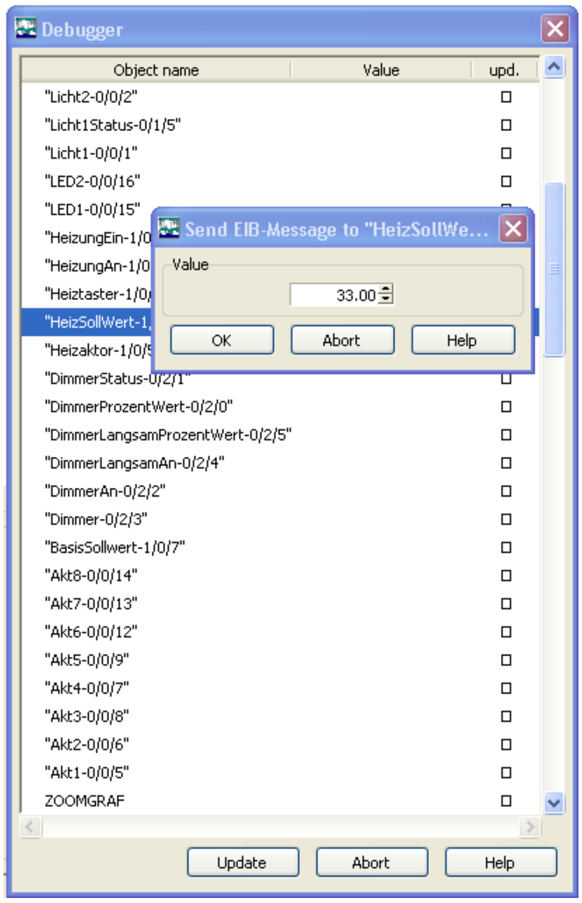
- Variables
- Constants
- Manual group addresses
- Imported group addresses

as well as write values to the bus and change the value of variables.

Important

Before you can debug a program or values of the Enertex® EibPC, you must have transferred the program to the Enertex® EibPC

For debugging, press or alternatively in the menu EIBPC – VARIABLES SETTING AND QUERY. A window appears (Figure 23) with variables defined in the program and imported or manual group addresses.



Object name	Value	upd.
"Licht2-0/0/2"		☐
"Licht1Status-0/1/5"		☐
"Licht1-0/0/1"		☐
"LED2-0/0/16"		☐
"LED1-0/0/15"		☐
"HeizungEin-1/0		
"HeizungAn-1/0		
"Heiztaster-1/0/		
"HeizSollWert-1,		
"Heizaktor-1/0/5		
"DimmerStatus-0/2/1"		☐
"DimmerProzentWert-0/2/0"		☐
"DimmerLangsamProzentWert-0/2/5"		☐
"DimmerLangsamAn-0/2/4"		☐
"DimmerAn-0/2/2"		☐
"Dimmer-0/2/3"		☐
"BasisSollwert-1/0/7"		☐
"Akt8-0/0/14"		☐
"Akt7-0/0/13"		☐
"Akt6-0/0/12"		☐
"Akt5-0/0/9"		☐
"Akt4-0/0/7"		☐
"Akt3-0/0/8"		☐
"Akt2-0/0/6"		☐
"Akt1-0/0/5"		☐
ZOOMGRAF		☐

Value: 33.00

Figure 23: Debugger

Here you have the possibility to query and modify values.

When you debug group addresses, the value which is saved for the group address by the Enertex® EibPC is displayed, (thus no read request on the bus).

In the message window the retrieved or written value is displayed.

For quick trouble-shooting

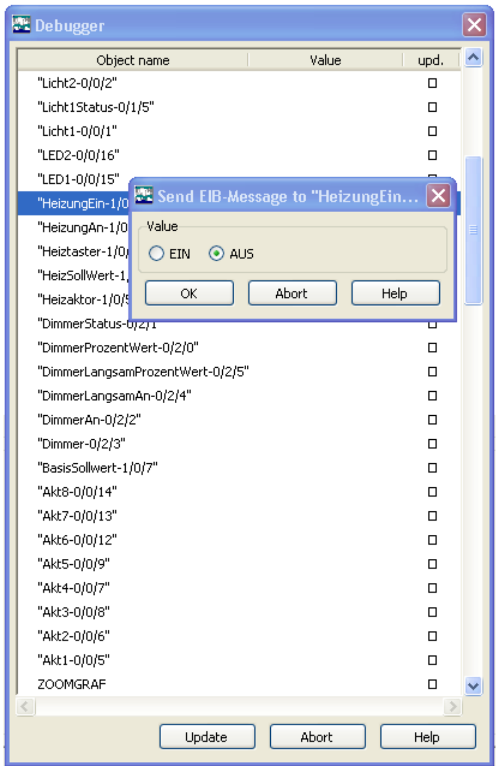
When you select an object, i. e. a variable, constant or a group address, and you press the right mouse button, there are three options available:

- Write value
- Read value from bus (only if you have selected a group address)
- Read value from EibPC

In case of imported group addresses, values can be directly written to or read from the EIB.

The Enertex® EibStudio automatically detects the data type and offers appropriate input options.

Setting and querying of variables



Object name	Value	upd.
"Licht2-0/0/2"		☐
"Licht1Status-0/1/5"		☐
"Licht1-0/0/1"		☐
"LED2-0/0/16"		☐
"LED1-0/0/15"		☐
"HeizungEin-1/0		
"HeizungAn-1/0		
"Heiztaster-1/0		
"HeizSollWert-1		
"Heizaktor-1/0/5		
"DimmerStatus-0/2/1		☐
"DimmerProzentWert-0/2/0"		☐
"DimmerLangsamProzentWert-0/2/5"		☐
"DimmerLangsamAn-0/2/4"		☐
"DimmerAn-0/2/2"		☐
"Dimmer-0/2/3"		☐
"BasisSollwert-1/0/7"		☐
"Akt8-0/0/14"		☐
"Akt7-0/0/13"		☐
"Akt6-0/0/12"		☐
"Akt5-0/0/9"		☐
"Akt4-0/0/7"		☐
"Akt3-0/0/8"		☐
"Akt2-0/0/6"		☐
"Akt1-0/0/5"		☐
ZOOMGRAF		☐

Value: EIN [✓ AUS]

Figure 24: Debugger

If you like to write a value to the EIB for the address “Light2-0/0/2“, as shown in Figure 24, you can choose between ON and OFF, since this is an address of the binary data type.

With the option “Read value from bus“ a read request is written to the bus. After that the bus device with the respective group address replies with its current value. Eibstudio evaluates the reply and displays the value in the respective column of the debugger. When a bus device with the respective group address does not reply, the term *not readable* is displayed in the column *value*.

Query more variables

With the button *Update* you can query more values from the bus at the same time. E.g. as seen in Figure 24 you can set crosses at variables of interest. Pressing the button *Update* will automaticly update the values of all marked variables.

Variables and manual group addresses can be changed and queried. Here, the input mask is also adapted to the data type, and the program directly responds to the change and initiates the proper actions.

Importing patch updates

By using the menu EibPC -Transfer Patch, a new patch update for the Enertex® EibPC is easily to import at the touch of a button. After an update starts, the Enertex® EibPC automatically starts. When the green indicator LED flashes again, the Enertex® EibPC is operational.

Updates are exclusively issued by the Enertex® Bayern GmbH.

Importing firmware updates

As seen in Figure 25, a new firmware update for the Enertex® EibPC can be imported simply by pressing a button on the menu EibPC. After an update, the Enertex® EibPC automatically restarts. If the green indicator LED flashes again, the Enertex® EibPC is operational.

Figure 25: Firmware update

Updates are exclusively issued by the Enertex® Bayern GmbH. In the menu EibPC, the firmware version can be checked any time. The version information will appear in the message window (see Figure 25).

You can reset the user program in the Enertex® EibPC. Thus, the present program is cleared and the Enertex® EibPC is set to idle. The power supply must be interrupted, because this process requires a hardware restart of the Enertex® EibPC.

Proceed as follows:

1. Select the item "Reset program of EibPC ..." in the menu Program (see Figure 21).
2. In the following dialog press "OK".
3. Disconnect the power supply of the Enertex® EibPC momentarily
4. Wait about 5 minutes until the Enertex® EibPC is operational again.
5. Now you can again transfer a new application program to the Enertex® EibPC (page 144).
6. If an installation location is entered which differs from the default, an additional computing time of 2 min is necessary once to create the new solar data.

Query of firmware version

System crash

When nothing works

Reset button

If even this approach does not help, you can press the reset button (see p. 19 for usage). In this case,

- the user program,
- data on the installation location and
- network settings

are deleted or reset to default values. **Note also the above item 6.**

Message

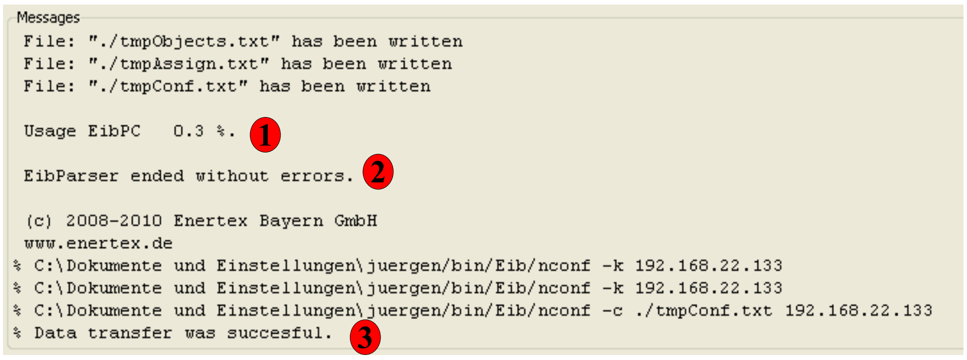
The message window is also used to display

- data from the bus monitor (see page 140)
- data in the debugger variables (see page 144)
- messages from the integrated compiler "EibParser". If errors occur during compilation, they will appear in this window area.

Bus monitor

Compiler

Data transfer



```
File: "./tmpObjects.txt" has been written
File: "./tmpAssign.txt" has been written
File: "./tmpConf.txt" has been written

Usage EibPC   0.3 %.

EibParser ended without errors.

(c) 2008-2010 Enertex Bayern GmbH
www.enertex.de
% C:\Dokumente und Einstellungen\juergen/bin/Eib/nconf -k 192.168.22.133
% C:\Dokumente und Einstellungen\juergen/bin/Eib/nconf -k 192.168.22.133
% C:\Dokumente und Einstellungen\juergen/bin/Eib/nconf -c ./tmpConf.txt 192.168.22.133
% Data transfer was succesful.
```

Figure 26: Messages

Status reports can be read in the message window.

1) indicates that the Enertex® EibPC is utilized to 0.3% by the memory consumption.
2) indicates that the user program is compiled without errors.
3) indicates that the data transmission was successful.

Time zone

Enertex® EibPC = Time master

or

KNX-BUS = Time master

Once an Internet connection is established by the Enertex® EibPC through your network, it detects via the NTP protocol the current time and synchronizes with this time. You also have the possibility to indicate up to nine NTP server.

The time synchronization can be inhibited by Options – NTP time synchronization.

Therefore it is necessary to determine whether the application program to be started only after the time synchronization or immediately after system start. By default, starts the application program when the EibPC is operational.

Your time zone can be modified in the menu EibPC at *"Set Time Zone*".

The Enertex® EibPC can also be used as the "time master" within your KNX installation, i.e. to periodically send the current system time and date as a KNX™ telegram to all bus users.

If you do not have an Internet connection, the Enertex® EibPC can be synchronized with KNX telegrams or the time can be manually entered. For further information, see page 174.

Position of the sun

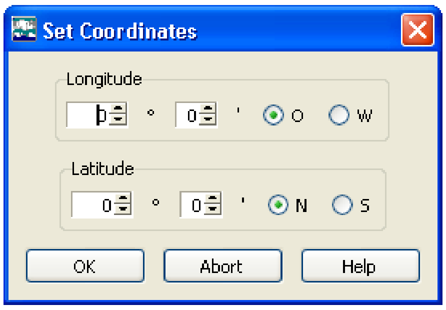


Figure 27: Message

As shown in Figure 27, the menu Options – Setup Coordinates for Calculation of Solar Altitude can be used for entering your longitude and latitude in order to calculate the solar altitude appropriately to your geographic location by the function sun(), as shown on page 150. The coordinates have to be provided also for the functions azimuth and elevation (page 150).

After exiting the above dialog, Enertex® EibStudio creates a section [Location] in the user program and writes the specified values. If this section does not exist, the Enertex® EibPC takes the default values as coordinates 11 ° 3' 29" E, 49° 43' 11"N; as decimal: 11.07 and 49.68.

Example:

```
[Location]
// Longitude and latitude of the installation location
49.0722222222222
11.0688888888889
```

Recalculation of the sun data takes a few minutes

After the change of the data and the start of the user program, the Enertex® EibPC calculates the sunrise and sunset as well as the position of the sun again. This process may take several minutes to complete and is indicated when you connect with the Enertex® EibPC.

Please note that during this time the Enertex® EibPC is blocked. If you are connected with the Enertex® EibPC via the menu EibPC → Retrieve EIB-Messages, you will see a status message in this case.

Note: You can modify the section [Location] in the user program directly.

You can find the information about your location e. g. in the Internet at www.geodatenzentrum.de.

Variables and KNX™ group addresses

Manual or imported group addresses

The group addresses which were created in an ets project can be imported easily. The approach on this is shown on page 151.

In addition, it is possible to directly write to the group address without this import, and thus without the ets project. Page 161 shows the necessary action.

Then, within the programming of the Enertex® EibPC in section [EibPC] you can access the variables and group addresses by using the syntax on page 153.

ets Project data

There a two operations necessary to use your ets data with the Enertex® EibStudio:

- Export of group addresses from the ets as a so-called ESF export.
- Import of group addresses into the Enertex® EibStudio.

ets Export of Group addresses

To export group addresses from the ets, proceed as follows: Within the ets, you open the export function “data exchange“ in the menu File (see Figure 1).

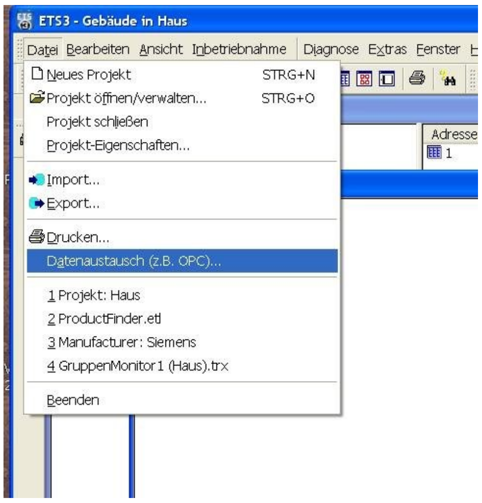
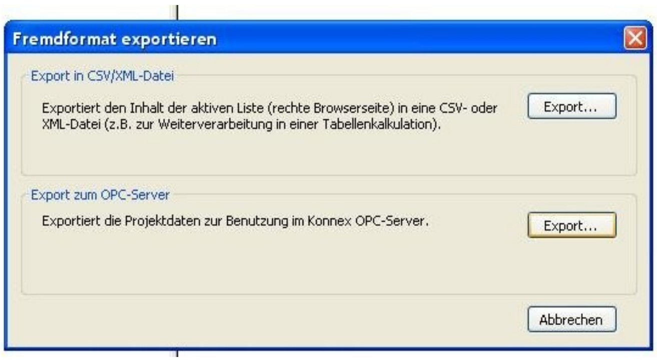


Figure 1: Export within ets

Then, the dialog appears according to figure Figure 2. Here, we choose the second button “Export to OPC-Server“. In the subsequent file dialog, the destination file is specified. In the following, this (text-) file is called ets-ESF-Export-File.



Exportiert den Inhalt der aktiven Liste (rechte Browserseite) in eine CSV- oder XML-Datei (z.B. zur Weiterverarbeitung in einer Tabellenkalkulation).

Exportiert die Projektdaten zur Benutzung im Konnex OPC-Server.

Figure 2: Export within ets to “OPC Server“

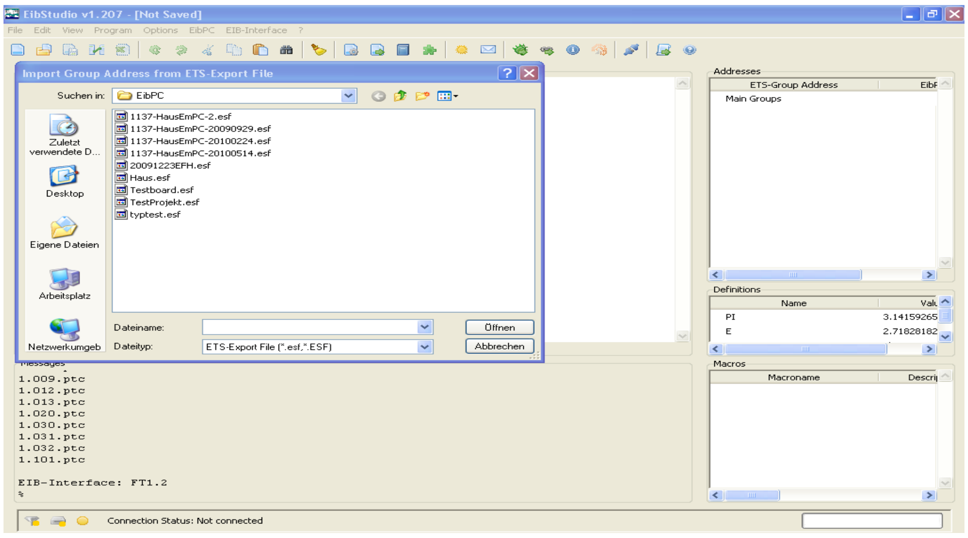
Most suitably, the ets-ESF-Export-File is put into the project directory.

Import of group addresses into the Enertex® EibStudio

In order to use the exported group addresses of the ESF data of the ets-project, this file must be written with its complete path and name in the section [ETS-ESF] of the user program. The Enertex® EibStudio completes this step for you, if you proceed as described now:

To load the ESF file, click:

File → Import Group Addresses from ETS-Export File



EibStudio v1.207 - [Not Saved]

Import Group Address from ETS-Export File

Suchen in: EibPC

- 1137-HausEmPC-2.esf
- 1137-HausEmPC-20090929.esf
- 1137-HausEmPC-20100224.esf
- 1137-HausEmPC-20100514.esf
- 20091223EFH.esf
- Haus.esf
- Testboard.esf
- TestProjekt.esf
- typtest.esf

Dateiname:

Dateityp: ETS-Export File (*.esf,*.ESF)

Messages

```
1.009.ptc
1.012.ptc
1.013.ptc
1.020.ptc
1.030.ptc
1.031.ptc
1.032.ptc
1.101.ptc

EIB-Interface: FT1.2
%
```

Connection Status: Not connected

Addresses

ETS-Group Address	EibP
Main Groups	

Definitions

Name	Valu
PI	3.14159265
E	2.71828182

Macros

Macroname	Descri

Figure 3: Loading ETS-export file

Now the Enertex® EibStudio automatically generates the section [ETS-ESF] in the user program as shown in Figure 4.

Figure 4: ETS-ESF-Section

In the ETS-ESF section, the absolute path to the exported ESF-file is now displayed. In the address field, the addresses imported from this file are visible and can be used by copy & paste. For comparison see page 144, Figure 22.

Use of variables and group addresses in the user program of the Enertex® EibPC

Telegram structure

Useful background information

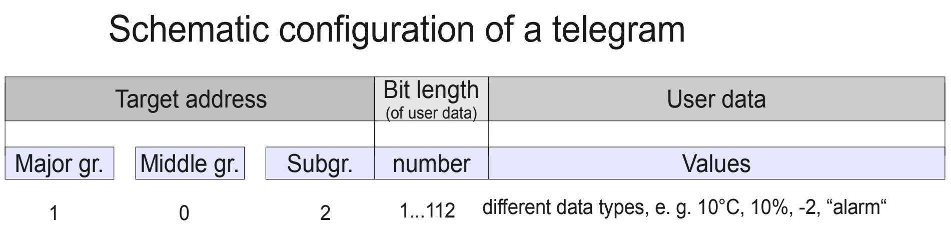


Figure 5: Schematic telegram structure

"Telegram" designates a complete block of data for the transmission of information. Besides the data to be transported, the telegram contains all additionally needed information in specific fields, as indicated in Figure 5.

Data types

Among the transported data, the bit length as well as the “interpretation“ of the data must be known, so that a meaningful programming is possible.

For example, there are numeric types with different bit lengths: The value “1“ can be transmitted with both 1-bit, 4-bit, 8 bits and so on. Because of this ambiguity, the so-called “data type“ must be accurately specified in the programming.

The term “data type“ summarizes the numeric type (“interpretation“) and bit length in one unit.

In the Enertex® EibStudio, the following syntax has been chosen:

The data type is appended to a number or a address.
The data type mostly consists of one letter and a 2-digit number.

Possible types (based on standard programming languages) are:

● Unsigned (positive) integers	Letter u (“unsigned“)
● Signed integers	Letter s (“signed“)
● Floating-point numbers	Letter f (“float“)
● Character string	Letter c (“char“)
● Date and time	Letter t or d or y (“time“, “day“, “year“)

The following bit lengths are possible

● 1 bit	01 digits
● 4 bit	04 digits
● 8 bit	08 digits
● 16 bit	16 digits
● 24 bit	24 digits
● 32 bit	32 digits
● 64 bit	64 digits
● 112 bit	14 digits
● 11200 bit (1400 characters)	no digits

Accordingly, u08 is a data type of length 8 bits and represents an unsigned (positive) integer.

Numbers (Constants)

By the help of the data type, numbers and constants can be declared in the Enertex® EibStudio.

For numbers, the number is preceded by the type of data, thus e. g.

● 2u08	Positive 8-bit-integer: 2
● 2.0f16	Floating point number 2.0
● -6s32	Integer with sign -6
● 33.2%	Percentage 33.2 (equivalent to 84)

Invalid syntax is recognized by the EibParser (integrated compiler in the Enertex® EibStudio) and generates an error message.

In case of unsigned integers with length 8 bits and of floating point numbers of length 16 bits, the specification of data types can be omitted, i.e. values in the form

- 0 ... 255 are of type u08,
- 2.0 (decimal point in number) are of type f16.

For these two types of numbers, the specification of data types is optional.

Special type: % (Percentage)

In the ETS programming, the percentages "%" are used. These are compatible to the data type "u08" and are internally adjusted by the KNX actuators by scaling. Here, to simplify programming, we have defined the percentage for constants. In this context, the percentage may be specified with a decimal point, e. g. 2.3%. Because of the scaling, 100% corresponds to a value of 255u08 or the conversion of a variable Y% is more generally as follows:

$$X[\mathrm{u08}] = \frac{Y[\%]}{100} \cdot 255 \qquad \text{for cutting off the decimal points}$$

The built-in compiler within the Enertex® EibStudio will make those adjustments for you, so that you can address actuators as usual

When different types of data are linked in your application program with each other, e.g. the sum of 2u08 and 2u32, then an error is reported by the integrated compiler in Enertex ® EibStudio. Therefore, accidental overflows, numerical problems, etc. cannot occur. To convert these numbers into yet another, and thus to be able to process them, use the convert function. Hence, even conversions from numbers to strings are possible. For further information, see page 208.

Hexadecimal representation

Unsigned integers (data type „u") also can be given in hexadecimal representation with the prefix "0x". The compiler converts this representation into the respective number.

- Data type u08: Two digits are required 0xF1 (= 241)
- Data type u08: Two digits are required 0xF1u08 (= 241)
- Data type u16: At least two digits and the data type „u16" are required: 0xF1A3u16 (= 61859u16)
- Data type u24: At least two digits and the data type „u24" are required: 0xF1A3u24 (= 61859u24)
- Data type u32: At least two digits and the data type „u32" are required: 0xF1A3u32 (= 61859u32)
- Data type u64: At least two digits and the data type „u64" are required: 0xF1A3u64 (= 61859u64)

Special type: Character string

Character strings are specified in the form

- $String$c14. Here, String represents any text. But this text consists of not more than 14 characters. This type is compatible to the KNX string (e. g.: display elements).
- $String$ (without the addition c14) is the second built-in data type. Here, the String represents any text, but may now consist of up to 1400 characters.

So one should distinguish:
$ Hello $c14: Character string with a maximum of 14 characters
$ Hello $: Character string with a maximum of 1400 characters

The two character strings can be transformed into each other by means of the convert-function (see page 208).

Special type: IP Address

IP addresses (add on Option NP) have the following syntax

- 192.168.22.100. An IP address is of data type u32.

Special type: Pyhsikali Address

Physikal KNX - addresses are defined as followed in the programm code

- 1.12.230. This address is of data type u16.

An overview of the data types

Type	Data type	Example of a constant	Usage	Range	EIS data type
Binary	b01	`1b01`	Switch actuator, sun-blind actuator	0, 1	EIS1/EIS7
2 bit	b02	`2b02`	Lock objects	0,1,2,3	EIS8
4 bit	b04	`10b04`	Dimming	0,1 ... 15	EIS2
Percentage	%	`85.3%`	Heating regulators, actuators	0,1.1 ... 100.0	EIS6/EIS14.001
8 bit integer without sign	u08	`255`	Simple numbers, programmable thermostats, etc.	0, ... 255	EIS6/EIS14.001
8 bit integer without sign	u08	`255u8`	Optional types		EIS6/EIS14.001
8 bit integer with sign	s08	`-45s08`	Temperature sensors	-128... 127	EIS14.000
16 bit integer without sign	u16	`45u16`		0 ... 65535	EIS10.000
16 bit integer with sign	s16	`-450s16`		-32768 ... 32767	EIS10.001
24 bit integer without sign	u24	`292235u24`		0 .. 16777216	EIS11.000
24 bit integer with sign	s24	`-92999s24`		-8388608 .. 8388607	EIS11.001
32 bit integer without sign	u32	`92235u32`		0 .. 4294967295	EIS11.000
32 bit integer with sign	s32	`-9999s32`		-2147483648 .. 2147483647	EIS11.001
64 bit integer without sign	u64	`92235u64`		0 .. 18446744073709551615	n.a.
64 bit integer with sign	s64	`-9999s64`		-9223372036854775808 9223372036854775807	.n.a.
Short float	f16	`4.0`	Wind sensors	-671088.64 .. 670760.96	EIS5
Short float	f16	`4.0f16`		-671088.64 .. 670760.96	EIS5
Float 32 bit	f32	`4.0e01f32`		-3.40282e+38 .. 3.40282e+38	EIS9
String	c14	`$HelloWorld$c14`	Display panels	14 digits	EIS15
String	(c1400)	`$HelloWorld$`	LAN telegrams	1400 digits	n.a.
IP address	(u32)	`192.168.22.100`	Fixed IP addresses at sendudp etc.		EIS11.000

Table 1: Data types

Note:

The data types d24, t24, Y64 are KNX DTP types handled properly by their definition in Enertex® EibPC. An input as a constant is not necessary and therefore not possible. These data types are needed only in connection with the functions getdate and gettime. Further information can be found on 174.

Variables

Variables start with letters, followed by any number and combination of letters or numbers, and the "_" character. Variables are initialized with a value or function. Opposed to keywords and function names, upper and lower case is respected.

Therefore, for example address and Address are different variables.

During the allocation of a variable and its processing, the compiler "EibParser" always checks the data type and prevents improper combinations of incompatible data types by an error message when generating the user program. Therefore, no accidental overflow, numerical problems, etc. may occur.

If you want to combine variables with different data types, use the convert-function (see page).

Each variable must be initialized only once. The declaration of variables must therefore be unique.

Some examples

```
a=123
A1=1b01
address=A1 or 0b01
Address=4%+5%+23u08
Value=4e4*0.2
w=4e16f32
```

Not permissible here...

Variables may not be defined depending on themselves ("recursion"). Therefore, the following expression is invalid as a definition:

```
a=a+1
```

In contrast, it is permissible to program a counter using variables in this way:

... but here

```
//Declaration
a=0
//Counting
if (sun()) then a=a+1 endif
```

No special characters in variable names

Umlauts are not allowed in variable names. Therefore, the following expression **is invalid**

```
KücheLightOn=1b01
```

Conversion

Only variables with a valid data type may be associated. Some functions require different or the same data types as arguments. To link any data types to each other, the convert-function (see page Fehler: Referenz nicht gefunden) has to be used.

For example: An 8 bit unsigned value is to be added with a signed 16 bit value:

```
Var1=10u08
Var2=300s16
// Var3=310
Var3=convert(Var1,Var2)+Var2
```

Implicit conversion

Both Var2, as well as convert(Var1,Var2) are defined with data type s16. This is recognized by the built-in compiler of the Enertex® EibStudio, and therefore it defines the data type of Var3 on its own. This is called **implicit conversion**, since the user does not explicitly set the type of Var3. In any case, the compiler checks the data types of variables and their linkages.

Predefined variables

In order to make the creation of a user program as simple as possible, variables are predefined in the Enertex® EibStudio. These variables are listed in the Enertex® EibStudio within the window Definitions. Predefined variables cannot be overwritten. An overview of the available variables is listed in side 339.

Imported KNX™ group addresses

Suppose, in the ets project you have defined the group address shown in Table 1, and then exported it from the ETS as described above. Finally, you have the group address imported into the Enertex® EibStudio as described in Chapter 144 and want to use this group address in your user program.

	Name	**Address**
Major group	Lighting, Power socket	1
Middle group	Basement	0
Subgroup	BasementWC	0

Table 2: ETS Group address

For this you need to know how such a variable is imported or how it is made accessible in the application program. According to generating law

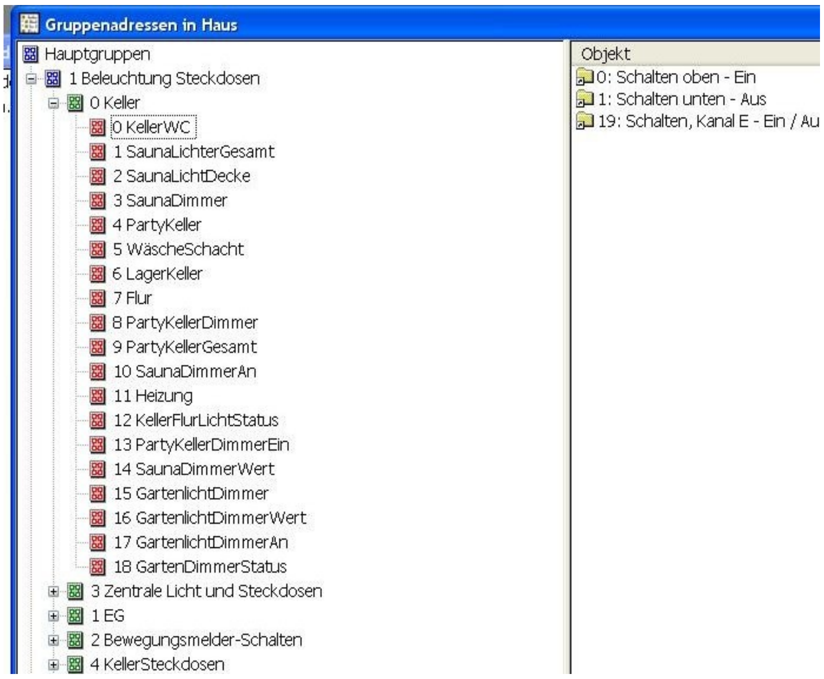
Generating law

Imported address: "Subgroup name-Group address"

the group addresses are automatically internally loaded by the indication of the export file in the section [ETS-ESF]. Thus, if the EIB objects were exported to the ets, you can access them any time.

Example: ETS project and group addresses

Suppose you have defined group addresses in the ETS, as shown in Figure 6 by the partial screenshot.


Gruppenadressen in Haus

Hauptgruppen
- 1 Beleuchtung Steckdosen
  - 0 Keller
    - 0 KellerWC
    - 1 SaunaLichterGesamt
    - 2 SaunaLichtDecke
    - 3 SaunaDimmer
    - 4 PartyKeller
    - 5 WäscheSchacht
    - 6 LagerKeller
    - 7 Flur
    - 8 PartyKellerDimmer
    - 9 PartyKellerGesamt
    - 10 SaunaDimmerAn
    - 11 Heizung
    - 12 KellerFlurLichtStatus
    - 13 PartyKellerDimmerEin
    - 14 SaunaDimmerWert
    - 15 GartenlichtDimmer
    - 16 GartenlichtDimmerWert
    - 17 GartenlichtDimmerAn
    - 18 GartenDimmerStatus
  - 3 Zentrale Licht und Steckdosen
  - 1 EG
  - 2 Bewegungsmelder-Schalten
  - 4 KellerSteckdosen

Objekt
- 0: Schalten oben - Ein
- 1: Schalten unten - Aus
- 19: Schalten, Kanal E - Ein / Au

Figure 6: Group address of a ets project

Using the import function, the addresses

- "BasementWC-1/0/0"
- "TotalSaunaLights-1/0/1"
- "SaunaDimmer-1/0/3"

Copy&Paste of imported addresses

etc. will be made available in the Enertex® EibStudio within the address window (see Figure 6). If you click on these addresses with the left mouse button, you can use the address by copy&paste.

Import of the ESF file

If you want to use these group addresses and their definitions in the Enertex® EibStudio, you must first export the group addresses as explained on page 151 and then insert the export file into the section [ETS-ESF] with path and name (on this, see example page 151).

Within the programming of the Enertex® EibPC in section [EibPC] you can access the group address by using the following syntax:

```
[EibPC]
a="BasementWC-1/0/0"
```

This can be done simply by copying the addresses in the window area of the Enertex® EibStudio and then paste them in the text window user program.

Defects in ets

Unfortunately, the ets does not export the appropriate numeric type for all data types (where the bit length is always given correctly.)

When compiling, the Enertex® EibStudio recognizes which addresses have not been fully defined by the ets and are therefore incomplete. However, based on the specifications of the data types, the compiler may conclude to the data type. Therefore, in the following example, the compiler recognizes that the variable *a* must be of the data type u08 (percentage).

```
[EibPC]
b=5%
a="Dimmer-5/1/0"+b
```

Implicit conversion

Explanation:

A sum may be made only with the same data types. Since b is clearly of type percentage, both parts of the sum must be of type percentage. Therefore, "Dimmer-5/1/0" is clearly to point as a percentage and thus the data type for the group address is indirectly (implicitly) defined. The user needs not further define the data type. However, if the ets-export has set a data type s08, of course the compiler delivers an error message. The same holds if the bit length does not match - in each case, this is indeed displayed correctly by the ets export function.

Unnecessary group addresses

If you want to make only certain addresses available to your customer, simply edit the ESF file directly in a text editor and delete the lines with the unneeded group addresses. If you deem it necessary or do not have access to the original ETS records, you can also directly edit the names of the group addresses. However, be careful not to accidentally generate mistakes in this approach.

“Manual“ group addresses

Besides the possibility to use group addresses by using the ets project data, you can define any group address itself without having to resort to the ets Now, you must only use the following notation:

Manual address: 'Group address'Data type

Group addresses without using the ets begin with a single quote, followed by the major group/middle group/subgroup (in numerical format), followed by a single quote and the data type, as was shown in Table 1.

Example:

```
'1/0/0'u08
'1/0/1'b01
'5/0/81's16
```

In the example above, the first group address 1/0/0 is of the type of an unsigned integer with 8 bits in length, the address 1/0/1 is of a binary type and 5/0/81 is of the type of a signed integer with 16 bits length. The simultaneous use of imported and manual addresses is possible at any time.

The if-statement

As could already be found within the step-by-step introduction (see page 32), the if statement, thus the conditional statement, is the central point of automation. The syntax of an if statement is as follows:

```
if (query condition)   then Statement endif
if (query condition)   then Statement1 else Statement2 endif
```

The query condition must be of type b01. If the query condition is not of this type, the integrated compiler of the Enertex® EibStudio issues an error.

Multiple statements can also be executed as a block. In this case, the instructions are separated by semicolons point. **After the last statement, there is no semicolon.**

A query condition is considered true when it takes the value ON (1b01). In this case, the then-branch is not executed. If the if query contains an else branch, so this is considered for non-compliance, i.e. the value of the query condition is OFF (0b01),

Example

```
if (a==23u08) then write('1/2/3's16,-234s16) else write('1/2/3's16,0s16)  endif
```

For the query condition of the if statement applies the validation scheme as described for the variables on the following page 162. This means that an if statement is evaluated only (once) when the query condition changes. If a assumes the value 23u08, then the then branch is executed once. In the subsequent cycle, if a does not change, the if function does not run again. If a changes, but the query condition is not met, the else branch is once performed at each of these changes. For further information refer to page 162 or the step-by-step introduction.

Reading from and writing to the KNX™ bus

Assigning variables and group address – Reading from the KNX™ bus

By means of a simple assignment, you can use the information sent by a sensor to a group address in a variable or function, i.e. with the statement *variable = Group address* the information which is sent to the *group address* is assigned to the *variable* in the Enertex® EibPC. As described above, the *group address* can be imported or "manually" defined in the Enertex® EibPC

Therefore, the use of variables and group addresses (imported or manual) is compatible, i.e. you can, instead of variables, use group addresses as arguments of functions.

Example Assign a "manual" KNX™ group address

```
a='1/0/0'u08
b='1/0/0'u08 + '1/0/1'u08
c= a+ '1/0/1'u08
```

In the example above, the contents of the group address 1/0/0 (more accurately: The contents of the user data of the telegram with this group address), which has the type of an unsigned integer, is assigned to the variable a. The variable b is the sum of the two values sent via the group addresses 1/0/0 and 1/0/1. The variable c is identical in content to the variable b, so you can use group addresses as variables.

The Enertex® EibPC always saves the current state of the contents of the group address. If the Enertex® EibPC is restarted, this must be taken into account. In this case, the Enertex® EibPC does not know previously sent telegrams and initializes the state with 0 (0.0, etc depending on the types of data.

Default and systemstart()

With the function systemstart(), see page 213, variables, which are defined depending on group addresses, can be preallocated with certain values at boot time.

A variable depending on a bus telegram in some way is only evaluated if the value which arrives in the bus telegram changes.

Example

```
c= 1u08+ '1/0/1'u08
if systemstart() then c=2u08 endif
if (c==2u08) then write('3/3/3'b01,EIN) endif
if change(c) then write('3/3/2'b01,EIN) endif
```

The Enertex® EibPC initializes the evaluation of the telegram with the group address 1/0/1 to 0, thus c is initialized with 1u08. By the statement if systemstart() c is set to 2u08. When a new telegram arrives on the bus to the group address 1/0/1, then c is re-evaluated.

Then the Enertex® EibPC verifies whether - due to the change of the variable c - further variables and other statements depending on c must also be re-evaluated. In the example, this is the case. When the value 1u08 is sent to the group address 1/0/1, the comparison in the if statement issues 1b01, and thus the Enertex® EibPC writes a telegram with a value of 1b01 to the group address 3/3/3 on the bus. The change function (page 208) checks whether a variable has changed in the last process cycle. In this case, its return value is ON (1b01). Therefore, the last if statement is always executed when the variable changes, and thus a telegram is written to group address 3/3/2 with the value ON (1b01).

The validation scheme

The Enertex® EibPC processes only changing terms and remembers the last state which was sent via a group address. The mapping of states of the received telegrams and variables is called validation scheme. It makes the operation and programming easy and intuitive, as long as one avoids nesting of statements are avoided. As shown in the example, this is always possible to avoid in a simple manner.

We advise beginners to avoid nested if statements.

Please, also refer to the example of the read function on page 165.

Example: Assigning imports group addresses

```
a="BasementWC-1/0/0"
b="SaunaLight-1/0/1"
c=max("BoilerRoom1-1/0/2","BoilerRoom2-1/0/3","BoilerRoom3-1/0/4")
d=min(c,"BoilerRoom3-1/0/4")
```

a and *b* are allocations of imported group addresses, which are of the data type defined in the ETS.

In order to process different numeric data types, you need the convert function (see page 208). The variable c in the example uses the max function, which is the maximum of any number of variables or group addresses.

The data types of group addresses have to be equal (arguments of the max function). Here, the variable c is of the data type of the arguments of the max function and therefore its data type is determined by the arguments of the max()-function (see also page 158)

The variable d is the minimum of the variables c and the imported group address "BoilerRoom3-1/0/4".

Writing to the KNX™ bus: write()

Writing information to the KNX™ bus is realized with the help of the write function.

Definition

- write(*GroupAddress, Value*)

Arguments

- 2 arguments of the same data type, but otherwise the data types are arbitrary..
- *GroupAddress*: Imported or manual KNX™ group address
- *Value*: The value which is to be written to the KNX™ group address (via the KNX™ bus)

Effect

- A valid KNX™ which writes the *value* to the *group address* is sent to the bus.

Data type result (return value)

- none

Example

```
write("BasementWC
write('1/0/1'u08,10%) endif
```

Note: The data types "u08" and "%" are equivalent and compatible (see also page 153).

Read request of a group address: read()

The read request of the value of an actuator using the corresponding group address from the KNX™ bus is realized by means of the read function.

Note:

The flag in the ETS program must also be set so that the actuator in the KNX™ network responds.

Definition

- read(*GroupAddress*)

Arguments

- *GroupAddress*: Imported or manual KNX™ group address
- The groupaddress can be optionally negated using the !-Sign.

Effect

- A valid KNX™ telegram with the "read-flag" set is sent to the bus. Confirm, that the actors are parameterized properly (set read flag).

Data type result (Return)

- none

Example: Querying the actual temperature from the bus

A temperature sensor can send a temperature value in floating point format f16 (16 bit) to the address 2/3/4. The bit "read request" is set in the ets, i.e. the temperature can be retrieved via a read request..
Every day at 18:30 clock and 20 seconds, the variable should be obtained from temperature sensor.

Implementation:

```
Temperature='2/3/4'f16
if chtime(18,30,20) then read('2/3/4'f16) endif
```

By means of the command *Variable = Group address* the information, which is sent to the group address triggered by the read function, is assigned to a variable.

Overall, the process of the example can be illustrated in Figure 7.

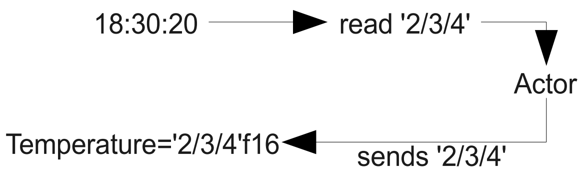


Figure 7: Operation of read

Once the time has been reached 18:30:20, chtime goes to ON, the condition of the if statement is true and the read sends the read request. Now the actuator responds and sends the value to the group address '2/3/4'f16.

Note:

Instead of using read('2/3/4'f16) it is possible to code with the invert-sign read(!'2/3/4'f16).

Initialize Groupaddresses

Before the Enertex® EibPC starts processing the user program, the user might want to initialize the images of the group addresses. The Enertex® EibPC always saves the current state of the contents of the group addresses as a kind of image in memory (see also gaimage() on p. 234). If started all group address images are set to 0, but as the KNX Bus is already running before the EibPC starts with processing, theses memory images will not hold the real state if they are different form zero (which will be most likely the case).

