enertexbayern gmbh
simulation entwicklung consulting

# Enertex® EibPC and EibPC²

# Manual

**Prerequisites**

| | |
|---|---|
| Requirements Enertex®EibPC: | Firmware 4.000 or higher<br>Options NP, V3 |
| **Enertex®EibStudio:** | **Version 4.000 or higher** |

This manual may be changed without notice or announcement and makes no claim to completeness or correctness.

# Contents

**Thank you for byuing an Enertex® EibPC oder EibPC²**

# Safety instructions

Please mind the following safety instructions

- Installation and assembly may only be performed by an authorized electrician.
- For connecting KNX interfaces, expert knowledge gained by KNX- trainings is assumed.
- Damages of the device, fire or other dangers could result from violating the instructions in the manual
- 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

- With purchasing the Enertex® EibPC, you are licensed to use the Enertex® EibStudio. The Enertex® EibStudio and all independently running components may only be used for 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.**

# Help

## E-Mail

Please send a support request to eibpc@enertex.de if you encounter problems with your Enertex® EibPC or EibPC².

## Support-Export

To simplify support, please attach your project in question to the support request. Click on Help → Export for support from the title menu and send the .esp file. The export is a .zip file containing your project and all uploaded webserver files, as well as machine-specific information (e.g., operating system) and the .log file. Private information (e.g., ftp, e-mail passwords) are **stripped** from the project.

## HOTLINE

You can also use our support via telefone at +49 9191 73395 0 (during business hours) free of charge.

## KNX-User-Forum

At http://knx-user-forum.de/eibpc a separate area for support of the Enertex ® EibPC is set up. You will also find direct advice from expert users and professionals.

## Videos

Please have a look at our Youtube channel https://www.youtube.com/user/eibpc

# Updates

Please find updates for the Enertex® EibPC and Enertex® EibPC² on our website www.eibpc.com.

# Enertex® EibPC²

## Overview

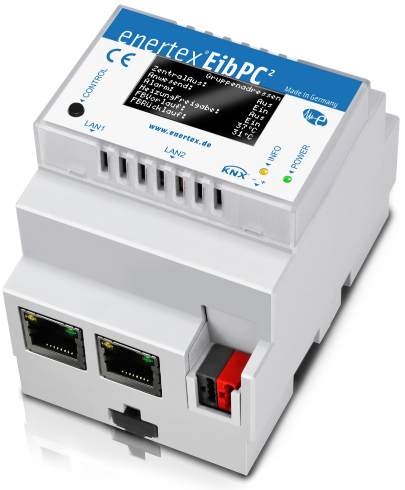


Figure 1: Enertex® EibPC²

*Summary*

The perfect control center for a smart future: EibPC². The new hardware platform with an ARM CPU for industrial applications, fast and low power DDR RAM and 8 GB flash memory guarantees performance and reliability for many years.

Simple logics or complex control flows – with the EibPC² it is easy to solve both tasks. If the built-in functions do not fit your ideas, you can freely create programs.

Keep the overview with our modern web-based visualization.

The integrated bus interface obviates the need for a dedicated power supply. The EibPC² can also be used as KNX interface (ETS) for programming your KNX devices. The integrated display shows important information.

Proven security features such as encrypted web server and VPN functionality, are of course available in the EibPC², too.

Our completely new designed, parametrization and visualization tool EibStudio V4 manages your existing EibPC or new EibPC² installation. EibStudio V4 is available free of-charge for Windows, OSX and Linux.

*KNX-Functions*

The Enertex® EibPC² offers the following functions for the KNX installation

- 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).

*LAN-Functions*

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

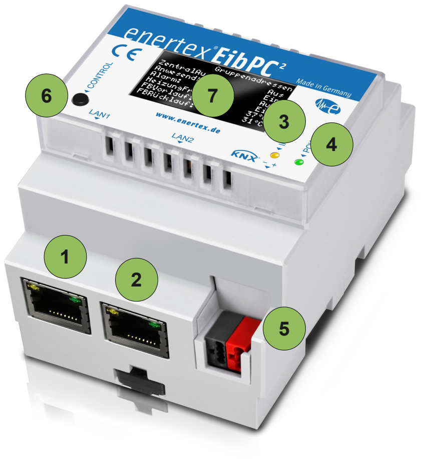


*Figure 2: Connectors and Control Elements*

### Connectors and Control Elements

See Figure 2 for the connectors and control elements:

1. LAN-Interface 1
2. LAN-Interface 2
3. Info-LED (orange)
4. Power-LED (green)
5. KNX
6. Control Button
7. Display

*Power supply*

The Enertex® EibPC² is powered directly from the KNX bus (required voltage: 27V – 30V). Check the voltage before installation if the device is not installed directly after the KNX power supply.