Using the [InitGA] Section

In order to synchronize with the KNX bus, some Group addresses have to be read by the EibPC. You can achieve this by using the section [InitGA] as follows:

Example

```
[InitGA]
// Temperature manually defined
'2/3/4'f16
“Heating-2/3/4”
“Lights-2/3/2”
[EibPC]
if “Lights-2/3/2” and '2/3/4'f16<10.0 then write(“Heating-2/3/4”,100%) endif
```

Notes

- You can use the context menu on the group address windows to enable initializing a group address
- You can easily write any valid group address in this section. If you use manually defined group addresses (as '2/3/4'f16 above), this group address must be really used at least once in your user program
- Initializing the group addresses means, the EibPC sends read telegrams to the group addresses. Each Groupaddress will be sent after waiting for a response with a timeout of 1500 ms.
- After the last read telegram is initiated, the EibPC starts processing the user program.
- All statements which include a group addresses listed in the section [InitGA] are invalid and therefore run at systemstart.
- If a group address is not responding to initGA, an event is generated.

Example 2

```
[InitGA]
“Lights-2/3/2”
[EibPC]
if “Lights-2/3/2” write(“Heating-2/3/4”,100%) endif
if !“Lights-2/3/2” write(“Heating-2/3/4”,0%) endif
```

Normally, on systemstart each instruction is set to valid.

So neither this

```
if “Lights-2/3/2” write(“Heating-2/3/4”,100%) endif
```

nor that

```
if !“Lights-2/3/2” write(“Heating-2/3/4”,0%) endif
```

is running.

Through the use of [InitGA] all group addresses listed in this section are invalid an therefore run at system start.

In the example above is written to the GA „Lights-2/3/2“ a read request. Depending on the answer is either the first or second statement executed.

Initga

Definition

- initga(*GroupAddress*)

Arguments

- *GroupAddress*: Imported or manual KNX™ group address
- The groupaddress can be optionally negated using the !-Sign.

Effect

- The effect of this function is same as if the *GroupAddress* was listed in the [InitGA]-section.
- The function can be used top-level only, which means, that it can not be used in a then or else branch of an if-query.
- The function can also be used in related to the function comobject (p. 208)

Data type result (Return)

- none

Alternatively to the syntax above the following is possible, too:

Example

```
[EibPC]
// Temperature manually defined
initGA('2/3/4'f16)
initGA("Heating-2/3/4")
initGA("Lights-2/3/2")
if "Lights-2/3/2" and '2/3/4'f16<10.0 then write("Heating-2/3/4",100%) endif
```

Example 2 - comobject

The following example shows the use in combination with the function comobject.

```
[EibPC]
initga(!"Licht KG Treppe-0/0/2")
initga(comobject("Licht EG -Decke Flur-0/0/14","Licht EG Speis-0/0/18"))
```

Both the use of negations and the function comobject are possible combined with the function initga. This has significant advantages of the programming of macros.

Bus-Activity:

Event

This function always responds when a telegram is written for the monitored address on the bus. It does not respond to variables.

In connection with UDP, TCPIP or RS232 telegrams, it reacts identically to the arrival of UDP messages (see page 235 and page 64, respectively) in the step-by-step introduction.

An event function is defined as follows:

Definition

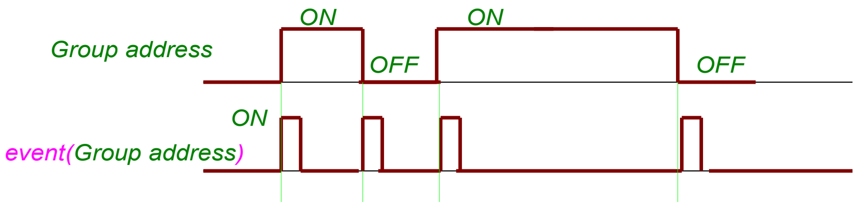
- Function event(*Group address*)

Arguments

- *Group address*: Imported or manual KNX™ group address
- The groupaddress can be optionally negated using the !-Sign.
- For UDP, TCPIP or RS232 telegrams (see page 135) the event function can be applied to the function readudp.

Effect

- Return value: 1b01 (ON pulse) when a telegram with the group address is on the KNX™ bus, regardless of its content.



Data type results (Return value)

- Data type b01

One special characteristic of the event functions is that this function may not be placed at if statements with else-branch. The event-function is only switched to ON for one processing cycle and will be execute the then-branch of the if-statement on the arrival of a telegram to the group address. In the next cycle, event returns to OFF and now the else branch is executed. To simplify programming, here the use of the event function is limited by the compiler (see page 43).

An example of using the event function.

Whenever the address "MotionDetector-3/2/3" or "MotionDetector-3/2/4" gets an event, the variable light is set to ON. After 3 minutes, the variable light should be reset to OFF.

The reaction is then:

```
if (event(“MotionDetector-3/2/3“)) or (event(!“MotionDetector-3/2/4“)) then Light=EIN endif
if(after(Light,30000u64)==EIN) then Light=AUS endif
```

The monitoring of bus activity to a group address will be realized with the help of the event function. For deeper analysis of the KNX telegrams the event-Functions described on the next pages can distinguish

1. a normal write,
2. a read
3. a response to a preceeding read.

EventRead

Definition

- Function eventread(*Group address*)

Arguments

- *Group address*: Imported or manual KNX™ group address
- The group address can be optionally negated using the !-Sign.

Effect

- Return value: 1b01 (ON pulse) when a Read-telegram with the group address has been written on the KNX™ bus, regardless of its content.

Data type results (Return value)

- Data type b01

Eventresponse

Definition

- Function eventresponse(*Group address*)

Arguments

- *Group address*: Imported or manual KNX™ group address
- The group address can be optionally negated using the !-Sign.

Effect

- Return value: 1b01 (ON pulse) when an answer to a Read-telegram with the group address has been written on the KNX™ bus, regardless of its content.

Data type results (Return value)

- Data type b01

Eventwrite

Definition

- Function eventwrite(*Group address*)

Arguments

- *Group address*: Imported or manual KNX™ group address
- The groupaddress can be optionally negated using the !-Sign.

Effect

- Return value: 1b01 (ON pulse) when an write-telegram with the group address has been written on the KNX™ bus, regardless of its content.

Data type results (Return value)

- Data type b01

Writersponse

Definition

- Function writeresponse(*Group address,value*)

Arguments

- *Group address*: Imported or manual KNX™ group address
- *Value:* The value which is to be written to the KNX™ group address (via the KNX™ bus)

Effect

- Responds to a read request by a valid telegram generated by KNX™ which writes the *value* to the *group address* is sent to the bus. The response flag is set in the telegram.

Data type results (Return value)

- none

Scene actuators

Scene actuators are also as special KNX™ functions. For further information see the step-by-step instructions on page 62 and on page 217.

Commands and functions

Note:

For all arguments or functions, the group addresses can also be used directly instead of variables.

Logical operators

AND-links

To create AND-links, the and instruction is provided. This statement is constructed as follows:

Definition

- *A* and *B* [and *C* ... etc.]

Arguments

- All arguments (*A, B, C* ...) are of the same data type. But otherwise, the data types are arbitrary.
- Any number of links

Effect

- The variable *A* is bitwise "ANDed" with the variable *B* (and the variable *C* etc.). The result of the operation and is zero (all bits), if one of the variables is zero (all bits). In the other case the result is a bitwise "ANDing", i.e. the n-th bit of the result is zero, once one of the bits of the input is zero. Otherwise, the n-th bit of the result is 1, i.e. each n-th bit of the two (or more) input variables is 1.

Data type result (Return)

- Data type of the arguments

Example: AND-Link

LightActuatorOn is the result of the AND operation of variable ButtonOn and variable LightRelease.

The implementation of the user program is then:

```
LightActuatorON = ButtonOn and LightRelease
```

If *ButtonOn* is 1b01 and *LightRelease* is 1b01, then LightActuatorOn is 1b01, otherwise it is 0b01.

Example: And-Link with different variables

If the variable ButtonOn is '1' and the variable wind speed is exactly 2.9 m/s, the variable LightActuatorOn has to be set to '1'.

For the implementation, we need the if statement and the comparison ==. (here, the whole if-query is to be set in parentheses). The implementation is then:

```
if ((ButtonOn==1u08) and (WindSpeed==2.9f16)) then LightActuatorOn=1u08 endif
```

OR-links

To create OR-links, the or statement is provided. This statement is organized as follows:

Definition

- *A* or *B* [or *C* ... etc.]

Arguments

- All arguments (*A, B, C* ...) are of the same data type. But otherwise, the data types are arbitrary.
- Any number of links

Effect

- The variable *A* is bitwise "ORed" with the variable *B* (and the variable *C* etc.), which means: The result of the operation or is zero, if both of the variables are zero. In the other case the result is a bitwise "ORing", i.e. the n-th bit of the result is one, once one of the bits of the input is one.

Data type result (Return)

- Data type of the arguments

Example: OR-link

LightActuatorOn is the result of the OR operation of variable ButtonON and variable LightRelease

The implementation is then:

```
LightActuatorOn = ButtonOn or LightRelease
```

If *TButtonOn* is 1b01 or *LightRelease* is 1b01 or both are 1b01, then *LightActuatorOn* is 1b01, otherwise it is 0b01.

Example: OR-link with different variables

If the variable ButtonOn is '1' or the variable WindSpeed is exactly 2.9 m/s, the variable LightActuatorOn is set to '1'.

For the implementation, we need the if statement and the comparison ==. Here, the entire if-query is set in parentheses. Then, the implementation reads:

```
if ((ButtonOn==1u08) or (WindSpeed==2.9f16)) then LightActuatorOn=1u08 endif
```

Exclusive-OR-links

To create exclusive-or-links ("either or"), the xor instruction is provided. This statement is constructed as follows:

Definition

- A xor B [xor C ... etc.

Arguments

- All arguments (*A, B, C* ...) are of the same data type. But otherwise, the data types are arbitrary.
- Any number of links

Effect

- The variable *A* is bitwise "XORed" with the variable *B* (and the variable *C* etc.), which means: the result of the operation xor is zero (bitwise), if both of the variables are zero or one. In the other case, the n-th bit of the result is one, if **only one** of the bits of the input is one.

Data type result (Return)

- Data type of the arguments

Example: XOR-Link

If either KEY1 (type b01) or KEY 2 (type b01) is pressed, the LightActuatorOn is to go to 1b01. If both are 0b01 and 1b01, LightActuatorOn is to go to 0b01.

The implementation is then:

```
LightActuatorOn = KEY1 xor KEY2
```

Comparison operators

To create Comparison-Links the following instructions are provided

Definition

- *A > B* greater
- *A < B* less
- *A == B* equal
- *A >= B* greater than or equal
- *A =< B* less than or equal
- *A !=B* not equal

Arguments

- 2 arguments (*A, B*) are of the same data type.
- Data types: uXX,sXX,fXX, with XX arbitrary bit lengths defined on page 154.

Effect

- The variable *A* is compared with the variables *B* – depending on the operator:

  The result of the operation > is 1b01, if the variable A is greater than variable B.

  The result of the operation < is 1b01, if the variable A is less than variable B.

  The result of the operation == is 1b01, if the variable A has the same value as the variable B.

  The result of the operation >= is 1b01, if the variable A is greater than or equal to the variable B.

  The result of the operation =< is 1b01, if the variable A is less than or equal to the variable B.

  The result of the operation != is 1b01, if the variable A does not have the same value as the variable B.

  In all other cases the result is 0b01.

Data type result (Return)

- Data type b01

Hysteresis-comparison

Definition

- Function hysteresis(*Var*,*LowerLimit*,*UpperLimit*)

Arguments

- 3 arguments (*Var*,*LowerLimit*,*UpperLimit*) of the same data type.
- Data types: uXX,sXX,fXX, with XX arbitrary bit lengths, defined on page 154.

Effect

- The argument *Var* is compared with the *LowerLimit* and *UpperLimit* of a hysteresis function.
- If the last comparison led to a result 0b01 and (*Var*≥*UpperLimit*) is true, the function assumes the value 1b01.
- If the last comparison led to a result 1b01 and (*Var*≥*LowerLimit*) is true, the function assumes the value 0b01.

Data type Result (Return)

- Data type b01

Example: Temperature-controlled shading

If a temperature actuator (Group address 1/3/4, data type f16) reports a temperature warmer than 25°C, the shading on the group address 4/5/77 should go to ON.
Only if the temperature falls below 23°C again, the shading is to boot again.

Implementation in the user program:

```
if hysteresis('1/3/4'f16,23f16,25f16) then write('4/5/77'b01,ON) \\
                                 else write('4/5/77'b01,OFF) endif
```

Inverting

For inverting binary values (data type b01), the following syntax is available

Definition

- *!A*

Arguments

- Argument *A* is of the data type b01

Effect

- The variable *A* is inverted.
  The result of the operation is 1b01, if the variable A is 0b01
  The result of the operation is 0b01, if the variable A is 1b01

Data type result (Return)

- Data type b01

Example: Inverted button

LightActuatorOn (b01) is to behave inversely to KEY1 (b01).

The reaction is then:

```
LightActuatorOn = !Button1
```

If *KEY1* is 1b01, then *LightActuator* is 0b01. If *KEY1* is 0b01, then *LightActuator* is 1b01.

Shift

The following function is available for shifting numeric data types:

Definition

- shift*(Operand, Number)*

Arguments

- Argument *Operand* of any numerical data type
- Argument *Number* of data type s08

Effect

- Arithmetic shift of the operand by *number*. With positive number shift to the left, with a negative number to the right. The number of bits of the number of the input is shortened.

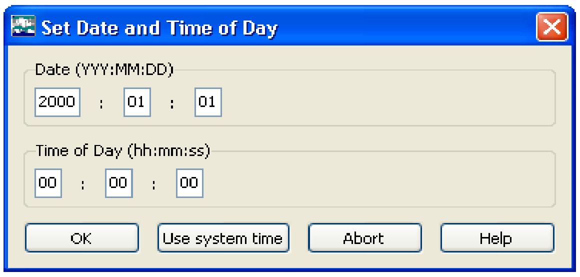
Data type result (return)

- as *Operand*

System time

Setting the time and date of the Enertex® EibPC manually

If you have no Internet connection, you can manually enter the time and date of the Enertex® EibPC. By EibPC – Time of Day and Date you reach the dialog shown in Figure 1. You can enter the current date and time in the dialog and press *OK*. Alternatively, you can press the button *Use system time* to set the system time of the PC, on which Eibstudio runs, equal to the system time of the EibPC.


Set Date and Time of Day

Date (YYY:MM:DD)
2000 : 01 : 01

Time of Day (hh:mm:ss)
00 : 00 : 00

Figure 1: Enter date and Time manually

Synchronization with an Internet timeserver (NTP)

The Network Time Protocol (NTP) is a standard for synchronizing clocks in computer systems via the Internet. The local clock is synchronized by external timing signals which are directly sent from an atomic clock of an NTP server. The synchronization takes place as soon as an Internet connection to the address pool.ntp.org exists.

Synchronization with the KNX™ bus

The system clock of the Enertex® EibPC can be controlled from the outside and can be read. Thus, actuators etc. within the KNX™ bus can be synchronized with the Enertex® EibPC or the system clock of the Enertex® EibPC can be adjusted.

There is both the possibility of sending data from the Enertex® EibPC to the KNX™ bus as well as data from the KNX bus to the Enertex® EibPC.

Moreover, there is the possibility to read and change the system time in the application program by means of variables. Please note that the timer functions can be disturbed by setting or changing, respectively, the system time.

Synchronization with the application program

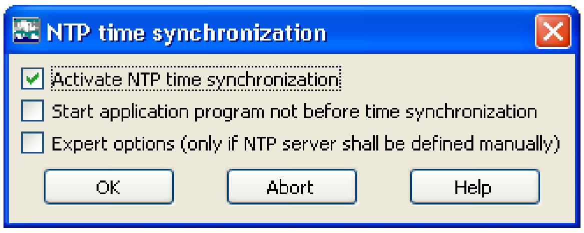
Concerning synchronization, the NTP server always takes precedence, i.e. if it is connected, the time is automatically re-written. You also have the possibility to indicate up to nine NTP server.

The time synchronization can be inhibited by Options – NTP time synchronization. (Fig. 2).

Therefore it is necessary to determine whether the application program to be started only after the time synchronization or immediately after the EibPC has started. If no NTP Server is reached after five minutes, the application program starts in any case.

By default, starts the application program when the EibPC is operational.

Every 10 minutes the NTP server synchronized.


NTP time synchronization

[✓] Activate NTP time synchronization
[ ] Start application program not before time synchronization
[ ] Expert options (only if NTP server shall be defined manually)

Figure 2: Enter NTP settings

Reset of the system time of the Enertex® EibPC

Definition

- Function gettime(*address*) with:

Arguments

- 1 Argument of data type t24

Effect

- The system clock of Enertex® EibPC is overwritten with the time stored in *address* and thus reset.

Data type result (Return)

- none

Note:

1. There is no assignment of the form *a*=gettime(*b*) possible (error message).
2. The function will only be executed, if the function is in a then or else branch of an if instruction.

Example: gettime

Weekly on Sunday at 00:00 clock, the system clock is to be synchronized with a radio clock existing in the KNX™ bus and to be reset.

Implementation in the user program:

```
if(cwtime(0,0,0,0)) then read(“RadioClock-1/2/1“) endif
if event (“RadioClock-1/2/1“) then gettime(“RadioClock-1/2/1“) endif
```

By the read function, a read request to the group address will be generated. The information which is then sent to the KNX bus is written into the system clock of the Enertex® EibPC by the gettime function.

Writing the system time of the Enertex® EibPC to the KNX™ bus

Definition

- Function settime()

Arguments

- none

Effect

- The system time is read from the Enertex® EibPC and assigned to a variable as a value. Return value is the current time in DPT format.

Data type result(Return)

- Data type t24

Example 1: settime

On the 1st of each month, the group address "WallClock-4/3/5" and the variable time are to be synchronized with the system clock (and thus be reset).

Implementation in the user program:

```
if (day(1)) then write(„WallClock24,settime()) endif
if (day(1)) then time=settime() endif
```

Reset of the date of the Enertex® EibPC

Definition

- Function getdate(*Address*) with:

Arguments

- 1 Argument of data type d24.

Effect

- The system clock of the Enertex® EibPC is overwritten with the time stored in *address* and thus reset.

Data type result (Return)

- none

Note:

1. There is no assignment of the form *a*=getdate(*b*) possible (error message).
2. The function will only be executed, if the function is in a then or else branch of an if instruction.

Example: GetDate

All six months, the system date is to be synchronized with a radio clock existing in the KNX bus and to be reset.

Implementation in the user program:

```
if (month(1,1) or month(1,7)) then read(“RadioClock-1/2/2“) endif
if event (“RadioClock-1/2/2“) then getdate(“RadioClock-1/2/2“) endif
```

Writing the date of the Enertex® EibPC to the KNX™ bus

Definition

- Function setdate()

Arguments

- none

Effect

- The system date is read from the Enertex® EibPC. The return value is the time in the format of type d24

Data type result (Return)

- Data type d24

Example: SetDate

On the 1st day of each year, the address "Date-3/5/3" is to be synchronized with the date of the Enertex® EibPC and to be reset.

Implementation in the user program:

```
if (month(1,1)) then write(“Date-3/5/3“d24, setdate()) endif
```

Reset of the time and the date of the Enertex® EibPC

Definition

- Function gettimedate(*address*) with:

Arguments

- 1 argument of data type y64

Effect

- The system clock and the system date of the Enertex® EibPC are overwritten with the time and the date stored in *address* and thus reset.

Data type result (Return)

- none

Note:

1. There is no assignment of the form *a*=gettimedate(*b*) possible (error message)
2. The function will only be executed, if the function is in a then or else branch of an if instruction.

Example: GetTimeDate

Every six months, the system time and the system date is to be synchronized with a radio clock existing in the KNX™ bus and to be reset.

Implementation in the user program:

```
if (month(1,1) or month(1,7)) then read(“RadioClock-1/2/3“) endif
if event (“RadioClock-1/2/3“) then gettimedate(“RadioClock-1/2/3“) endif
```

Writing the time and the date of the Enertex® EibPC to the KNX™ bus

Definition

- Function settimedate()

Arguments

- none

Effect

- The system time and system date are read from the Enertex® EibPC and assigned to a variable as a value

Data type result (Return)

- Data type y64

Example: SetDate

On the 1st day of each year, the address "RadioClock-1/2/1" is to be synchronized with the system time and the system date of the Enertex® EibPC and to be reset.

Implementation in the user program:

```
if (month(1,1)) then write(“RadioClock-1/2/1“d24, settimedate()) endif
```

Hour

Definition

- Function hour()

Arguments

- none

Effect

- The system time (hour) is stored in a variable

Data type result (Return)

- Data type u08

Example:

Stop watch see page 178

Minute

Definition

- Function minute()

Arguments

- none

Effect

- The system time (minute) is stored in a variable

Data type result (Return)

- Data type u08

Example:

Stop watch see page 178

Second

Definition

- Function second()

Arguments

- none

Effect

- The system time (second) is stored in a variable

Data type result (Return)

- Data type u08

Example:Stop watch

Timing the seconds at which the variable Stopper_Go has the value ON. A c1400 text string shall be given that prints the time in the format 000d:000h:000m:000s (days, hours, minutes, seconds).

Here the implementation, at which the seconds can be found in the variable *Stopper_time* and the formatted output in *Stopper*. Cf.example Stop watch V2 on page 224).

Stringformat for a formatted output/conversion

```
[EibPC]
Stopper=$$
Stopper_start=0s32
Stopper_time=1s32
Stopper_Go=AUS

// Start the stop watch (calculate offset)
if (Stopper_Go) then {
        Stopper_start=-convert(hour(),0s32)*3600s32-convert(minute(),0s32)*60s32-
convert(second(),0s32)
} endif
if change(dayofweek()) then Stopper_start=Stopper_start+86400s32 endif

// End of stop time
if !Stopper_Go then {

Stopper_time=convert(hour(),0s32)*3600s32+convert(minute(),0s32)*60s32+convert(second(),
0s32)+Stopper_start;
        Stopper=stringformat(Stopper_start/86400s32,0,3,3,3)+$d:$+\\
            stringformat(mod(Stopper_start,86400s32)/3600s32,0,3,3,3)+$h:$+\\
            stringformat(mod(Stopper_start,3600s32)/60s32,0,3,3,3)+$m:$+\\
            stringformat(mod(Stopper_start,60s32),0,3,3,3)+$s$
} endif
```

Changehour

Definition

- Function changehour(*arg*)

Arguments

- *arg*, Data type u08

Effect

- The system time (hour) is set to the value of *arg*.
- Please note that the timer functions can be disturbed by setting or changing, respectively, the system time.
- If your Enertex® EibPC establishes an NTP connection, the time is reset again (cf. note on page 174).

Data type result (Return)

- none

Changeminute

Definition

- Function changeminute(*arg*)

Arguments

- *arg*, Data type u08

Effect

- The system time (minute) is set to the value of *arg*.
- Please note that the timer functions can be disturbed by setting or changing, respectively, the system time.
- If your Enertex® EibPC establishes an NTP connection, the time is reset again (cf. note on page 174).

Data type result (Return)

- none

Changesecond

Definition

- Function changesecond(*arg*)

Arguments

- *arg*, Data type u08

Effect

- The system time (second) is set to the value of *arg*.
- Please note that the timer functions can be disturbed by setting or changing, respectively, the system time.
- If your Enertex® EibPC establishes an NTP connection, the time is reset again (cf. note on page 174).

Data type result (Return)

- none

Utc

Definition

- function utc(*Zeit*)

Arguments

- *time* as string in format $*YYYY-MM-DD HH:MM:SS*$, data type c1400

Effect

- Converts the time value in YYYY-MM-DD HH:MM:SS format back into a UTC-value. This value is compatible to the Unix time stamp, this value is shown instead of seconds in milliseconds.

Data type result (return)

- u64

Utctime

Definition

- function utctime()

Arguments

- none

Effect

- Shows the number of elapsed milliseconds since 1970-01-01 00:00:00. This function is compatible to the Unix time stamp.

Data type result (return)

- u64

Utcconvert

Definition

- Function utcconvert(*utc*)

Arguments

- *utc* time in ms, data type u64

Effect

- Converts the time specification from the utc format into a c1400 string format YYYY-MM-DD HH:MM:SS.

Data type result (return)

- string

Example: UTC Transformation

Conversion in the user program:

```
// Gibt aktuelle Zeitangabe im UTC-Format zurück
utcZeit=utctime()

// Convertiert UTC-Format in YYYY-MM-DD HH:MM:SS
DateTime=utcconvert(1364826122000u64)

// Wandelt 2012-09-03 20:00:17 in UTC-Format um
utcZ=utc($2012-09-03 20:00:17$)
```

Data control

Date comparison

A date comparison is defined as follows:

Definition

- Function date(*dd,mm,yyy*) with:
  dd: Day (1..31)
  mm: Month (1=January, 12=December)
  yyy: Years Difference (0..255) from year 2000

Arguments

- All of the data type u08

Effect

- The output is 1b01, if the date is reached or already passed. If the date is before the set value, the output goes to 0

Data type Result (Return)

- Data type b01

Example: Date comparison timer

On 01 October 2009 the variable a is to be set to 1u08.

Implementation in the user program:

```
if date(10,1,09) then a=1 endif
```

Monthly comparison

A monthly comparison is defined as follows:

Definition

- Function month(dd,mm) with:
  dd: Day (1..31)
  mm: Month (1=January, 12=December)

Arguments

- 2 arguments are of data type u08

Effect

- The output is 1b01, if the date is reached or already passed. If the date is before the set value, the output goes to 0b01. With the beginning of a new year (January 1) the output goes to 0b01, until the month and day reach the set value.

Data type Result (Return)

- Data type b01

Example: Monthly comparison timer

Every year on 01 December, the variable ChristmasLightingOn is to be set on 1.

Implementation in the user program:

```
if month(1,12) then ChristmasLightingOn=1 endif
```

Example: Definition of variable “summer“

A variable summer shall be defined, which is 1b01 (On) from 1.5. until 30.9. of each year.

Implementation in the user program:

```
Summer=month(01,05) and !month(30,09)
```

Daily comparison

A daily comparison is defined as follows:

Definition

- Function day(*dd*) with:

  dd: Day (1..31)

Arguments

- Argument of data type u08

Effect

- The output is 1b01 when the day is reached or already passed. If the day is before the set value, the output goes to 0b01. With the beginning of a new month, the output goes to 0b01 until the day meets the set value.

Data type result (Return)

- Data type b01

Example: Day timer comparison

Every 6th in the month, the variable SprinklerOn is to be set to 1.

The implementation in the user program then reads:

```
if day(6) then SprinklerOn=1 endif
```

DayOfWeek

Definition

- Function dayofweek() with:

Arguments

- none

Effect

- The output returns the current day of the week [0{Sunday}..6{Saturday}.

Data type result (Return)

- Data type u08

Example: Day timer comparison

Request the current day of the week. In case it is Sunday, the variable SprinklerOn is to be set to 1.

The implementation in the user program then reads:

```
if dayofweek()==SUNDAY then SprinklerOn=1 endif
```

Easter Day

Definition

- Function easterday(*Offset*)

Arguments

- Argument *Offset* Data type s16

Effect

- Calculate the day of Easter Sunday. An offset for the calculation is indicated, e.g. Easter Sunday +40 days, Easter Sunday - 30 days.

Data type result (Return)

- Data type u08

Eastermonth

Definition

- Function eastermonth(*Offset*)

Arguments

- Argument *Offset* Data type s16

Effect

- Calculate the month of Easter Sunday. An offset for the calculation is indicated, e.g. Easter Sunday +40 days, Easter Sunday - 30 days..

Data type result (Return)

- Data type u08

Example: Calculation of Ash Wednesday; (Ash Wednesday is 46 days before Easter Sunday:)

```
uAschermittwochTag=easterday(-46s16)
uAschermittwochMonat=eastermonth(-46s16)
```

Shading and the position of the sun

Sun - Day or night?

The function sun returns whether it is day or night. It requires the Enertex® EibPC's knowledge of the longitude and latitude of the concerned location.

These can be entered in Enertex® EibStudio.

Definition

- Function sun()

Effect

- Return Value: The return value is 1 binary, if it is day and 0 binary, if it is night.

Data type result (Return)

- Data type b01

Example 2: Solar altitude

If it is day, the variable SunblindsOn should be set to 0.

The implementation in the user program is then:

```
if (sun()==1b01) then SunblindsOn=0 endif
if (sun()==BRIGHT) then SunblindsOn=0 endif
```

“BRIGHT“ is a predefined variable (see page 339) with the binary value 1b01 and hence can be stated as a comparison operator instead of 1b01.

Azimuth

Definition

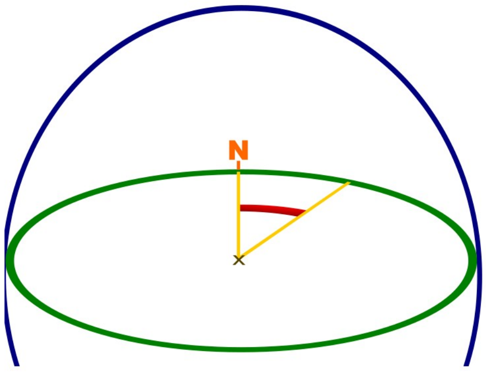
- Function azimuth()

Arguments

- None. However, the Enertex® EibPC should know the longitude and latitude of the place. These can be entered in Enertex® EibStudio (see page 184).

Effect

- This function cyclically (time frame: 5 minutes) calculates the azimuth of the sun in degrees, north through east.



(Source: Wikipedia)

Data type (Return)

- Data type f32

Example 3: Calculate azimuth

Calculate the azimuth angle of the sun for the location of the Enertex® EibPC every 5 minutes.

The implementation in the user program then reads:

```
AAngle=azimuth()
```

Note:

This function is needed in house awnings. In the library EnertexBeschattung.lib you will find detailed examples.

Elevation

Definition

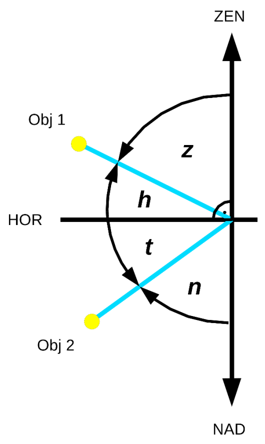
- Function elevation()

Arguments

- None. However, the Enertex® EibPC should know the longitude and latitude of the concerned location. These can be entered in Enertex® EibStudio (see page 184).

Effect

- This function cyclically (time frame: 5 minutes) calculates the elevation angle of the sun in degrees.



(Source: Wikipedia)

Data type result (Return)

- Data type f32

Example 4: elevation

At 5:00, calculate the elevation angle of the sun at the location of the Enertex® EibPC.

The implementation in the user program then reads:

```
HAngle=0f32
if htime(5,00) then HAngle=elevation() endif
```

Note:

This function is needed in house awnings. In the library EnertexBeschattung.lib you will find detailed examples.

Presun

Definition

- Function presun(*hh,mm*)

  hh: hours (0... 23)

  mm: minutes (0... 59)

Arguments

- two arguments of data type u08

Effect

- Shows 1 to 0 in the specified time before chancing from day to night. The programm has to know the geographics length an width of the specified location.

Data type result (Return)

- Sun position, 1= Day, 0 = Night of data type u08

```
s=$$
if presun(1,30) then s=$Eine Stunde vor Sonnenaufgang$ endif
if !presun(0,20) then s=$20 Minuten vor Sonnenuntergang$ endif
```

Sunrisehour - hour at sunrise

Definition

- Function sunrisehour()

Arguments

- none

Effect

- The hour (0 to 23) at sunrise is returned.

Data type result (Return)

- Data type u08

Sunriseminute - minute at sunrise

Definition

- Function sunriseminute()

Arguments

- none

Effect

- The minute (0 to 59) at sunrise is returned.

Data type result (Return)

- Data type u08

Example: Visualize the sunrise

Write the time at sunrise to the group address 1/4/4 (data type c14).

The implementation in the user program then reads:

```
if htime(sunrisehour(),sunriseminute(),0) then \\
  write('1/4/4'c14, convert(sunrisehour(),$$c14)+$:$c14+convert(sunriseminute(),$$c14))  \\
endif
```

Sunsethour - hour at sunset

Definition

- Function sunsethour()

Arguments

- none

Effect

- The hour (0 to 23) at sunset is returned.

Data type result (Return)

- Data type u08

Sunsetminute - minute at sunset

Definition

- Function sunsetminute()

Arguments

- none

Effect

- The minute (0 to 59) at sunset is returned.

Data type result (Return)

- Data type u08

Example: see the above example "visualize the sunrise"

```
if htime(sunsethour(),sunsetminute(),0) then \\
  write('1/4/4'c14, convert(sunsethour(),$$c14)+$:$c14+convert(sunsetminute(),$$c14))  endif
```

Time switch

Basics

Time switches are functions which change their return value from OFF to ON and then back to OFF upon entering the specified time of day for one processing cycle of the Enertex® EibPC. Time switches are objects which trigger regular activities, for example every night at 1:00 clock the garage lighting turns off etc.

To facilitate the application, we distinguish four types of time switches:

- The weekly time switch which triggers one action per week,
- the daily time switch which runs one action every day,
- the hourly time switch which is active once hourly, and finally
- the minute time switch which triggers one action per minute.

To perform the action, the time switches have to reach exactly the specified time. This should be considered when programming. As the reference time for all time switches, the system time of the Enertex® EibPC is used (see also page 149), which is given the Enertex® EibPC either by the Internet or via a KNX system device.

Weekly time switch

Definition

- wtime(*hh,mm,ss,dd*) with:
  hh: Hour (0..23)
  mm: Minutes (0..59)
  ss: Seconds (0..59)
  dd: Day (0=Sunday, 6=Saturday,7=Weekdays, 8=Weekends)

Arguments

- 4 arguments are of data type u08

Effect

- The return value is 0b01, if the current time and date of the Enertex® EibPC's system clock are not equal to *hh*:*mm*:*ss* and *dd*. When the time is reached (and matches exactly), the output value rises to 1b01 (if the time is exceeded, it returns to 0b01).

Data type result (Return)

- Data type b01

Example: Weekly time switch

Every Tuesday at 01:00 Clock, 30 seconds, the variable LightActuatorOn is set to 0b01.

Implementation in the user program:

```
if wtime(TUESDAY,01,00,30) then LightActuatorOn=0b01 endif
```

Note:

For the days weekend and weekday constants (written in capitals) are defined (MONDAY, TUESDAY, WEEKDAYS, WEEKENDS, etc.)

Daily time switch

Definition

- htime(*hh,mm,ss*) with:
  hh: Hour (0..23)
  mm: Minutes (0..59)
  ss: Seconds (0..59)

Arguments

- 3 Arguments are of data type u08

Effect

- The return value is 0b01, if the current time of Enertex® EibPC-system clock is not equal to *hh*:*mm*:*ss*. When the time is reached (and matches exactly), the output value rises to 1b01 (if the date is exceeded, it returns to 0b01).

Data type result (Return)

- Data type b01

Example: Daily timer

Every day, 22:04 Clock, 7 seconds, the variable LightActuatorOn is to set '0'.

Implementation in the user program:

```
if htime(22,04,07) then LightActuatorOn=0b01 endif
```

Hourly time switch

The hourly timer is defined as follows:

Definition

- mtime(*mm,ss*) with:
  mm: Minutes (0..59)
  ss: Seconds (0..59)

Arguments

- 2 arguments are of data type u08

Effect

- The return value is 0b01, if the current minute-second-time of the Enertex® EibPC's system clock is not equal to *mm*:*ss* (the hour is not relevant). When the time is reached (and matches exactly), the output value is set to 1b01 (if the date is exceeded, it returns to 0b01).

Data type result (Return)

- Data type b01

Example: Example hour time switch

Every hour, always 22 minutes, 7 seconds after a full hour, the variable LightActuatorOn will be set to '0'.

Implementation in the user program:

```
if mtime(22,07) then LightActuatorOn=0b01 endif
```

Minute time switch

The minute timer is defined as follows:

Definition

- stime(*ss*) with:
  ss: Seconds (0..59)

Arguments

- 1 argument is of data type u08

Effect

- The return value is 0b01, when the current second-time of the Enertex® EibPC's system clock is not equal to *ss* (hour and minute are not relevant). When the time is reached (and matches exactly), the output value is set to 1b01 (if the date is exceeded, it returns to 0b01).

Data type result (Return)

- Data type b01

Example: Example minute time switch

Always after 34 seconds after a full minute, the variable WindowContacts should be set to '0'.
Always after 5 seconds after a full minute, the variable should be set to '1'.

Implementation in the user program:

```
if stime(34) then WindowContacts=0 endif
if stime(5) then WindowContacts=1 endif
```

Comparator time switches

Basics

Comparator time switches are objects that allow a time comparison. Depending on the result of the comparison, a bus telegram can then be initiated, for example, every night from 1:00 to 6:00 the garage lights are turned off. If the set time is reached, they are 1b01 until the next day, in contrast to the time switches, which jump only at the exact time to 1b01 and immediately after back to 0b01. Thus, comparison time switches are very similar to the more common timers, but have the advantage, that the time must be not be reached accurately (e. g. power failure, reboot).

As the reference time for all comparator time switches, the system time of the Enertex® EibPC is used (see also page 149), which is given the Enertex® EibPC either by the Internet or via a KNX system device.

To facilitate the application, we distinguish four types of comparator time switches:

- The weekly comparator time switch which triggers one action per week,
- the daily comparator time switch which runs one action every day,
- the hourly comparator time switch which is active once hourly, and finally
- the minute comparator time switch which triggers one action per minute.

Weekly comparator time switch

A weekly comparator time switch is defined as follows:

Definition

- cwtime(*hh,mm,ss,dd*) with:
  ss: Seconds (0..59)
  mm: Minutes (0..59)
  hh: Hours (0..23)
  dd: Day (0 = Sunday, 6 = Saturday)

Arguments

- 4 arguments are of data type u08

Effect

- The return value is 0b01, if the current time and day of Enertex® EibPC's system clock are not equal to *hh*:*mm*:*ss* and *dd*. When the time is reached, the output value rises to 1b01 and remains at this value until the following Sunday, 00:00:00.

Data type result (Return)

- Data type b01

Example: Week comparator time switch

Every week from Tuesday at 01:00 Clock, 30 seconds, the variable LightActuatorOn is to be set to '0'. With the beginning of a new week, the variable should be set back to '1'.

Implementation in the user program:

```
if cwtime(01,00,30,THUSDAY) then LightActuatorOn=0 else LightActuatorOn=1 endif
```

Note:

1. For the days weekdays and weekend, constants are defined (written in capitals), e. g.
```
if cwtime(01,00,30,WEEKEND) then LightActuatorOn=0 else LightActuatorOn=1 endif
```
2. cwtime and WEEKDAYS returns a constant values of 1b01.

Daily comparator time switch

A daily comparator time switch is defined as follows:

Definition

- chtime(*hh,mm,ss*) with:
  ss: Seconds (0..59)
  mm: Minutes (0..59)
  hh: Hour (0..23)

Arguments

- 3 arguments are of the data type u08

Effect

- The return value is 0b01, when the current time of the Enertex® EibPC's system clock is not equal to *hh*:*mm*:*ss*. When the time is reached, the output value is set back to 1b01 and remains at this value until the next day (i.e. 00:00:00).

Data type result (Return)

- Data type b01

Example: Daily comparator time switch

Every day from 22:04 Clock, 7 seconds, the variable LightActuatorOn is set to '0'. With the beginning of a new day, the variable is set back to '1'.

Implementation in the user program:

```
if chtime(22,04,07) then LightActuatorOn=0 else LightActuatorOn=1 endif
```

Hourly comparator time switch

A hourly comparator time switch is defined as follows:

Definition

- cmtime(*mm,ss*) with:
  ss: Seconds (0..59)
  mm: Minutes (0..59)

Arguments

- 2 arguments are of the data type u08

Effect

- The return value is 0b01, if the current minute-second-time of the Enertex® EibPC's system clock is not equal to *mm*:*ss*. When the time is reached, the output value is set to 1b01 and remains at this value until the next hour.

Data type result (Return)

- Data type b01

Example: Hour comparator time switch

Every hour, always after 22 minutes, 7 seconds, the variable LightActuatorOn is set to '0'. On the hour, the variable should be set back to '1'.

Implementation in the user program:

```
if cmtime(22,07) then LightActuatorOn=0 else LightActuatorOn=1 endif
```

Minute comparator time switch

A minute comparator time switch is defined as follows:

Definition

- cstime(*ss*) with:
  ss: Seconds (0..59)

Arguments

- 1 argument of the data type u08

Effect

- The return value is 0b01, when the current second-time of the Enertex® EibPC's system clock is not equal to *ss*. When the time is reached, the output value is set on 1b01 and remains at this value until the next minute.

Data type result (Return)

- Data type b01

Example: Minutes comparator time switch

Always after 34 seconds after a full minute, the variable WindowContacts is to be set to '0'. At the beginning of a new minute until it reaches the preset time, the variable should be set to '1'.

Implementation in the user program:

```
if cstime(34) then WindowContacts=0 else WindowContacts=1 endif
```

Special time functions

Precision timer - programmable delay

With the help of delay and after, very short time constants can be generated, as needed for example in the control of motion detectors (light duration, debounce against restart) or certain control algorithms. The Enertex® EibPC responds even in the microsecond range.

The minimum delay time is 1 ms, the maximum adjustable delay time is approximately 30 years.

Delay

Definition

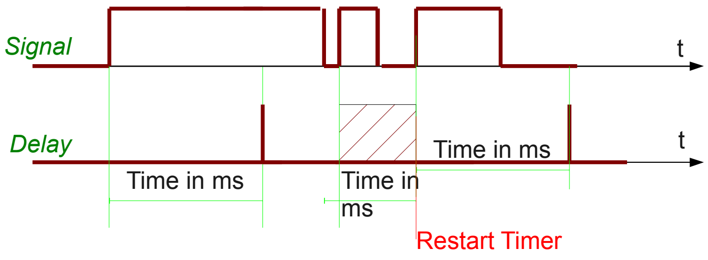
- Function delay(*Signal*, *Time*)

Arguments

- Argument *Signal* of the data type b01
- Argument *Time* of the data type u64

Effect

- The function starts a timer at the transition of the variable *signal* from OFF to ON and sets the return value of the function for one cycle to ON, if the time delay is reached.



- When a new OFF-ON pulse occurs during the internal timer is running, the timer restarts.

Data type result (Return)

- Data type b01

Note:

- Do not use delay in the then or else branch of an if statement.
- If the delay (using an if statement and a write) writes a telegram, there can arise an additional delay time of a few ms - depending on the bus load and the bus speed.

Example: Delayed variable assignment

If the variable LightActuator (Date type f16) is less than 1000f16, the variable light (data type b01) is to go to *ON* after 10s for 1 cycle

Implementation in the user program:

```
Light=!delay(LightActuator<1000f16,10000u64)
```

Example: Delayed variable assignment

If LightButton (Type b01) is *ON*, the variable LightActuator (Type b01) is to go to *ON* after 1300 ms.

Implementation in the user program:

```
if delay(LightButton,1300u64) then LightActuator=1b01 endif
```

Alternative 1

```
if delay(LightButton==1b01,1300u64) then LightActuator=1b01 endif
```

Alternative 2

```
if (delay(LightButton,1300u64)==1b01) then=1b01 endif
```

Note that "LightActuator" is only set, but not deleted. See also the following example.

Example: Switch off delay

If the LightButton (data type b01) is *OFF*, the variable LightActuator is to go to *OFF* after 4000 ms.

Then, the implementation in the user program reads:

```
if (delay(LightButton==OFF,4000u64)) then LightActuator=0b01 endif
```

Example: Different On- and Off-delay

If LightButton (data type b01) is *ON*, the variable LightActuator (data b01) is to go to *ON* after 1300 ms. If LightButton (data type b01) is *OFF*, the variable LightActuator (data b01) is to go to *OFF* after 4000 ms.

Implementation in the user program:

```
if (delay(LightButton==ON,1300u64)) then LightActuator=ON endif
if (after(LightButton==OFF,4000u64)) then LightActuator=OFF endif
```

Delayc

Definition

- Function delayc(*Signal*, *Time, xT*)

Arguments

- Argument *Signal* of the data type b01
- Argument *Time* of the data type u64
- Argument *xT* of the data type u64

Effect

- Works as delay (p. 192).
- The remaining time of the internal timer can be read with variable *xT.*

  CAUTION: If you use the same variable *xT* for different delayc in the programm code, a non predictable behavoir will be the consequence.

Data type result (Return)

- Data type b01

Note:

- Do not use delayc in the then or else branch of an if statement.
- If the delayc (using an if statement and a write) writes a telegram, there can arise an additional delay time of a few ms - depending on the bus load and the bus speed.

Example: Delayed variable assignment

If LightButton (Type b01) is *ON*, the variable LightActuator (Type b01) is to go to *ON* after 1300 ms. The remaining time starting from the change to *ON* til end of the 1300ms period will be written to address '2/2/2' every 300 ms.

Implementation in the user program:

```
xT=0u64
debug='2/2/2'u64
if delayc(LightButton,1300u64,xT) then LightActuator=1b01 endif
if (change(xT/300u64)) then write('2/2/2'u64, xT) endif
```

After

Definition

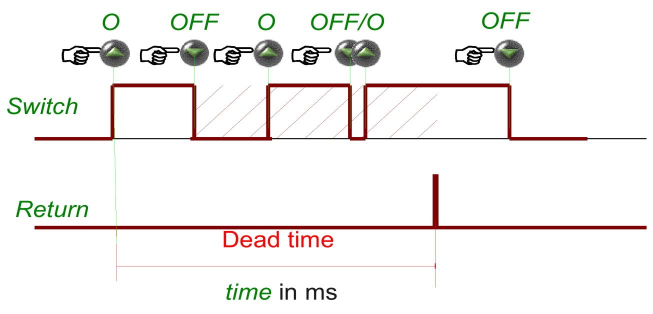
- Function after(*Signal*,*Time*)

Arguments

- Argument *Signal* is of data type b01
- Argument *Time* is of data type u64

Effect

- The function starts a timer at the transition of the variable *signal* from OFF to ON and sets the return value of the function for one after to ON, if the time delay is reached.



- During the dead time interval the function is blocked, i.e. new incoming pulses are ignored.

Data type result (Return)

- Data type b01

Note:

- If the after (using an if statement and a write) writes a telegram, there can arise an additional delay time of a few ms - depending on the bus load and the bus speed.

Example: On- and Off-delay

The variable light sensors (data type b01) is to follow the variable LightButton (data type b01) after 1000 ms.

Implementation in the user program:

```
LightActuator = after(LightButton,1000u64)
```

Example: On-delay

If LightButton (data type b01) is *ON*, the variable LightActuator (data type b01) is to be set to *ON* after 1300 ms.

Implementation in the user program:

```
if (after(LightButton,1300u64)==1b01) then LightActuator=1b01 endif
```

Alternative 1

```
if after(LightButton==1b01,1300u64) then LightActuator=1b01 endif
```

Alternative 2

```
if after(LightButton,1300u64) then LightActuator=1b01 endif
```

Note that "LightActuator" is only set to 1b01 (ON), but not re-set to 0b01 (OFF). See also the following example.

Example: Off-delay

If the LightButton is (data type b01) is *OFF*, the variable LightActuator is to be set after 4000 ms.

Then, the implementation in the user program is :

```
if (after(LightButton==OFF,4000u64)) then LightActuator=0b01 endif
```

Example: Different On- and Off-delay

If LightButton (data type b01) is *ON*, the variable LightActuator (data type b01) is set to *ON* after 1300 ms, if LightActuator (data type b01) is *OFF*, the variable LightActuator (data type b01) is set to *OFF* after 4000 ms.

Implementation in the user program:

```
if (after(LightButton==ON,1300u64)) then LightActuator=ON endif
if (after(LightButton==OFF,4000u64)) then LightActuator=OFF endif
```

Example: Double assignment of a button:

See page 63.

Afterc

Definition

- Function afterc(*Signal*,*Time,xT*)

Arguments

- Argument *Signal* is of data type b01
- Argument *Time* is of data type u64
- Argument *xT* of the data type u64

Effect

- Works exactly as after (p. 193).
- The remaining time of the internal timer can be read with variable *xT.*
  CAUTION: If you use the same variable *xT* for different delayc in the programm code, a non predictable behavoir will be the consequence.

Data type result (Return)

- Data type b01

Note:

- If the afterc (using an if statement and a write) writes a telegram, there can arise an additional delay time of a few ms - depending on the bus load and the bus speed.

Example: On-delay

If LightButton (data type b01) is *ON*, the variable LightActuator (data type b01) is to be set to *ON* after 1300 ms. The remaining time starting from the change to *ON* til end of the 1300ms period will be written to address '2/2/2' every 300 ms.

Implementation in the user program:

```
xT=0u64
if (afterc(LightButton,1300u64)==1b01,xT) then LightActuator=1b01 endif
if (change(xT/300u64)) then write('2/2/2'u64, xT) endif
```

Cycle timer - cycle

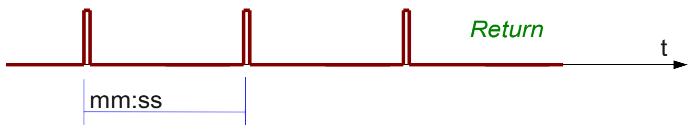
Definition

- Function cycle(*mm,ss*) with:
  mm: minutes (0...255)
  ss: seconds (0..59)

Arguments

- 2 arguments *mm,ss* of the data type u08

Effect



- The return value is periodically set to 1b01 for one processing cycle, otherwise it is 0b01. The repetition time is defined in mm:ss (minutes:seconds).

Data type result (Return)

- Data type b01

Example: Cycle

Always after 1 minutes and 5 seconds a, read request is to be sent to the address "Light1-0/0/1".

Implementation in the user program:

```
if cycle(01,05) then read("Light1-0/0/1") endif
```

Remanent memory

You can use the Flash-Memory of the Enertex® EibPC to store variables. Therefore 1000 memory cells are provided, which can store variables of each data type. This memory is touched neither by firmware updates nor by hardware resets nor by transfering patches and nor by changeing the application program.

Storing data of a variable in a flash memory cell stores only binary data and not the type of the variable. So, when data is red from the flash memory cell and wrote back into a variable you must pay attention to keep the data type of the variable, which was stored previous in the flash memory cell, equal to that, in which the value is wrote back. Every flash memory cell contains 1400 Bytes. The number of variables, which can be stored in the Flash-Memory, depends on the data type or their bit length, respectively, of the stored variables (see page 154).

Pay atteintion to the following note:

> **Writing on the flash memory can lead to errors in the file system on the Enertex EibPC®, when the application program is faulty, or when voltage supply is interrrupted, or a similar problem occurs. By the way writing on the flash many times stresses the flash memory and shortens its life time.**
>
> *To avoid errors in the file system of the Enertex EibPC®, Enertex® Bayern GmbH provides the patch 1.100, which runs a checkdisk on the flash memory at each start of the EibPC. Because this patch re-initializes the flash memory, it can only be transferred into the Enertex EibPC® in the factory. Contact Enertex® Bayern GmbH for further details.*
>
> **Without the patch 1.100 we do not guarantee for file system errors, if the customer applies the function to store variables into the flash memory. We will prove it via a log function, whether this function was applied or net.**

Readflash

Definition

- Function readflash(*Variable*, *Flash memory cell)*

Arguments

- *Variable* arbitrary data type
- *Flash memory cell* of data type u16. Valid values are from 0u16 to 999u16

Effect

- The data of the flash memory cell (Number 0u16 to 999u16) is red and wrote to the variable *Variable* until the memory cell of the variable *Variable* is full (see bit length on page 154). The return value is 0b01, when the read process was successful. If the read process failed, the function returns 1b01.

Data type result (Return)

- Data type b01

  (The return value goes asynchronous to the main development loop - see p. 124)

Writeflash

Definition

- Function writeflash(*Variable*, *Flash memory cell*)

Arguments

- *Variable* arbitrary data type
- *Flash memory cell* of data type u16. Valid values are from 0u16 to 999u16

Effect

- The binary data of the variable *Variable* is stored in the flash memory cell at the position (Number 0u16 to 999u16). The return value is 0b01, when the write process was successful. If the write process failed, the function returns 1b01.

Data type result (Return)

- Data type b01

  (The return value is asynchronous to the main development loop - see p. 124)

Example:

At system start ten 1400 byte strings (c1400) should be wrote on the first ten flash memory cells and afterwards they should be read again. If problems occur during writing or reading, then an error message should be displayed at the group address '8/5/2'c14.The result of the read process should be also wrote at the group address.

```
[EibPC]
a=$: No$
nr=0u16
read_nok=OFF
write_nok=OFF
new_r=ON
new_w=ON
TestGA='8/5/2'c14

if cycle(0,1) and nr<10u16 then write_nok=writeflash(convert(nr,$$)+a,nr);
nr=nr+1u16;new_w=!new_w endif
if cycle(0,1) and nr>9u16 then {
            read_nok=readflash(a,nr-10u16);
            nr=nr+1u16;
            if (nr<20u16) then new_r=!new_r endif
} endif

if write_nok then write('8/5/2'c14,$W-Err: $c14+convert(nr,$$c14)) endif
if change(new_w) then write('8/5/2'c14,convert(convert(nr,$$)+a,$$c14)) endif

if read_nok then write('8/5/2'c14,$R-Err: $c14+convert(nr-10u16,$$c14)) endif
if change(new_r) then write('8/5/2'c14,convert(a,$$c14)) endif
```

Example 2:

The last value that is sent on the bus should be stored in flash and after a restart automatically sent to the bus.

```
Value=0u08
if change("Wohnküche RTR Modus-5/1/7") then {
            writeflash("Wohnküche RTR Modus-5/1/7",0u16)
} endif
if systemstart() then readflash(Value, 0u16) endif
if after(systemstart(),1000u64) then write("Wohnküche RTR Modus-5/1/7",Value) endif
```

Readflashvar

Definition

- Function readflashvar(*Variable)*

Arguments

- *Variable* arbitrary data type

Effect

- In the built-in flash, the binary data is written back to the memory of the *Variable*, as it can be recorded (see bit length, page 154)). The return value is 0b01 when reading was successful, otherwise 1b01 is returned.
- The reading or de-referencing is performed via the variable name. When the user installs a new program, the variable is overwritten with the last value stored in the flash, regardless of the program changes.

Data type result (Return)

- Data type b01
  (The return value is asynchronous to the main processing loop - see p.124)

Writeflashvar

Definition

- Function writeflashvar(*Variable)*

Arguments

- *Variable* arbitrary data type

Effect

- The binary data of the memory content (see bit length, page 154) of the *Variable* are stored in the built-in flash. The return value is 0b01 if the writing was successful, otherwise 1b01 is returned.
- The writing or referencing is carried out exclusively via the variable name.

Data type result (Return)

- Data type b01
  (The return value is asynchronous to the main processing loop - see p.124)

Example:

The last value of a variable is to be stored in the flash at midnight or before a new user programming is installed and automatically loaded into the variable after a restart.
Note: The predefined variable SHUTDOWN is automatically set to ON by the Enertex® EibStudio before importing a new user program, so that the application is given sufficient time, e.g. to store values to the flash (see p. 213)

```
ValuePowerK1="K1-Wirkenergiezähler (Verbrauch)-14/2/76"
if htime(0,0,0) or SHUTDOWN then {
            writeflashvar(ValuePowerK1)
} endif
if systemstart() then readflashvar(ValuePowerK1) endif
```

Data savings

In addition to the values in the remanent memory, which can addressed with the functions readflash and writeflash. Are additionally archived on the flash of the Enertex® EibPC:

- scene
- Time series of diagramms (“timebuffer”)
- Pictures and datas for the internal webserver
- and (depending on users' choice) the whole project.

These datas can be loaded and seperately saved by the Enertex® EibStudio of Enertex® EibPC.

Project data

If transmitting a program with the option “Uploading project datas shown as plain text to the EibPC” is crossed, so the whole datas will be archived including used macro datas and ESF etc. of the EibStudio to a separate directory in the flash.

At a later date this data could be reconstructed with the menu task DATA/PROJECTDATA UPLOAD AS PLAINTEXT ON EIBPC.

Pictures, Scene, Time series

You can find a up- and downloadmanger in the menu task EIBPC/DATATRANSFER DIALOG.

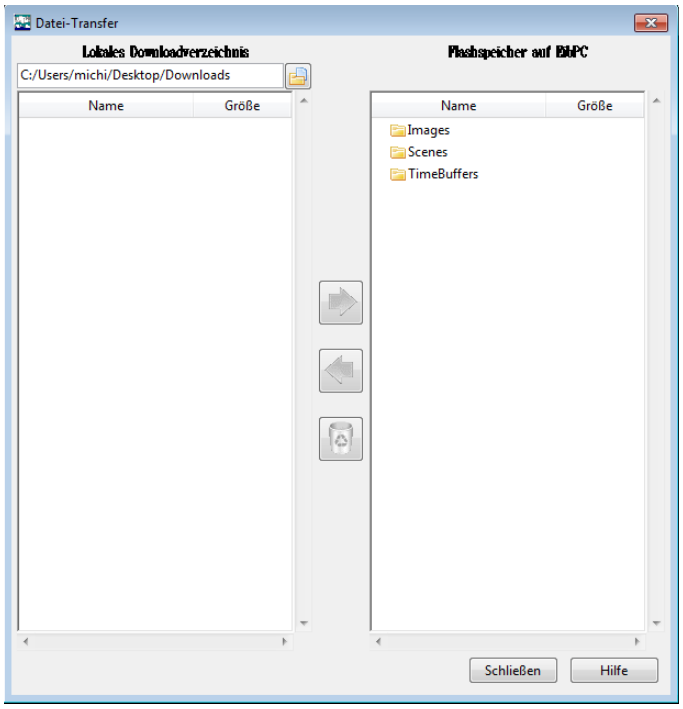


Abbildung 3: Up- und Downloads von Dateien auf den internen Flash

Therewith you can upload e.g. pictures on the Enertex® EibPC, which you can use on the web server. The stored time series from the Enertex® EibPC can parallel be saved on a local pc in a backup data. Enertex® EibPC.

The up- and downloadmanager transfers depending on ending of the data the allocation for the converting on the Enertex® EibPC. Picture data files must have the ending “jpg” or “png”, scene has to follow the intern procedure of the Enertex® EibPC, time series can be identified an allocated by the data name.

I.e. depending on the data name the Enertex® EibStudio identifies automatically the destination directory.

Arithmetic operations ("Calculations")

Basics

Not only (logical and temporal) processes can be programmed by Enertex® EibPC, but also mathematical expressions can be evaluated and hence appropriate responses to the KNX network, e.g. caused by sending of the corresponding addresses, can be produced.

For all the arguments of functions, group address can also be directly used instead of variables.

Absolute value

Definition

- Function abs(*variable*)

Arguments

- Data type: uXX, sXX and fXX, with XX arbitrary bit length defined on page 92

Effect

- Return value: Absolute of *variable*

Data type result (Return)

- Data type of arguments

Example absolute value:

Calculate the absolute value of a (= 2.5f23) and save it as b.

Then, the implementation in the user program is:

```
a=-2.5f32
b=abs(a)
```

Addition

Definition

- *variable1 + variable2 [...]*

Arguments

- All arguments are of the same data type
- Data type: uXX, sXX and fXX, with XX arbitrary bit length defined on page 154

Effect

- The values of the variables are added. Only values of the same type can be added. If you nevertheless want to add e.g. an unsigned 8 bit value and a signed 16 bit value, use the convert function (see page Fehler: Referenz nicht gefunden)

Data type result (Return)

- Data type of the arguments

Note:

With the same syntax, you can concatenate character strings (see page 220).

Arc cosine

Definition

- Function acos(*variable*)

Arguments

- 1 argument *variable* is of data type f32

Effect

- Calculation of the arc cosine of the *variable* given in RAD
- If the argument is greater than 1f32 or smaller than -1.0f32, there is no calculation

Data type result (Return)

- Data type f32

Example arccosine:

In variable b is the result of the arccosine of variable a.

Then, the implementation in the user program is:

```
a=5f32
b=acos(a)
```

Arc sine

Definition

- Function asin(*variable*)

Arguments

- 1 argument *variable* is of data type f32

Effect

- Calculation of the arc sine of the *variable* given in RAD
- If the argument is greater than 1f32 or smaller than -1.0f32, there is no calculation

Data type result (Return)

- Data type f32

Example Arcsine:

```
In variable b is the result of the arcsine of variable a.
```

Implementation in the user program:

```
a=5f32
b=asin(a)
```

Arc tangent

Definition

- Function atan(*variable1*)

Arguments

- 1 argument *variable* is of data type f32

Effect

- Calculation of the arc tangent of the *variable* given in RAD

Data type result (Return)

- Data type f32

Example Arctangent:

```
In variable b is the result of the arctangent of variable a.
```

Implementation in the user program:

```
a=5f32
b=atan(a)
```

Cosine

Definition

- Function cos(*variable1*)

Arguments

- 1 argument *variable* is of data type f32

Effect

- Calculation of the cosine of the *variable* given in RAD

Data type result (Return)

- Data type f32

Example Cosine:

```
In variable b is the result of the cosine of variable a.
```

Implementation in the user program:

```
a=5f32
b=cos(a)
```

Division

Definition

- *variable1 / variable2 [...]*

Arguments

- all arguments are of the same data type
- Data type: uXX, sXX and fXX, with XX arbitrary bit length defined on page 154

Effect

- Calculation of the quotient of Variable1 and Variable2

Data type result (Return)

- Data type of arguments

Example

The flow of the flow temperature should be adjusted independently of the outdoor temperature. In case the outdoor temperature is below 0°C, the flow temperature reaches 55°C. At an outdoor temperature of 30°C, the flow temperature is adjusted to 30°C.

OutdoorTemperature = 15°C

FlowTemperature = 30 + 25/30 * (30 - OutdoorTemperature)

Implementation in the user program:

```
FlowTemperature = 30f16 + 25f16 / 30f16 * (30f16 – "OutdoorTemperature-3/5/0"f16)
```

Average

Definition

- Function average(*variable1*, *variable2, [...]*)

Arguments

- all arguments are of the same data type
- Data type: uXX, sXX and fXX, with XX arbitrary bit length defined on page 92

Effect

- Return value: The average value of the given variables which must all be of the same data type (instead of variables, manual or ets-imported group addresses can be used). The precision of the calculation depends on the data type.

Data type result (Return)

- Data type of arguments

Example: Calculate the average value

The average value of the heating actuators shall be determined.

Implementation in the user program:

```
c=average("HeatingBasement1-1/0/2","HeatingBasement2-1/0/3","HeatingBasement3-1/0/4" /
        "HeatingBasement4-1/0/5","HeatingBasement5-1/0/6")
```

Exponential function

Definition

- Function exp(*variable*)

Arguments

- 1 argument *variable* of data type f32

Effect

- Calculation of the exponential function of *variable*

Data type result (Return)

- Data type f32

Example exponential function:

Variable b is the result of the exponential function of variable a.

Implementation in the user program:

```
a=5f32
b=exp(a)
```

Logarithm

Definition

- Function log(*variable1, variable2*)

Arguments

- 2 arguments of data type f32
- *variable1*: base
- *variable2*: argument

Effect

- Return value: The result of the logarithm calculation
- If the argument and/or the base is not positive, no calculation is performed.

Data type result (Return)

- data type f32

Maximum value

The maximum value function is defined as follows:

Definition

- Function max(*variable1*, *variable2, [...])*

Arguments

- all arguments are of the same data type
- Data type: uXX, sXX and fXX, with XX arbitrary bit length defined on page 92

Effect

- Return value: The maximum value of the given variables which must all be of the same data type

Data type result (Return)

- Data type of arguments

Example: Maximum value of 5 percentage values

The maximum value of the heating actuators shall be determined.

Implementation in the user program:

```
c=max("HeatingBasement1-1/0/2","HeatingBasement2-1/0/3","HeatingBasement3-1/0/4" /
        "HeatingBasement4-1/0/5","HeatingBasement5-1/0/6")
```

Minimum value

The minimum value of an arbitrary number of variables is calculated as follows:

Definition

- Function min(*variable1*, *variable2, [...])*

Arguments

- all arguments are of the same data type
- Data type: uXX, sXX and fXX, with XX arbitrary bit length defined on page 154

Effect

- Return value: The minimum value of the given variables which must all be of the same data type

Data type result (Return)

- Data type of arguments

Example: Minimum value of 5 percentage values

The minimum value of the heating actuators shall be determined.

Implementation in the user program:

```
c=min("HeatingBasement1-1/0/2","HeatingBasement2-1/0/3","HeatingBasement3-1/0/4" /
        "HeatingBasement4-1/0/5","HeatingBasement5-1/0/6")
```

Multiplication

Definition

- *variable1 * variable2 [...]*

Arguments

- all arguments are of the same data type
- Data type: uXX, sXX and fXX, with XX arbitrary bit length defined on page 92

Effect

- The values of the variables are multiplied.

Data type result (Return)

- Data type of arguments

Power

Definition

- Function pow(*variable1, variable2*)

Arguments

- 2 arguments of data type f32
- *variable1*: Base
- *variable2*: Exponent

Effect

- Return value: The result of the power calculation.
- Both variables must be of data type f32. See Table 1 and the explanations there.
- If the base is negative, no calculation is performed.

Data type result (Return)

- Data type f32

Example: *See dew-point calculation, page 59*

Square root

Definition

- Function sqrt(*variable*)

Arguments

- 1 argument of data type f32

Effect

- Square root of *variable. variable* must be of data type f32. See Table 1 and the explanations there.
- If *variable* is negative, no calculation is performed.

Data type result (Return)

- Data type f32

Example Square root:

```
Variable b is the result of the square root of variable a.
```

Implementation in the user program:

```
a=5f32
b=sqrt(a)
```

Sine

Definition

- Function sin(*variable*)

Arguments

- 1 argument of data type f32

Effect

- Return value: Sine of *variable* in radian.

Data type result (Return)

- Data type f32

Example Sinus:

```
Variable b is the sine of variable a.
```

Implementation in the user program:

```
a=4f32
b=sin(a)
```

Subtraction

Definition

- *variable1 - variable2 [...]*

Arguments

- all arguments are of the same data type
- Data type: uXX, sXX and fXX, with XX arbitrary bit length defined on page 92

Effect

- *variable1* is subtracted from *variable2*

Data type result (Return)

- Data type of arguments

Tangent

Definition

- Function tan(*variable*)

Arguments

- 1 argument of data type f32

Effect

- Tangent of *variable*

Data type result (Return)

- Data type f32

Example tangent:

```
Variable b is the tangent of variable a.
```

Implementation in the user program:

```
a=5f32
b=tan(a)
```

Special functions

Change

This function reacts to changes of the supervised address or variable written to the bus.

Definition

- Function change(*variable*)

Arguments

- 1 argument of arbitrary data type

Effect

- Return value: ON, if a change of the supervised address or variable is detected. Reset to OFF after one processing pass of the Enertex® EibPC.

Data type result (Return)

- Data type b01

As a peculiarity, the change function must not depend on if statements with else branch.
Similarly to the event function (see page 168), the change function assumes the value ON only for one processing pass and then executes the then branch of the if function. At the next pass, change returns to OFF, an the else branch would be executed. To make programming easier for the user, the usage of the change function is restricted by the compiler.

The change-Function is activated in next processing cycle of the change of its argument.

Example: Change

If the maximum heating output changes, the flow temperature shall be readjusted.

Implementation in the user program:

```
if change(HeatingMax) then write("FlowTemperature-0/0/1",HeatingNeed) endif
```

Comobject - communication object

Definition

- Function comobject(*variable1*, *variable2, [...])*

Arguments

- all arguments are of the same data type
- Data type: uXX, sXX and fXX, with XX arbitrary bit length defined on page 92

Effect

- Return value: The value of the variable which has changed most recently.

Data type result (Return)

- Data type of arguments

Example: An actuator with multiple variables – determine the status

You want to determine the status of an actuator (1 bit). The actuator is accessed through the group addresses "GA_a-1/2/3","GA_b-1/2/4" and "GA_c-1/2/5".
If the actuator has been switched on for 3 minutes and has not yet been switched off manually, it shall be switched off.

Implementation in the user program:

```
StatusActuator=comobject("GA_a-1/2/3","GA_b-1/2/4","GA_c-1/2/5")
if delay(StatusActuator==EIN,180000u64) and StatusActuator==EIN then write("GA_a-1/2/3",
AUS) endif
```

Convert

Definition

- Function convert(*variable1*, *variable2*)

Arguments

- 2 arguments of arbitrary data type

Effect

- Converts the data type of *variable1* to the data type of *variable2*.
- Any conversion is permitted, with the exception of conversions from b01 to f16 or f32 and vice versa.
- If data type f16 is converted to data type c14 or c1400, the resulting string is a floating point notation with two decimal places.
- If data type f32 is converted to data type c14 or c1400, the resulting string is an exponential notation with two decimal places.
- The value of *variable2* will always be ignored. This argument's sole purpose is the specification of the target data type.

Data type result (Return)

- The result of the conversion from *variable1* to the data type of *variable2*.

Note:

Information may be lost by the conversion of data types, e.g. by the truncation of bits.

Example: Convert function

An unsigned 8-bit value shall be added to a signed 16-bit value.

Implementation in the user program:

```
Var1=10u08
Var2=300s16
Var3=convert(Var1,Var2)+Var2
```

Devicenr

Definition

- Function devicenr()

Arguments

- none

Effect

- Serial number inquery of EibPC

Data type result (Return)

- data type u32

Example: devicenr

The serial number should be assigned to the variable SNR.

Implementation in the user program:

```
SNR=devicenr()
```

Elog

Definition

- Function elog()

Arguments

- none

Effect

- Reading the oldest event stored item.
- After reading the log the entry is deleted.

Data type result (Return)

- data type c1400 string

Example: see example elognum p.210

Elognum

Definition

- Function elognum()

Arguments

- none

Effect

- Returns the number of entries returned in the error memory.

Data type result (Return)

- data type u16

Example: elognum

Read the last event number and reset the memory by one.

Implementation in the user program:

```
EventInfo=$$
EventNr=elognum()
if change(EventNr) then EventInfo=elog() endif
```

Eval

Definition

- Function eval(*arg*)

Arguments

- 1 argument of arbitrary data type

Effect

- The evaluation of the expression will be carried out independently of the validation scheme. This is particularly important for the if-statement when nestings shall be implemented in the usual syntax of C programs.

Data type result (Return)

- Data type of argument

Example: Counter

You want to program a counter which increases a variable by 1 with every processing pass of the Enertex® EibPC until it reaches 100.

Implementation in the user program:

```
Counter=0
if eval(Counter<100) then Counter=Counter+1 endif
```

Note:

Programming with the help of the validation scheme guarantees a stable and optimized event-based processing of the telegrams: An expression/variable/function becomes invalid only on change, so that the Enertex® EibPC **only** processes the expressions depending thereof. The function eval interrupts the validation scheme while processing and hence generates a higher system load.

If you used instead of

```
if '1/0/0'b01 then write('1/0/1'b01,AUS) endif
```

if eval('1/0/0'b01) inadvertently, you could cause your KNX™ installation to crash. We recommend the use of the function eval only to experienced programmers, because the validation scheme is optimized for the Enertex® EibPC and its programming.

A statement

```
if Counter<100 then Counter=Counter+1 endif
```

normally would be executed only once at system start or when setting the variable *Counter* e.g. from 102 to 10 as *Counter*<100 is valid and a further evaluation is not planned.

For nestings, we recommend to use and instead of the function eval, if possible.

Processingtime

Definition

- Function processingtime()

Arguments

- none

Effect

- The EibPC requires a certain amount of time for the processing of its program per cycle. This processing time is returned with this function in ms.

Data type result (Return)

- Processing time in ms as data type u16.

Example:

The max. Duration of processing per second should be visualized in a diagram. The maximum value over all cycles should also be indicated.

```
[WebServer]
page(1) [$Test$,$Processingtime$]
mtimechart(1)[EXTLONG,AUTOSCALE,256,0,10,0,1]($Time in ms $,LEFTGRAF, Buffer0)
[EibPC]
Buffer0=0
timebufferconfig(Buffer0, 0, 3600u16, t)

// per Second
t=0u16
if t < processingtime() then t=processingtime() endif
// Maximum
m=0u16
if m < processingtime() then m=processingtime() endif

// write to chart
if cycle(0,1) then {
		timebufferadd(Buffer0,t);
		t=0u16;
} endif

// Generate some load
y=0f32
if cycle(0,10) then
y=cos(34f32)+sqrt(234f32)+tan(34f32)*7f32+cos(34f32)+sqrt(234f32)+tan(34f32)*7f32+cos(34f
32)+sqrt(234f32)+tan(34f32)*7f32+cos(34f32)+sqrt(234f32)+tan(34f32)*7f32+cos(34f32)+sqrt(
234f32)+tan(34f32)*7f32+cos(34f32)+sqrt(234f32)+tan(34f32)*7f32+cos(34f32)+sqrt(234f32)+t
an(34f32)*7f32 endif
```

System start

Definition

- Function systemstart()

Arguments

- none

Effect

- After transferring a new application program or rebooting the Enertex® EibPC, this function changes from ON to OFF during the first processing pass.

Data type result (Return)

- data type b01

Example: systemstart

At system start time, the variables LightsOff and BlindsUp shall be set to 0b01 once.

Implementation in the user program:

```
if systemstart() then LightsOff=OFF; BlindsUp=DOWN endif
```

End of program

There is no end of the program at the Enertex® EibPC. An Enertex® EibPC program is terminated by either disconnecting the power supply or by the user entering a new program. In the latter case, Enertex® EibStudio sets the built-in variable SHUTDOWN ON so that the appropriate program can be executed in the user program. Enertex® EibStudio then waits 5 seconds before the application program is stopped. Ongoing running of the Flash is still running properly.

Example see p. 199

Random (random number)

Definition

- Function random(max)

Arguments

- 1 argument max of data type u32

Effect

- Returns a random number in the range of 0 to max.

Data type result (Return)

- Data type u32

Example: Turn-on pulse at random time

Every evening at 22:00 plus a random time of up to 3 minutes, the variable BlindsDown shall be set to ON.

Implementation in the user program:

```
// Random number from 0 to 180 (32-bit unsigned)
RandomNumber=convert(random(180u32),0u08)
// Conversion to minutes and seconds
Min=RandomNumber/60
Sec=RandomNumber-Min*60
if htime(22, Min, Sec) then BlindsDown=AUS endif
```

Sleep - passive mode

Definition

- Function sleep(status)

Arguments

- 1 argument status of data type b01.

Effect

- If the input's value is OFF, the Enertex® EibPC sends outbound EIB telegrams and UDP packets to their respective output queue. If the input's value is ON, outbound EIB telegrams and UDP packets are discarded, i.e. they are not sent to their respective output queue. Data which are already located in an output queue are not discarded and are written to the bus or the Ethernet in case of the availability of the respective interface.

Data type result (Return)

- none

Example: Put the Enertex® EibPC to passive mode

You want to put an Enertex® EibPC to passive mode through the group address 2/5/6 (b01).

Implementation in the user program:

```
if '2/5/6'b01 then sleep(EIN) else sleep(AUS) endif
```

Note:

This function is helpful when testing a program in an existing system without actually writing to the bus. Without disrupting users or the program of another Enertex® EibPC, new programs can be tested (the web server can be accessed in the usual way). If the Enertex® EibPC is in passive mode, its internal program runs normally, i.e. variables are being calculated, states changed, the web server adjusted, etc.

Eibtelegramm

This function creates KNX telegrams at lowest application level. For instance, devices can be addressed with their physical address, which is the case of the programming of application data. The Enertex® EibPC internally works in the group message mode and therefore only logs group telegrams sent to a group address. Should other messages (e.g. sent to a physical address) is observed in the integrated bus monitor of EibStudio. In order to watch such telegrams, use the ETS bus monitor with a bus monitor enabled interface. Such interfaces are:

- EIBMarkt IF-RS232
- EIB-IP-Router N146 (Work with ets bus monitor and in routing-mode)
- EIB KNX IP interface PoE (Work with ets bus monitor)

Definition

- Function eibtelegramm(Conntrolfield, Destination, Telegramminfo, data1 ... data18)

Argumente

- Conntrolfield data type u08

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	1	0	W	1	P1	P0	0	0
	1	0	1	1	1	1	0	0
	1*128 + 0*64 + 1*32 + 1*16 + 1*8 + 1*4 + 0*2 + 0*1							
u08 Datentyp	188							

Figure 4: Controlfield of a KNX Telegram

Bit W: Repeat; is normally set to 1.

P1 and P0 define the priority level. Normally a telegram is sent with low priority: P1=P0=1

A normal telegram therefore will have a Conntrolfield : 10111100b = 188u08

- Destination (physical address or group address) with Data type u16

Bit:	16 .. 12	11 .. 8	7 .. 0
Adress	main	middle	low
Expample	1	3	5
Binär:	0001	0011	0101
	1*4096 +	1*512+1*256	+ 0*8+1*4+0*2+1*1
u16-Data type	4869		

Figure 5: Physically Addressing of an Actor with 1/3/5

- Telegraminfo data type u08, split into

a) the type of the given address in Bit 7 (MSB)

value = 0 → physical address

value = 1 → group address

b) routing-Counter Bits 4..6

Counter 7:	A telegram will be sent without change through any coupler
Counter 6..1:	A telegram will be sent through any coupler, but the counter will be decremented by 1 when passing it
Counter 0:	A telegram will not be sent through any coupler

c) The length of the given data Bits 0..3

The length is calculated by the given data and therefore this will be calculated properly by the Enertex® EibPC itself. The given value will be ignored.

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	0	1	1	1	0	0	0	0
	0*128 + 1*64 + 1*32 + 0*16 + 0*8 + 0*4 + 0*2 + 0*1							
U16	112							

Figure 6: Physically Addressing of an Actor with 1/3/5

- date1 .. data18 of data type u08

  Depending on the *Controlfield* the first two bytes e.g. contain the command to run, and in most cases the information to be transmitted.
- For an available commands, please refer to the literature.

Effect

The state of the input objects are copied to an KNX Telegram object. The individual address of the sender can not be given, as It will be set to the address of the bus access unit (= interface connected to the Enertex ® EibPC).

Data type result (Return)

- none

Example: physical Addressing

Every 10 minutes a read request is to be sent to the actuator with the physical address of 1/3/5

```
if cycle(10,0) then eibtelegramm(188u08,4869u16,112u08,0u08) endif
// you could also use hex-values
//if cycle(10,0) then eibtelegramm(0xbc,0x1105u16,0x70,0x00) endif
```

Lighting scenes

Scene actuator - scene

Up to 64 scenes per scene function ("scene actuator") can be stored and recalled. The number of scene functions ("scene actuators") is not limited - only by the number of maximum possible group addresses in the ets.

Stored scenes also persist when interrupting the Enertex® EibPC's power supply or after changing the application program. Only a change of the group addresses relevant to the scenes requires resetting the scenes (menu EIBPC → DELETE SCENES ...).

Definition

- Function scene(*GroupAddressSceneActuator*, *Act1, Act2, ..., ActN*)

Arguments

- *GroupAddressSceneActuator* of data type u08, the other arguments group addresses of arbitrary data types
- *ActXX*, XX from 0 to max. 65000: A group address or variable (see Example presetscene p. 217).

Effect

- A KNX™ scene actuator with the group address defined in *ActXX* (XX 1 to 65000) is implemented. It can be accessed by means of KNX™ switches and an appropriate ETS parametrization or via the below-mentioned functions storescene or callscene.
- You can define an arbitrary number of scene actuators.
- You can preset the scenes with presetscene p. 217.

Data type result (Return)

- none

Note:

1. It is possible to deactivate inputs differently in each scene number, see presetscene p. 217.
2. You can (like other functions) define an arbitrary number of scene actuators.
3. Please see page 62 for detailed application information.

Example: Lighting scenes

You want to realize a scene actuator for a dimmer and a lamp.

Implementation in the user program:

```
scene("SceneActuator-1/4/3"u08, "Dimmer-1/1/2", "DimmerValue-1/1/3", "Lamp-1/1/1")
```

Presetscene

Definition

- Function presetscene(*GroupAddressSceneActuator, SceneNumber, OptionOverwrite, ValVar1,KonfVar1,[ValVar2,KonfVar2,..., ValVarN,KonfVarN])*

Arguments

- *GroupAddressSceneActuator* and *SceneNumber* of data type u08
- *OptionOverwrite* of data type b01
- *ValVarXX* with the same data type as *Variable* respectively *GroupaddressActor* which is defined in function scene
- *KonfVar* of data type b01

Effect

- Create default settings for the sceneactuator with the group address *GroupAddressSceneActuator* and *SceneNumber.*
- If *OptionOverwrite* is set to 1b01, an existing dataset will be overwritten on restart of the programm. By a setting to 0b01, a previously saved scene is not pre-written.
- *SceneNumber* a value 0 to 63 of data type u08, which indicates the szene number, which is to be pre-defined.
- *KonfVarXX,* XX from 0 to max. 65000, indicates, if the corresponding input object is active in this scene number. Active at 1b01, inactive at 0b01. If acitve, the Value *ValVarXX* is the corresponding preset value.

Data type result (Return)

- none

Example: Lighting scenes with presetscene

You want to realize a scene actuator for a dimmer and a lamp.
Also variable Var1 and Var2 shall change.
Scene actuator SceneActuator-1/4/3"u08, number 13 sould be preallocated like this:
- scenes that have been already saved will be overwritten
- the dimmer should be inactive in Szene-number 13
- the lamp an the two variables Var1 and Var2 should be active (send an ON signal to "Lamp-1/1/1" , set Var1 to -20 and Var2 to "scene on")

Implementation in the user program:

```
Var1=123s32
Var2=$scene off$c14

scene("SceneActuator-1/4/3"u08, "Dimmer-1/1/2", "DimmerValue-1/1/3", "Lamp-1/1/1", Var1,
Var2)

presetscene("SceneActuator-1/4/3"u08, 13, ON, ON, OFF, 50%, OFF,ON, ON, -20s32, ON,
$scene on$, ON)
```

Remark:

The functions scene and presetscene are „toplevel", which means independent of an if-condition.
The macro library EnertexScene.lib uses this functions and make the handling of this easier.

Store a scene- storescene

Definition

- Function storescene(*GroupAddressSceneActuator*, *number*)

Arguments

- 2 arguments: *GroupAddressSceneActuator* and *number* of data type u08

Effect

- This function requires the parametrization of a scene actuator to this group address (either KNX™ scene actuators or scene functions).
- The function triggers a telegram to *GroupAddressSceneActuator* and thereby storing the scene *number*.

Data type result (Return)

- none

Example: storescene

You want to store the scene defined in the above example of scene in number 1.

Implementation in the user program:

```
if ButtonStoreScene==ON then storescene("SceneActuator-1/4/3"u08,1) endif
```

Recall a scene - callscene

Definition

- Function callscene(*GroupAddressSceneActuator*, *number*)

Arguments

- 2 arguments: GroupAddressSceneActuator and number of data type u08

Effect

- This function requires the parametrization of a scene actuator to this group address (either KNX™ scene actuators or scene functions).
- The function triggers a telegram to *GroupAddressSceneActuator* and thereby recalling the scene *number*.

Data type result (Return)

- none

Example: Callscene

You want to recall the scene defined in the above example of scene in number 1.

Implementation in the user program:

```
if ButtonRecallScene==EIN then callscene(“SceneActuator-1/4/3“u08,1) endif
```

Stringfunctions

Strings can be defined variable from 1 to 65534 bytes. Thereby the corresponding endpoint has to be specified behind the character string. E.g. a string with the length of 55 bytes will be defined as follows: string= $$c55

The data type c14 will be treated seperately by the compiler because he is compatible with the KNX data type EIS15 and has in contrast to all other strings any zero termination at the end, Gegensatz zu allen anderen Strings keine Nullterminierung am Ende hat, as well as any special characters are not allowed. C.f. the notes to the encoding too on p. 108.

Concatenate

Definition

- *string1* + *string2* [+ *string3* ... *stringN*]

Arguments

- An arbitrary number of arguments, but either all of data type c14 or all of data type c1400.

Effect

- The character strings are concatenated. If the resulting length exceeds the maximum length of the data type, the result is truncated to this length.

Data type result (Return)

- Data type of arguments

Example: Addition of character strings

The character strings string1 and string2 shall be “added“ or concatenated.

Implementation in the user program:

```
string1=$Character$
string2=$String$
// result: “CharacterString”
result=string1+string2
```

Find

Definition

- Function find(*string1, string2, pos1*)

Arguments

- 3 arguments, *string1, string2* of data type c1400, *pos1* of data type u16

Effect

- *string1*: Character string a (partial) character string shall be searched for in.
- *string2*: Character string to be searched for.
- *pos1*: Ignore the first *pos1* incidences of the character string to be searched for.
- The function returns the position of the first character of the found character string (0..1399u16). It returns 1400u16 if the character string has not been found
- For 65534u16, the constant END has been defined.

Data type result (Return)

- Data type u16

Example: Search a character string

In the variable String=$CharacterString$, the character string “String” shall be searched for. No (0) incidences shall be ignored.

If “String” is not found, the variable Error shall be set to 1.

Implementation in the user program:

```
Error
String=$CharacterString$
Find=$String$
Result=find(String,Find,0u16)
if Result==1400u16 then Error=EIN endif
```

Stringcast

Definition

- Function stringcast(*string, data, pos*)

Arguments

- 3 arguments: *string* of data type c1400, *data* of arbitrary data type, *pos* of data type u16

Effect

- *string*: Character string (1400 bytes) a certain number of bytes of which shall be copied to another data type. The number of bytes is defined by the data type of *data*. At this, only the raw data will be copied (cast) and no conversion of the data types is performed.
- *pos*: The position of the 1st character of the character string to be copied to the target type.

Data type result (Return)

- n Bits (n = length of *data* in bytes) from *string*, i.e. raw data are returned.

Example: Conversion of a string into a floating point number

In the variable a=98, the first two bytes character shall be written to a floating point number

Implementation in the user program:

```
a=$98$
z=stringcast(a,0.0,0u16)
// z interprets 0x39 0x38 (ASCII „98“) as „72.9600000“
```

Note:

In connection with stringset and stringcast, c1400 character strings can be used to manage data arrays. See the example of stringset on page 221.

Stringset

Definition

- Function stringset(*string, data, pos*)

Arguments

- 3 arguments: *string* of data type c1400, *data* of arbitrary data type, *pos* of data type u16

Effect

- *string*:Character string one ore more bytes of which shall be replaced.
- *data*: This bytes (= characters) replace characters of *string*.
- *pos*: The position of the bytes in *string* to be replaced. The number of bytes arises from the data type of *data.*

Data type result (Return)

- none

Example: Replace a character sequence

In the variable a=$ nnette$, the 1st character shall be set to 65 =('A').

Implementation in the user program:

```
a=$ nnette$
if systemstart() then stringset(a,65,0u16) endif
```

Example: Create and read a data array

The 15-min-values of the temperature from group address '1/1/1'f16 shall be stored in a data array. At the same time, the temperature difference of the last change shall be extracted from this data array.

The implementation is as follows. Note, the user has to be aware of the byte length of the data.

By means of the debugger (page. 144), you can also view the “raw data” in the data array. However, this should make sense only for integers.

1400 Bytes of the character string can be used.

```
[EibPC]
array=$$
Var='1/1/1'f16
ReadVar=0.0
// Bytessize of f16 == 2
ByteSize=2u16
Pos=0u16

if cycle(15,0) then {
		Pos=Pos+ByteSize;
		stringset(array,Var,Pos);
		if Pos==END then Pos=0u16 endif
} endif
if cycle(15,0) then {
		if (Pos>2u16) then {
				ReadVar=stringcast(array,Var,Pos-ByteSize)-stringcast(array,Var,Pos)
		} endif
} endif
```

String format

Definition

- Function stringformat(*data, conversion_type, format, field_width,[precision]*)

Arguments

- Argument *data* of data type uXX, sXX, fXX with arbitrary XX as defined on page 154.
- Arguments *format, field_width, precision, conversion_type* of data type u08

Effect

- *conversion_type*
  - 0: uXX / iXX → decimal notation
  - 1: uXX / iXX → octal notation
  - 2: uXX / iXX → hexadecimal notation ('x')
  - 3: uXX / iXX → hexadecimal notation ('X')
  - 4: fXX → floating-point notation
  - 5: fXX → exponential notation ('e')
  - 6: fXX → exponential notation ('E')
- *format* defines formatting as follows:
  - 0: (no effect)
  - 1: A blank before a positive number (only permitted if *data* is of data type sXX or fXX and no conversion into octal or hexadecimal notation)
  - 2: A sign before a positive number (only permitted if *data* is of data type sXX or fXX and no conversion into octal or hexadecimal notation)
  - 3: Zero-padded (ignored if *data* is of data type uXX or sXX and a *precision* is given)
  - 4: Zero-padded and a blank before a positive number (only permitted if *data* is of data type sXX or fXX and no conversion into octal or hexadecimal notation; ignored if *data* is of data type uXX or sXX and a *precision* is given)
  - 5: Zero-padded and a sign before a positive number (only permitted if *data* is of data type sXX or fXX and no conversion into octal or hexadecimal notation; ignored if *data* is of data type uXX or sXX and a *precision* is given)
  - 6: Left-justified
  - 7: Left-justified and a blank before a positive number (only permitted if *data* is of data type sXX or fXX and no conversion into octal or hexadecimal notation)
  - 8: Left-justified and a sign before a positive number (only permitted if *data* is of data type sXX or fXX and no conversion into octal or hexadecimal notation)
  - 9: Alternative notation (s. man 3 printf) (only permitted if no conversion into decimal notation)
  - 10: Alternative notation (s. man 3 printf) and a blank before a positive number (only permitted if *data* is of data type fXX)
  - 11: Alternative notation (s. man 3 printf) and a sign before a positive number (only permitted if *data* is of data type fXX)
  - 12: Alternative notation (s. man 3 printf) and zero-padded (only permitted if no conversion into decimal notation; ignored if *data* is of data type uXX or sXX and a *precision* is given)
  - 13: Alternative notation (s. man 3 printf), zero-padded and a blank before a positive number (only permitted if *data* is of data type fXX)
  - 14: Alternative notation (s. man 3 printf), zero-padded and a sign before a positive number (only permitted if *data* is of data type fXX)
  - 15: Alternative notation (s. man 3 printf) and left-justified (only permitted if no conversion into decimal notation)
  - 16: Alternative notation (s. man 3 printf), left-justified and a blank before a positive number (only permitted if *data* is of data type fXX)

  - 17: Alternative notation (s. man 3 printf), left-justified and a sign before a positive number (only permitted if *data* is of data type fXX)
  - 18: Prefix 0x also for a zero and zero-padded (only permitted for a conversion into hexadecimal notation 'x'; ignored if *precision* is given).
  - 19: Prefix 0x also for a zero and left-justified (only permitted for a conversion into hexadecimal notation 'x').
  - 20: Prefix 0X also for a zero and zero-padded (only permitted for a conversion into hexadecimal notation 'X'; ignored if *precision* is given).
  - 21: Prefix 0X also for a zero and left-justified (only permitted for a conversion into hexadecimal notation 'X').
- *field_width:* Definition of the minimum field width
- *precision:* Definition of the precision

Data type result (Return)

- Data type c1400

Example: Stop watch V2 (Cf. Example:Stop watch, page 178).

Timing the seconds at which the variable Stopper_Go has the value ON. A c1400 text string shall be given that prints the time in the format 000d:000h:000m:000s (days, hours, minutes, seconds).

Here the implementation, at which the seconds can be found in the variable *Stopper_time* and the formatted output in *Stopper*. In contrast to Example:Stop watch (page 178), the time difference is counted by means of after.

```
[EibPC]
Stopper=$$
Stopper_time=0s32
Stopper_Go=AUS
if (Stopper_Go) then {
            Stopper_time=1s32;
            write(address(85u16),$Start$c14)
} endif
if after(change(Stopper_time),1000u64) then Stopper_time=Stopper_time+1s32 endif

// End of stop time
if !Stopper_Go then {
            Stopper=stringformat(Stopper_time/86400s32,0,3,3,3)+$d:$+\\
                  stringformat(mod(Stopper_time,86400s32)/3600s32,0,3,3,3)+$h:$+\\
                  stringformat(mod(Stopper_time,3600s32)/60s32,0,3,3,3)+$m:$+\\
                  stringformat(mod(Stopper_time,60s32),0,3,3,3)+$s$
} endif
```

Split

Definition

- Function split(*string*, *pos1, pos2*)

Arguments

- 3 arguments, *string* of data type c1400, *pos1* and *pos2* of data type u16

Effect

- *string*: Character string a character string shall be extracted from.
- *pos1*: Position of the first character of the character string to be extracted (0...1399u16).
- *pos2*: Position of the last character of the character string to be extracted (0...1399u16). If *pos2* equals 65534u16 (predefined constant END), the character string will be separated up to its end.
- The variable *string* must be of data type c1400.
- Return value: The character string extracted from string.

Data type result (Return)

- A character string of data type c1400.

Example: split

The character string „String“ shall be extracted from the variable string=$CharacterString$.

The first character of the character string to be separated has position 8 (counting starts at 0), the last character has position 13.

Implementation in the user program:

```
string=$CharacterString$
result=split(string, 8u16,13u16)
```

Example: Search a character string (2)

The character string “Hello” shall be separated from the variable

string=$CharacterString:Hello$.

Implementation in the user program:

```
String=$CharacterString:Hello$
PartialString=split(String,find(String,$:$,0u16),1399u16)
```

Size

Definition

- Function size(*string*)

Arguments

- 1 argument *string* of data type c1400

Effect

- The length of character string *string* shall be determined. The length is given by the termination character ”\0“ at the end of character strings.

Data type result (Return)

- Data type u16

Example: size

The length of string=$CharacterString$ shall be determined.

Implementation in the user program:

```
string=$CharacterString$
result=size(string)
```

Capacity

Definition

- Function capacity(*String*)

Arguments

- An argument, *string* of data type c1400 respectively with a self defined string length

Effect

- From the string band *String* the maximum available length is to be determined

Data type result (Return)

- Data type u16

Example: capacity

The maximum available length of the string=$string band$ is to be determined.

Implementation in the user program:

```
string=$string band$
result=capacity(string)
```

Tostring

Definition

- Function tostring(*char1[,char2, ... charN]*)

Arguments

- At least one argument, char1 of the data type u08 as the character code of the UTF-8 encoding (see http://de.wikipedia.org/wiki/UTF-8)

Effect

- A string from the individual bytes is formed, the terminating zero is automatically appended

Data type result (Return)

- Data type c1400

Example: capacity

The maximum available length of the string=$string band$ is to be determined.

Implementation in the user program

```
Eurosign=tostring(0xE2,0x82,0xAC)
```

Encode

Definition

- Function encode(*string, source encoding, target encoding*)

Arguments

- An argument, *string* of data type c1400 respectively with a self defined string length
- *Source encoding* with the usual designation, e.g. „UTF-8“
- *Target encoding* with the usual designation, e.g. „UTF-8“

Effect

- A string band *string*, which is present in the source encoding, is going to be transferred in the target encoding.

Data type result (return)

- Data type string format

Example: encode

Recode a string from UTF-8 to ISO-8859

Implementation in the user program:

```
// String
s1=$Hallöchen$c4000

// String code from UTF to Windows (German);
sDE=encode(s1,$UTF-8$c14,$ISO-8859-15$c14)
```

Recode a string from EISO-8859 to UTF-8

```
// String code from UTF to Windows (Europe):
sEU=encode(s1,$UTF-8$c14,$ISO-8859-1$c14)
sUTF=encode(sDE,$ISO-8859-1$c14,$UTF-8$c14)
```

Please also read the notes on p. 108

Urldecode

Definition

- Function urldecode(*string, source encoding, target encoding*)

Arguments

- *String* data type c1400 or with a user-defined string length
- *Source encoding* with the usual designations, e.g. „UTF-8“
- *Target encoding* with the usual designations, e.g. „UTF-8“

Effect

- A string *String*, which is in source encoding, is transmitted to the target encoding using the URL encoding.

Data type result (return)

- Data type string format

Example: encode

Recode a string $ÜberMich.de$

Implementation in the user program

```
// String:org: $Hallöchen auf http:\\enertex.de$
org=urldecode($Hall%c3%b6chen%20auf%20http%3a%5c%5cenertex.de$,$utf-8$c14,$utf-
8$c14)
```

Please also read the notes on p. 108.

Urlencode

Definition

- Function urlencode(*string, source encoding, target encoding*)

Arguments

- *String* data type c1400 or with a user-defined string length
- *Source encoding* with the usual designation, e.g. „UTF-8“
- *Target encoding* with the usual designation, e.g. „UTF-8“

Effect

- A string *String*, which is in source encoding, is transmitted to the target encoding using the URL encoding.

Data type result (return)

- Data type string format

Example: encode

Recode a string $ÜberMich.de$

Implementation in the user program

```
// String ulr=$Hall%c3%b6chen%20auf%20http%3a%5c%5cenertex.de$
url=urlencode($Hallöchen auf http:\\enertex.de$,$utf-8$c14,$utf-8$c14)
```

Please also read the notes on p. 108.

RS232 interface

If you establish your KNX^{TM} connection with an IP interface, you can freely program the RS232 to have access to other devices via the RS232. The syntax is analogous to the network functions for reading and writing on UDP and TCP interfaces, respectively. Cf. the statements on page 64 and page 71.

Configuration

You have to configure the RS232 Interface to fit to your application. Insert the section [RS232] into the application program or use the Dialog OPTIONS - RS232 SETTINGS.

```
[RS232]
// Baud rate of the RS-232 user interface: Decimal notation.
// Permitted values: 0 , 50 , 75 , 110 , 134 , 150 , 200 , 300 , 600 , 1200 , 1800 , 2400 , 4800
// 9600 , 19200, 38400 , 57600 , 115200 , 230400
9600
//Data bits of the RS-232 user interface: Decimal notation. Permitted values: 5, 6, 7, 8
8
//Stop bits of the RS-232 user interface: Decimal notation. Permitted values: 1, 2
1
//Parity of the RS-232 user interface: Decimal notation. OFF = 0 / EVEN = 1 / ODD = 2
0
//Flow control of the RS-232 user interface: Decimal notation.
//OFF = 0 / RTS/CTS = 1 / Xon/Xoff = 2
0
```

Readrs232

Definition

- Function readrs232(*arg 1*[*, arg2, ... argN*])

Arguments

- *arg2* to *argN* arbitrary

Effect

- If an arbitrary RS232 telegram is sent to the Enertex® EibPC, every function readrs232 updates its arguments. If this is the case, the arguments of the function are “filled” with data until the amount of received data complies with the data length of the arguments of the function readrs232.
- To detect incoming telegrams, the function event can be applied to readrs232. This will become necessary if telegrams with identical content have to be evaluated (see below).

Data type result (Return)

- none

Remark

Depending on the configuration of the RS232-Interface (Baudrate) more than one character can be in the buffer, while the Enertex® EibPC is running a process cycle. The lengh of the buffer is provided with the 2nd argument.

Example: Reading RS232 Data

New data shall be written into a string buffer.

```
[EibPC]
rawdata=$$
len=0u16
Buffer=$$
if event(readrs232(rawdata,len)) then {
  Buffer=Buffer + split(rawdata,0u16,len);
  len=0u16
} endif
```

Example: Reading exactly 10 Bytes from RS232

```
[EibPC]
rawdata=$$
len=0u16
Buffer=$$
if event(readrs232(rawdata,len)) and len>9u16 then {
  Buffer=Buffer + split(rawdata,0u16,9u16);
  len=len-10u16;
  rawdata=split(rawdata,10u16,EOS)
} endif
```

Resetrs232

Definition

- Function resetrs232()

Arguments

- none

Effect

- Performs a reset for the RS232 Interface

Data type result (Return)

- none

Sendrs232

Definition

- Function sendrs232(*arg 1*[*, arg2, ... argN*])

Arguments

- *arg2* to *argN* arbitrary

Effect

- “User data” to be transmitted are arbitrary in number and data type.
- If *arg2* to *argN* are data type c1400, the terminating zero of the string will not be transferred.

Data type result (Return)

- none

KNX-Telegram-Routing

With help of the functions address and readknx the Enertex® EibPC can used as an free programmable router for KNX™ telegrams. If e.g. the group address is sent (as number) to the Enertex® EibPC via TCP/IP client, it is possible to write via the function address to this group address a given value, without any additional program code. Similar an incoming KNX™ telegram will be signaled by the readknx function to the TCP/IP client. The Opensource project "EibPC-Homecontrol" uses this functionality. The function address can be used as first argument instead of the group address in the functions: event, write, scene et cetera.

Address

This function generates a group address from a u16 number to be used when accessing the bus.

Definition

- Function address(*variable*)

Arguments

- 1 argument of data type u16

Effect

- Return value: A group address as it can be used with write, read etc..

Data type result (Return)

- Data type group address

As a particular feature of the bus access functions, they expect group addresses as arguments.
E.g. the 1st argument of write('5/3/11'b01, ON) has to be a group address. The function address converts a u16 number into a group address. This number is calculated as address= [main group] x 2048+[middle group] x 256 + [subgroup], with [main group]=5, [middle group]=3 and [subgroup]=11 for the example '5/3/11'. You have to calculate this number by yourself or you can use the function getaddress.

Example: address

You want to write ON to group address *'5/3/11'b01*at system startup.

Implementation in the user program:

```
if systemstart() then write(address(11019u16),ON) endif
```

Readknx

Definition

- Function readknx(*Number*, *Output*)

Arguments

- *Number* of data type u16
- *Output* of data type c1400

Effect

- An incoming KNX™ telegram will make the function wriingt the group address of the telegram in the variable named *Number*. The binary data of the telegram is stored in the variable named *Output*. *Output* is changing its type to that of the last incoming telegramm To convert it back, use convert as shown in the example.

Data type result (Return)

- Result of the conversion of the KNX™ telegrams binary data

Note:

The function event can used with readknx function (see example).

Example: Sending all incoming KNX™ telegrams via UDP:

Following code will send all telegrams received from the KNX™ bus via UDP to the client with the IP 192.168.22.199. The group address of the telegram is sent in u16 format and the information as a string in the format GA:XXXXX INF:YYYYYYY .

```
adr=0u16
info=$$
if event(readknx(adr,info)) then {
            sendudp (5000u16, 192.168.22.199,$GA:$+convert(adr,$$)+$INF:$+info)
}endif
```

Readrawknx

Definition

- Function *readrawknx(control field, phyAddress, targetAddress, IsGroubAddress, routingCounter, bitLength, userData)*

Arguments

- *control field* of data type u08
- *phyAddress* of data type u16 (he transmitter's address in the usual notation, e.g. 2.4.13)
- *targetAddress* of data type u16
- *IsGroubAddress* of data type b01
- *routingCounter* of data type u08
- *bitLength* of data type u08
- *userData* of data type c1400

Find further information about the telegram structure on p. 215

Effect

- If a KNX telegram observed, every function readrawknx updates its arguments. The arguments of the readrawknx function are filled with data up to the length of its arguments. In any case, the variables *phyAddress* and *groubAddress* of the function readrawknx are overwritten with the current data of the transmitter every time a KNX telegram is received.
- The physically address (variable *phyAddress*) is defined in the usual notation (e.g. 2.4.13)
- The *IsGroubAddress* shows, wheather the telegram is addressed to a physical address or a group adress.
- To detect incoming telegrams, the function event can be applied to readrawknx. This will become necessary ,if telegrams with identical content have to be evaluated.

Data type result (Return)

- none

Example: Write data received from KNX telegrams to the KNX^{TM} bus

Count telegrams who were send by physically address 1.3.14

Implementation in the user program:

```
Raw_Kontroll=0
Raw_Sender=10.2.1
Raw_GA=0u16
Raw_IsGa=OFF
Raw_RoutingCnt=0
Raw_Len=0
Raw_Data=$$
count=0u08
if event(readrawknx(Raw_Kontroll,Raw_Sender,Raw_GA,Raw_IsGa,Raw_RoutingCnt,
Raw_Len,Raw_Data)) and Raw_Sender==1.3.14 and Raw_GA==getaddress('2/4/44'b01) and
Raw_IsGa  then {
            count=count+1
} endif
```

Example: monitoring actuator

It checks whether from a KNX device at least 120 minutes a telegram arrives.

In addition, a few statistics about the bus.

Implementation in the user program:

```
// ----------------------------------------
// physical device address
// ----------------------------------------
Raw_Dev=1.1.60

// evaluation
// ----------------------------------------
// max time between two telegrams from one device since recording
Raw_MaxTime=0u16
// min time between two telegrams from one device since recording
Raw_MinTime=65365u16
// last determined time
Raw_CalcTime=0u16
// Average value over all telegrams of the same equipment
Raw_AvgTime=0u64

// errortime: When an error is to be recognized
Raw_TimeWatch=120u64*60000u64


// arguments from readrawknx:
Raw_Kontroll=0
Raw_Sender=0.0.0
Raw_GA=0u16
Raw_IsGa=AUS
Raw_RoutingCnt=0
Raw_Len=0
Raw_Data=$$


// ----------------------------------------
// assistant variables
Raw_AvgTrigger=0u64
Raw_Error=AUS
Raw_AvgTimeSum=0u64
// timescale:  1000  accuracy in seconds
//             60000  accuracy in minutes
Raw_TimeScale=1000u64

Raw_Time=Raw_TimeWatch

// Respond only to group messages on the EibPC and only if the sender address is correct

if event(readrawknx(Raw_Kontroll,Raw_Sender,Raw_GA,Raw_IsGa,Raw_RoutingCnt,Raw_Len,Raw_Data))
and  Raw_Sender==Raw_Dev and Raw_IsGa then {
        // change time to seconds and calculate min and max values
        // evaluate Raw_Time
        Raw_CalcTime=convert((Raw_TimeWatch-Raw_Time)/Raw_TimeScale,0u16);
        if Raw_MaxTime<Raw_CalcTime then Raw_MaxTime=Raw_CalcTime endif;
        if Raw_MinTime>Raw_CalcTime then Raw_MinTime=Raw_CalcTime endif;
        // avarage=Raw_AvgTime/Raw_Trigger
        Raw_AvgTimeSum=Raw_AvgTimeSum+convert(Raw_CalcTime,0u64);
        Raw_AvgTrigger=Raw_AvgTrigger+1u64;
        Raw_AvgTime=Raw_AvgTimeSum/Raw_AvgTrigger;
} endif
```

```
// expect a telegram every Raw_TimeWatch: then delay will retrigger
// otherwise error condition!
if delayc(change(Raw_AvgTrigger),Raw_TimeWatch,Raw_Time) then {
                Raw_Error=EIN
} endif
```

Note:

The function event can used with readrawknx function (see example).

GetAddress

Definition

- Function getaddress(*Groupaddress*)

Arguments

- Groupaddress any imported (or manually given) Group Address

Effect

- The function is returning the unsigned 16-Bit Value of the groupaddress as its address number.

Data type result (Return)

- u16

At 12:00 AM the Group Address 1/1/27 shall be read and at 12:30 a 10% value shall be written to the same group address

```
[EibPC]
a=getaddress("Dimmer-1/1/27")
if htime(12,00,00) then read(address(a)) endif
if htime(12,30,00) then write(address(a),16) endif
```

Note:

Normally you don't need this function, you could directly code read("Dimmer-1/1/27") etc. This function is provided for enhanced coding styles.

Gaimage

Definition

- Function gaimage(*Number*)

Arguments

- *Number* of data type u16

Effect

- The function is returning the actual image of a group address stored in the Enertex® EibPC. The group address of the telegram is given with the variable named *Number*. The binary data of the telegram is converted into a string (see convert) and given as the return value of this function.

Data type result (Return)

- c1400

Note:

The Number is calculated as address= [main group] x 2048+[middle group] x 256 + [subgroup]. As an example with [main group]=5, [middle group]=3 and [subgroup]=11 the telegramm imaga of '5/3/11' is addressed. You have to calculate this number by yourself or you can use the function getaddress.

Getganame

Definition

- Function getganame(*Groupaddress, Coding)*

Arguments

- *Groupaddress* any imported Group Address
- Coding with the usual designation, e.g. $ UTF-8 $ c14 as c14 string, is used to directly convert the GA to any system encoding.

Effect

- The function returns the name of the group address in the Enertex® EibPC format when this group address has been imported into the application program (ESF import)

Data type result (Return)

- c1400

The name of a group address should be stored as a text in the standard Windows encoding (iso8859-15) in a variable.

```
// MyVar=$"VentilateWorking-0/0/2"$
MyVar=getganame("VentilateWorking-0/0/2",$utf-8$c14)
```

Network functions

Activation codes

To use the network functions of this chapter, the option NP must be activated in the Enertex® EibPC. You have to know the serial number of your Enertex® EibPC and have to acquire the respective additional package. The activation code is linked to the serial number of a single device and cannot be used for other devices.

To transfer a valid activation code to the Enertex® EibPC, choose the menu EibPC → Transfer Activation Code ... and follow the instructions.

Name resolution

For some network functions (sendmail, resolve), the Enertex® EibPC requires a connection to a DNS server. Information about the configuration or function test of a DNS server can be found on page 137.

Standard-Ports

The ports via which the Enertex® EibPC communicates can be changed via the Options / Ports menu. This menu generates a [Ports] section, which then receives the following entries

```
[Ports]
///UDP InPort
12600
// UDP OutPort
3245
// TCP Port
23456
//Modbus Port
5901
```

UDP telegrams

The Enertex® EibPC itself sends the data of a UDP transfer always from its port 4807, whereas the receiver's port can be chosen arbitrarily.

UDP Ports

The Enertex® EibPC receives the data of a UDP transfer always from its port 4806. Therefore, the transmitter must use this port as destination. The port the transmitter send its data from can be determined by the Enertex® EibPC.

Readudp

Definition

- Function readudp(*port, ip, arg 1*[*, arg2, ... argN*])

Arguments

- Argument *port* of data type u16 (the transmitter's outbound port; the transmitter's destination port must always be port 4806).
- Argument *ip* of data type u32 (the transmitter's address in the usual notation, e.g. 192.168.22.100)
- *arg2* to *argN* of arbitrary data type

Effect

- Received “user data” start with the 3rd argument. Their number and data type is arbitrary.
- If a UDP telegram is sent to the Enertex® EibPC, every function readudp updates its respective arguments. The arguments of the readudp function are filled with data up to the length of its arguments. In any case, the variables *port* and *ip* of the function readudp are overwritten with the current data of the transmitter every time a UDP telegram is received.
- The IP address (variable *ip*) is defined in the usual notation (xxx.xxx.xxx.xxx with xxx: number between 0 and 255).
- If your LAN device can be addressed by a name and DNS, the function resolve can replace an explicit IP address.
- To detect incoming telegrams, the function event can be applied to readudp. This will become necessary if telegrams with identical content have to be evaluated (see below).
- The Enertex® EibPC always receives from port 4806. This port cannot be changed and must be taken into consideration by a UDP transmitter.

Data type result (Return)

- none

Example: Write data received from UDP telegrams to the KNX™ bus

A UDP telegram is sent by the transmitter 122.32.22.1 to the Enertex® EibPC via the transmitter's port 2243u16. The user data consist of three u08 values and shall be sent to the group addresses 3/4/0,3/4/1,3/4/2 whenever a UDP telegram arrives.

Implementation in the user program:

```
Port=0u16
IP=0u32
Data1=0;Data2=0;Data3=0
telegram=event(readudp(Port, IP,Data1,Data2,Data3))
if (Port==2243u16) and (IP==122.32.22.1) and telegram then \\
                        write('3/4/0'u08,Data1);                  \\
                        write('3/4/1'u08,Data2);                  \\
                        write('3/4/2'u08,Data3)                   \\
                        endif
```

Note:

The function event, or rather the link with *telegram* in the if statement ensures that the then branch is called in any case, thus sending the data to the bus, even if identical UDP telegrams are sent multiple times (and because of the validation scheme, also see pages 50 and 162).

Sendudp

Definition

- Function sendudp(*port, ip, arg 1*[*, arg2, ... argN*])

Arguments

- Argument *port* of data type u16
- Argument *ip* of data type u32 (the receiver's address in the usual notation, e.g. 192.168.22.100)
- *arg2* to *argN* of arbitrary data type

Effect

- Argument *port* is the destination port of the data sent by the Enertex® EibPC.
- The Enertex® EibPC itself sends the data from its port 4807.
- Transmitted “user data” start with the 3rd argument. Their number and data type is arbitrary.
- The IP address (variable *ip*) is defined in the usual notation (xxx.xxx.xxx.xxx with xxx: number between 0 and 255).
- If your LAN device can be addressed by a name and DNS, the function resolve can replace an explicit IP address.
- If *arg2* to *argN* are data type c1400, the terminating zero of the string will be transferred, too.

Data type result (Return)

- none

Example: Send UDP telegrams

Every 2 minutes, a UDP telegram shall be sent by the Enertex® EibPC to the port 5555u16 of the receiver www.enertex.de. The user data to be transmitted shall comprise a 32-bit counter for the telegrams and the character string “I'm still alive”.

Implementation in the user program:

```
Count=0u32
if cycle(2,00) then sendudp(5555u16,resolve($www.enertex.de$, Count,$I'm still alive$); \\
                Count=Count+1u32 endif
```

Sendudparray

Definition

- Function sendudparray(*port, ip, arg,Nr*)

Arguments

- Argument *port* of data type u16
- Argument *ip* of data type u32 (the receiver's address in the usual notation, e.g. 192.168.22.100)
- *arg* of data type c1400
- *Nr* of data type u16

Effect

- Argument *port* is the destination port of the data sent by the Enertex® EibPC.
- Received "user data" start with the 3rd argument. Their number and data type is arbitrary.
- The IP address (variable *ip*) is defined in the usual notation (xxx.xxx.xxx.xxx with xxx: number between 0 and 255).
- If your LAN device can be addressed by a name and DNS, the function resolve can replace an explicit IP address.
- Sends *Nr* Bytes of *arg* via UDP Protocol.

Data type result (Return)

- none

Example: Send UDP telegrams

Every 2 minutes, a UDP telegram shall be sent by the Enertex® EibPC to the port 5555u16 of the receiver www.enertex.de. The user data to be transmitted is the first 5 characters of the string "I'm still alive".

Implementation in the user program:

```
Count=0u32
if cycle(2,00) then sendudparray(5555u16,resolve($www.enertex.de$),$I'm still alive$,5u16)
endif
```

TCP server and client

Server and client

The Enertex® EibPC functions both as a server and as a client. Every 100 ms, it responds to a new connection request. If the Enertex® EibPC is connected, it answer the requests with the cycle time of the processing cycle.

TCP ports

The TCP/IP server of the Enertex® EibPC receives connection requests always via its port 4809.

Connecttcp

Definition

- Function connecttcp(*port, ip*)

Arguments

- Argument *port* of data type u16
- Argument *ip* of data type u32 (the destination's address in the usual notation, e.g. 192.168.22.100)

Effect

- The Enertex® EibPC functions as a client. It establishes a connection to the given destination (defined by *ip* address and *port)*.
- The function returns its processing status:
  - successful = 0
  - in progress = 1
  - error= 2
  - error due to an already existing connection = 3
  - error caused by too many active connections = 4
  - connection automatically closed due to a timeout (not responding) = 6
  - connection closed by user with closetcp= 7
  - TCP counterpart closed the connection = 8
  - Initial value = 9
- After 30 seconds of inactivity of an existing connection, the Enertex® EibPC disconnects automatically

Data type result (Return)

- u08 (The return value goes asynchronous to the main development loop - see p. 124)

Closetcp

Definition

- Function closetcp(*port, ip*)

Arguments

- Argument *port* of data type u16
- Argument *ip* of data type u32 (the destination's address in the usual notation, e.g. 192.168.22.100)

Effect

- The Enertex® EibPC closes the connection to the given destination (defined by *ip* address and *port)*.
- The function returns its processing status:
  - successful = 0,
  - in progress = 1 and
  - error = 2
  - error, the connection does not exist = 5

Data type result (Return)

- u08

Readtcp

Definition

- Function readtcp(*port, ip, arg 1*[*, arg2, ... argN*])

Arguments

- Argument *port* of data type u16 (the transmitter's outbound port)
- Argument *ip* of data type u32 (the transmitter's address in the usual notation, e.g. 192.168.22.100)
- *arg2* to *argN* of arbitrary data type

Effect

- Received "user data" start with the 3rd argument. Their number and data type is arbitrary.
- If a TCP/IP telegram is sent to the Enertex® EibPC, every function readtcp updates its respective arguments. The arguments of the readtcp function are filled with data up to the length of its arguments. In any case, the variables *port* and *ip* of the function readtcp are overwritten with the current data of the transmitter every time a TCP/IP telegram is received.
- The IP address (variable *ip*) is defined in the usual notation (xxx.xxx.xxx.xxx with xxx: number between 0 and 255).
- If your LAN device can be addressed by a name and DNS, the function resolve can replace an explicit IP address.
- To detect incoming telegrams, the function event can be applied to readtcp. This will become necessary if telegrams with identical content have to be evaluated (see below).

Data type result (Return)

- none

Sendtcp

Definition

- Function sendtcp(*port, ip, arg 1*[*, arg2, ... argN*])

Arguments

- Argument *port* of data type u16
- Argument *ip* of data type u32 (the receiver's address in the usual notation, e.g. 192.168.22.100)
- *arg2* to *argN* of arbitrary data type

Effect

- Argument *port* is the destination port of the data sent by the Enertex® EibPC.
- The "user data" starts with the 3rd argument. Their number and data type is arbitrary.
- The IP address (variable *ip*) is defined in the usual notation (xxx.xxx.xxx.xxx with xxx: number between 0 and 255).
- If your LAN device can be addressed by a name and DNS, the function resolve can replace an explicit IP address.
- If *arg2* to *argN* are data type c1400, the terminating zero of the string will be transferred, too.

Data type result (Return)

- none

Example: Send TCP telegrams

Every 2 minutes, a TCP telegram shall be sent by the Enertex® EibPC to the port 5555u16 of the receiver www.enertex.de. The user data to be transmitted is the string "I'm still alive". The socket is already open and ready to send (IP and Port open).

Implementation in the user program:

```
Count=0u32
if cycle(2,00) then sendtcp(5555u16,resolve($www.enertex.de$),$I'm still alive$) endif
```

Example: See page 71.

Sendtcparray

Definition

- Function sendtcparray(*port, ip, arg,Nr*)

Arguments

- Argument *port* of data type u16
- Argument *ip* of data type u32 (the receiver's address in the usual notation, e.g. 192.168.22.100)
- *arg* of data type c1400
- *Nr* of data type u16

Effect

- Argument *port* is the destination port of the data sent by the Enertex® EibPC.
- Received “user data” start with the 3rd argument. Their number and data type is arbitrary.
- The IP address (variable *ip*) is defined in the usual notation (xxx.xxx.xxx.xxx with xxx: number between 0 and 255).
- If your LAN device can be addressed by a name and DNS, the function resolve can replace an explicit IP address.
- Sends *Nr* Bytes of *arg* via TCP/IP Protocol.

Data type result (Return)

- none

Example: Send TCP telegrams

Every 2 minutes, a TCP telegram shall be sent by the Enertex® EibPC to the port 5555u16 of the receiver www.enertex.de. The user data to be transmitted is the first 5 Bytes of the string “I'm still alive”.
The socket is already open and ready to send (IP and port).

Implementation in the user program:

```
Count=0u32
if cycle(2,00) then sendtcparray(5555u16,resolve($www.enertex.de$),$I'm still alive$,5u16)
endif
```

Md5sum

Definition

- Function md5sum(*string*)

Arguments

- Argument *string* of any length

Effect

- The MD5 sum of the string is calculated. The result is returned as a string.
- **Result (Return)**
- Data type cXXXXX with the same string length as the output string.

Example ping

The value of the MD5 sum of the string $ fdzehkdkhfckdhk %% $ is to be determined

```
string=$fdzehkdkhfckdhk%%$
md5=md5sum(string)
```

Ping

Definition

- Function ping(*IP*)

Arguments

- The IP address (variable *ip*) is defined in the usual notation (xxx.xxx.xxx.xxx with xxx: number between 0 and 255).

Effect

- Execution of the ping command
- The function returns its processing status:
  - successful = 0,
  - in progress = 1 and
  - error = 2

Data type result (Return)

- u08

  (The return value is asynchronous to the main development loop - see p. 124)

Example ping

The address www.enertex.de should be pinged shortly after systemstart.

```
IP=0u32
a=3
If after(systemstart(),10u64) then IP=resolve($www.enertex.de$) endif
If after(systemstart(),10u64) then a=ping(IP) endif
if a==0 then write('2/2/2'c14,$found$c14) endif
```

Resolve

Definition

- Function resolve(*hostname*)

Arguments

- 1 argument *hostname* of data type c1400

Effect

- The function determines the IP address of the given hostname.
- If an error occurs, 0u32 is returned.

Data type result (Return)

- Data type u32
  (The return value goes asynchronous to the main development loop - see p. 124)

Example resolve

The hostname enertex.de shall be resolved.

Implementation in the user program:

```
hostname=$www.enertex.de$
IP=resolve(hostname)
```

Before the function sendmail can be used, the basic e-mail configuration has to be done (see page 137).

Sendmail

Definition

- Function sendmail(*destination, subject, message*)

Arguments

- 3 arguments of data type c1400

Effect

- A *message* with *subject* is sent to the *destination* (character string).
- All character strings are restricted to a maximum length of 1400 characters.
- A line break can be achieved by using the two characters '\n' in the string,
- Return value: 0 = e-mail successfully sent
  1 = in progress
  2 = error

Data type result (Return)

- Data type u08
  (The return value goes asynchronous to the main development loop - see p. 124)

Example: sendmail

Every Monday at 08:00, an e-mail shall be sent to eibpc@enertex.de.
The subject is “EibPC“ and the message contains 2 lines “I'm still alive“ and “Here we go!”

Implementation in the user program:

```
email=$eibpc@enertex.de$
subject=$EibPC$
message=$I'm still alive\nHere we go$
if wtime(08,00,00,MONTAG) then sendmail(email, subject, message) endif
```

Note:
If you want to send html - formatted mails, use the sendhtmlmail Function (page 243)

Sendhtmlmail

Before the function sendhtmlmail can be used, the basic e-mail configuration has to be done (see page 137).

Definition

- Function sendhtmlmail(*destination, subject, message*)

Arguments

- 3 arguments of data type c1400

Effect

- A *message* with *subject* is sent to the *destination* (character string).
- All character strings are restricted to a maximum length of 1400 characters.
- A line break can be achieved by using the two characters '\n' in the string,
- Return value: 0 = e-mail successfully sent
  1 = in progress
  2 = error

Data type result (Return)

- Data type u08

Example: sendhtmlmail

Every Monday at 08:00, an e-mail shall be sent to eibpc@enertex.de.
The subject is “EibPC“ and the message contains 2 lines “Hello World,“ (in bold) and “Here we go!”

Implementation in the user program:

```
email=$eibpc@enertex.de$
subject=$EibPC$
message=$<html><head><meta name="qrichtext" content="1" /></head><body style="font-size:11pt;font-family:Sans Serif"> <p><span style="font-weight:600">Hello World, </span></p> <p>a message from the EibPC</p> </body></html>$
if wtime(08,00,00,MONTAG) then sendhtmlmail(email, subject, message) endif
```

Note:
If you don't want to send html - formatted mails, use the sendmail Function (page 242).

VPN Server

The Enertex® EibPC can operate as an configurable VPN Server and VPN Router in your LAN. To use this functionality your Enertex® EibPC must have a valid activation code for Option NP.

In the router port forwarding for UDP port 1194 must be aktivated.

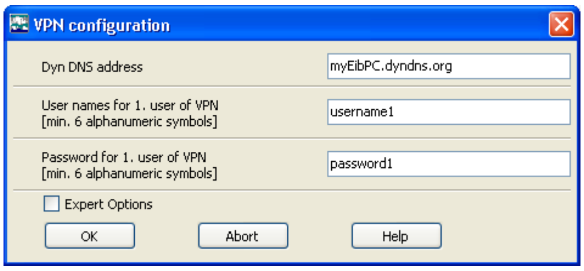
With the Enertex® EibPC VPN Services can be directly started and stopped by the running KNX Installation. Also different users' access can be opened and closed.

First, configure the VPN with the help of OPTIONS-VPN CONFIGURATION. (Fig. 7)

It will be determined

- which network address the VPN Network will have
- which network address the remote LAN has
- which IPs will be accessible through the VPN in the remote LAN
- which users will have access

Furthermore it is necessary to create key and certificate files for the VPN operation. They are unique generated by the Enertex® EibPC. You can find a demo-video (for now German language only) on www.youtube.com showing more details on this.You can configure the VPN Server using the wizard of Enertex® EibStudio at EIBPC – VPN CONFIGURATION. The configuration is stored in the [VPN] section.


VPN configuration

Dyn DNS address: myEibPC.dyndns.org

User names for 1. user of VPN [min. 6 alphanumeric symbols]: username1

Password for 1. user of VPN [min. 6 alphanumeric symbols]: password1

☐ Expert Options

Figure 7: VPN configuration dialog

With the Info-Button in EibStudio can be read whether the VPN service is running and which users are enabled.

Startvpn

Definition

- Function startvpn()

Arguments

- none

Effect

- Starts the VPN Service on the Enertex® EibPC. The VPN must be configured with Enertex® EibStudio before.
- After a reboot the VPN is stopped per default. The VPN should therefore started with an if systemstart() construction (see example)
- All in the past enabled users (to open a user's VPN access use openvpnuser) are immediately opened after this function call.
- If a new user progamm is downloaded to an EibPC, the VPN service remains open. An recommended additional startvpn()-call does not make an interruption on the running service. Only if the system is rebooted the Service will be stoppped.
- With the Info-Button in EibStudio can be read whether the VPN service is running and which users are enabled.

Data type result (Return)

- none

Stopvpn

Definition

- Function stopvpn()

Arguments

- none

Effect

- Stops the VPN Service on the Enertex® EibPC.
- After a reboot the VPN is stopped per default.
- All in the past enabled users (to open a user's VPN access use openvpnuser) are immediately closeed after this function call.
- With the Info-Button in EibStudio can be read whether the VPN service is running and which users are enabled.

Data type result (Return)

- **none**

Openvpnuser

Definition

- Function openvpnuser(*username*)

Arguments

- *username* is a c1400 Type ($$)

Effect

- Opens a user's VPN access. The access becomes active only, if a startvpn() is already executed .
- After a reboot the VPN access itself remains enabled, but the VPN service has to be started with startvpn() separately.
- With the Info-Button in EibStudio can be read whether the VPN service is running and which users are enabled.

Data type result (Return)

- **none**

Closevpnuser

Definition

- Function closevpnuser(*username*)

Arguments

- *username* is a c1400 Type ($$)

Effect

- Closes a user's VPN access. The access becomes inactive independently whether the VPN Service is running or not.
- After a reboot the VPN is still open, but the VPN service has to be started with startvpn().
- With the Info-Button in EibStudio can be read whether the VPN service is running and which users are enabled.

Data type result (Return)

- **none**

Remark

closevpnuser does not effect an already open VPN user access. The access will denied, if the user is logged out and will try to re-login or the VPN Service is completely stopped and started again.

Example:

The access of *User1* should be opened, once there is an ON Signal (1b01) sent at groupaddress 1/1/1. If there is an OFF signal (0b1) the user shall be closed. A second user shall be opened with address 1/1/2.

The VPN Service should be started 500ms after systemstart and closed with an ON, if 1/1/3 is receiving a signal.

```
[EibPC]
if after(systemstart(),500u64) then startvpn() endif
if "OpenUser1-1/1/1"==ON then openvpnuser($User1$) else closevpnuser($User1$) endif
if "OpenUser2-1/1/2"==ON then openvpnuser($User2$) else closevpnuser($User2$) endif
if "StopVPN-1/1/3"==ON then stopvpn() endif
```

Web server functions

Button (Webbutton)

Definition

- Function button(*id*)
- Identical to function webbutton of former releases.

Arguments

- Argument *id* of data type u08. This argument must not change at the runtime of the program.

Effect

- By operating the button of a web button element (e.g. *button* or *shifter*) with the *id* (page 265 and the following), the function assumes a value not equal to zero for the duration of one processing pass, and zero in all other cases.
- For a *button* element, the return value when operated is 1.
- For a *shifter* element, the return value when operated is 1, 2, 3 or 4 (u08) depending on the actually operated element of the web button. The return values refer to the order of the buttons (from left to right).

Data type result (Return)

- Data type u08, values 0,1,2,3,4

Example: sendmail

A detailed example can be found on page 75 and the following.

Chart (Webchart)

Definition

- Function chart(*id, var, x1, x2*)
- compatible with function webchart

Arguments

- Arguments *id, var* of data type u08
- Arguments *x1, x2* of data type c14

Effect

- This function addresses the XY diagram *chart*. If there are multiple occurrences of *id*, all elements of this id are addressed.
- When calling this function, the XY diagram of the value *var* is activated. Values in the range of 1...30 can be displayed. 0 refers to the value not being displayed, and values greater than 30 are not permitted and are interpreted like 0. Every call of the function displays the values beginning from the left side. When the end is reached after 47 function calls, the values are shifted to the left.
- The labeling of the x-axis is given by the arguments *x1, x2* (data type c14).

Data type result (Return)

- Data type u08 (internal state of the webchart)

Example display percentage value

In an XY diagram of the web server (element *chart*), a percentage shall be displayed.

Implementation in the user program:

```
[WebServer]
chart(ChartWebID)[$0%$,$50%$,$100%$]
[EibPC]
PercentageValue='1/3/5'u08
ChartWebID=0
if stime(0) then\\
webchart(ChartWebID,convert(convert(PercentageValue,0f32)/8.5f32,0), $now$c14,$-
47min$c14) endif
```

Display (Webdisplay)

Definition

- Function webdisplay(*id, text, icon, state, style, [mbutton]*)

Arguments

- Arguments *id, icon, state, style* and *mbutton* of data type u08
- Argument *text* of arbitrary data type

Effect

- The function addresses the web button (*button* or *shifter*). If there are multiple web buttons with *id*, they all will be addressed.
- With the optional argument *mbutton* the list of the drop-down menu can be changed.
- Calling this function sets the icon of the web element with *id* to the symbol defined by *icon* (data type u08). Possible images are shown on page 291 and are selected by predefined numbers (data type u08). In addition, predefined constants facilitate the choice. Their respective allocation is listed in Table 2 (page 159)
- The argument *text* denominates an arbitrary variable the value of which, converted to a character string, is displayed in the variable text line of the web element.
- Every icon has at least the states ACTIVE (==1), INACTIVE (==2), DARKRED (==0) and BRIGHTRED (==9). One of these states can be submitted as the argument *state*. For an overview of the possible states see Table 3 (page 159).
- The text to be displayed can be represented in the stylesGREY (==0), GREEN (==1), BLINKRED(==2) and BLINKBLUE (==3).

Data type result (Return)

- none

Example show current time

A *button* element shall display the current time.

Implementation in the user program:

```
[WebServer]
button(ClockWebID)[CLOCK]$Uhrzeit$2
[EibPC]
ClockWebID=0
if stime(0) then webdisplay(ClockWebID,settime(),CLOCK,INACTIVE,GREY) endif
```

Note:

1. The data type of the return value of *settime*() is t24. In this case, it is converted to a readable character string of the notation „Fr. 12:33:55“.
2. You can access to variables defined in the section [EibPC]. But consider, the webserver evaluates the variable statically. When the variable *ClockWebID* is changing during run-time, the index *ClockWebID* will still use its initial value, which is 0.

Getslider

Definition

- Function getslider(*id*)

Arguments

- Argument *id* of data type u08. This argument must not change at the runtime of the program.

Effect

- The function addresses the *slider* and returns its position (0 to 255). If there are multiple occurrences of *id*, all elements of this id are addressed.

Data type result (Return)

- Data type u08

Getpslider

Definition

- Function getpslider(*id, page_id)*

Arguments

- Argument *id* of data type u08. This argument must not change at the runtime of the program.
- Argument *page_id* of data type u08. This argument must not change at the runtime of the program.

Effect

- The function addresses the *pslider* that refers to a page and returns its position (0 to 255). If there are multiple occurrences of *id*, all elements of this id on the web page with *page_id* are addressed.

Data type result (Return)

- Data type u08

Geteslider

Definition

- Function geteslider(*id*)

Arguments

- Argument *id* of data type u08. This argument must not change at the runtime of the program.

Effect

- The function addresses the *eslider* and returns its position (0 to 255). If there are multiple occurrences of *id*, all elements of this id are addressed.

Data type result (Return)

- Data type f32

Getpeslider

Definition

- Function getpeslider(*id, page_id)*

Arguments

- Argument *id* of data type u08. This argument must not change at the runtime of the program.
- Argument *page_id* of data type u08. This argument must not change at the runtime of the program.

Effect

- The function addresses the *peslider* that refers to a page and returns its position (0 to 255). If there are multiple occurrences of *id*, all elements of this id on the web page with *page_id* are addressed.

Data type result (Return)

- Data type f32

link

Definition

- Function link(*id, text, icon, page_id, website*)

Arguments

- Arguments *id, icon* and *page_id* of data type u08
- Argument *text* of arbitrary data type
- Argument *website* of data type c1400

Effect

- The function addresses the web button that refers to a page (*link*). If there are multiple web buttons with *id* on the web page of *page_id*, they all will be addressed.
- Calling this function sets the icon of the web element with *id* to the symbol defined by *icon* (data type u08). Possible images are shown on page 291 and are selected by predefined numbers (data type u08). In addition, predefined constants facilitate the choice. Their respective allocation is listed in Table 2 (page 159).
- The argument *text* denotes an arbitrary variable the value of which, converted to a character string, is displayed in the variable text line of the web element.
- Every icon has at least the states ACTIVE (==1), INACTIVE (==2), DARKRED (==0) and BRIGHTRED (==9). One of these states can be submitted as the argument *state*. For an overview of the possible states see Table 3 (page 159).
- The text to be displayed can be represented in the styles GREY (==0), GREEN (==1), BLINKRED(==2) and BLINKBLUE (==3).
- The argument *website* (http address (incl. path and leading http://) of the destination site) specified the new destination. The link is shortened to 479 characters due to compatibilities restrictions.

Data type result (Return)

- none

Mbutton

Definition

- Function mbutton(*id, selection*)

Arguments

- Argument *id* of data type u08. This argument must not change at the runtime of the program.
- Argument *selection* of data type u08

Effect

- By operating the button of a multi button element and the given selection with index *selection* (e.g. *mbutton* or *mshifter*) with the *id* (page 265 and the following), the function assumes a value not equal to zero for the duration of one processing pass, and zero in all other cases.
- For a *mbutton* element, the return value when operated is 1.
- For a *mshifter* element, the return value when operated is 1, 2, 3 or 4 (u08) depending on the actually operated element of the web button. The return values refer to the order of the buttons (from left to right).

Data type result (Return)

- Data type u08, values 0,1,2,3,4.

Mchart

Definition

- Function mchart(*id, x, y, index*)

Arguments

- Arguments *id, index* of data type u08
- Arguments *x, y* of data type f16

Effect

- This function addresses the element *mchartf* of the given *id*. If there are multiple occurrences of *id*, all elements of this id are addressed.
- One *mchart* displays four different graphs. *index* (0,1,2,3) defines the graph to be addressed.
- Up to 48 values are stored. If more than 48 values are stored in the same *index* of mchart, the value stored in the first location is lost.
- The placement of the values in the graph is performed by the specification of the pairs of variates.
- The labeling is generated automatically.

Data type result (Return)

- u08 (internal state).

Mpchart

Definition

- Function mpchart(*id, x, y, index, page_id*)

Arguments

- Arguments *id, page_id, index* of data type u08
- Arguments *x, y* of data type f16

Effect

- This function addresses the element *mpchart* that refers to a page of the given *id*. If there are multiple occurrences of *id*, all elements of this id are addressed.
- One *mpchart* displays four different graphs. *index* (0,1,2,3) defines the graph to be addressed.
- Up to 48 values are stored. If more than 48 values are stored in the same *index* of mpchart, the value stored in the first location is lost.
- The placement of the values in the graph is performed by the specification of the pairs of variates.
- The labeling is generated automatically.

Data type result (Return)

- u08 (internal state).

Mpbutton

Definition

- Function mpbutton(*id, selection, page_id*)

Arguments

- Argument *id* of data type u08. This argument must not change at the runtime of the program.
- Argument *page_id* of data type u08. This argument must not change at the runtime of the program.
- Argument *selection* of data type u08.

Effect

- By operating the button of a multi button element that refers to a page and the given selection with index *selection* (e.g. *mpbutton* or *mpshifter*) with the *id* (page 265 and the following), the function assumes a value not equal to zero for the duration of one processing pass, and zero in all other cases.
- For a *mpbutton* element, the return value when operated is 1.
- For a *mpshifter* element, the return value when operated is 1, 2, 3 or 4 (u08) depending on the actually operated element of the web button. The return values refer to the order of the buttons (from left to right).

Data type result (Return)

- Data type u08, values 0,1,2,3,4.

Plink

Definition

- Function plink(*ID, Text, Icon, Iconzustand, PageID, PageSprungIndex*)

Arguments

- Argument *ID* of data type u08.
- Argument *Text* of arbitrary data type
- Argument *Icon* of data type u08
- Argument *Iconzustand* of data type u08
- Argument *PageID* of data type u08
- Argument *PageSprungIndex* of data type u08

Effect

- The function responds to the (*plink*). If *ID* is more than one, then all elements of this ID are addressed to the page with the *PageID*.
- When calling the function, the icon of the web element with the *ID* on the webpage with the *PageID* is set to the symbol with the number (type u08) *icon*. Possible graphics are shown on page 291 and are selected using predefined numbers (u08). The allocation for this is listed in Table 2 (page 291).
- The argument *text* is an arbitrary variable whose value, converted into a string, is output to the text line of the web element.
- The *PageSprungIndex* argument specifies the page number of the new jump destination.

Data type result (Return)

- none

Example

Dynamic Change of Web-Links

```
[WebServer]
page (1) [$Haus$,$OG$]
plink(2) [INFO] [3] $Zu Seite 3$
picture(3) [DOUBLE,ZOOMGRAF] ($Wetter$,
$http://eur.yimg.com/w/wcom/eur_germany_outlook_DE_DE_440_dmy_y.jpg$)
link(4) [BLIND]
[$http://eur.yimg.com/w/wcom/eur_germany_outlook_DE_DE_440_dmy_y.jpg$] $Mein Link$

page (2) [$Haus$,$Seite2$]
plink(2) [INFO] [3] $Zu Seite 3$

page (3) [$Haus$,$Seite3$]
plink(2) [WEATHER] [1] $Zu Seite 1$

[EibPC]
SprungZiel=3
if after(systemstart(),5000u64) then plink(2,$Doch zu Seite 2$,MONITOR,DISPLAY,
1,SprungZiel) endif

// Achtung: picture verwendet nur die ersten 479 Zeichen für den Link
if after(systemstart(),5000u64) then picture(3,$Neues
Wetter$,1,$http://eur.yimg.com/w/wcom/eur_satintl_440_dmy_y.jpg$) endif

// Achtung: link verwendet nur die ersten 479 Zeichen für den Link
if after(systemstart(),5000u64) then link(4,$Neuer
Link$,MONITOR,DISPLAY,1,$http://eur.yimg.com/w/wcom/eur_satintl_440_dmy_y.jpg$) endif
```

Setpeslider

Definition

- function setpeslider(*ID, Value, Icon, State, PageID*)

Arguments

- *ID, Icon, State, PageID* of data type u08
- *Value* of data type f32

Effect

- The function addresses the page-related peslider at the *ID* of the Page *PageID* and sets it to the value of *Value.* If *ID* occurs multiple times, then all elements of this ID on the website with the *PageID* are addressed.
- When calling the function, the icon is also placed on the icon with the number (type u08) *Icon*.Possible graphics are shown on page 291 and the graphics are selected via pre-defined values (u08). In addition, predefined constants facilitate the selection. The assignment of this is listed in Table 2 (Page 291).
- Each icon has at least the states ACTIVE (== 1), INACTIVE (== 2), DARKRED (== 0) and brightred (== 9). One of these states can be passed as the argument *State*. An overview of the possible states are Table 3 (Page 291).

Datentyp Ergebnis (Rückgabe)

- no

Timebufferconfig

Definition

- Function timebufferconfig(*ChartBufferID, MemTyp, Laenge, DataTyp*)

Arguments

- *ID* of data type u08
- *MemTyp* Memory Type, with "0" ring memory and "1" represents a linear memory.
- *Length* of the data in the puffer. Maximum 65535 records with max. 4 bytes in length. The data type has to be u16.
- The memory is of data type *DataTyp* of the input object.

- **Effect**
  - There is a pair of values buffer is created or configured here. It can be set using the memory type, if this becomes full after filling with the values or if the oldest value is discarded.
  - CAUTION: The EibPC has a RAM of 64MB, of which about 40 MB can be used by the user maximum.

    To ensure proper operation, the buffer and arts must be sized so that the memory of the EibPC is not overloaded. Using the function to buffer 255 for storing history data can be defined. The following applies for the necessary storage capacity = (number of values) * 12 Thus, for example, has a buffer with 65000 values about 780 kB.
  - You can store them in the Flash buffer at any time, so when you restart the values are not lost, see time buffer gates 256 and timebufferread 256.

Data type result (Return)

- Values: 0 success, 1 Error: exceeded maximum number of time buffers, 2 Error: time buffer already defined.

Detailed Example: P. 115

Timebufferadd

Definition

- Function timebufferadd(*ChartBufferID, Daten*)

Arguments

- *ID* of data type u08
- *Data* Value (max 32 bits), which has to be inserted into the memory at the end.

- **Effect**
  - There is a pair of values inserted into the buffer at the end of time here.

Data type result (Return)

- 0 success, 1 error

  Detailed example: p. 115

Timebufferclear

Definition

- Function timebufferclear(*ChartBufferID*)

Arguments

- *ChartBufferID* of data type u08

- **Effect**

  - Delete the current time buffer (in the memory and, if necessary, on the flash, if existing)

Data type result (Return)

- Level of the time buffer of the data type u16

Example

```
if systemstart() then timebufferclear(2) endif
```

Timebufferstore

Definition

- Function timebufferstore(*ChartBufferID*)

Arguments

- *ChartBufferID* of data type u08

- **Effect**

- It is permanently stored in a flash buffer.

Datentyp Ergebnis (Rückgabe)

- 0 success, 1 error, 2 ongoing processing

Detailed example: P. 115

Timebufferread

Definition

- Function timebufferread(*ChartBufferID*)

Arguments

- *ChartBufferID* of data type u08

- **Wirkung**

- A buffer is selected from the Flasch.

Datentyp Ergebnis (Rückgabe)

- 0 success, 1 error, 2 ongoing processing, data type u08

Detailed example: P. 115

Timebuffersize

Definition

- Function timebuffersize(*ChartBufferID*)

Arguments

- *ChartBufferID* of data type u08

- **Effect**

- Show the current level of the time buffer.

Data type result (Return)

- Level of the time buffer of data type u16

Detailed example: P. 115

Timebuffervalue

Definition

- Function timebuffervalue(*ChartBufferID, utcZeit,Data, utcZeitWert)*

Arguments

- *ID of* data type u08
- *utcZeit* of data type u64, which is indicated by the time stamp which is greater than or equal to the time of the next data point in the time series.
- *Data* Value (max 32 bits), which should be inserted into the memory at the end. The function changes the value of this argument to the stored value at the time when it is called. The data type must match the data type of the timebuffer (timebufferconfig).
- *utcZeitWert* The exact time of the recording time of the *Data* value. The function changes the value of this argument to the value when it is called

- **Effect**
  - A value pair is searched for in the time buffer.

Data type result (Return)

- 0 success, 1 error, 2 persistent processing.

See a detailed example on timebuffering : p. 115

Example: Reading values

A timebuffer has f16 data types and records since 1.1.2016. The value in the time buffer at the time 12:00:00 on 2.1.2016 daily should be read at 9:30:00. If a value is present in the buffer written to the buffer with plus or minus one second at this time with timebufferadd, this value is to be output to the GA '1/2/3'f16.

```
uBf=0
timebufferconfig(uBf,0,2500u16,0f16)
// requested Time
uTime=utc($2016-01-02 12:00:00$)
fVal=0f16
uSampleTime=0u64
uRet=3

if htime(9,30,00) then {
  uRet=timebuffervalue(uBf,uTime,fVal,uSampleTime);
} endif
if uRet==0 then {
      if hysteresis(uSampleTime, uTime-1000u64,uTime+1000u64) then {
           write('1/2/3'f16, fVal) ;
     }  endif
} endif
```

Webinput

Definition

- Funkcion webinput(ID)

Arguments

- *ID* of Webinput element data type u08

Effect

- reads out the webinput field and sends the result to the return value.
- Webinput elements are all globally

Data type result (Return)

- string c1400 as result

s. example weboutput S.258, S. 112)

Weboutput

Definition

- Function weboutput(ID,Data)

Argumente

- *ID* of Webinput element data type u08
- *Data* to show at weboutput field

Wirkung

- sends the string to the corresponding weboutput field in the webserver
- Weboutput elements are all globally

Data type result (Return)

- none

Example: webinput, weboutput, see p. 112

```
WebServer]
page(1)[$Enertex$,$Webserver$]
webinput(1)[INFO] $Eingabe hier -> Ausgabe in Outputfeldern$
weboutput(2)[SINGLE,ICON]

[EibPC]
inputstring=webinput(1)
if change(inputstring) then weboutput(2,inputstring) endif
```

mtimechartpos

Definition

- Function mtimechartpos(TimeChartID,ChartIdx,ChartBuffer,StartPos,EndPos)

Arguments

- TimeChartID of datatyp u08
- ChartIdx Index of charts (0..3)
- ChartBuffer Handle to the time buffer to be displayed by the web element. The Webelement has to be configured accordingly.
- StartPos Starting position of the display
- EndPos Ending position of the display

Effect

- Specify the displayed portion of a time buffer for the web element.

Data type result (Return)

- none

Examples p. 115 to 119

mtimechart

Definition

- Function mtimechart(TimeChartID,ChartIdx,ChartBuffer,StartZeit,EndZeit)

Arguments

- TimeChartID of Datatyp u08
- ChartIdx-Index of charts (0..3)
- ChartBuffer Handle to the time buffer to be displayed by the web element. The Webelement has to be configured accordingly.
- StartZeit Starting position of the display used as UTC Time-Tics
- EndZeit Ending position of the display used as UTC Time-Tics

Effect

- Specify the displayed portion of a time buffer for the web element.

Data type result (Return)

- no

Examples P. 115 to 119

Pdisplay

Definition

- Function pdisplay(*id, text, icon, state, style, page_id, [mbutton]*)

Arguments

- Arguments *id, icon, state, style* and *page_id* of data type u08
- Argument *text* of arbitrary data type

Effect

- The function addresses the web button that refers to a page (*pbutton* or *pshifter*). If there are multiple web buttons with *id* on the web page of *page_id*, they all will be addressed.
- By means of the optional argument *mbutton*, the displayed selection of the drop-down box can be changed.
- At function plink this argument specifies the jump index.
- Calling this function sets the icon of the web element with *id* to the symbol defined by *icon* (data type u08). Possible images are shown on page 291 and are selected by predefined numbers (data type u08). In addition, predefined constants facilitate the choice. Their respective allocation is listed in Table 2 (page 159).
- The argument *text* denotes an arbitrary variable the value of which, converted to a character string, is displayed in the variable text line of the web element.
- At function link this argument specifies the new link.
- Every icon has at least the states ACTIVE (==1), INACTIVE (==2), DARKRED (==0) and BRIGHTRED (==9). One of these states can be submitted as the argument *state*. For an overview of the possible states see Table 3 (page 159).
- The text to be displayed can be represented in the styles GREY (==0), GREEN (==1), BLINKRED(==2) and BLINKBLUE (==3).

Data type result (Return)

- none

Pchart

Definition

- Function pchart(*id, var, x1, x2, page_id*)

Arguments

- Arguments *id, var, page_id* of data type u08
- Arguments *x1, x2* of data type c14

Effect

- This function addresses the XY diagram *chart*. If there are multiple occurrences of *id*, all elements of this id on the web page of *page_id* are addressed.
- When calling this function, the XY diagram of the value *var* is activated. Values in the range of 1...30 can be displayed. 0 refers to the value not being displayed, and values greater than 30 are not permitted and are interpreted like 0. Every call of the function displays the values beginning from the left side. When the end is reached after 47 function calls, the values are shifted to the left.
- The labeling of the x-axis is given by the arguments *x1, x2* (data type c14).

Data type result (Return)

- Data type u08 (internal state of the webchart).

Pbutton

Definition

- Function pbutton(*id,page_id*)

Arguments

- Argument *id* of data type u08. This argument must not change at the runtime of the program.
- Argument *page_id* of data type u08. This argument must not change at the runtime of the program.

Effect

- By operating the button of a web button element that refers to a page (e.g. *pbutton* or *pshifter*) with the *id* (page 265 and the following) on the web page of *page_id*, the function assumes a value not equal to zero for the duration of one processing pass, and zero in all other cases.
- For a *pbutton* element, the return value when operated is 1.
- For a *pshifter* element, the return value when operated is 1, 2, 3 or 4 (u08) depending on the actually operated element of the web button. The return values refer to the order of the buttons (from left to right).

picture

Definition

- Function picture(*id, label, page_id, www-LINK*)

Arguments

- Arguments *id* and *page_id* of data type u08
- Argument *text* of arbitrary data type
- Argument *www-LINK* of data type c1400

Effect

- The function addresses the web button that refers to a page (*picture*). If there are multiple web buttons with *id* on the web page of *page_id*, they all will be addressed.
- Calling this function sets the icon of the web element with *id* to the symbol defined by *icon* (data type u08). Possible images are shown on page 291 and are selected by predefined numbers (data type u08). In addition, predefined constants facilitate the choice. Their respective allocation is listed in Table 2 (page 159).
- The argument *text* denotes an arbitrary variable the value of which, converted to a character string, is displayed in the variable text line of the web element.
- The argument *www-LINK* Valid WWW address (incl..Path and leading http://) to the external image specified the new destination. The link is shortened to 479 characters due to compatibilities restrictions.

Data type result (Return)

- none

plink

Definition

- Function plink(*id, text, icon, page_id, pageDestination*)

Arguments

- Arguments *id, icon, page_id* and *pageDestination* of data type u08
- Argument *text* of arbitrary data type

Effect

- The function addresses the web button that refers to a page (*plink*). If there are multiple web buttons with *id* on the web page of *page_id*, they all will be addressed.
- Calling this function sets the icon of the web element with *id* to the symbol defined by *icon* (data type u08). Possible images are shown on page 291 and are selected by predefined numbers (data type u08). In addition, predefined constants facilitate the choice. Their respective allocation is listed in Table 2 (page 159).
- The argument *text* denotes an arbitrary variable the value of which, converted to a character string, is displayed in the variable text line of the web element.
- The argument *pageDestination* specified the page id as new destination

Data type result (Return)

- none

Setslider

Definition

- Function setslider(*id, value, icon, state*)

Arguments

- All arguments of data type u08

Effect

- The function addresses the *slider* and sets its value to *value*. If there are multiple occurrences of *id*, all elements of this id are addressed.
- A call of the function sets the icon to the symbol with the number *icon*. Possible symbols are shown on page 291 and are referenced with predefined values (u08). Further predefined constants make the choice easier. Table 2 (page 159) lists the assignment.
- Every icon has at least the states ACTIVE (==1), INACTIVE (==2), DARKRED (==0) and BRIGHTRED (==9). One of these states can be set in the argument *state*. Table 3 (page 159) provides an overview over all possible states.

Data type result (Return)

- none

Setpslider

Definition

- Function setpslider(*id, value, icon, state page_id*)

Arguments

- All arguments of data type u08

Effect

- The function addresses the *pslider* that refers to a page at the *id* on page *page_id* and sets it to the value *value.* If there are multiple occurrences of *id*, all elements of this id on the web page with *page_id* are addressed.
- A call of the function sets the icon to the symbol with the number *icon*. Possible symbols are shown on page 291 and are referenced with predefined values (u08). Further predefined constants make the choice easier. Table 2 (page 159) lists the assignment.
- Every icon has at least the states ACTIVE (==1), INACTIVE (==2), DARKRED (==0) and BRIGHTRED (==9). One of these states can be set in the argument *state*. Table 3 (page 159) provides an overview over all possible states.

Data type result (Return)

- none

Seteslider

Definition

- Function seteslider(*id, value, icon, state*)

Arguments

- All arguments of data type u08

Effect

- The function addresses the *eslider* and sets its value to *value*. If there are multiple occurrences of *id*, all elements of this id are addressed.
- A call of the function sets the icon to the symbol with the number *icon*. Possible symbols are shown on page 291 and are referenced with predefined values (u08). Further predefined constants make the choice easier. Table 2 (page 159) lists the assignment.
- Every icon has at least the states ACTIVE (==1), INACTIVE (==2), DARKRED (==0) and BRIGHTRED (==9). One of these states can be set in the argument *state*. Table 3 (page 159) provides an overview over all possible states.

Data type result (Return)

- none

Setpeslider

Definition

- Function setpeslider(*id, value, icon, state page_id*)

Arguments

- All arguments of data type u08

Effect

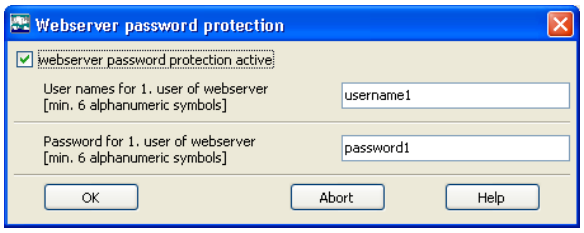
- The function addresses the *peslider* that refers to a page at the *id* on page *page_id* and sets it to the value *value.* If there are multiple occurrences of *id*, all elements of this id on the web page with *page_id* are addressed.
- A call of the function sets the icon to the symbol with the number *icon*. Possible symbols are shown on page 291 and are referenced with predefined values (u08). Further predefined constants make the choice easier. Table 2 (page 159) lists the assignment.
- Every icon has at least the states ACTIVE (==1), INACTIVE (==2), DARKRED (==0) and BRIGHTRED (==9). One of these states can be set in the argument *state*. Table 3 (page 159) provides an overview over all possibel states.

Data type result (Return)

- none

HTTPS

With Enertex ® EibPC secure communications via HTTPS between Web servers and browsers is possible. Therefore choose the menu ® Options → Webserver Password protection and follow the instructions (Figure 8). You also need to configure port forwarding for port 443 in your router.



☑ webserver password protection active

User names for 1. user of webserver [min. 6 alphanumeric symbols]: username1

Password for 1. user of webserver [min. 6 alphanumeric symbols]: password1

Figure 8: HTTPS configuration dialog

Configuration of the web server

Mozilla Firefox as a client

The built-in web server allows the visualization of data and automation processes. For this, you only need a web browser.

The web server of the Enertex® EibPC has been successfully tested with the web browsers Microsoft Internet Explorer 7 and Mozilla Firefox 3. Best performance is achieved with Mozilla Firefox 3.

The design of the web server

The design of the buttons is predefined

The integrated web server arranges its elements which have either a predefined single or double length (cf. Figure 9) like a checkerboard pattern. There is basically the option not to use some fields and to insert dividers.

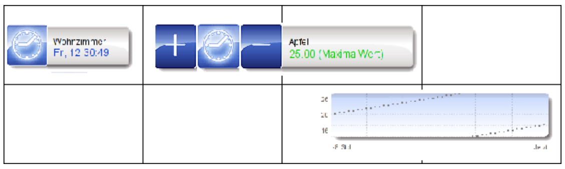


Figure 9: Scheme of the web server

The arrangement and the configuration of the web elements are carried out in the section [WebServer]. Due to the fixed specification of icons, lengths, and variables, the configuration can take place without graphical effort, and a professional web interface can be established in a simple way. The icons (overview on page 291) have a consistent design.

You can configure the web server with a single page (as provided in firmware versions < 1.200) or in the multiple-page version.

You can configure and use a maximum of 40 elements per page. The graphic design of each button is fixed to a given design set. You can change the design set of a complete set, though (see page Fehler: Referenz nicht gefunden).

Navigation with multiple pages

When using multiple pages, you can assign every page to a group. Via the list box of the page navigation (cf. Figure 10), you can select the grouped pages.

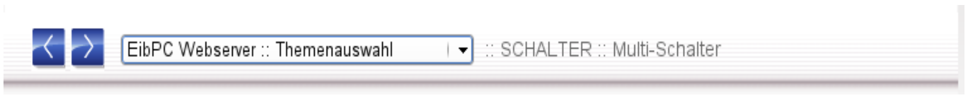

EibPC Webserver :: Themenauswahl
:: SCHALTER :: Multi-Schalter

Figure 10: Page navigation

Also the page navigation is generated automatically. At this, the pages are defined by the page configuration command in section [WebServer]. If a page is assigned to a group by this command, it appears in the selection box in this order. In this manner, groups like “basement", “first floor” etc. can be generated.

The quick selection (forward and back button, respectively, in Figure 10) is given by the order of the definition.

There are the following groups of elements for the visualization, with these constituting global or local elements. Global means that the element can be used on multiple pages, but has been generated only once. One access / change of the element in the application program is therefore identical for all pages.

On the contrary, a local element can be changed only on one page. Concerning the design, local and global elements are identical and marked by the addition “p” (page).

User administration

As of Patch-Version 3.xxx a page-related user administration of the webserver is possible. Every page can be saved with an userword and a password. Therewith more than one user can be tolerated by one page.

For each user a password can be allocated, which has to be indicated in the first definition of the username.

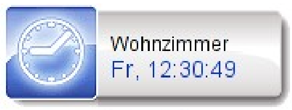
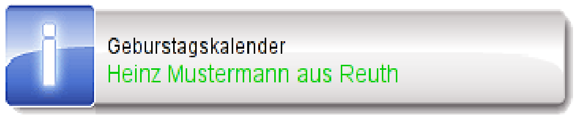
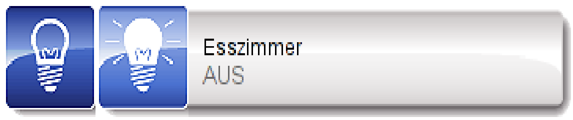
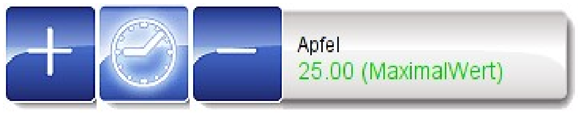
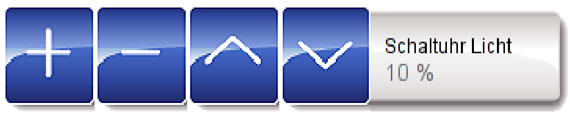
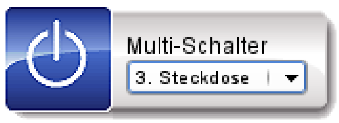
Example:

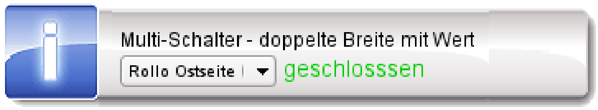
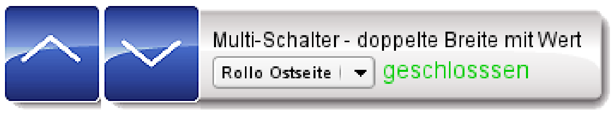
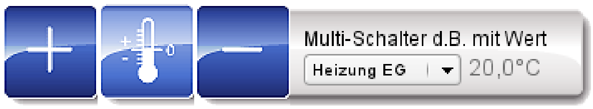
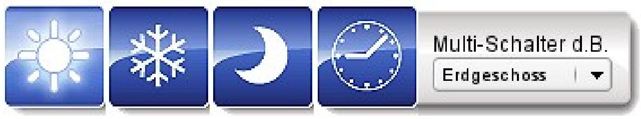
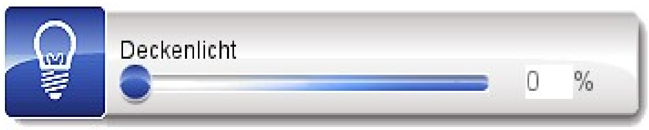
```
[WebServer]
page(1) [$User administration$,$page 1$]
user $Michael$ [PasswordM]
user $Florian$ [PasswordF]
button(1) [INFO] $page 1$

page(2) [$user administration$,$page 2$]
// Passwords are going to overtaken
user $Michael$
user $Florian$
button(1) [INFO] $page 2$

page(3) [$user administration$,$page 3$]
// This page is only for Michael
// Password is going to overtaken
user $Michael$
button(1) [INFO] $page 3$

page(4) [$user administration$,$page 4$]
// This page is only for Stefanie
// Password has to be specified, because this user was not mentioned on the pages before
user $Stefanie$ [Sgood]
button(1) [INFO] $page 4$

page(5) [$user administration$,$Seite 5$]
// All users
button(1) [INFO] $page 5$
```

Elements of the web server

There a two groups of elements for displays, the *Webbutton* and the *Webdisplay*. Of these, the element *button* (sub-group of W*ebbutton*) is the only element which exhibits the single width. At last, there are also design elements to mention: The header and footer (*header*) and the divider (*line*) (cf. Table 1). The following pictures are screen shots of the standard blue design (see also p. Fehler: Referenz nicht gefunden).

Every web button can modify a graphic and a line of text dynamically at the runtime of the program.

Group	*Element*	*Description*
button		
	button, pbutton	 Wohnzimmer Fr, 12:30:49 The graphic constituting the actual control panel can be modified by the user program. The first line of text is static (only changeable at the configuration). The second line can be modified by the user program, e.g. to display variables.
	shifter, pshifter	 Geburstagskalender Heinz Mustermann aus Reuth The graphic can be modified by the user program. The first line of text is static (only changeable at the configuration). The second line can be modified by the user program.
	shifter, pshifter	 Esszimmer AUS The **right** graphic can be modified by the user program. The **left** graphic can be modified only at the configuration. The first line of text is static (only changeable at the configuration). The second line can be modified by the user program.
	shifter, pshifter	 Apfel 25.00 (MaximalWert) The **middle** graphic can be modified by the user program. The **outer** graphics can be modified only at the configuration. The first line of text is static (only changeable at the configuration). The second line can be modified by the user program.
	shifter	 Schaltuhr Licht 10 % The **right** graphic can be modified by the user program. The other graphics can be modified only at the configuration. The first line of text is static (only changeable at the configuration). The second line can be modified by the user program.
mbutton		
	mbutton, mpbutton	 Multi-Schalter 3. Steckdose The graphic constituting the actual control panel can be modified by the user program. The first line of text is static (only changeable at the configuration). The active selection can be modified by the user program, with the latter having to adjust the state of the graphic. No text can be displayed in the second line. The listbox can administer a maximum of 254 entries. By operating the listbox, a signal which can be queried by the functions mbutton (page 249) and mpbutton (page 252), respectively, is sent to the application program.

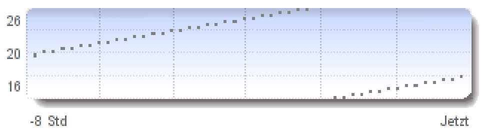
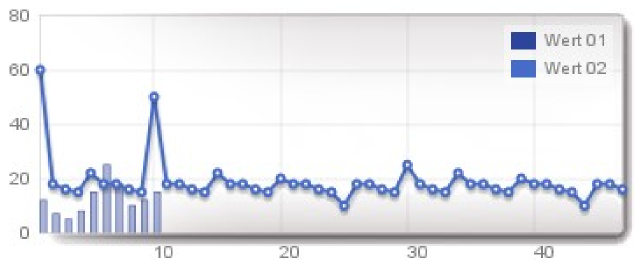
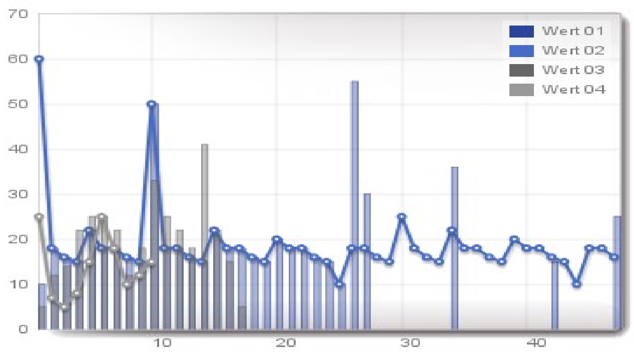
Group	Element	Description
	mshifter, mpshifter	 Multi-Schalter - doppelte Breite mit Wert Rollo Ostseite geschlosssen The graphic constituting the actual control panel can be modified by the user program. The first line of text is static (only changeable at the configuration). The second line can be modified by the user program. The listbox can administer a maximum of 4 entries. By operating the listbox, a signal which can be queried by the functions mbutton (page 249) and mpbutton (page 252), respectively, is sent to the application program.
	mshifter, mpshifter	 Multi-Schalter - doppelte Breite mit Wert Rollo Ostseite geschlosssen The **right** graphic can be modified by the user program. The **left** graphic can be modified only at the configuration. The first line of text is static (only changeable at the configuration). The second line can be modified by the user program. The listbox can administer a maximum of 4 entries. By operating the listbox, a signal which can be queried by the functions mbutton (page 249) and mpbutton (page 252), respectively, is sent to the application program.
	mshifter, mpshifter	 The **middle** graphic can be modified by the user program. The **outer** graphics can be modified only at the configuration. The first line of text is static (only changeable at the configuration). The second line can be modified by the user program. The listbox can administer a maximum of 4 entries. By operating the listbox, a signal which can be queried by the functions mbutton (page 249) and mpbutton (page 252), respectively, is sent to the application program.
	mshifter, mpshifter	 The **right** graphic can be modified by the user program. The other graphics can be modified only at the configuration. The first line of text is static (only changeable at the configuration). No text can be displayed in the second line. The listbox can administer a maximum of 4 entries. By operating the listbox, a signal which can be queried by the functions mbutton (page 249) and mpbutton (page 252), respectively, is sent to the application program.
	slider *pslider*	 The image and the position of the sliders can be set in the application porgramm with the functions setslider and setpslider. Clicking the button element triggers the functions mbutton (page 249) and mpbutton (page 252), respectively.
	eslider *peslider*	The image and the position of the sliders can be set in the application porgramm with the functions setslider and setpslider. Clicking the button element triggers the functions mbutton (page 249) and mpbutton (page 252), respectively. The mininum, the maximum value and the increment can be parametrized.

By means of mchart, you can plot up to four different graphs ...

... either of "single" ...

.. or of "double" height ...

... or integrate an external image source...

Group	Element	Description
chart		
	chart, pchart	 This element serves the purpose of visualizing a time series. The labeling of the y-axis is defined at the configuration. The labeling of the x-axis can be modified by the user program. When calling the function webdisplay, the XY diagram is activated. Values from the field 1...30 can be represented. 0 means no representation. The values are displayed starting from the left. When the end is reached after 47 calls, the values are shifted to the left.
	mchart mpchart	 The pairs of variates are addressed by the application program via the function mchart (page Fehler: Referenz nicht gefunden). One element mchart administers up to 4 XY charts that can be supplied with data via the identical function mchart in the application program. A maximum of 4 diagrams can be defined, each having a labeling of its own (inserted in the top right corner). Up to 47 floating-point values are displayed. The scale is generated automatically.
	mchart mpchart	 like above, though double height.
	picture	°C Vorhersage für 07/05/10 Frankfurt Nürnberg Stuttgart München 06 May 2010 05:26 GMT weather.com Anzeige eines externen Bildes z. B. Wetterkarte für Deutschland An external link to a graphic is integrated. The graphic can be left-justified, centered or right-justified.

	picture	°C Vorhersage für 07/05/10 Hamburg Berlin Hannover Essen Leipzig Frankfurt Nürnberg Stuttgart München 06 May 2010 05:26 GMT The Weather Channel weather.com Anzeige eines externen Bildes z. B. Wetterkarte für Deutschland An external link to a graphic is integrated. The graphic can be left-justified, centered or right-justified.

... or integrate complete web sites ...

Group	***Element***	***Description***
Link		
	frame dframe	Embedding an external website
	pLink	Link to an internal page (simple button)
	Link	Link to an external page (simple button)
Decorations		
	line	Überschrift Enforces an empty line with a divider in the web server arrangement. The caption is optional.
	header	Header. Can be switched off to make the handling of touchpanels easier. Likewise, a link to an external image source is possible. At this, the scale should be adapted to the size.
	footer	Footer. Can be switched off to make the handling of touchpanels easier. Likewise, a link to an external image source is possible. At this, the scale should be adapted to the size.
	none	An empty field of single width.

Table 1: Overview of web elements.

Configuration

The design of the web server is integrated into the Enertex® EibPC in a fixed way. The scheme according to page 265, Figure 9 can be extended to ten columns. The web server administers up to 60 (IDs from 0 to 59) web elements on one web page.

The configuration of the web server takes place in the section [WebServer] in the user program. For this, the elements which are arranged in a line simply have to be - separated by one or more space or tab characters - configured as follows. The compiler detects the number of elements per line and configures the "checkerboard pattern" automatically (Figure 9). Each element must be indexed so that it can be accessed by the user program via the respective functions.

compact

Element *Compact*

- *compact* (STATE)

Arguments

- State value of 0 / 1 or ON/OFF

The web server is built in unit sizes. All elements fit into this grid or are integer multiples thereof, as already explained in Figure 9 . Therefore, when a four-fold height element (e.g., mpchart) is configured next to a simple-height element,

An example without „compact" mode

```
[WebServer]
page(1) [$Demo$,$Compact$]
// the next command is default
compact(off)
// Two elements
mpchart(1) [DOUBLE, SXY]($Description1$,LINE) mpshifter(2) [$Basement$,$OG$][WEATHER, ICE, NIGHT, CLOCK]
$Multi$
```

a clearance is created in the representation as shown in Figure 11.

generates a clearance under the 4-way multi button

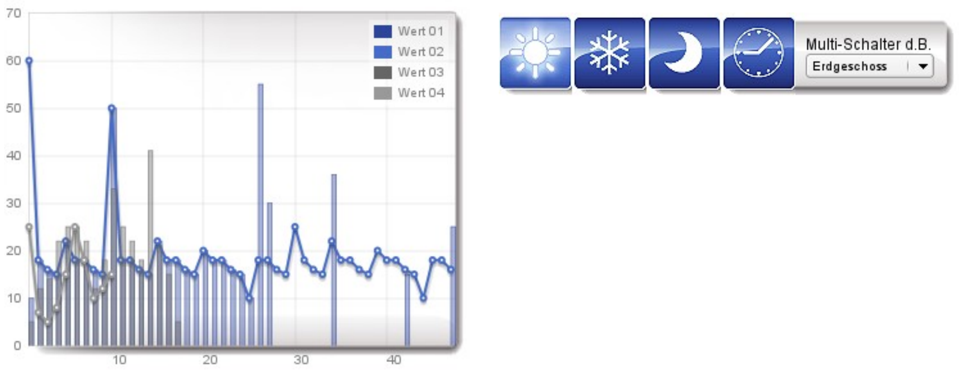


Figure 11: Clearance

When configuring the Web server, each line of the text configuration represents a web server display line. In the "switched off" (compact (off)) mode, the elements of different heights are always arranged in one line, that is, the actual line height of the representation is indicated by the max. Height of all elements in the respective line. This creates the clearance in the web server. In other words, in the representation additional non-visible elements are placed under the elements. Figure 12 shows this "allocation" of the unit sizes (shown in blue) of the above web configuration.

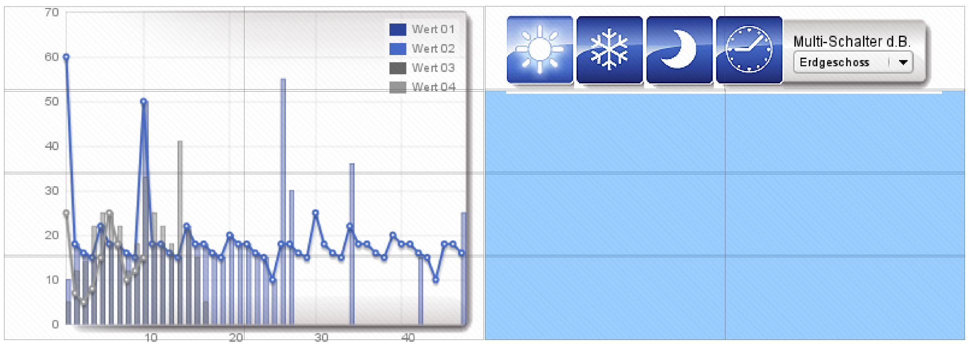


Figure 12: Illustration of the unit sizes

The eibparser already displays the configuration in the Messages window:

The output of the eibparser

```
====== Seite: 01/Demo  ======
 mchart (1)   -  mpshifter (2)    -
      |       |         o         o
      |       |         o         o
      |       |         o         o
```

In this case, a cross-bar ("-") means that the element to the right occupies this "place", i.e. this unit size, a vertical bar "|" means that the element above occupies this place. A round circle is an empty element (none) generated automatically or by the user. In Figure 12 the automatic generated free spaces are shown in blue. This output thus clearly illustrates the user's visualization of the structure as it is displayed by the web server. Compare the above output of the eibparser with Figure 12.
If you now want to use the free space to the right of the diagram, the configuration has to be changed. e.g.: one would like to set additional multibuttons beside the graphics (Figure 13).

Compact mode

```
page(1) [$Demo$,$Compact$]
// the next command is default
compact(on)
mpchart(1) [DOUBLE, SXY]($Description1$,LINE) mpshifter(2) [$Basement$,$OG$][WEATHER, ICE, NIGHT, CLOCK] $Multi$
mpshifter(3) [$Keller$,$OG$][PLUS, TEMPERATURE, Minus] $Multi$
mpshifter(4) [$Keller$,$OG$][PLUS, TEMPERATURE, Minus] $Multi$
mpshifter(5) [$Keller$,$OG$][PLUS, TEMPERATURE, Minus] $Multi$
```

Transfer of occupied unit elements across lines

The first line is as before. Now the clearances of Figure 12 can be used when working in Compact mode. In Compact mode, the elements are not arranged in rows at different heights. Since the line

```
mpchart(1) [DOUBLE, SXY]($Description1$,LINE) mpshifter(2) [$Basement$,$OG$][WEATHER, ICE, NIGHT, CLOCK] $Multi$
```

configures a mpchart with a double-width and four-fold height, its display projects down into three further lines.

In the lines

```
mpshifter(3) [$Basement$,$OG$][PLUS, TEMPERATURE, Minus] $Multi$
mpshifter(4) [$Basement$,$OG$][PLUS, TEMPERATURE, Minus] $Multi$
mpshifter(5) [$Basement$,$OG$][PLUS, TEMPERATURE, Minus] $Multi$
```

elements with double width and simple height are installed. Through the first element two additional unit elements in the line are already "invisible". The eibparser already outputs this line overflow by issuing the "-" or "|" characters: aus:

```
 ====== Seite: 01/Demo  ======
 mchart (1)   -  mpshifter (2)    -
      |       |  mpshifter (3)    -
      |       |  mpshifter (4)    -
      |       |  mpshifter (5)    -
```

See Figure 13, which is now output by the web server:

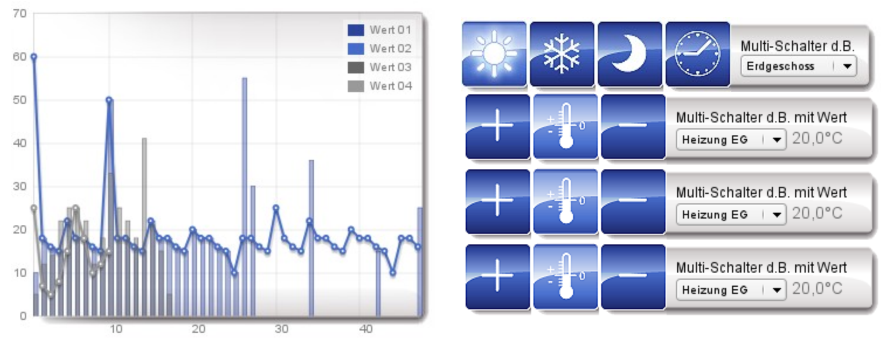


Figure 13: Compact mode

The compact (ON) statement can be used to enable the placement of elements of different heights next to each other. The web server itself calculates the heights overflow in the next line. The user may not place any **none** elemente elements here, if the width is not to be increased. Figure 14 shows again schematically the arrangement of the elements, as is already output in the eibparser

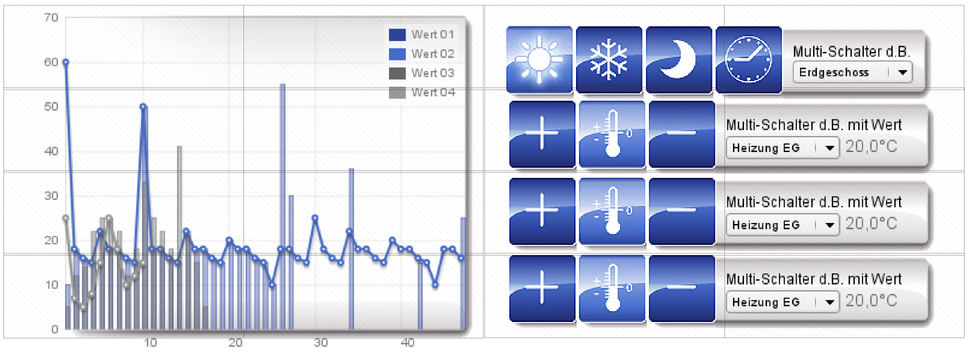


Figure 14: „compact“ with grid (for illustrative purposes)

In the mode with *compact* (on) of the web server, the user must therefore take into account the size of the web element in the next line of the configuration in order to control the arrangement of the web elements. If you want to generate a free line with consideration of line overflows, you must work with the empty element.
The following example illustrates this

```
page(1) [$Demo$,$Compact$]
// the next command is default
compact(on)
mpchart(1) [DOUBLE, SXY]($Description1$,LINE) mpchart(2) [DOUBLE, SXY]($Description$,LINE)
mpshifter(3) [$Basement$,$OG$][WEATHER, ICE, NIGHT, CLOCK] $Multi$
```

The first two elements occupy 2 unit widths and 4 unit heights. After the line break in the configuration of the two mpcharts a new line starts in the representation. This has a "carry" of two times two occupied unit elements. Then a mpshifter is configured in the next line. Therefore, the side must be at least 6 unit elements wide. This is also output by the eibparser:

```
====== Seite: 01/Demo  ======

 mchart (1)    -  mchart (2)   -        o         o
      |       |       |       |  mpshifter (3)    -
      |       |       |       |         o         o
      |       |       |       |         o         o
```

Ultimately, the Web server will output a representation as in Figure 15:

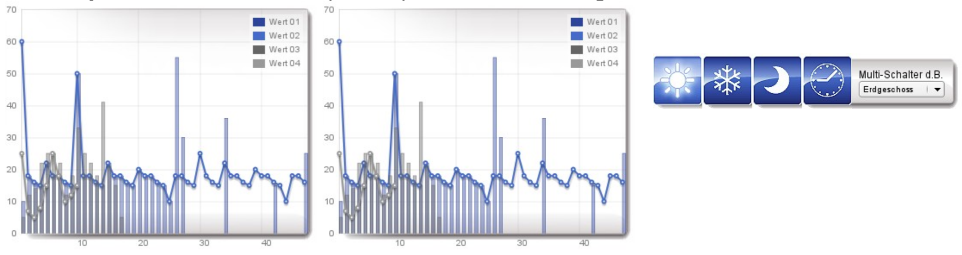


Figure 15: Representation example for line feed

If you now want the four-button button to be displayed below the two graphs, empty elements must be configured as follows:

```
page(1) [$Demo$,$Compact$]
// the next command is default
compact(on)
mpchart(1) [DOUBLE, SXY]($Description1$,LINE) mpchart(2) [DOUBLE, SXY]($Description1$,LINE)
empty
empty
empty
mpshifter(3) [$Basement$,$OG$][WEATHER, ICE, NIGHT, CLOCK] $Multi$
```

The three Empty elements now insert empty lines or skip one line in the display. Also here this can already be recognized in advance by means of the output specified by the eibparser in the message window:

```
====== Seite: 01/Demo  ======

 mchart (1)        -  mchart (2)    -
        |         |       |        |
        |         |       |        |
        |         |       |        |
 mpshifter (3)     -       o        o
```

The side index of local elements - elements, which are assigned to only one side,- is this one, given by the previous page command.

Button

Element *button*

- button(ID)[Image] $Text$

Arguments

- ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access variables of the section [EibPC].
- Image: A value between 0 and 99. To arrange the application more clearly, constants have been predefined (page 291).
- Text: A static labeling text (first line).

Access by the user program

- The image and the text are accessed by the function Display (Webdisplay) (page 248).
- It is a global button. I. e. if the there are equal definitions on more than one pages, all buttons with this ID are affected at all pages.
- Activation of the buttons has to be evaluated by the function Button (page 247).

Design

Element *design*

- design $DESIGNSTRING$ [$Link/Path$]

Arguments

- $DESIGNSTRING$ can be $black$ for a black design (well suited for wall mounted touch panels or smart phones)
- $DESIGNSTRING$ can be $blue$ for a blue design shown in the screen shots.
- The design command can configure each site differently
- $Link/Path$ is a link to an internal stored image (see p. 199) or to an external server providing the image. The image will not be scaled. The position of the web elements is not influenced by this image, none-elements will be transparent.

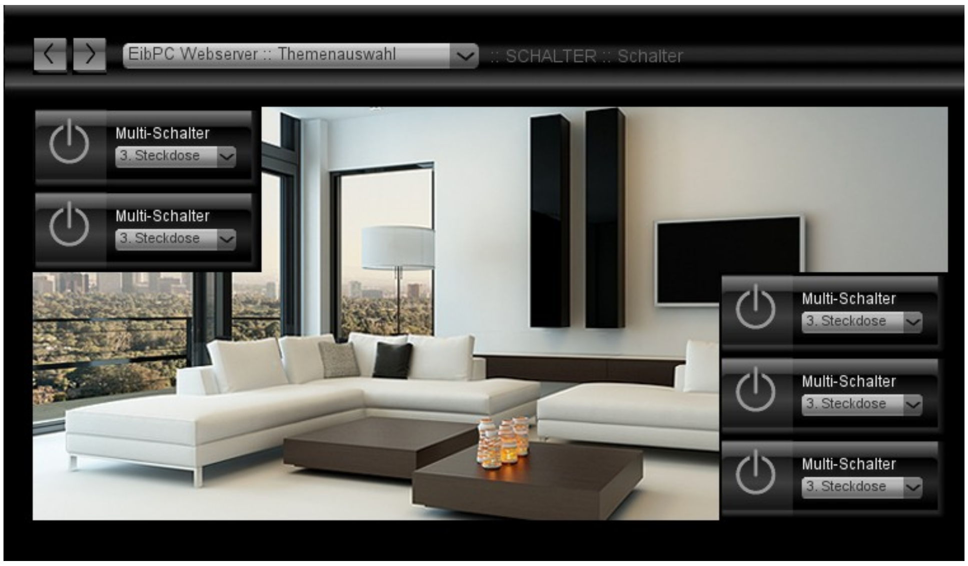


Figure 16: background graphics

Mobilezoom

Element *mobilezoom*

- *mobilezoom(Factor)*

Arguments

- *Factor*: integer value from 0 to 255 as a zoom factor in percent for the zoom of the visualization on mobile devices or Android-bayed panels. The zoom factor only affects the page that was initially defined with a previous page configuration

Mbutton

Element *mbutton*

- *mbutton*(ID)[$Text1$,$Text2$,... $Text254$][Image] $Label$

Arguments

- ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access variables of the section [EibPC].
- Text1, Text2, .. Text254: label texts for *mbutton*. The second and following elements are optional.
- Image: A value between 0 and 99. To arrange the application more clearly, constants have been predefined (page 291).
- Label: A static labeling text (first line).

Access by the user program

- The image and the text are accessed by the function Display (Webdisplay) (page 248).
- It is a global button. I. e. if the there are equal definitions on more than one pages, all buttons with this ID are affected at all pages.
- Activation of the buttons has to be evaluated by the function Button (page 247).
- Switching of the listbox (providing the active listbox element) is arranged by the function Display (Webdisplay) (page 248)

Pbutton

Element *pbutton*

- *pbutton*(ID)[Image] $Text$

Arguments

- ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access variables of the section [EibPC].
- Image: A value between 0 and 99. To arrange the application more clearly, constants have been predefined (page 291).
- Text: A static labeling text (first line).

Access by the user program

- The image and the text are accessed by the function pdisplay (page 260).
- The element is assigned to only one side
- Activation of the buttons has to be evaluated by the function pbutton (page 261).

Mpbutton

Element *mpbutton*

- *mpbutton*(ID) [$Text1$,$Text2$,...$Text254$][Image] $Label$

Arguments

- ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access variables of the section [EibPC].
- Text1, Text2, .. Text254: label texts for *mbutton*. The second and following elements are optional.
- Image: A value between 0 and 99. To arrange the application more clearly, constants have been predefined (page 291).
- Label: A static labeling text (first line).

Access by the user program

- The image and the text are accessed by the function pdisplay (page 260). Switching of the listbox (providing the active listbox element) is also arranged by this function.
- Activation of the buttons has to be evaluated by the function mpbutton (page 252).

Shifter

Element *shifter*

- shifter(ID)[Image1, Image2, Image3, Image4]$Text$

Arguments

- ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access variables of the section [EibPC].
- Image1 to Image4: A value between 0 and 99. To arrange the application more clearly, constants have been predefined (page 291).
- Image2 to Image4 are optional.
- If only three images are defined, the element has only three buttons etc..
- Text: A static labeling text (first line).

Access by the user program

- The image and the text are accessed by the function pdisplay (S. 260), as explained in Table 1.
- The operation of the buttons has to be evaluated by the function button (page 247).

Pshifter

Element *pshifter*

- *pshifter*(ID)[Image1, Image2, Image3, Image4]$Text$

Arguments

- ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access variables of the section [EibPC].
- Image1 to Image4: A value between 0 and 99. To arrange the application more clearly, constants have been predefined (page 291).
- Image2 to Image4 are optional.
- If only three images are defined, the element has only three buttons etc..
- Text: A static labeling text (first line).

Access by the user program

- The image and the text are accessed by the function pdisplay (S. 260).
- The operation of the buttons has to be evaluated by the function pbutton (page 261).

Mshifter

Element *mshifter*

- *mshifter*(ID)[$Text1$,$Text2$,...,$Text254$][Image1, Image2, Image3, Image4]
  $Label$

Arguments

- ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access variables of the section [EibPC].
- Image1 to Image4: A value between 0 and 99. To arrange the application more clearly, constants have been predefined (page 291).
- Image2 to Image4 are optional.
- If only three images are defined, the element has only three buttons etc.
- Text1, Text2, .. Text254: labels for the *mshifter. The second and following elements are optional.*
- Label: A static labeling text (first line).

Access by the user program

- The image and the text are accessed by the function pdisplay (page 260). Switching of the listbox (providing the active listbox element) is also arranged by this function.
- Activation of the buttons has to be evaluated by the function mbutton (page 247).

Mpshifter

Element *mpshifter*

- *mpshifter*(ID)[$Text1$,$Text2$,...,$Text254$][Image1, Image2, Image3, Image4] $Label$

Argumente

- ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access u08 variables of the section [EibPC].
- Image1 to Image4: A value between 0 and 99. To arrange the application more clearly, constants have been predefined (page 291).
- Image2 to Image4 are optional.
- If only three images are defined, the element has only three buttons etc.
- Text1, Text2, .. Text254: labels for the *mpshifter*. The second and following elements are optional.
- Label: A static labeling text (first line).

Access by the user program

- The Image and the text are accessed by the function pdisplay (page 260). Switching of the listbox (providing the active listbox element) is also arranged by this function.
- Activation of the buttons has to be evaluated by the function mbutton (page 247).

Chart

Element *chart*

- chart(ID)[$Y0$,$Y1$,$Y2$]

Arguments

- ID: A value between 0 and 255 as an index for programming and the access to this element.
- $Y0$, $Y1$,$Y2$: Labeling of the y-axis.

Access by the user program

- The y-values are accessed in the user program by the function Chart (page 247), as explained in Table 1.
- Values from the field 1...30 can be represented. With every call of this function, the values are displayed starting from the left. When the end is reached after 47 calls, the values are shifted to the left.

Mchart

Element *mchart*

- *mchart*(ID) [Height,Type]($Label1$,Style1, $Label2$,Style2, $Label3$,Style3, $Label4$,Style4)

Arguments

- ID: Value between 0 and 59 as an index for programming and the access to this element.
- *Height*:Value 0 or 1 (or constant SINGLE and DOUBLE)
- *Type*: Value 9 (or constant SXY) for plots with sorted X-Y sets (well suited for time-based plots)
- *$Label1$ .. $Label2$* Legend of the graph
- *Style1, Style2, Style3, Style4:* value 0,1,2 or 3 (constant LINE, DOTS, LINEDOTS, COLUMN)

Access by the user program

- XY values are accessed with the function mchart (page Fehler: Referenz nicht gefunden) in the user program as explained in in Table 1. A *mchart* manages up to 4 XY diagrams. The number of diagrams is specified through the number of arguments.
- Each XY diagram has a legend. When you display 4 XY diagrams, also 4 legend are displayed.
- 47 floating point values are display in a diagram. The scale is generated automatically. Please consider the additional information given by the function mchart() on page Fehler: Referenz nicht gefunden.

Mtimechart

mtimechart element

- mtimechart (ID) [size, type, length, YLMIN, YLMAX, YRMIN, YRMAX] ($Description1$, ChartPos1, Buffer1, $Description2$, ChartPos2, BUFFER2, $Description3$, ChartPos3, buffer3, $Description4$, ChartPos4, Buffer4)
- $Description1$, CHARTPOS1, verkn.Buffer1, $Description2$, ...(up to 4 graphs)

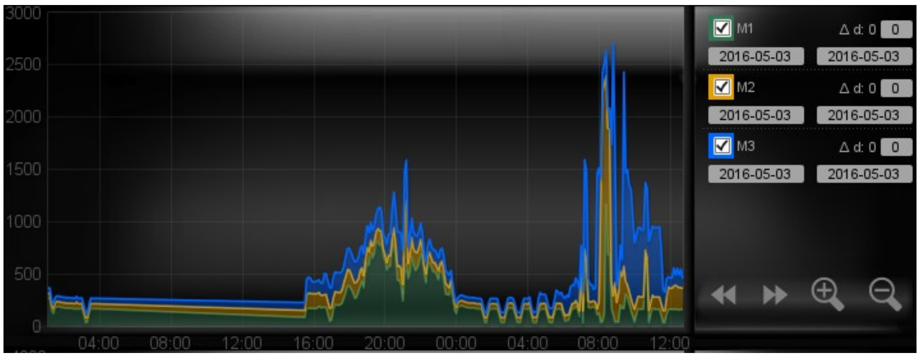
Arguments

- ID: A value between 0 and 59 as an index for programming and access to this element.
- Format: DOUBLE, TRIPLE , QUAD , LONG , EXTDOUBLE , EXTTRIPLE , EXTLONG
- Type : 0 for auto scale to the left axis , in this case YLMAX is ignored etc. (0=AUTOSCALELEFT)

  1 for autoscale the right axis , in this case YRMAX is ignored etc. (1=AUTOSCALERIGHT)

  2 for auto scale of the two axes (2=AUTOSCALE)

  3 for no autoscale (3=NOAUTOSCALE)
- Length: Maximum number of pairs of values that can be displayed per graph (Possible values : from 32 to 256)
- YLMIN : Minimum value left y - axis , floating point numbers
- YLMAX : Maximum value left y - axis , floating point numbers
- YRMIN : minimum value right y - axis , floating point numbers
- YRMAX : maximum value right y - axis , floating point numbers
- $Description1$ ... $Description4$ Legend of the corresponding graphs
- ChartPos : 0 (LEFTGRAF) or 1 (RIGHTGRAF) (0 for marking on the left y-axis, for one caption on the right y-axis) or 2 (STACK) for graphically adding two graphs: The outermost envelope is to be understood as the total sum of the individual graphs:



- Buffer: ID of the graphs associated with the respective time buffer. Values between 0 and 255 as an index for the programming and the access .
- CAUTION: The EibPC has a RAM of 64MB .

  To ensure proper operation , the buffer and arts must be dimensioned so that the memory of EibPC is not overloaded . See here under timebufferconf (p. 254) for more details.
- The formats EXTDOUBLE , EXTTRIPLE , EXTLONG are Count with integrated zoom , shift function and time delay setting.

Access in the user program

- The XY values in the user program using the function p timebufferadd p.254 and timebufferconf p.254 addressed. An art manages up to 4 XY charts. The number of charts is determined by the number of arguments.
- Each XY chart has a legend. In Preparation of 4 XY graphs in the diagram 4 legends are displayed.
- Up to 65535 floating-point values are presented. For scaling note here notes in the description of user functions timebufferadd p. 254 and timebufferconf p. 254
- mtimecharts are always global.

Detailed example: p. 115

timechartcolor

Element *timechartcolor*

- *timechartcolor* ID *#HtmlFarbCode*
  Changes the color value of the graph with the ID (1,2,3,4) of the timecharts. The formatting is identical to the usual HTML color coding function, see (https://wiki.selfhtml.org/wiki/Grafik/Farbpaletten)
- This setting is valid globally for all graphs and is placed behind a page command.

Example

```
[WebServer]
page (wsMeter) [$Smartmeter$, $Measuring$
timechartcolor 1 #337755
timechartcolor 2 #e5a000
timechartcolor 3 #0066ff
timechartcolor 4 #ffff00
```

Picture

Element *picture*

- *picture* (ID) [*Height,Type*]($Label$,$www-LINK$)

Arguments

- *ID*: Value between 0 and 59 as an index for programming and the access to this element.
- *Height*:Value 0 or 1 (or constant SINGLE and DOUBLE)
- *Typ*: Value 0,1,2 (or LEFTGRAF, CENTERGRAF, ZOOMGRAF): left aligned, centered or streched embedding of the image
- www-Link: Valid WWW address (incl..Path and leading http://) to the external image

Access by the user program

- Label and link can be changed during runtime with the function picture (S. 262).

Mpchart

Element *mpchart*

- *mpchart*(ID) [*Height,Type*]($Label1$,Style1, $Label2$,Style2, $Label3$,Style3, $Label4$,Style4)

Arguments

- *ID*: Value between 0 and 59 as an index for programming and the access to this element.
- *Height*: Value 0 or 1 (or constant SINGLE and DOUBLE)
- *Type*: Value 8 (or constant XY) for plots
- *Type*: Value 9 (or constant SXY) for plots with sorted X-Y sets (well suited for time-based plots)
- *$Label1$ .. $Label2$* Legend of the graph
- *Style1, Style2, Style3, Style4*: value 0,1,2 or 3 (constant LINE, DOTS, LINEDOTS, COLUMN)

Access by the user program

- XY values are accessed with the function mpchart (page 252) in the user program as explained in in Table 1. A *mchart* manages up to 4 XY diagrams. The number of diagrams is specified through the number of arguments.
- Each XY diagram has a legend. When you display 4 XY diagrams, also 4 legend are displayed.
- 47 floating point values are display in a diagram. The scale is generated automatically. Please consider the additional information given by the function mpchart() on page 252.

Pchart

Element p*chart*

- pchart(ID)[$Y0$,$Y1$,$Y2$]

Arguments

- ID: A value between 0 and 255 as an index for programming and the access to this element.
- $Y0$, $Y1$,$Y2$: Labeling of the y-axis.

Access by the user program

- The y-values are accessed in the user program by the function Chart (page 247), as explained in Table 1.
- Values from the field 1...30 can be represented. With every call of this function, the values are displayed starting from the left. When the end is reached after 47 calls, the values are shifted to the left.

Slider

Element *slider*

- *slider*(ID)[Image]$Label$

Arguments

- ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access u08 variables of the section [EibPC].
- Image: A value between 0 and 99. To arrange the application more clearly, constants have been predefined (page 291).
- Label: A static labeling text (first line).

Access by the user program

- The image and the text are accessed by the function display (page 248) as explained in Table 1.
- Activation of the slider has to be evaluated by the function getslider (page 249).
- Changing the slider level has to be done by the function setslider (page 263).
- Activation of the button has to be evaluated by the function Button (page 247).
- The input field can be used to directly manipulate the slider value in the web interface.

Pslider

Element *pslider*

- *pslider*(ID)[Image]$Label$

Arguments

- ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access u08 variables of the section [EibPC].
- Image: A value between 0 and 99. To arrange the application more clearly, constants have been predefined (page 291).
- Label: A static labeling text (first line).

Access by the user program

- The image and the text are accessed by the function pdisplay (page 260) as explained in Table 1.
- Activation of the slider has to be evaluated by the function getslider (page 249).
- Changing the slider level has to be done by the function setslider (page 263).
- Activation of the button has to be evaluated by the function Button (page 247).
- The input field can be used to directly manipulate the slider value in the web interface.

Eslider

Element e*slider*

- *eslider*(ID)[Image] (Min,Increment, Max) $Description$ $Label$

Arguments

- ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access u08 variables of the section [EibPC].
- Image: A value between 0 and 99. To arrange the application more clearly, constants have been predefined (page 291).
- Min: slider minimum value
- Increment: slider increment
- Max: slider maximum value
- Description: A static labeling text (first line).
- Label: a static labeling text, max. two places

Access by the user program

- The image and the text are accessed by the function display (page 248) as explained in Table 1.
- Activation of the slider has to be evaluated by the function getslider (page).
- Changing the slider level has to be done by the function setslider (page).
- Activation of the button has to be evaluated by the function Button (page 247).
- The input field can be used to directly manipulate the slider value in the web interface.

FTP-Funktionen

FTP transfer to any data logging.

The FTP transfer writes files to a remote FTP server, the maximum file size is 64 kB.

To this end, various handles can be created, which in turn create buffered queue by up to 64 kB large file on the server. The files are via timeout earlier (and then fewer bytes if necessary) written or initiated by flushftp () by the user.

The files are named automatically by the firmware by date and time.

Strings can be written as input. The file is in ASCII format and therefore the function sendftp() P. 283 is wirtten in the queue.

In this case an LF CR (newline suitable for Windows) is inserted at the end of the data transmission of sendftp. A call to sendftp can pass more than one substring, but no more than 1400 bytes assume total. It can not handle more than four are defined. On P. 110 you can find a detailed example. This is not to be confused with the periodic outsourcing of the KNX^{TM} telegramms (vgl. P. XX).

Ftpconfig

Definition

- Function ftpconfig(server,user,password,path,timeout)

Arguments

- Argument *server* of data type c1400
- Argument *user* of data type c1400
- Argument *password* of data type c1400
- Argument *path* of data type c1400
- Argument *timeout* of data type u32 in seconds

Effect

- Configuration of an FTP server
- Updating the dependencies for value change or during the possible invocation of the startup function.
- The FTP transfer writes files to a remote FTP server, the maximum file size is 64 kB. To this end, various handles can be created, which in turn create buffered queue by up to 64 kB large file on the server. The files are via timeout earlier (and then fewer bytes if necessary) written or initiated by flushftp () by the user. The files are automatically named by the firmware by date and time.
- More than four handles cannot be defined.

Data type result (return)

- In case of failure = 0
- On sucess a handle number 1 to 4 will return

Sendftp

Definition

- Function sendftp(handle,data1,[data2],[...])

Arguments

- Argument *handle* of data type u08
- Argument *data[x]* of any data type, a maximum of 1400 bytes.

Effect

- Any data written to the queue of the handle.
- The assignment is done asynchronously.

Data type result (return)

- if it is successful = 0
- In the case of failure= 1

Ftpstate

Definition

- Function ftpstate(handle)

Arguments

- Argument *handle of* data type u08

Effect

- Returns information about the status of the FTP configuration.

Data type result (return)

- u08
- Configures / error-free = 0
- Last transmission error-free = 1
- Server not available = 2
- Password/User not allowed = 3
- Error Directory does not exist and cannot be created = 4
- Queue overflow, when previously error = 5
- Don't handle defined = 6

Ftptimeout

Definition

- Function ftptimeout(handle)

Arguments

- Argument *handle* of data type u08

Effect

- Returns the elapsed time in seconds back since the last transfer

Data type result (return)

- u32

Ftpbuffer

Definition

- Function ftpbuffer(handle)

Arguments

- Argument *handle* of data type u08

Effect

- Gives the fill level of the queue of transfers back.

Datentyp Ergebnis (Rückgabe)

- u16

Flushftp

Definition

- Function flushftp(handle)

Arguments

- Argument *handle* of data type u08

Effect

- Write data manually on the FTP server

Data type result (return)

- Success = 0
- Server not available = 1
- Error while uploading the file = 2
- Password/User not allowed = 3
- Error Directory does not exist and cannot be created = 4
- Transmission is just performed (asynchronous update) = 5

Webinput

Element *webinput*

- *webinput*(ID)[Graphic] $labelling$

Arguments

- *ID*: Value between 0 until 59 as index for programming and access to this element. You can also access to u08 variable definition in the section [EibPC].
- *Graphic*: Value between 0 and 99. In order to design the implementation clearly are predifined terms defined (page 291).
- *Labelling*: A statical labelling text
- *Style* is optional. Possible characteristics are
  - PASSWORD: In this case, the input is hidden with asterisks or characters specified by the web browser.
  - DATEPICK: Enter a date using a standard dialog (depending on the web browser). The output with webinput (p. 258) is a string in the representation $ YYYY-MM-DD $
  - TIMEPICK: Enter a time using a standard dialog (depending on the web browser). The output with webinput (p. 258) is given as a string in the representation $ HH-MM-SS $
  - COLORPICK: The input of an RGB color using a standard dialog (depending on the web browser). The output with webinput (p. 258)) is a 24-bit string.

Access to the user program

- The element is addressed via function web input (p. 258).
- Elements of web input are always global.

weboutput

Element *weboutput*

- *weboutput*(ID)[Dimension,style]

Arguments

- *ID*: Value between 0 until 59 as index for programming and access to this element. You can also access to u08 variable definition in the section [EibPC].
- *Dimension*: Value 0, 1 or 2...5(respectively constant SINGLE, DOUBLE and QUAD,respectively Width times Height: any number for height and width as factor of the unit size of the elements of the web server.)
- *Style*: Value 0,1,2 (respectively constant ICON and NOICON, NOCOLOR)

Access to the user program

- The element is addressed via function weboutput (p. 258).
- Elements of weboutput are always global.

Example look at p. 112

Peslider

Element *peslider*

- *peslider*(ID)[Image] (Min,Increment, Max) $Description$ $Label$

Arguments

- ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access u08 variables of the section [EibPC].
- Image: A value between 0 and 99. To arrange the application more clearly, constants have been predefined (page 291).
- Min: slider minimum value
- Increment: slider increment
- Max: slider maximum value
- Description: A static labeling text (first line).
- Label: a static labeling text, max. two places

Access to the user program

- The image and the text are accessed by the function pdisplay (page 260) as explained in Table 1.
- Activation of the slider has to be evaluated by the function getslider (page).
- Changing the slider level has to be done by the function setslider (page).
- Activation of the button has to be evaluated by the function Button (page 247).
- The input field can be used to directly manipulate the slider value in the web interface.

Page

Element *page*

- *page*(ID)[$Group$,$Name$]

Arguments

- ID: Value between 1 and 100 as an site index for programming and the access to local site elements (first letter 'p'). You can also access u08 variables of the section [EibPC]. Quick selection (Next- and Previous page button according Figure 10) is given by order of page definitions. You have to define all elements of a page between the respective page definition and the definition of the next page.
- Group: Assignment of the page to a group. When a page is assigned to a group, the order of definitions of the pages determine the order of pages in the selection box. In this manner you can create groups like "Cellar", "Ground floor", et. cetera.
- Name: A static labeling text (first line).

Access to the user program

- none

User

Element *user*

- *user* $Name$ [Password]

Arguments

- Name: Username. This user has access to the correspondent page.
- Password: The defined user needs this password in order to have access to the correspondent page.

Access to the user program

- none

ADVICE:

The user query and the password are not "safe", but merely serves to trap user input errors on the web server easily. The achieved IT security is considered to be low and no way comparable to the HTTPS access.

Line

Element *line*

- line [$Text$]

Arguments

- None. The element inserts a divider between two lines.
- The text is fixed at the divider and is optional.

Access to the user program

- none

Header

Element *header*

- header(number) $www.link$

Arguments

- If number assumes the value 0, header is hidden. You can also access u08 variables of the section [EibPC].
- The link (incl. path and leading http://) is optional. The URL can access an extern resource. In this case the number must be set to 2.
- The header is configurable, but then equal for each site.

Access to the user program

- none

Footer

Element *footer*

- *footer*(number) $WWW-Link$

Arguments

- If number assumes the value 0, footer is hidden. You can also access u08 variables of the section [EibPC].
- The link (incl. path and leading http://) is optional. The URL can access an extern resource. In this case the number must be set to 2.
- The footer is configurable, but then equal for each site.

Access to the user program

- none

None

Element *None*

- none

Arguments

- None. An empty element of single width is inserted into the web server.

Access to the user program

- none

Plink

Element plink (Link to other page of webserver)

- *plink*(ID)[Image] [PageID] $Text$

Arguments

- ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access u08 variables of the section [EibPC]. (This element is optically identic to the element button)
- Image: A value between 0 and 99. To arrange the application more clearly, constants have been predefined (page 291).
- PageID: Value between 1 and 100 as index of the page, to which the user jumps, when the link is activated. You can also access u08 variables of the section [EibPC].
- Label: A static labeling text (first line).

Access to the user program

- The image and the text are accessed by the function pdisplay (page 260) as explained in Table 1.
- With the function plink (page 262) link, icon and text can be changed dynamically at run time.

Link

Element link (Link to external web site)

- *link*(ID)[Image][$Website$] $Text$

Arguments

- ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access u08 variables of the section [EibPC]. (This element is optically identical to the element button)
- $Website$ http address (incl. path and leading http://) of the destination site
- Image: A value between 0 and 99. To arrange the application more clearly, constants have been predefined (page 291).
- Label: A dynamically labeling text (first line).

Access to the user program

- With the function link (page 250) the web site, icon and text can be changed dynamically at run time.

Frame

Element frame (Embedded HTML site)

- *frame* [$Text$]

Arguments

- Text: A WWW link (incl. path and leading http://) to a external HTML site, which is integrated in the webserver

Access to the user program

- none

Dframe

Element dframe (Embedded HTML site)

- *dframe* [$Text$]

Arguments

- Text: A WWW link (incl. path and leading http://) to an external HTML site, which is integprated in the webserver. The embedded window is twice as high as this from the *frame* element.

Access to the user program

- none

Section [WebServer]

In the section [WebServer] in the user program, the web server is configured. The web server is arranged like a checkerboard pattern (cf. Figure 9).

A line of this pattern corresponds to a line in this section. The elements of a line have to be arranged separated by one or more space or tab characters according to the above shown syntax.

The elements *header* and *line* have to occur singly in a line in [WebServer].

If you specify less than four elements in a line, the web server fills the remainder of the line automatically with *none* elements. The maximum number of columns is restricted to ten, at which it has to be considered that the elements *shifter* and *chart* exhibit twice the normal length.

The minimum design consists of five lines and four columns so that a visualization can be created on touchpanels with a resolution of 800x600 pixels. However, elements not required do not have to be configured.

A possible configuration with the help of icons (page 291) then reads:

The configuration of the web server is kept simple

```
WebServer]
button(0)[INFO]$InfoText$      None       button(1)[Clock]$Today is$
line
chart(0)[$10 C$,$21.5 C$,$33 C$] shifter(2)[PLUS,TEMPERATURE,MINUS]$SetpointValue$
shifter(3)[PLUS,MINUS]$2er Knopf$  shifter(4)[mail]$1er Knopf$
shifter(5)[PLUS,MINUS,UP,DOWN]$4er Knopf$
[EibPC]
```

This user program can be transmitted directly, i.e. the section [EibPC] does not need to contain additional programming, which is particularly important for the design of the visualization. After the transmission of the program to the Enertex® EibPC, the web server is being started if the option NP is activated for this device.

It takes about 15 seconds for the web server to be available after the data transmission at system start.

Whereas the user program with its access to the KNX™ interface takes about eight seconds to start, the web server requires about 15 seconds.

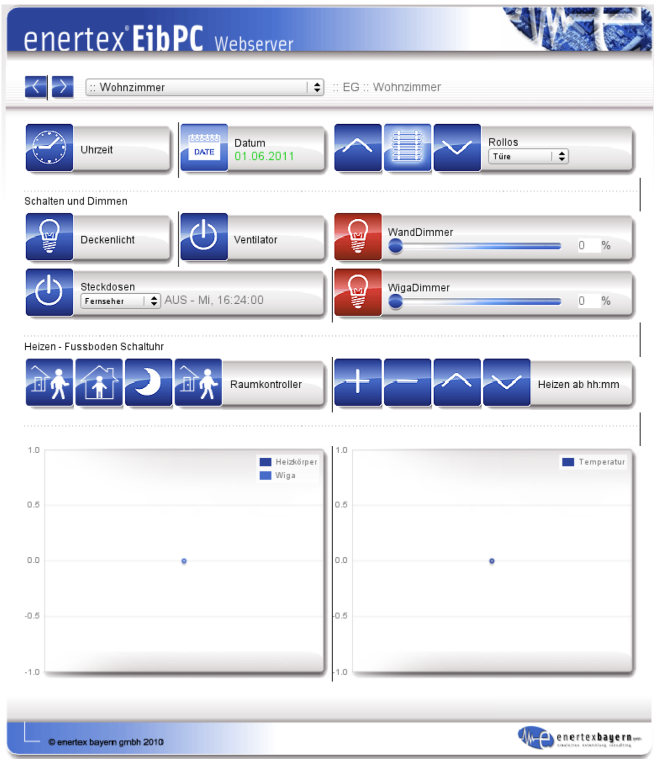


Figure 17: The web server – multipage-version, blue-Design

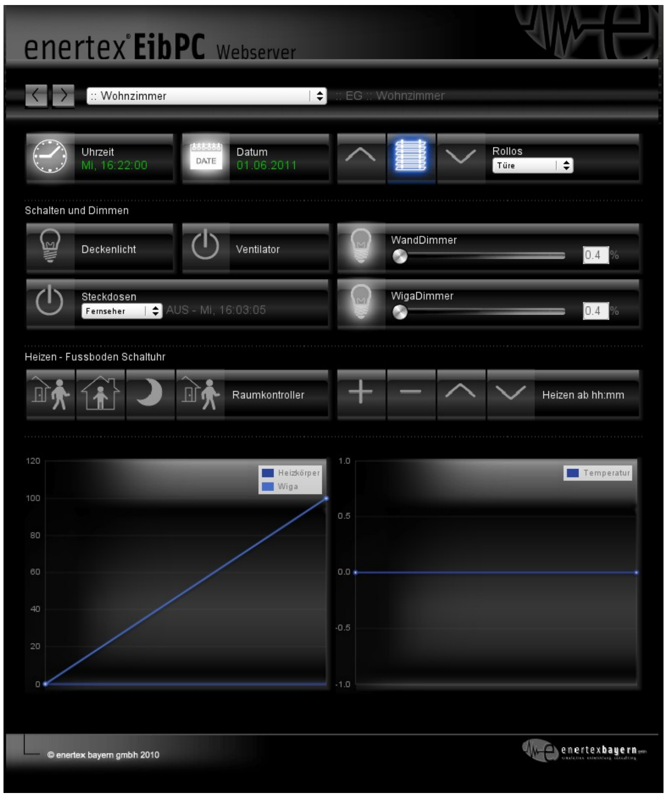


Figure 18: The web server – multipage-version, black-Design

Figure 19: Web server

Note:

Initialization

Each icon of the web server has different occurrences (color etc.). In the configuration file, for the initialization always state 1 (INACTIVE) is set. Additional states (see page 291) can be set by the function Display (Webdisplay) (page 248).

Each icon can have different decorations, symbol variations etc. In the configuration the compiler sets automatically the state1 (INACTIVE). More states (see page 291) can be set with the function Display (Webdisplay) (page 248).

Web icons

The Enertex® EibPC has a built-in set of graphics at his disposal. These can be addressed directly by their index (group of symbols) and their sub-index (design).

The following symbol groups exist, which can be addressed in the section [WebServer] as well as in the user program as a corresponding argument of Display (Webdisplay) directly via the name or the number.

Symbol	Index
INFO	0u08
SWITCH	1u08
UP	2u08
DOWN	3u08
PLUS	4u08
MINUS	5u08
LIGHT	6u08
TEMERATURE	7u08
BLIND	8u08
STOP	9u08
MAIL	10u08
SCENES	11u08
MONITOR	12u08
WEATHER	13u08
ICE	14u08
NIGHT	15u08
CLOCK	16u08
WIND	17u08
WINDOW	18u08
DATE	19u08
PRESENT	20u08
ABSENT	21u08
REWIND	22u08
PLAY	23u08
PAUSE	24u08
FORWARD	25u08
RECORD	26u08
STOP	27u08
EJECT	28u08

Table 2: Overview of symbol groups

Each symbol of a group can be displayed in different occurrences. For this, up to ten states exist which are again addressed both in the section [WebServer] and in the user program as a corresponding argument of Display (Webdisplay) directly via the name or the number.

Note: Not every symbol group implements all possible states. (see also below).

Symbol	Index
DARKRED	0u08
INACTIVE	1u08
ACTIVE	2u08
DISPLAY	3u08
STATE4	4u08
STATE5	5u08
STATE6	6u08
STATE7	7u08
STATE7	8u08
BRIGHTRED	9u08

Table 3: Overview of states.

GREY	0u08
GREEN	1u08
BLINKRED	2u08
BLINKBLUE	3u08

Table 4: Overview of styles

Note on ***BLINKRED*** and ***BLINKBLUE:***

In most browsers, the flashing function is disabled. To activate it again you need the appropriate plugin. For Firefox this would be the *Blink Enable 1.1* plugin.

The following are the symbol groups. The file name is specified in the form *Symbol_group_state.png*.

Behavior of the web server at user interaction

The integrated web server is designed in such a way that it immediately responds to pressing the buttons in the web browser and sends corresponding information to the processing loop. Moreover, when pressing a button, the icon of a symbol group always changes to the state ACTIVE immediately, which is characterized by a lighting effect. This aims at facilitating the detection of the activity.

The application program can now react to this keypress, e.g. by changing the display state with webdisplay or webchart and modifying the HTML file of the web server. Approximately 0.6 seconds after actuation, the web server sends an update command to the browser, which implies the call of the new HTML file. You can set these time intervals. For further information for setting these time intervals see Performance settings on page 127.

Again, here all symbols in an overview:

Symbol	Index	DARKRED 0u08	INACTIVE 1u08	ACTIVE 2u08	DISPLAY 3u08	STATE4 4u08	STATE5 5u08	STATE6 6u08	STATE7 7u08	STATE8 8u08	BRIGHTRED 9u08
INFO	0u08	ℹ	ℹ	ℹ	ℹ						ℹ
SWITCH	1u08	⏻	⏻	⏻							⏻
UP	2u08	∧	∧	∧							∧
DOWN	3u08	∨	∨	∨							∨
PLUS	4u08	+	+	+							+
MINUS	5u08	−	−	−							−
LIGHT	6u08	💡	💡	💡	💡	💡	💡	💡	💡		💡
TEMPERATURE	7u08	🌡	🌡	🌡	🌡						🌡

BLIND	8u08	▤	▤	▤	▤	▤	▭	▭	▭		▭
STOP	9u08	STOP	STOP	STOP							STOP
MAIL	10u08	@	@	@	@						@
SCENES	11u08	SCENE	SCENE	SCENE	SCENE						SCENE
MONITOR	12u08	▣	▣	▣	▣						▣
WEATHER	13u08	☀	☀	☀	☀	⛅	☁	🌧	❄		☀
ICE	14u08	❄	❄	❄	❄						❄
NIGHT	15u08	☾	☾	☾	☾						☾

CLOCK	16u08	🕒	🕒	🕒	🕒						🕒
WIND	17u08	🧭	🧭	🧭	🧭						🧭
WINDOW	18u08	🚪	🚪	🚪	🚪	🚪	🚪	🚪	🚪		🚪
DATE	19u08	DATE	DATE	DATE	DATE						DATE
PRESENT	20u08	🏠	🏠	🏠	🏠						🏠
ABSENT	21u08	🚶	🚶	🚶	🚶						🚶
REWIND	22u08	⏪	⏪	⏪							⏪
PLAY	23u08	▶	▶	▶							▶
PAUSE	24u08	⏸	⏸	⏸							⏸

FORWARD	25u08	⏩	⏩	⏩							⏩
RECORD	26u08	●	●	●							●
STOP	27u08	■	■	■							■
EJECT	28u08	⏏	⏏	⏏							⏏
NEXT	29u08	⏭	⏭	⏭							⏭
PREVIOUS	30u08	⏮	⏮	⏮							⏮
LEFT	31u08	‹	‹	‹							‹
RIGHT	32u08	›	›	›							›
CROSSCIRCLE	33u08	⊗	⊗	⊗							⊗

OKCIRCLE	34u08	🟥	🟦	🟦							🟥
STATESWITCH	35u08	🟥	🟦	🟦							🟥
PLUG	36u08	🟥	🟦	🟦	🟦						🟥
METER	37u08	🟥	🟦	🟦	🟦						🟥
PVSOLAR	38u08	🟥	🟦	🟦	🟦						🟥
THERMSOLAR	39u08	🟥	🟦	🟦	🟦						🟥
PUMP	40u08	🟥	🟦	🟦	🟦						🟥
HEATINGUNIT	41u08	🟥	🟦	🟦	🟦						🟥
HEATINGPUMP	42u08	🟥	🟦	🟦	🟦						🟥

FLOORHEATING	43u08	🟥	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟥
WALLHEATING	44u08	🟥	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟥
COOLER	45u08	🟥	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟥
MICRO	46u08	🟥	🟦	🟦	🟦	🟦	🟦	🟦			🟥
SPEAKER	47u08	🟥	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟥
RGB	48u08	RGB	RGB	RGB	RGB	RGB	R	G	B		RGB
LUX	49u08	k Lux	k Lux	k Lux	k Lux						k Lux
RAIN	50u08	%	%	%	%						%
KEY	51u08	🟥	🟦	🟦	🟦						🟥

WASTE	52u08	🗑	🗑	🗑	🗑	🗑	🗑	🗑	🗑	🗑	🗑
ASK	53u08	?	?	?	?						?
WARN	54u08	⚠	⚠	⚠	⚠	△	△	△	△		⚠
NEAR	55u08	((•))	((•))	((•))	((•))	)) ((	((•))	((•))	((•))		((•))
CAMERA	56u08	📹	📹	📹	📹						📹
SIGNAL	57u08	🚨	🚨	🚨	🚨	FIRE	OIL	WATER	GAS		
DOOR	58u08	🚪	🚪	🚪	🚪	🚪	🚪	🚪			🚪
GARAGE	59u08	▤	▤	▤	▤	🚗	🚗	🚗	🚗		🚗
CURTAIN	60u08	▥	▥	▥	▥	▥	▥	▥	▥		▥

ANGLE	61u08	🟥	🟦	🟦	🟦	🟦	🟦	🟦	🟦		🟥
ROLLER	62u08	🟥	🟦	🟦	🟦	🟦	🟦	🟦	🟦		🟥
EMAIL	63u08	🟥	🟦	🟦	🟦 MAIL IN						
PETS	64u08	🟥	🟦	🟦	🟦	🟦					🟥
PHONE	65u08	🟥	🟦	🟦	🟦						🟥
PERSON	66u08	🟥	🟦	🟦	🟦						🟥
TV	67u08	🟥	🟦	🟦	🟦						🟥
BEAMER	68u08	🟥	🟦	🟦	🟦						🟥
RADIO	69u08	🟥	🟦	🟦	🟦						🟥

RECIEVER	70u08										
MEDIA	71u08										
STOVE	72u08										
FRIDGE	73u08										
WASHER	74u08										
DISHWASHER	75u08										
HOLIDAY	76u08										
SLEEP	77u08										

UPDATE	78u08	🔄	🔄	🔄	🔄						🔄

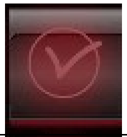
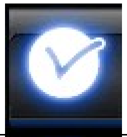
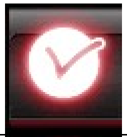
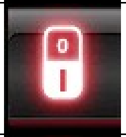
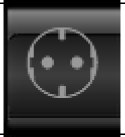
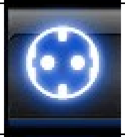
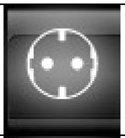
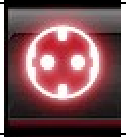
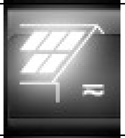
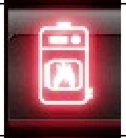
Table 5: Overview icons – blue design

Symbol	Index	DARKRED 0u08	INACTIVE 1u08	ACTIVE 2u08	DISPLAY 3u08	STATE4 4u08	STATE5 5u08	STATE6 6u08	STATE7 7u08	STATE8 8u08	BRIGHTRED 9u08
INFO	0u08										
SWITCH	1u08										
UP	2u08										
DOWN	3u08										
PLUS	4u08										
MINUS	5u08										
LIGHT	6u08										
TEMPERATURE	7u08										

BLIND	8u08										
STOP	9u08										
MAIL	10u08										
SCENES	11u08										
MONITOR	12u08										
WEATHER	13u08										
ICE	14u08										
NIGHT	15u08										

CLOCK	16u08										
WIND	17u08										
WINDOW	18u08										
DATE	19u08										
PRESENT	20u08										
ABSENT	21u08										
REWIND	22u08										
PLAY	23u08										
PAUSE	24u08										

FORWARD	25u08										
RECORD	26u08										
STOP	27u08										
EJECT	28u08										
NEXT	29u08	⏭	⏭	⏭							⏭
PREVIOUS	30u08	⏮	⏮	⏮							⏮
LEFT	31u08	<	<	<							<
RIGHT	32u08	>	>	>							>
CROSSCIRCLE	33u08	⊗	⊗	⊗							⊗

OKCIRCLE	34u08										
STATESWITCH	35u08										
PLUG	36u08										
METER	37u08										
PVSOLAR	38u08										
THERMSOLAR	39u08										
PUMP	40u08										
HEATINGUNIT	41u08										
HEATPUMP	42u08										

FLOORHEATING	43u08										
WALLHEATING	44u08										
COOLER	45u08										
MICRO	46u08										
SPEAKER	47u08										
RGB	48u08	RGB	RGB	RGB	RGB	RGB	R	G	B		RGB
LUX	49u08	k Lux	k Lux	k Lux	k Lux						k Lux
RAIN	50u08	%	%	%	%						%
KEY	51u08										

WASTE	52u08	■	■	■	■	■	■	■	■	■	■
ASK	53u08	■	■	■	■						■
WARN	54u08	■	■	■	■	■	■	■	■		■
NEAR	55u08	■	■	■	■	■	■	■	■		■
CAMERA	56u08	■	■	■	■						■
SIGNAL	57u08	■	■	■	■	FIRE	OIL	WATER	GAS		■
DOOR	58u08	■	■	■	■	■	■	■			■
GARAGE	59u08	■	■	■	■	■	■	■	■		■
CURTAIN	60u08	■	■	■	■	■	■	■	■		■

ANGLE	61u08	■	■	■	■	■	■	■	■		■
ROLLER	62u08	■	■	■	■	■	■	■	■		■
EMAIL	63u08	■	■	■	■	MAIL IN	MAIL OUT	■			■
PETS	64u08	■	■	■	■	■					■
PHONE	65u08	■	■	■	■						■
PERSON	66u08	■	■	■	■						■
TV	67u08	■	■	■	■						■
BEAMER	68u08	■	■	■	■						■
RADIO	69u08	■	■	■	■						■

RECIEVER	70u08										
MEDIA	71u08										
STOVE	72u08										
FRIDGE	73u08										
WASHER	74u08										
DISHWASHER	75u08										
HOLIDAY	76u08										
SLEEP	77u08										
UPDATE	78u08										

Table 6: Overview icons – black design

Macros functional blocks

Basics

With the aid of macros, also designated (ready-to-use) functional blocks, programming the Enertex® EibPC is

- substantially simplified for the beginner and
- schematized for the experienced user. The user can separate out complete code fragments of program parts he repeatedly uses into a library of his own and hence re-use the programming in different projects at any time.
- You can use the macro-wizard, which guides you if you parametrize a macro. This means dialogs with explanation on every arguments are given by EibStudio. If you change any argument later on, again the wizards can be opened and help you re-parametrizing the macro.
- You can use a macro guided by the macro-assistant or as a “normal function” in your application program. In this case the assistant is not available.

Available libraries

On page 24 you will find the step-by-step explanation of the application of already existing libraries.

Programming a macro

Basics

A macro is (a part of) a user program which is separated out into a library. As an independent part of another user program, these macros can be integrated into other projects. Within the macro, you can define various inputs (arguments) containing project-specific data.

Definition

Most conveniently, the programming of macros can be explained by means of an example. You have programmed the double occupancy of a KNX™ button: Pressing the button sends an ON telegram to the address 0/0/1. If the button is pressed twice within 800ms, the Enertex® EibPC shall send an ON telegram to the address 3/4/6, if it is pressed only once, it shall send an ON telegram to the address 3/4/5: The following user program arises:

```
[EIBPC]
DoubleClick=0
if event('0/0/1'b01) and ('0/0/1'b01==EIN) then DoubleClick=DoubleClick+1 endif
if after(DoubleClick==1, 800u64) then write('3/4/5'b01, EIN) endif
if after(DoubleClick==1, 800u64) and DoubleClick==2 then write('3/4/6'b01, EIN) endif
if after(DoubleClick==1, 1000u64) then DoubleClick=0 endif
```

To transfer this functionality to additional buttons and group addresses, you can change the text by way of copy & paste in the text editor of the Enertex® EibStudio.

However, this method possibly may become error-prone.

With a macro your are capable of creating templates in such situations which make programming easy. To this end, you create a new text file (ending „.lib“) and write now:

A macro starts with :begin

```
:begin DoubleClick(Name,ButtonGA,ButtonValueClick1GA,Click1Value,Click2GA,Click2Value)
Name^DoubleClick=0
if event(ButtonGA) and (ButtonGA==ButtonValue) then Name^DoubleClick=Name^DoubleClick+1 endif
if after(Name^DoubleClick==1, 800u64) then write(Klick1GA,Klick1Wert) endif
if after(Name^DoubleClick==1, 800u64) and Name^DoubleClick==2 then write(Klick2GA,Klick2Wert) endif
if after(Name^DoubleClick==1, 1000u64) then Name^DoubleClick=0 endif
:end
```

... ends with :end

A macro starts with the keyword :begin and ends with :end. The definition itself is the name of the macro, followed by comma-separated arguments which are confined by parentheses, and is positioned directly after :begin.

The arguments of the macro are used as pure text replacements in the macro code. The syntax is exactly the same as that of the “normal” user program. The code generated from the macros as it were from text templates is bound internally by the compiler to the section [EibPC]. You can look at your macro code generated by the compiler also in the file „tmpMacroOut.txt“ in the working directory of the Enertex® EibStudio.set

If the above macro is saved e.g. as myMakros.lib, the “double-click” on a KNX™ button is simplified:

```
[Macros]
DoubleClick(Basement,'0/0/1'b01,ON,'3/4/5'b01,ON,'3/4/6'b01,ON)
[MacroLibs]
myMakros.lib
[EibPC]
```

Now the compiler writes in our example „tmpMacroOut.txt“ (in the working directory of the Enertex® EibStudio):

The expansion is located in the file „tmpMacroOut.txt“ in the working directory

```
BasementDoubleClick=0
if event('0/0/1'b01) and ('0/0/1'b01==EIN) then BasementDoubleClick=BasementDoubleClick+1 endif
if after(BasementDoubleClick==1, 800u64) then write('3/4/5'b01,EIN) endif
if after(BasementDoubleClick==1, 800u64) and BasementDoubleClick==2 then write('3/4/6'b01,EIN) endif
if after(BasementDoubleClick==1, 1000u64) then BasementDoubleClick=0 endif
```

Special characters

The “^” character is a special character at replacing text. By means of this character, the text replacement can be extended in such a way that variables comprising two words are generated. At this, the „^“ character is deleted. The same effect is achieved by the „_“ character, whereas this character is not deleted. By this procedure, variables can be generated in macros (indirectly), which are as it were “encapsulated” due to the naming.

That way you now can “encapsulate” variables similarly to object-oriented programming languages. In the example, the variable „DoubleClick“ is used repeatedly. If not every macro had its “own” double-click variable, the program would generate a faulty behavior.

Runtime errors and syntax errors

Runtime errors or syntax errors due to the erroneous use of e.g. group address assignments first occur at the “expansion” of the macro.

Macro wizard

You can document your macros directly in the source code for the application. For this, the keyword :info exists. At the first position after the keyword the description of the function is located, followed by a description of each argument. The descriptions are enclosed by two “$” character.

You can generate the description by yourself with “:info”.

Each description of the arguments is enclosed by two $ characters.

```
:info $With this function block, you can realize a double-click on a button:\\
      If you press the button twice within 0.8 seconds, another function is triggered than if you press once.\\
      You can control both actions by this function block macro$\\
   $Name of the button (for the purpose of unambiguousness)$\\
   $Group address to which the button sends values$\\
   $The value sent by the button (e.g. ON or OFF)$\\
   $Group address for a telegram at single-click$\\
   $Value for the telegram at single-click (e.g. ON or OFF or 23%)$\\
   $Group address for a telegram at double-click$\\
   $Value for the telegram at double-click (e.g. ON or OFF or 23%)$
```

In order to use a the wizard or re-parametrize your macros, these have to be coded in the [Macros] section.

Local Variables and Return Values.

Macros can define local variables, which are used in a local context of the macro only. If a macro is expanded serveral times, each of the local variables are used separately in each expansion of the macro. A local variable is defined with the :var VARNAME@. Note, the @-character at the end of the name is mandatory, whereas VARNAME can be a valid variable name (combination of letters and numbers and “_” characters).

Each macro has an return value. Either it is defined with the macro command line :return *Expression* or if not defined it will be the last line before the :end command.

If we want to define a function $\cosh(x) = \frac{e^x - e^{-x}}{2}$ we can define the following macro

You can define as many local varoables as you like, but the memory usage will be increased

```
:begin cosh(x)
:info Calculates the cosh-function
:var sum@
:var p_ex@
:var m_ex@
p_ex@=exp(x)
m_ex@=-exp(-x)
sum@=p_ex@+m_ex@
:return sum@ / 2.f32
:end
```

Of course, in this case the local variables *sum@, p_ex@* and *m_ex@* are not really necessary and we could code instead:

Compact design

```
:begin cosh(x)
:info Calculates the cosh-function
:return (exp(x)-exp(-x))/2f32
:end
```

Additionally the return command could be left (due to compatibility reasons to older macros), so the code

Compact design

```
:begin cosh(x)
:info Calculates the cosh-function
(exp(x)-exp(-x))/2f32
:end
```

is still equivalent to the code above. If the last line before :end is empty or only spaces, no return value is defined. So it is a good coding style always to use :return. :return can be placed anywhere in the code of the macro.

empty line before :end means no return value (if :return is not defined)

```
:begin cosh(x)
:info Calculates the cosh-function
(exp(x)-exp(-x))/2f32

:end
```

Once defined in a macro-lib and added to the [MacroLibs] section, the macro can be used as a built-in function:

Use it as built-in

```
[EibPC]
MyVar=cosh(2.3f32)
MyVar2=cosh(cosh('1/3/2'f32)) +cosh('1/3/3'f32) + 32f32
```

Online debugging at runtime

If variables are to be monitored at runtime, it is recommended to debug with UDP telegrams and a netcat client (see https://de.wikipedia.org/wiki/Netcat).
The following code is used as a debug macro, assuming that the remote 192.168.1.18 listens on port 9000, e.g. Configured with the Unix tool netcat -ul 9000:

Sending a string with CR to a UDP client

Empty macro

```
#define DEBUG
#ifdef DEBUG
// Debugger an 192.168.1.118 an Port 9000u16
:begin vmDebugUDP(cString)
:return  {
        sendudp(9000u16, 192.168.1.18, cString+tostring(0x0d,0x0a));
}
:end
#endif
#ifndef  DEBUG
:begin vmDebugUDP(cString)
:return __EMPTY()
:end
#endif
```

Depending on whether debugging is enabled with #define DEBUG, a message is sent via UDP. In the event that the #define DEBUG is uncommented, no messages will be sent. A special feature is the use of __EMPTY(). This statement ensures that the macro does not expand and does not generate any code.

Efficient for inactive #define of DEBUG

```
x=3
If x>5 then {
     x=x*2;
    vmDebugUDP($x ist nun $+convert(x,$$));
} endif
```

Now with active #define DEBUG via UDP the value is automatically transferred to the receiver at runtime of the program. If // #define DEBUG is uncommented, the line vmDebugUDP ($ x is now $ + convert (x, $$)) does not create any overhead.

If, on the other hand, an If statement is **just** set up for debug purposes, for example:

Inefficient for inactive #define of DEBUG - if query that is used only for debugging.

```
x=3
If x>5 then {
    vmDebugUDP($x ist nun $+convert(x,$$));
} endif
```

the compiler does not create any objects for vmDebugUDP, but a "referenced" ifx> 5 object is created. This type of automatic debugging should therefore be avoided or completely disabled with #define in the code:

... then rather this way..

```
    x=3
    #ifdef DEBUG
    If x>5 then {
        vmDebugUDP($x ist nun $+convert(x,$$));
    } endif
#endif
```

Technical specifications

Power supply: The Enertex® EibPC requires an external DC power supply in the range of 5 to 30 V DC. The power input is about 1.2 W. At activity of the LAN it increases to 1.7 W.

Access: In order to access the KNX™ bus, the Enertex® EibPC requires a KNX™ interface. For this, either an external EIB-RS232 interface for the FT1.2 protocol or an IP interface is required.

Operating system: Debian Linux: 2.6.29 -rt1.

The corresponding software sources are provided by us for a handling fee at any time.

Keywords - reference

Logical instructions	
if ... then ... endif	If – then
if ... then ... else ... endif	If – then – else
!Var	Bitwise inverting
Var1 or Var2	Bitwise or
Var1 and Var2	Bitwise and
Var1 xor Var2	Bitwise exclusive-or
Var1 > Var2	Comparison of sizes
Var1 < Var2	Comparison of sizes
Var1 == Var2	Comparison of sizes
Var1 >= Var2	Comparison of sizes
Var1 =< Var2	Comparison of sizes
Var1 != Var2	Comparison of sizes
Hysteresis (Var,LowerLimit,UpperLimit)	The argument Var is compared with the variables LowerLimit and UpperLimit with a hysteresis function.
Conversion	
convert(Var1, Var2)	Converts the data type of Var1 to that of Var2 (Caution: Data may be lost!).
Arithmetic operators	
Var1 + Var2 + VarN	Addition
Var1 – Var2 - VarN	Subtraction
Var1 * Var2 * VarN	Multiplication
Var1 / Var2 / VarN	Division
abs(Var1)	Absolute value
acos(Var1)	Arc cosine
asin(Var1)	Arc sine
atan(Var1)	Arc tangent
cos(Var1)	Cosine

exp(Var1)	Exponential function

log(Var1, Var2)	Logarithm:
	Var1 = Base
	Var2 = Argument

pow(Var1, Var2)	Power:
	Var1= Base
	Var2= Exponent

sin(Var1)	Sine
sqrt(Var1)	Square root
tan(Var1)	Tangent
Measurements	
average(Var1, Var2, ... VarN)	Return value: Average of the given variables which have all to be of the same data type.
min(Var1, Var2, ... VarN)	Return value: The minimum of the given variables which have all to be of the same data type.
max(Var1, Var2, ... VarN)	Return value: The maximum of the given variables which have all to be of the same data type.
Time-based control (timer)	
after(Control, msTime)	Control: Binary value msTime: Time in ms (<2^{64})
afterc(Control, msTime, xT)	Control: Binary value msTime: Time in ms (<2^{64}) xT: remaining time (in ms)
delay(Signal, Time)	At the transition of the variable Signal from OFF to ON, the function starts a timer and sets the return value of the function to ON. After the expiry of the time in ms, the output returns to OFF.
delay(Signal, Time, xT)	At the transition of the variable Signal from OFF to ON, the function starts a timer and sets the return value of the function to ON. After the expiry of the time in ms, the output returns to OFF. xT: Remaining time

cycle(mm,ss)	The return value is not equal to zero when the time has been reached. When the time is reached (and matches exactly), the return value assumes 1.
	mm= Minutes (0...255)
	ss = Seconds (0..59)

wtime(dd,hh,mm,ss)	Weekly time switch: ss: Seconds (0..59) mm: Minutes (0..59) hh: Hours (0..23) dd: Day (0=Sunday, 6=Saturday)

htime(hh,mm,ss)	Daily time switch: ss: Seconds (0..59) mm: Minutes (0..59) hh: Hours (0..23)
mtime(mm,ss)	Hourly time switch: ss: Seconds (0..59) mm: Minutes (0..59)
stime(ss)	Minute time switch: ss: seconds (0..59)
cwtime(hh,mm,ss,dd)	Weekly comparator time switch: ss: Seconds (0..59) mm: Minutes (0..59) hh: Hours (0..23) dd: Day (0=Sunday, 6=Saturday)
chtime(hh,mm,ss)	Daily comparator time switch: ss: Seconds (0..59) mm: Minutes (0..59) hh: Hours (0..23)
cmtime(mm,ss)	Hourly comparator time switch: ss: Seconds (0..59) mm: Minutes (0..59)
cstime(ss)	Minute comparator time switch: ss: Seconds (0..59)
Synchronization with the KNX™ bus	
gettime(address)	Sets the system time of the Enertex® EibPC anew
settime()	Writes the system time of the Enertex® EibPC to the KNX™ bus
getdate(address)	Sets the date of the Enertex® EibPC anew
setdate()	Writes the date of the Enertex® EibPC to the KNX™ bus
gettimedate(address)	Sets the system time and the date of the Enertex® EibPC anew
settimedate()	Writes the system time and the date of the Enertex® EibPC to the KNX™ bus
Date-based control	
date(yyy,mm,dd)	Date comparison: yyy: Years (0..255) since the year 2000 mm: Month (1=January, 12= December) dd: Day (1..31)

month(mm,dd)	Monthly comparison: mm: Month (1=January, 12= December) dd: Day (1..31)
day(dd)	Daily comparison: dd: Day (1..31)
Position of the sun	
azimuth()	This function cyclically (time frame: 5 minutes) calculates the azimuth of the sun in degrees, north through east.
elevation()	This function cyclically (time frame: 5 minutes) calculates the elevation angle of the sun in degrees.
sun()	Returns whether it is day or night. Requires the Enertex® EibPC's knowledge of the longitude and latitude of the concerned location.
sunriseminute()	Minute at sunrise of the current day
sunrisehour()	Hour at sunrise of the current day
sunsetminute()	Minute at sunset of the current day
sunsethour()	Hour at sunset of the current day
Reading and writing	
write(GroupAddress, Value)	A valid EIB telegram is generated on the bus.
read(GroupAddress)	An EIB telegram with a read request to the group address is generated on the bus. If the actuators etc. reply, the result is determined as the return value of the function.
Bus-Activity	
event(GroupAddress)	Return value: 1b01 (ON pulse) when a telegram with the group address is on the KNX™ bus, regardless of its content.
eventread(GroupAddress)	Return value: 1b01 (ON pulse) when a Read-telegram with the group address has been written on the KNX™ bus, regardless of its content.
eventresponse(GroupAddress)	Return value: 1b01 (ON pulse) when an answer to a Read-telegram with the group address has been written on the KNX™ bus, regardless of its content.
eventwrite(GroupAddress)	Return value: 1b01 (ON pulse) when an write-telegram with the group address has been written on the KNX™ bus, regardless of its content.
writeresponse(GroupAddress, Value)	Responds to a read request by a valid telegram generated by KNX™ which writes the value to the group address is sent to the bus.
Lighting scenes	
scene (GroupAddressSceneActuator, GroupAddressActuator1, ... GroupAddressActuatorN)	A KNX™ scene actuator with the group address defined in GroupAddressSceneActuator is implemented.

storescene (GroupAddressSceneActuator, Number)	A scene actuator shall store its scene with the corresponding number.
callscene (GroupAddressSceneActuator, Number)	A scene actuator shall recall its scene with the corresponding number.
presetscene(GroupAddressSceneActuator,SceneNumber,OptionOverwrite,ValueVariable1,StatusValueVariable1, [ValueVariable2,StatusValueVariable2,ValueVariablen,StatusValueVariablen])	Default settings for the scene actuator with the group address with the corresponding number create.
Special functions	
change(Var1)	Return value: 1b01 on change of the supervised address or variable
devicenr()	Return value: serial number of Enertex® EibPC
Elog()	Reading the oldest event
elognum()	Returns the number of entries in the error memory.
comobject(Var1, Var2, ... VarN)	The value of the variable or group address that changed most recently is returned.
event(readudp(...))	Return value: 1b01, when a LAN UDP telegram arrives.
event(GroupAddress)	Return value: 1b01 (ON impulse), when a telegram is sent on the KNX™ bus, regardless of its content.
initGA(GroupAddress)	Send a read telegramm before processing user programm
random(Max)	Returns a random number in the range of 0 to Max.
systemstart()	At system start, all given variables are updated.
Network functions	
closetcp(port, address)	The Enertex® EibPC closes a TCP connection.
connecttcp(port, address)	The Enertex® EibPC establishes a TCP connection.
ping(address)	The Enertex® EibPC implement a ping to the given address.
readtcp(port, address, arg 1 [, arg2, ... arg n]	The Enertex® EibPC receives TCP telegrams.
readudp (port, address, arg 1 [, arg2, ... arg n])	The Enertex® EibPC receives UDP telegrams.
resolve(hostname)	The function establishes the IP address of the given host name.
sendmail(recipient, subject, message)	An e-mail with the subject (character string) and the message (character string) is sent to the recipient (character string).
sendhtmlmail(recipient, subject, message)	An e-mail with the subject (character string) and the message (character string) is sent to the recipient (character string). The message can be html-formated.

sendtcp(port, address, arg 1 [, arg2, ... arg n])	The Enertex® EibPC sends TCP telegrams.
sendtcparray(port, address, arg 1 [, arg2, ... arg n[, size)	The Enertex® EibPC sends TCP telegrams, without zero termination.
sendudp(port, address, arg 1 [, arg2, ... arg n])	The Enertex® EibPC sends UDP telegrams.
sendudparray(port, address, arg 1 [, arg2, ... arg n],size)	The Enertex® EibPC sends UDP telegrams, without zero termination.
webbutton(Index)	Returns the event which the web element with Index (0...255) has triggered at actuation.
webdisplay(Index,Text,Graphic)	Writes the Text to the web element Index (0...255) and sets the Graphic. The available standard graphics are selectable (to the lower right in the Enertex® Enertex® EibStudio) encoded as predefined values.
webchart(ID, Var, X1, X2)	The function addresses the xy diagram chart. When called, the xy representation of the value Var is activated. ID, Var of data type u08 X1, X2 of data type c14
Character string functions	
String1 + String2 [+ String3 ... String n]	The character strings are concatenated. If the resulting length exceeds the maximum length of the data type, the result is truncated to this length.
find(String1, String2, Pos1)	String1: Character string a (partial) character string shall be searched for in. String2: Character string to be searched for. Pos1: Ignore the first pos1 incidences of the character string to be searched for. The function returns the position of the first character of the found character string (0..1399u16). It returns 1400u16 if the character string has not been found For 65534u16, the constant END has been defined.
size(String)	The length of character string string shall be determined. The length is given by the termination character "\0" at the end of character strings.
split(String, Pos1, Pos2)	String: Character string a character string shall be extracted from. Pos1: Position of the first character of the character string to be extracted (0...1399u16). Pos2: Position of the last character of the character string to be extracted (0...1399u16). If pos2 equals 65534u16 (predefined constant END), the character string will be separated up to its end. The variable string must be of data type c1400. Return value: The character string extracted from String.
VPNServer (Option NP)	
startvpn()	Stats the VPN Service
stopvpn()	Stops the VPN Service
openvpnuser(Name)	Opens access of an user

closevpnuser(Name)	Closes access of an user
KNX^{TM} Routing (Option NP)	
readknx(Adr,String)	Converting of a telegram an decoding its information
readrawknx(control field, phyAddress, targetAddress, IsGroubAddress, routingCounter, bitLength, userData)	Converting of a telegram an decoding its information
address(Number)	Converting a number to a group address
gaimage(Number)	Returning the internal image of a group address
getaddress(GroupAddress)	Converting a group address to its corresponding unsigned 16-Bit Value
Webserver elements configuration	
design $DESIGNSTRING$	$DESIGNSTRING$ can be $black$ or $blue$. The design command can configure each site differently
page(ID)[$Groub$,$Name$]	ID: Value between 1 and 100 as an site index for programming and the access to local site elements (first letter 'p'). Group: Assignment of the page to a group. Name: A static labeling text (first line).
header(Number) $www.link$	If number assumes the value 0, header is hidden. You can also access u08 variables of the section [EibPC]. The link (incl. path and leading http://) is optional. The URL can access an extern resource. In this case the number must be set to 2. The header is configurable, but then equal for each site.
footer(Number) $WWW-Link$	If number assumes the value 0, footer is hidden. You can also access u08 variables of the section [EibPC]. The link (incl. path and leading http://) is optional. The URL can access an extern resource. In this case the number must be set to 2. The footer is configurable, but then equal for each site.
Line $Text$	The element inserts a divider between two lines.
none	An empty element of single width is inserted into the web server.
button(ID)[Image] $Text$	ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access variables of the section [EibPC]. Image: A value between 0 and 99. Text: A static labeling text (first line).
pbutton(ID)[Image] $Text$	ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access variables of the section [EibPC]. Image: A value between 0 and 99. Text: A static labeling text (first line).

mbutton(ID)[$Text1$,$Text2$,... $Text254$][Image] $Label$	ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access variables of the section [EibPC]. Text1, Text2, .. Text254: label texts for mbutton. The second and following elements are optional. Image: A value between 0 and 99. To arrange the application more clearly, constants have been predefined (page 291). Label: A static labeling text (first line).
mpbutton(ID)[$Text1$,$Text2$,... $Text254$][Image] $Label$	ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access variables of the section [EibPC]. Text1, Text2, .. Text254: label texts for mbutton. The second and following elements are optional. Image: A value between 0 and 99. To arrange the application more clearly, constants have been predefined (page 291). Label: A static labeling text (first line).
shifter(ID)[Image1,Image2, Image3, Image4]$Text$	ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access variables of the section [EibPC]. Image1 to Image4: A value between 0 and 99. Image2 to Image4 are optional. If only three images are defined, the element has only three buttons etc.. Text: A static labeling text (first line).
pshifter(ID)[Image1,Image2, Image3, Image4]$Text$	ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access variables of the section [EibPC]. Image1 to Image4: A value between 0 and 99. Image2 to Image4 are optional. If only three images are defined, the element has only three buttons etc.. Text: A static labeling text (first line).
mshifter(ID)[$Text1$,$Text2$,...,$Text254$][Image1,Image2, Image3, Image4]$Label$	ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access variables of the section [EibPC]. Image1 to Image4: A value between 0 and 99. Image2 to Image4 are optional. If only three images are defined, the element has only three buttons etc. Text1, Text2, .. Text254: labels for the mshifter. The second and following elements are optional. Label: A static labeling text (first line).
mpshifter(ID)[$Text1$,$Text2$,...,$Text254$][Image1,Image2, Image3, Image4]$Label$	ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access variables of the section [EibPC]. Image1 to Image4: A value between 0 and 99. Image2 to Image4 are optional. If only three images are defined, the element has only three buttons etc. Text1, Text2, .. Text254: labels for the mshifter. The second and following elements are optional. Label: A static labeling text (first line).
chart(ID)[$Y0$,$Y1$,$Y2$]	ID: A value between 0 and 255 as an index for programming and the access to this element. $Y0$, $Y1$,$Y2$: Labeling of the y-axis.
pchart(ID)[$Y0$,$Y1$,$Y2$]	ID: A value between 0 and 255 as an index for programming and the access to this element. $Y0$, $Y1$,$Y2$: Labeling of the y-axis.

mchart(ID)[Height,Typ]($Label1$,Style1, $Label2$,Style2, $Label3$,Style3, $Label4$,Style4)	ID: Value between 0 and 59 as an index for programming and the access to this element. Height: Value 0 or 1 (or constant SINGLE and DOUBLE) Type: Value 8 (or constant XY) for plots Type: Value 9 (or constant SXY) for plots with sorted X-Y sets (well suited for time-based plots) $Label1$ .. $Label2$ Legend of the graph Style1, Style2, Style3, Style4: value 0,1,2 or 3 (constant LINE, DOTS, LINEDOTS, COLUMN)
mpchart(ID)[Height,Typ]($Label1$,Style1, $Label2$,Style2, $Label3$,Style3, $Label4$,Style4)	ID: Value between 0 and 59 as an index for programming and the access to this element. Height: Value 0 or 1 (or constant SINGLE and DOUBLE) Type: Value 8 (or constant XY) for plots Type: Value 9 (or constant SXY) for plots with sorted X-Y sets (well suited for time-based plots) $Label1$ .. $Label2$ Legend of the graph Style1, Style2, Style3, Style4: value 0,1,2 or 3 (constant LINE, DOTS, LINEDOTS, COLUMN)
slider(ID)[Image]$Label$	ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access u08 variables of the section [EibPC]. Image: A value between 0 and 99. To arrange the application more clearly, constants have been predefined (page 291). Label: A static labeling text (first line).
pslider(ID)[Image]$Label$	ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access u08 variables of the section [EibPC]. Image: A value between 0 and 99. To arrange the application more clearly, constants have been predefined (page 291). Label: A static labeling text (first line).
eslider(ID)[Image] (Min,Increment, Max) $Description$ $Label$	ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access u08 variables of the section [EibPC]. Image: A value between 0 and 99. To arrange the application more clearly, constants have been predefined (page 291). Min: slider minimum value Increment: slider increment Max: slider maximum value Description: A static labeling text (first line). Label: a static labeling text, max. two places
peslider(ID)[Image] (Min,Increment, Max) $Description$ $Label$	ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access u08 variables of the section [EibPC]. Image: A value between 0 and 99. To arrange the application more clearly, constants have been predefined (page 291). Min: slider minimum value Increment: slider increment Max: slider maximum value Description: A static labeling text (first line). Label: a static labeling text, max. two places
picture(ID) [Height, Typ]($Label$,$www-LINK$)	ID: Value between 0 and 59 as an index for programming and the access to this element. Height:Value 0 or 1 (or constant SINGLE and DOUBLE) Typ: Value 0,1,2 (or LEFTGRAF, CENTERGRAF, ZOOMGRAF): left aligned, centered or streched embedding of the image www-Link: Valid WWW address (incl..Path and leading http://) to the external image
link(ID)[Image][$Website$] $Text$	Link to external web site
plink(ID)[Image] [PageID] $Text$	Link to other page of webserver
frame[$Text$]	Embedded HTML site

dframe[$Text$]	Embedded HTML site. The embedded window is twice as high as this from the *frame* element.
mtimechart(ID) [Format,Type,Length,YLMAX,YLMIN,YRMAX,YRMIN ($Description1$,ChartPos1,Buffer1, [$Description2$,ChartPos2,Buffer2, $Description3$,ChartPos3,Buffer3,$Description4$,ChartPos4,Buffer4])	ID: Value between 0 to 59 as an index for programming and accessing this element. Format: DOUBLE|TRIPLE|QUAD|LONG|EXTDOUBLE|EXTTRIPLE|EXTLONG Type: 0 for autoscale of the left axis, in this case YLMAX etc. is ignored 1 for autoscale of the right axis, in this case YRMAX etc. is ignored 2 for autoscale of the two axis 3 for no autoscale Length: Maximum number of pairs of values that can be displayed per graph. YLMAX: Maximum value, left y-axis, floating point numbers YLMIN: Minimum value, left y-axis, floating point numbers YRMAX: Maximum value right y-axis, floating point numbers YRMIN: Minimum value right y-axis, floating point numbers $Description1$ .. $Description4$ Legend of the corresponding graph ChartPos: Value 0 or 1 (0 for label on the left y-axis, 1 for label on the right y-axis) Buffer: ID of the timebuffer associated with the respective graph
webinput(ID)[Graphic] $Description$	ID: Value between0 to 59 as an index for programming and accessing this element. You can also access V08 variable definitions in the [EibPC] section. Graphic: Value between 0 and 99. Predefined constants are defined to make the application clearer (page 291). Description: A static caption text.
weboutput(ID)[Height]	ID: Value between 0 to 59 as an index for programming and accessing this element. You can also access V08 variable definitions in the [EibPC] section. Height: Value 0, 1 or 2 (or constant SINGLE, DOUBLE and HALF)
Webserver elements functionality	
button(ID)	By operating the button of a web button element (e.g. button or shifter) with the id, the function assumes a value not equal to zero for the duration of one processing pass, and zero in all other cases. Identical to function webbutton of former releases.
pbutton(ID, PageID)	By operating the button of a web button element that refers to a page (e.g. pbutton or pshifter) with the id on the web page of pageID, the function assumes a value not equal to zero for the duration of one processing pass, and zero in all other cases.
mbutton(ID, Selection)	By operating the button of a multi button element and the given selection with index selection (e.g. mbutton or mshifter) with the id, the function assumes a value not equal to zero for the duration of one processing pass, and zero in all other cases.
mpbutton(ID, Selection, PageID)	By operating the button of a multipagebutton element and the given selection with index selection (e.g. mbutton or mshifter) with the id on the web page of PageID, the function assumes a value not equal to zero for the duration of one processing pass, and zero in all other cases.
chart(ID, Var, X1, X2)	This function addresses the XY diagram chart. If there are multiple occurrences of id, all elements of this id are addressed. X1, X2: The labeling of the x-axis. Compatible with function webchart.
pchart(ID, Var, X1, X2, PageID)	This function addresses the XY diagram chart. If there are multiple occurrences of id on the web page of PageID, all elements of this id are addressed. X1, X2: The labeling of the x-axis.

mchart(ID, VarX, VarY, Index)	This function addresses the element mchart of the given id. If there are multiple occurrences of id, all elements of this id are addressed. One mchart displays four different graphs. index (0,1,2,3) defines the graph to be addressed. X1, X2: The labeling of the x-axis.
mpchart(ID, VarX, VarY, Index,PageID)	This function addresses the element mpchart that refers to a page of the given id. If there are multiple occurrences of id, all elements of this id are addressed. One mpchart displays four different graphs. index (0,1,2,3) defines the graph to be addressed. X1, X2: The labeling of the x-axis.
display(ID, Text, Icon, State, TextStil,[Mbutton])	The function addresses the web button (button or shifter). If there are multiple web buttons with id, they all will be addressed. Compatible with function webdisplay.
pdisplay(ID, Text, Icon, State, TextStil, PageID, [Mbutton])	The function addresses the web button that refers to a page (pbutton or pshifter). If there are multiple web buttons with id on the web page of page_id, they all will be addressed.
getslider(ID)	The function addresses the slider and returns its position (0 to 255). If there are multiple occurrences of id, all elements of this id are addressed.
getpslider(ID,PageID)	The function addresses the pslider that refers to a page and returns its position (0 to 255). If there are multiple occurrences of id, all elements of this id on the web page with page_id are addressed.
geteslider(ID)	The function addresses the eslider and returns its position (0 to 255). If there are multiple occurrences of id, all elements of this id are addressed.
getpeslider(ID, PageID)	The function addresses the peslider that refers to a page and returns its position (0 to 255). If there are multiple occurrences of id, all elements of this id on the web page with page_id are addressed.
setslider(ID,Value,Icon, State)	The function addresses the slider and sets its value to value. If there are multiple occurrences of id, all elements of this id are addressed.
setpslider(ID,Value,Icon, State, PageID)	The function addresses the pslider that refers to a page at the id on page page_id and sets it to the value value. If there are multiple occurrences of id, all elements of this id on the web page with page_id are addressed.
seteslider(ID,Value,Icon, State)	The function addresses the eslider and sets its value to value. If there are multiple occurrences of id, all elements of this id are addressed.
setpeslider(ID,Value,Icon, State,PageID)	The function addresses the peslider that refers to a page at the id on page page_id and sets it to the value value. If there are multiple occurrences of id, all elements of this id on the web page with page_id are addressed.

Keywords - reference - alphanumeric order

!Var	Bitwise inverting
abs(Var1)	Absolute value
acos(Var1)	Arc cosine
address(Number)	Converting a number to a group address
after(Control, msTime)	Control: Binary value msTime: Time in ms (<2^{64})
afterc(Control, msTime, xT)	Control: Binary value msTime: Time in ms (<2^{64}) xT: remaining time (in ms)
asin(Var1)	Arc sine
atan(Var1)	Arc tangent
average(Var1, Var2, ... VarN)	Return value: Average of the given variables which have all to be of the same data type.
azimuth()	This function cyclically (time frame: 5 minutes) calculates the azimuth of the sun in degrees, north through east.
button(ID)	By operating the button of a web button element (e.g. button or shifter) with the id, the function assumes a value not equal to zero for the duration of one processing pass, and zero in all other cases. Identical to function webbutton of former releases.
button(ID)[Image] $Text$	ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access variables of the section [EibPC]. Image: A value between 0 and 99. Text: A static labeling text (first line).
callscene (GroupAddressSceneActuator, Number)	A scene actuator shall recall its scene with the corresponding number.
change(Var1)	Return value: 1b01 on change of the supervised address or variable
chart(ID, Var, X1, X2)	This function addresses the XY diagram chart. If there are multiple occurrences of id, all elements of this id are addressed. X1, X2: The labeling of the x-axis. Compatible with function webchart.
chart(ID)[$Y0$,$Y1$,$Y2$]	ID: A value between 0 and 255 as an index for programming and the access to this element. $Y0$, $Y1$,$Y2$: Labeling of the y-axis.
chtime(hh,mm,ss)	Daily comparator time switch: ss: Seconds (0..59) mm: Minutes (0..59) hh: Hours (0..23)
closetcp(port, address)	The Enertex® EibPC closes a TCP connection.
closevpnuser(Name)	Closes access of an user

Keywords - reference - alphanumeric order

cmtime(mm,ss)	Hourly comparator time switch: ss: Seconds (0..59) mm: Minutes (0..59)
comobject(Var1, Var2, ... VarN)	The value of the variable or group address that changed most recently is returned.
connecttcp(port, address)	The Enertex® EibPC establishes a TCP connection.
convert(Var1, Var2)	Converts the data type of Var1 to that of Var2 (Caution: Data may be lost!).
cos(Var1)	Cosine
cstime(ss)	Minute comparator time switch: ss: Seconds (0..59)
cwtime(hh,mm,ss,dd)	Weekly comparator time switch: ss: Seconds (0..59) mm: Minutes (0..59) hh: Hours (0..23) dd: Day (0=Sunday, 6=Saturday)

cycle(mm,ss)	The return value is not equal to zero when the time has been reached. When the time is reached (and matches exactly), the return value assumes 1.
	mm= Minutes (0...255)
	ss = Seconds (0..59)

date(yyy,mm,dd)	Date comparison: yyy: Years (0..255) since the year 2000 mm: Month (1=January, 12= December) dd: Day (1..31)
day(dd)	Daily comparison: dd: Day (1..31)
delay(Signal, Time, xT)	At the transition of the variable Signal from OFF to ON, the function starts a timer and sets the return value of the function to ON. After the expiry of the time in ms, the output returns to OFF. xT: Remaining time
delay(Signal, Time)	At the transition of the variable Signal from OFF to ON, the function starts a timer and sets the return value of the function to ON. After the expiry of the time in ms, the output returns to OFF.
design $DESIGNSTRING$	$DESIGNSTRING$ can be $black$ or $blue$. The design command can configure each site differently
devicenr()	Return value: serial number of Enertex® EibPC
dframe[$Text$]	Embedded HTML site. The embedded window is twice as high as this from the *frame* element.
display(ID, Text, Icon, State, TextStil,[Mbutton])	The function addresses the web button (button or shifter). If there are multiple web buttons with id, they all will be addressed. Compatible with function webdisplay.
elevation()	This function cyclically (time frame: 5 minutes) calculates the elevation angle of the sun in degrees.

Keywords - reference - alphanumeric order

Elog()	Reading the oldest event
elognum()	Returns the number of entries in the error memory.
eslider(ID)[Image] (Min,Increment, Max) $Description$ $Label$	ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access u08 variables of the section [EibPC]. Image: A value between 0 and 99. To arrange the application more clearly, constants have been predefined (page 291). Min: slider minimum value Increment: slider increment Max: slider maximum value Description: A static labeling text (first line). Label: a static labeling text, max. two places
event(GroupAddress)	Return value: 1b01 (ON impulse), when a telegram is sent on the KNX™ bus, regardless of its content.
event(GroupAddress)	Return value: 1b01 (ON pulse) when a telegram with the group address is on the KNX™ bus, regardless of its content.
event(readudp(...))	Return value: 1b01, when a LAN UDP telegram arrives.
eventread(GroupAddress)	Return value: 1b01 (ON pulse) when a Read-telegram with the group address has been written on the KNX™ bus, regardless of its content.
eventresponse(GroupAddress)	Return value: 1b01 (ON pulse) when an answer to a Read-telegram with the group address has been written on the KNX™ bus, regardless of its content.
eventwrite(GroupAddress)	Return value: 1b01 (ON pulse) when an write-telegram with the group address has been written on the KNX™ bus, regardless of its content.
exp(Var1)	Exponential function
find(String1, String2, Pos1)	String1: Character string a (partial) character string shall be searched for in. String2: Character string to be searched for. Pos1: Ignore the first pos1 incidences of the character string to be searched for. The function returns the position of the first character of the found character string (0..1399u16). It returns 65534u16 if the character string has not been found For 65534u16, the constant END has been defined.
footer(Number) $WWW-Link$	If number assumes the value 0, footer is hidden. You can also access u08 variables of the section [EibPC]. The link (incl. path and leading http://) is optional. The URL can access an extern resource. In this case the number must be set to 2. The footer is configurable, but then equal for each site.
frame[$Text$]	Embedded HTML site
gaimage(Number)	Returning the internal image of a group address
getaddress(GroupAddress)	Converting a group address to its corresponding unsigned 16-Bit Value
getdate(address)	Sets the date of the Enertex® EibPC anew
geteslider(ID)	The function addresses the eslider and returns its position (0 to 255). If there are multiple occurrences of id, all elements of this id are addressed.

Keywords - reference - alphanumeric order

getpeslider(ID, PageID)	The function addresses the peslider that refers to a page and returns its position (0 to 255). If there are multiple occurrences of id, all elements of this id on the web page with page_id are addressed.
getpslider(ID,PageID)	The function addresses the pslider that refers to a page and returns its position (0 to 255). If there are multiple occurrences of id, all elements of this id on the web page with page_id are addressed.
getslider(ID)	The function addresses the slider and returns its position (0 to 255). If there are multiple occurrences of id, all elements of this id are addressed.
gettime(address)	Sets the system time of the Enertex® EibPC anew
gettimedate(address)	Sets the system time and the date of the Enertex® EibPC anew
header(Number) $www.link$	If number assumes the value 0, header is hidden. You can also access u08 variables of the section [EibPC]. The link (incl. path and leading http://) is optional. The URL can access an extern resource. In this case the number must be set to 2. The header is configurable, but then equal for each site.
htime(hh,mm,ss)	Daily time switch: ss: Seconds (0..59) mm: Minutes (0..59) hh: Hours (0..23)
Hysteresis (Var,LowerLimit,UpperLimit)	The argument Var is compared with the variables LowerLimit and UpperLimit with a hysteresis function.
if ... then ... else ... endif	If – then – else
if ... then ... endif	If – then
initGA(GroupAddress)	Send a read telegramm before processing user programm
Line $Text$	The element inserts a divider between two lines.
link(ID)[Image][$Website$] $Text$	Link to external web site

log(Var1, Var2)	Logarithm:
	Var1 = Base
	Var2 = Argument

max(Var1, Var2, ... VarN)	Return value: The maximum of the given variables which have all to be of the same data type.
mbutton(ID, Selection)	By operating the button of a multi button element and the given selection with index selection (e.g. mbutton or mshifter) with the id, the function assumes a value not equal to zero for the duration of one processing pass, and zero in all other cases.
mbutton(ID)[$Text1$,$Text2$,... $Text254$][Image] $Label$	ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access variables of the section [EibPC]. Text1, Text2, .. Text254: label texts for mbutton. The second and following elements are optional. Image: A value between 0 and 99. To arrange the application more clearly, constants have been predefined (page 291). Label: A static labeling text (first line).

Keywords - reference - alphanumeric order

mchart(ID, VarX, VarY, Index)	This function addresses the element mchart of the given id. If there are multiple occurrences of id, all elements of this id are addressed. One mchart displays four different graphs. index (0,1,2,3) defines the graph to be addressed. X1, X2: The labeling of the x-axis.
mchart(ID)[Height,Typ]($Label1$,Style1, $Label2$,Style2, $Label3$,Style3, $Label4$,Style4)	ID: Value between 0 and 59 as an index for programming and the access to this element. Height: Value 0 or 1 (or constant SINGLE and DOUBLE) Type: Value 8 (or constant XY) for plots Type: Value 9 (or constant SXY) for plots with sorted X-Y sets (well suited for time-based plots) $Label1$ .. $Label2$ Legend of the graph Style1, Style2, Style3, Style4: value 0,1,2 or 3 (constant LINE, DOTS, LINEDOTS, COLUMN)
min(Var1, Var2, ... VarN)	Return value: The minimum of the given variables which have all to be of the same data type.
month(mm,dd)	Monthly comparison: mm: Month (1=January, 12= December) dd: Day (1..31)
mpbutton(ID, Selection, PageID)	By operating the button of a multipagebutton element and the given selection with index selection (e.g. mbutton or mshifter) with the id on the web page of PageID, the function assumes a value not equal to zero for the duration of one processing pass, and zero in all other cases.
mpbutton(ID)[$Text1$,$Text2$,... $Text254$][Image] $Label$	ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access variables of the section [EibPC]. Text1, Text2, .. Text254: label texts for mbutton. The second and following elements are optional. Image: A value between 0 and 99. To arrange the application more clearly, constants have been predefined (page 291). Label: A static labeling text (first line).
mpchart(ID, VarX, VarY, Index,PageID)	This function addresses the element mpchart that refers to a page of the given id. If there are multiple occurrences of id, all elements of this id are addressed. One mpchart displays four different graphs. index (0,1,2,3) defines the graph to be addressed. X1, X2: The labeling of the x-axis.
mpchart(ID)[Height,Typ]($Label1$,Style1, $Label2$,Style2, $Label3$,Style3, $Label4$,Style4)	ID: Value between 0 and 59 as an index for programming and the access to this element. Height: Value 0 or 1 (or constant SINGLE and DOUBLE) Type: Value 8 (or constant XY) for plots Type: Value 9 (or constant SXY) for plots with sorted X-Y sets (well suited for time-based plots) $Label1$ .. $Label2$ Legend of the graph Style1, Style2, Style3, Style4: value 0,1,2 or 3 (constant LINE, DOTS, LINEDOTS, COLUMN)
mpshifter(ID)[$Text1$,$Text2$,...,$Text254$][Image1,Image2, Image3, Image4]$Label$	ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access variables of the section [EibPC]. Image1 to Image4: A value between 0 and 99. Image2 to Image4 are optional. If only three images are defined, the element has only three buttons etc. Text1, Text2, .. Text254: labels for the mshifter. The second and following elements are optional. Label: A static labeling text (first line).

Keywords - reference - alphanumeric order

mshifter(ID)[$Text1$,$Text2$,...,$Text254$][Image1,Image2, Image3, Image4]$Label$	ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access variables of the section [EibPC]. Image1 to Image4: A value between 0 and 99. Image2 to Image4 are optional. If only three images are defined, the element has only three buttons etc. Text1, Text2, .. Text254: labels for the mshifter. The second and following elements are optional. Label: A static labeling text (first line).
mtime(mm,ss)	Hourly time switch: ss: Seconds (0..59) mm: Minutes (0..59)
none	An empty element of single width is inserted into the web server.
openvpnuser(Name)	Opens access of an user
page(ID)[$Groub$,$Name$]	ID: Value between 1 and 100 as an site index for programming and the access to local site elements (first letter 'p'). Group: Assignment of the page to a group. Name: A static labeling text (first line).
pbutton(ID, PageID)	By operating the button of a web button element that refers to a page (e.g. pbutton or pshifter) with the id on the web page of pageID, the function assumes a value not equal to zero for the duration of one processing pass, and zero in all other cases.
pbutton(ID)[Image] $Text$	ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access variables of the section [EibPC]. Image: A value between 0 and 99. Text: A static labeling text (first line).
pchart(ID, Var, X1, X2, PageID)	This function addresses the XY diagram chart. If there are multiple occurrences of id on the web page of PageID, all elements of this id are addressed. X1, X2: The labeling of the x-axis.
pchart(ID)[$Y0$,$Y1$,$Y2$]	ID: A value between 0 and 255 as an index for programming and the access to this element. $Y0$, $Y1$,$Y2$: Labeling of the y-axis.
pdisplay(ID, Text, Icon, State, TextStil, PageID, [Mbutton])	The function addresses the web button that refers to a page (pbutton or pshifter). If there are multiple web buttons with id on the web page of page_id, they all will be addressed.
peslider(ID)[Image] (Min,Increment, Max) $Description$ $Label$	ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access u08 variables of the section [EibPC]. Image: A value between 0 and 99. To arrange the application more clearly, constants have been predefined (page 291). Min: slider minimum value Increment: slider increment Max: slider maximum value Description: A static labeling text (first line). Label: a static labeling text, max. two places
picture(ID) [Height, Typ]($Label$,$www-LINK$)	ID: Value between 0 and 59 as an index for programming and the access to this element. Height:Value 0 or 1 (or constant SINGLE and DOUBLE) Type: Value 0,1,2 (or LEFTGRAF, CENTERGRAF, ZOOMGRAF): left aligned, centered or streched embedding of the image www-Link: Valid WWW address (incl..Path and leading http://) to the external image

Keywords - reference - alphanumeric order

ping(address)	The Enertex® EibPC implement a ping to the given address.
plink(ID)[Image] [PageID] $Text$	Link to other page of web server

pow(Var1, Var2)	Power:
	Var1= Base
	Var2= Exponent

presetscene(GroupAddressSceneActuator,SceneNumber,OptionOverwrite,ValueVariable1,StatusValueVariable1, [ValueVariable2,StatusValueVariable2,ValueVariablen,StatusValueVariablen])	Default settings for the scene actuator with the group address with the corresponding number create.
pshifter(ID)[Image1,Image2, Image3, Image4]$Text$	ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access variables of the section [EibPC]. Image1 to Image4: A value between 0 and 99. Image2 to Image4 are optional. If only three images are defined, the element has only three buttons etc.. Text: A static labeling text (first line).
pslider(ID)[Image]$Label$	ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access u08 variables of the section [EibPC]. Image: A value between 0 and 99. To arrange the application more clearly, constants have been predefined (page 291). Label: A static labeling text (first line).
random(Max)	Returns a random number in the range of 0 to Max.
read(GroupAddress)	An EIB telegram with a read request to the group address is generated on the bus. If the actuators etc. reply, the result is determined as the return value of the function.
readknx(Adr,String)	Converting of a telegram an decoding its information
readrawknx(control field, phyAddress, targetAddress, IsGroubAddress, routingCounter, bitLength, userData)	Converting of a telegram an decoding its information
readtcp(port, address, arg 1 [, arg2, ... arg n]	The Enertex® EibPC receives TCP telegrams.
readudp (port, address, arg 1 [, arg2, ... arg n])	The Enertex® EibPC receives UDP telegrams.
resolve(hostname)	The function establishes the IP address of the given host name.
scene (GroupAddressSceneActuator, GroupAddressActuator1, ... GroupAddressActuatorN)	A KNX™ scene actuator with the group address defined in GroupAddressSceneActuator is implemented.
sendhtmlmail(recipient, subject, message)	An e-mail with the subject (character string) and the message (character string) is sent to the recipient (character string). The message can be html-formated.
sendmail(recipient, subject, message)	An e-mail with the subject (character string) and the message (character string) is sent to the recipient (character string).

Keywords - reference - alphanumeric order

sendtcp(port, address, arg 1 [, arg2, ... arg n])	The Enertex® EibPC sends TCP telegrams.
sendtcparray(port, address, arg 1 [, arg2, ... arg n[, size)	The Enertex® EibPC sends TCP telegrams, without zero termination.
sendudp(port, address, arg 1 [, arg2, ... arg n])	The Enertex® EibPC sends UDP telegrams.
sendudparray(port, address, arg 1 [, arg2, ... arg n],size)	The Enertex® EibPC sends UDP telegrams, without zero termination.
setdate()	Writes the date of the Enertex® EibPC to the KNX™ bus
seteslider(ID,Value,Icon, State)	The function addresses the eslider and sets its value to value. If there are multiple occurrences of id, all elements of this id are addressed.
setpeslider(ID,Value,Icon, State,PageID)	The function addresses the peslider that refers to a page at the id on page page_id and sets it to the value value. If there are multiple occurrences of id, all elements of this id on the web page with page_id are addressed.
setpslider(ID,Value,Icon, State, PageID)	The function addresses the pslider that refers to a page at the id on page page_id and sets it to the value value. If there are multiple occurrences of id, all elements of this id on the web page with page_id are addressed.
setslider(ID,Value,Icon, State)	The function addresses the slider and sets its value to value. If there are multiple occurrences of id, all elements of this id are addressed.
settime()	Writes the system time of the Enertex® EibPC to the KNX™ bus
settimedate()	Writes the system time and the date of the Enertex® EibPC to the KNX™ bus
shifter(ID)[Image1,Image2, Image3, Image4]$Text$	ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access variables of the section [EibPC]. Image1 to Image4: A value between 0 and 99. Image2 to Image4 are optional. If only three images are defined, the element has only three buttons etc.. Text: A static labeling text (first line).
sin(Var1)	Sine
size(String)	The length of character string string shall be determined. The length is given by the termination character "\0" at the end of character strings.
slider(ID)[Image]$Label$	ID: Value between 0 and 59 as an index for programming and the access to this element. You can also access u08 variables of the section [EibPC]. Image: A value between 0 and 99. To arrange the application more clearly, constants have been predefined (page 291). Label: A static labeling text (first line).
split(String, Pos1, Pos2)	String: Character string a character string shall be extracted from. Pos1: Position of the first character of the character string to be extracted (0...1399u16). Pos2: Position of the last character of the character string to be extracted (0...1399u16). If pos2 equals 65534u16 (predefined constant END), the character string will be separated up to its end. The variable string must be of data type c1400. Return value: The character string extracted from String.

Keywords - reference - alphanumeric order

sqrt(Var1)	Square root
startvpn()	Stats the VPN Service
stime(ss)	Minute time switch: ss: seconds (0..59)
stopvpn()	Stops the VPN Service
storescene (GroupAddressSceneActuator, Number)	A scene actuator shall store its scene with the corresponding number.
String1 + String2 [+ String3 ... String n]	The character strings are concatenated. If the resulting length exceeds the maximum length of the data type, the result is truncated to this length.
sun()	Returns whether it is day or night. Requires the Enertex® EibPC's knowledge of the longitude and latitude of the concerned location.
sunrisehour()	Hour at sunrise of the current day
sunriseminute()	Minute at sunrise of the current day
sunsethour()	Hour at sunset of the current day
sunsetminute()	Minute at sunset of the current day
systemstart()	At system start, all given variables are updated.
tan(Var1)	Tangent
Var1 – Var2 - VarN	Subtraction
Var1 != Var2	Comparison of sizes
Var1 * Var2 * VarN	Multiplication
Var1 / Var2 / VarN	Division
Var1 + Var2 + VarN	Addition
Var1 < Var2	Comparison of sizes
Var1 =< Var2	Comparison of sizes
Var1 == Var2	Comparison of sizes
Var1 > Var2	Comparison of sizes
Var1 >= Var2	Comparison of sizes
Var1 and Var2	Bitwise and
Var1 or Var2	Bitwise or
Var1 xor Var2	Bitwise exclusive-or
webbutton(Index)	Returns the event which the web element with Index (0...255) has triggered at actuation.

Keywords - reference - alphanumeric order

webchart(ID, Var, X1, X2)	The function addresses the xy diagram chart. When called, the xy representation of the value Var is activated. ID, Var of data type u08 X1, X2 of data type c14
webdisplay(Index,Text,Graphic)	Writes the Text to the web element Index (0...255) and sets the Graphic. The available standard graphics are selectable (to the lower right in the Enertex® Enertex® EibStudio) encoded as predefined values.
write(GroupAddress, Value)	A valid EIB telegram is generated on the bus.
writeresponse(GroupAddress, Value)	Responds to a read request by a valid telegram generated by KNX™ which writes the value to the group address is sent to the bus.
wtime(dd,hh,mm,ss)	Weekly time switch: ss: Seconds (0..59) mm: Minutes (0..59) hh: Hours (0..23) dd: Day (0=Sunday, 6=Saturday)

Predefined variables

Variable	Value
PI	3.141592654
E	2.718281828
ON	1b01
OFF	0b01
LEIGHT	1b01
DARK	0b01
UP	0b01
DOWN	1b01
MONDAY	1u08
TUESDAY	2u08
WEDNESDAY	3u08
THURSDAY	4u08
FRIDAY	5u08
SATURDAY	6u08
SUNDAY	0u08
WORKDAY	7u08
WEEKEND	8u08
END	65534u16
EOS	65535u16
GREY	1u08
GREEN	2u08
BLINKRED	3u08
BLINKBLUE	4u08
INFO	0u08
SWITCH	1u08
UP	2u08
DOWN	3u08
PLUS	4u08
MINUS	5u08
LIGHT	6u08
TEMERATURE	7u08
BLIND	8u08
STOP	9u08
MAIL	10u08
SCENES	11u08
MONITOR	12u08
WEATHER	13u08
ICE	14u08
NIGHT	15u08
CLOCK	16u08
WIND	17u08
WINDOW	18u08
DATE	19u08
DARKRED	0u08
INACTIVE	1u08
ACTIVE	2u08

DISPLAY	3u08
STATE4	4u08
STATE5	5u08
STATE6	6u08
STATE7	7u08
STATE8	8u08
BRIGHTRED	9u08

Questions and answers

ErrorCode Eventlog

Error code	explanation
ERR_PROC_OBJECT	An object (a function) could not be processed. This can have several, function-specific causes. Please pay attention to more error messages.
ERR_PROC_OBJECT_MSG_OUT	An output object could not be processed. This can have the following functions relate to: 1 write access to the KNX bus 1.1 settime 1.2 setdate 1.3 settimedate 1.4 write 1.5 read 1.6 write response 1.7 scene 1.8 store scene 1.9 callscene 1:10 eibtelegramm 2 Network Functions 2.1 closetcp 2.2 ConnectTCP 2.3 ping 2.4 resolve 2.5 send html mail 2.6 sendmail 2.7 sendtcp 2.8 sendtcparray 2.9 sendudp 2:10 sendudparray 3 RS232 interface 3.1 resetrs232 3.2 sendrs232 4 VPN Server 4.1 closevpnuser openvpnuser 4.2 4.3 4.4 startvpn stopvpn Please check if an appropriate connection exists
ERR_PROC_REPETITIONS	An endless loop has been detected. Processing was therefore canceled.
ERR_POW_OF_NEG_BASE	During the processing of a function pow an error was detected, the base is negative. The calculation is thereforenot processed.
ERR_LOG_OF_NON_POS_BASE_OR_ARG	During the processing of the log function, an error has been recognized that the base or the argument is not positive. The calculation is therefore not processed.
ERR_SQRT_OF_NON_POS_ARG	The error is sqrt When processing function detected that the argument is negative. The calculation is therefore carried out.
ERR_ASIN_OF_ARG_OUT_OF_RANGE	The error was asin When processing function detected that the argument outside the interval [-1; +1] is. The calculation is therefore carried out.
ERR_ACOS_OF_ARG_OUT_OF_RANGE	When processing the acos function the error was detected that the argument outside the interval [-1; +1] is. The calculation is therefore carried out.
ERR_DIVISION_BY_ZERO	During processing of a division of the error has been detected, the divisor is equal to 0. The calculation is therefore carried out.
ERR_EIBNET_IP_SETSOCKOPT_0	It is an error in the preparation of the compound occurred to a KNXnet / IP interface.
ERR_EIBNET_IP_SETSOCKOPT_1	s.a.
ERR_EIBNET_IP_SETSOCKOPT_2	s.a.
ERR_EIBNET_IP_SENDTO_0	An error has occurred while sending a message to a KNXnet / IP interface.
ERR_EIBNET_IP_SENDTO_1	s.a.
ERR_EIBNET_IP_SENDTO_2	s.a.
ERR_EIBNET_IP_SENDTO_3	s.a.
ERR_EIBNET_IP_SENDTO_4	s.a.
ERR_EIBNET_IP_SENDTO_5	s.a.
ERR_EIBNET_IP_TIMEOUT_SEARCH	There could be found no KNXnet / IP interface. Please check whether an operational KNXnet / IP interface is connected to the same network as the EibPC.
ERR_EIBNET_IP_DISCONNECT_REQUEST_IN	The connection between EibPC and KNXnet / IP interface has been disconnected.
ERR_EIBNET_IP_DISCONNECT_REQUEST_OUT	s.a.
ERR_EIBNET_IP_TIMEOUT_CONNECTIONSTATE_REQUEST	s.a.
ERR_EIBNET_IP_E_CONNECTION_ID	s.a.
ERR_EIBNET_IP_E_DATA_CONNECTION	The KNXnet / IP interface has detected an error connecting to the EibPC.
ERR_EIBNET_IP_E_KNX_CONNECTION	The KNXnet / IP interface has detected an error in the connection to the KNX bus.

ERR_EIBNET_IP_TUNNELLING_TIMEOUT_0	A message was sent again to KNXnet / IP interface, because an error has occurred.
ERR_EIBNET_IP_TUNNELLING_TIMEOUT_1	The connection between EibPC and KNXnet / IP interface has been disconnected.
ERR_EIBNET_IP_L_DATA_CON	It was received for a message sent to this email a confirmation of the KNXnet / IP interface.
ERR_FT12_LINE_IDLE_TIMEOUT_0	It is an error when connecting to the FT1.2 interface occurred.
ERR_FT12_LINE_IDLE_TIMEOUT_1	s.a.
ERR_FT12_SELECT	s.a.
ERR_FT12_INVALID_TELEGRAM	s.a.
ERR_FT12_READ	s.a.
ERR_FT12_RESET_REQ_IN	The connection to FT1.2 interface has been reset.
ERR_FT12_STATUS_REQ_IN	It has received a status request from the FT1.2 interface.
ERR_FT12_L_BUSMON_IND	It has received a message from the KNX bus via the FT1.2 interface.
ERR_FT12_FIX_LENGTH_END	A message from the FT1.2 interface was faulty.
ERR_FT12_FIX_LENGTH_CHECKSUM	s.a.
ERR_FT12_VAR_LENGTH_LENGTH_0	s.a.
ERR_FT12_VAR_LENGTH_LENGTH_1	s.a.
ERR_FT12_VAR_LENGTH_START	s.a.
ERR_FT12_VAR_LENGTH_CHECKSUM	s.a.
ERR_FT12_VAR_LENGTH_END	s.a.
ERR_FT12_L_DATA_CON	It was received for a message sent to this email a confirmation of the FT1.2 interface.
ERR_FT12_IN_BUFFER_FULL	It is an error when connecting to the FT1.2 interface occurred.
ERR_MEM_OBJECTS_COUNT	Obsolete in V3
ERR_MEM_OBJECT_OBJECT_TYPE	Obsolete in V3
ERR_MEM_OBJECT_CALC_TYPE	Obsolete in V3
ERR_MEM_OBJECT_BIT_LEN	Obsolete in V3
ERR_MEM_OBJECT_DATA_SIZE	Obsolete in V3
ERR_MEM_OBJECT_NAME	Obsolete in V3
ERR_MEM_OBJECT_EXPRESSION	Obsolete in V3
ERR_MEM_OBJECT_INPUT_COUNTER_0	Obsolete in V3
ERR_MEM_OBJECT_INPUTS_0	Obsolete in V3
ERR_MEM_OBJECT_DEPENDENCY_COUNTER_0	Obsolete in V3
ERR_MEM_OBJECT_DEPENDENCIES_0	Obsolete in V3
ERR_MEM_OBJECT_DEPENDENCY_COUNTER_1	Obsolete in V3
ERR_MEM_OBJECT_DEPENDENCIES_1	Obsolete in V3
ERR_MEM_OBJECT_NULL	Obsolete in V3
ERR_MEM_OBJECT_NO_ERROR	Obsolete in V3
ERR_MSGSND_ASYNC_SERIAL_0	An error in the communication with the asynchronous serial user interface has been determined because an internal queue was not available. Perhaps the EibPC with the current application program is temporarily overloaded.
ERR_MSGSND_ASYNC_SERIAL_1	s.a.
ERR_MSGSND_MSGOUT_0	Access to the KNX bus has not been possible because an internal queue was not available. Perhaps the EibPC with the current application program is temporarily overloaded.
ERR_MSGSND_MSGOUT_1	s.a.
ERR_MSGSND_MSGOUT_2	s.a.
ERR_MSGSND_MSGOUT_3	s.a.

ERR_MSGSND_MSGOUT_4	s.a.
ERR_MSGSND_MSGOUT_5	s.a.
ERR_MSGSND_RESOLVE_0	The resolve function could not be executed because an internal queue was not available. Perhaps the EibPC with the current application program is temporarily overloaded.
ERR_MSGSND_INTERFACE_IN_0	A received from the KNX bus message could not be passed to the application program, because an internal queue was not available. Perhaps the EibPC with the current application program is temporarily overloaded.
ERR_MSGSND_INTERFACE_IN_1	s.a.
ERR_MSGSND_INTERFACE_IN_2	s.a.
ERR_MSGSND_MAIL_0	An e-mail message could not be sent because an internal queue was not available. Perhaps the EibPC with the current application program is temporarily overloaded.
ERR_MSGSND_MAIL_1	s.a.
ERR_MSGSND_TCP_OUT_0	A TCP message could not be sent because an internal queue was not available. Perhaps the EibPC with the current application program is temporarily overloaded.
ERR_MSGSND_TCP_OUT_1	A TCP connection could not be established because an internal queue was not available. Perhaps the EibPC with the current application program is temporarily overloaded.
ERR_MSGSND_TCP_OUT_2	A TCP connection could not be disconnected because an internal queue was not available. Perhaps the EibPC with the current application program is temporarily overloaded.
ERR_MSGSND_TCP_IN_0	A received TCP message could not be passed to the application program, because an internal queue was not available. Perhaps the EibPC with the current application program is temporarily overloaded.
ERR_MSGSND_UDP_OUT_0	A UDP message could not be sent because an internal queue was not available. Perhaps the EibPC with the current application program is temporarily overloaded.
ERR_MSGSND_UDP_IN_0	A received UDP message could not be passed to the application program, because an internal queue was not available. Perhaps the EibPC with the current application program is temporarily overloaded.
ERR_MSGSND_PING_0	The ping function could not be executed because an internal queue was not available. Perhaps the EibPC with the current application program is temporarily overloaded.
ERR_MSGSND_TCP_OUT_3	A TCP message without zero termination could not be sent because an internal queue was not available. Perhaps the EibPC with the current application program is temporarily overloaded.
ERR_MSGSND_UDP_OUT_1	A UDP message without zero termination could not be sent because an internal queue was not available. Perhaps the EibPC with the current application program is temporarily overloaded.
ERR_MSGSND_ASYNC_SERIAL_2	An error in the communication with the asynchronous serial user interface has been determined because an internal queue was not available. Perhaps the EibPC with the current application program is temporarily overloaded.
ERR_EXIT_NCONF_0	The application program was terminated. This process was triggered by an action in EibStudio.
ERR_EXIT_NCONF_1	s.a.
ERR_EXIT_NCONF_2	s.a.
ERR_EXIT_NCONF_3	s.a.
ERR_EXIT_MAIN_0	The application program was terminated due to an internal error.
ERR_EXIT_MAIN_1	The application program was terminated due to an internal error.
ERR_EXIT_MAIN_2	The application program was terminated due to an internal error.
ERR_EXIT_MAIN_3	The application program was terminated due to an internal error.
ERR_EXIT_MAIN_4	The application program was terminated due to an internal error.
ERR_LED_MUTEX_TRYLOCK	Obsolete in V3

ERR_READ_GROUP_ADDRESS	A group address has been configured with initga, but does not respond to the read request.
ERR_ERRNO	An internal error has been detected. The type of error can be more accurately determined by the manufacturer based on the error code.
ERR_ASYNC_SERIAL_0	There was an error accessing the asynchronous serial user interface.
ERR_ASYNC_SERIAL_1	s.a.
ERR_ASYNC_SERIAL_2	s.a.
TIMEBUFFER_DATATYPE_ERROR	Obsolete in V3
TIMEBUFFER_DATATYPE_ERROR	Obsolete in V3
TIMEBUFFER_DATATYPE_ERROR	Obsolete in V3

Problems and solutions

Error message	Solution
! Default value is too big for given data type in >xy< !	The value must be given with a data type, e.g. Brightness<2000u16
! Make use of convert-functions: Datatypes of parameters are not the same: >Var1+Var2< !	Var3=convert(Var1,Var2) + Var2
Syntax error in line:[17] >if (("EntireKitchen-1/1/9"==On) and wtime(23,00,00,00)) < Valid until position: STOP--> and wtime(23,00,00,00))	The instruction must be positioned in one line or the line must be finished with ' \\'. if .... and \\ then ....
! Predefined variable cannot be re-defined in >EIN=1b01< !	In the Enertex® EibParser, variables are predefined to make the construction of a user program as simple as possible. The predefined variables are listed in the Enertex® EibStudio in the right section of the window. They cannot be defined again.
Datatypes of parameters are not the same: >sun()==1< !	The return value of the function is binary. A number without the definition of a data type is always an unsigned 8 bit value. As a relational operator, a binary value must be given. sun()==1b01
Syntax error in line:[13] >a=4,6e1f32< Valid until position: STOP--> ,6e1f32	As a decimal point, always "." has to be used.
Syntax error in line:[21] >"Akt1-0/0/5"=after(a,5000u64)<	A direct assignment is only possible for variables, not for addresses. Writing information to the KNX™ bus is realized with the help of the write function. write(„Akt1-0/0/5", 1b01)
Syntax error in line:[19] >if (a==EIN) then write("Akt1-0/0/5",EIN) write("Akt2-0/0/6",EIN);write("Akt3-0/0/8",EIN); write("Ak4-0/0/7",EIN) endif<	Multiple instructions in an if statement must be separated by ";". if(a=EIN) then write(b=EIN); write(c=AUS) endif
Syntax error in line:[26] >write(on,ON)< data type is unkown in >write(on<	The write function can only affect group addresses (1st argument), not variables.
Deklaration der Variable muss eindeutig sein in >u=convert(z,r)-r-e<	Every variable may be declared only once. An additional declaration produces this error messages.
Wrong data type in >cycle(0.5,5<	Only integer values may be entered.

Changelog

Version 30 (from Patch 3.100, EibStudio 3.103)

- Windows 7 and Windows 8.1 and Firewalls p. 20.
- Complete web server example for building automation p. 86
- New function processingtime p.211.
- New function urlencode p. 227.
- New function urldecode p. 227.
- New function getganame p. 234.
- New function md5sum p. 241.
- New directive #addto p. 128.
- New keyword mobilezoom for webserver p. 275
- Note on Blinkred and Blinkblue p. 292
- Number of web elements increased to 60 per page p. 271
- Number of global web elements increased to 60 p. 271
- Online Debugging at runtime– direct in code p. 316
- Mpchart with 4-fold width (LONG) p. 252
- timechart with option of stacking the graphs (STACK) p. 279
- Additions to connecttcp p. 238
- compact of web elements / Filling out of clearances in the web server - p. 271.
- Color wheel - RGB choice with the web server / webinput p. 285
- Time input with the web server / webinput p. 285
- Data input with the web server / webinput p. 285
- Password input with the web server/ webinput p. 285
- New dimensions for weboutput (p. 285)
- New function easterday p. 182
- New function eastermonth p. 183
- SHUTDOWN variable for controlling storage in the user program p. 213.
- New function timebuffervalue p. 257
- New function timebufferclear S. 255
- New function writeflashvar p. 199.
- New function readflashvar p. 198
- Changing communication ports UDP and TCP of the Enertex® EibPC (p. 235).
- New web server element timechartcolor (p. 280)
- New function tostring S. 226.
- Description of the mtimechartpos function improved
- Arbitrary sizes for the weboutput field (S. 285)
- Correct div. mistakes

Version 26 (Patches 3.0xx, EibStudio 3.0xx)

- Code table for event logs p. 341
- New Icon p. 302.
- Backgroundimages for webserver p. 274

Version 24 (Patches 3.0xx, EibStudio 3.0xx)

New functions to the access for the internal flash

- New chapter data saving p. 199
- Data of project saving on the Enertex® EibPC p. 199
- Pictures, data, time series saving on the Enertex® EibPC p. 200

New webserver-function:

- User administration webserver p. 266, 286
- New web element and function weboutput p. 285
- New web element and function webinput p. 258
- Completion function plink p. 253
- Completion function link p. 288.
- picture element can treat relative path (on internal field), p. 281

- Own field for uploading of data onto the Enertex® EibPC p. 200
- Detailed example to the weboutput p. 112
- Detailed example for uploading of graphics for picture and header and footer p. 114
- Detailed example mtimechart respectively visualising of time series with the EXT-mtimechart p. 115
- Detailed example mtimechart respectively visualising of time series, single mtimechart p. 118
- Change of indicated graphs with mtimechart p. 119
- 49 new groups of webicon S. 291

New chart-functions:
- New webelement mtimechart p. 279
- New function mtimechartpos p.259
- New function mtimechart p.259
- New function timebufferconfig p. 254
- New function timebufferadd p. 254
- New function timebuffersize p. 256
- New function timebufferstore p. 256
- New function timebufferread p. 256

New FTP-functions:
- New function ftptimeout p. 284
- New function ftpbuffer p. 284
- New function ftpstate p.284
- New function flushftp p. 284
- New function sendftp p. 283
- New function ftpconfig p. 283
- Detailed example for using the new FTP functions p. 110.

New and enlarged functions of time series:
- New function encode p. 226 and p. 108
- Variable length of time series are possible p. 220
- New function capacity p. 226
- Definition changed EOS on 65535
- Definition changed END on 65534
- Detailed example for encoding of c14-strings and encode p. 108
- Conatenating strings of different length p. 109

New function for processing of date- and time data:
- New function utcconvert p. 180
- New function utctime p. 180
- New function utc p. 180

Miscellaneous Innovations:
- Code of mistakes of the event memory are listed p. 341
- Explanation of the visualising assistent p. 72
- New function shift p. 173
- New function presun p. 185
- Explanation sendmail extended
- New constant HALF of charts and picture
- Correct div. mistakes
- New chapter „Programming for experts“ p.101 ff.
- Completion Enertex® KNXNet/IP router p. 16
- Revision of the example p. 52

Version 22 (Patches 2.303, EibStudio 2.305)

- New function picture p. 262
- New function plink p. 262
- New function link p. 250
- Supplement plink p. 288

- Supplement link p. 288
- Extended example for function readrawknx p. 231

Version 21 (Patches 2.30x, EibStudio 2.30x)

- Supplement new macro-libs p. 24
- Supplement to the webelement link p. 288.
- Supplement picture p. 281
- new function resetrs232 p. 229
- Supplement menubar
- Supplement keywords reference
- Supplement of syntax for use of physical KNX - addresses S. 155.
- New function readrawknx p. 231
- New function writeresponse p. 169
- New function setpeslider p. 264
- New function seteslider p. 263
- New function getpeslider p. 249
- New function geteslider p. 249
- New function peslider p. 286
- New function eslider p. 282
- Supplement function pslider p. 282
- Supplement function slider p. 282
- Supplement scene p. 217
- New function presetscene p. 217
- New function elog p. 209
- New function elognum p. 210
- New function devicenr p. 209
- New function ping p. 241
- New function afterc p. 195
- New function delayc p. 193
- Supplement firewall problems and solutions
- Supplement NTP functions 174
- Correction of typos
- Supplement to readflash and writeflash p. 197
- Supplement to connecttcp p. 238
- New function sendtcparray p. 240
- New function sendudparray p. 237

Version 19 (Patches 2.10x, EibStudio 2.10x)

- Supplement to text syles
- Supplement to VPN p. 244
- Added icons for black-design p. 312
- Correction of typos
- Supplement to connecttcp p 238
- Functionality section [InitGA] p. 166
- Asynchronous processing p. 124
- Supplement to FTP p. 142

Version 17 (Firmware 2.00x, Patches 2.00x, EibStudio 2.00x)

- New NTP functions 174
- Update menubar
- HTTPS p. 264
- Correction of typos
- Validating scheme introduced more detailed, S. 121
- New option SXY for m(p)chart-webelement p. 278
- New keyword design for Webserver p. 278
- VPN server p. 244
- gaimage-function p. 234

- getaddress-function p. 234
- Updating available macro-libs p. 24
- Macro now with return values, local variables and function like usability, p. 313
- Black and blue design of webserver p. 274
- New section [VPN] and [InitGA]. p. 125
- [InitGA]. p. 166 and p. 41.
- New eibtelegramm- function p. 215
- Changes in behavoir of gettime/getdate and gettimedate p.176
- event can be used with negated groupaddresses p. 168
- read can be used with negated groupaddresses p. 165
- New initga-function p. 167
- New #include directive p. 129
- New #break_if_older_version p. 128
- Linebreak in sendmail function p. 242
- New function sendhtmlmail p. 243
- #define directive p. 129
- #ifdef directive p. 129
- #endif directive p. 129
- #undef directive p. 129
- #ifndef directive p. 129
- New eventread-function p. 169
- New eventresponse-function p. 169
- New eventwrite-function p. 169

Version 15 (Firmware 1.30x, Patches 1.30x, EibStudio 1.30x)

- Curly brackets at if-statements p. 38, p. 126
- Address function p. 230
- Eval function p. 211
- Stringcast function p. 221
- Stringset function p. 221
- Button function p. 247
- Chart function p. 247
- Getslider function p. 249
- Getpslider function p. 249
- Getpeslider function p. 249
- Pbutton function p. 261
- Mpchart function p. 252
- Mpbutton function p. 252
- Pdisplay function p. 260
- Pchart function p. 260
- Setslider function p. 263
- Setpslider function p. 263
- String format function p. 223
- Compiler capable of hexadecimal constants p. 154
- Synchronization with the application program, p. 174
- Second function p. 178
- Minute function p. 178
- Hour function p. 177
- Changehour function p. 179
- Changeminute function p. 179
- Changesecond function p. 179
- Complete revision of Elements of the web server (p. 267).
- Single-page version of the web server, p. 75
- Multiple-page version of the web server, p. 79
- Visualization with the iPhone, p. 30
- An overview of the data types, supplement EIS types p. 156
- TCP/IP server and client, p. 71
- RS232 interface, p. 228

- Reference to changelog Updates, p. 12
- Supplements to the web server: Single- and multiple-page version: p. 265
- Pdisplay function: p. 265
- Readflash function: p. 197
- Writeflash function: p. 197
- readknx and the chapter concerning routing with the Enertex® EibPC, p. 230
- Extented information to variable definitions, p. 45
- New symbol template WebElement for formating the web configuration commands, e.g. *page*
- Adaption of the intern config file 5 (web elements)
- Storing of telegrams at a FTP 142
- Auslagern auf FTP-Verzeichnis, p. 60 and 142.
- Performance settings, p. 127
- All Eibstudio screenshots have been updated. Screenshots show now the Eibstudio V1.207 (Win XP).
- New Debugger explained at p 144.