If the internal KNX interface is not required, a regular power supply can be used.

The KNX power supply must provide at least 3.2 W at its output (110 mA at 29 V Bus voltage).

*Power consumption ~1.5 Watt at 30 VDC*

*KNX bus*

The Enertex® EibPC² has an integrated KNX bus interface. A dedicated KNXnet/IP-Interface can be configured, and the Enertex® EibPC² can be installed separately of the KNX installation..

All certified KNXnet/IP interfaces can be used with the Enertex® EibPC².

We recommend one of the following:

- Enertex® KNX IP Secure Router
- Enertex® KNX IP Secure Interface
- Enertex® KNXnet/IP Router
- Enertex® KNXnet/IP Interface

The Enertex® EibPC² uses KNX net/IP Tunnelling. Once connected, the tunnel is not available to other devices or the ETS.

## Installation

*Integrated KNX interface*

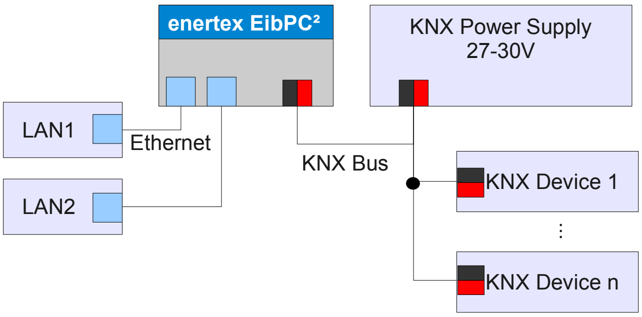


*Figure 3: Connection of the Enertex EibPC² to the KNX bus*

Figure 3 shows how the installation of the Enertex® EibPC².

Installation steps:

1. Connect to LAN using LAN 1 oder LAN 2 (1,2).
2. The other LAN interface can be used to connect other devices.
3. Connect Enertex® EibPC² with a (KNX) power supply.

*Integrated Ethernet-Switch*

**Please mind:** LAN 1 and LAN 2 are connected by an internal switch, and the Enertex® EibPC² must be started for the switch to operate.

When the Enertex® EibPC² is (re)starting, the connection between LAN1 and LAN2 is interrupted. Restarting the user program does not interrupt the connection.

*Dedicated KNXnet/IP interface*

When the internal interface is not used, connect the device as shown in Figure 4.

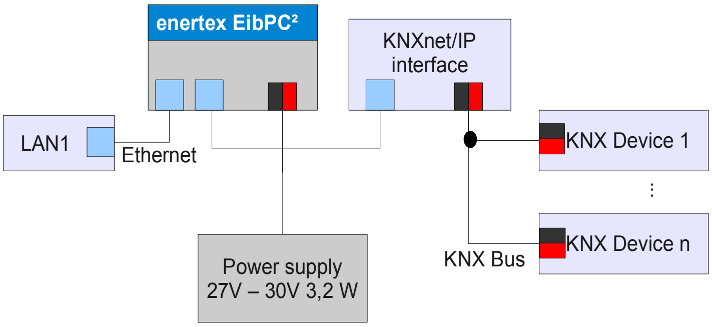


*Figure 4: Using a dedicated KNXnet/IP interface*

## Device Start

After the device has been plugged-in or restarted using EibStudio, the start procedure is as follows:

1. Info- and Power-LED are both on during system boot.
2. The Power-LED stays on during operation.
3. After ~2 min. the Info-LED blinks once every second. A factory reset can be issued (see below).
4. Initialize bus connection. The info-LED flickers.
5. After booting, the display shows system information, including the IP. The display stays on for 30 s. By pressing the Control button, the display can be reactivated.
6. Normal operation. The Info-LED blinks once every few seconds and when KNX telegrams are received.

## Firmware Update

Firmware updates are installed using Enertex® EibStudio. Download the Firmware file from our website, extract it (update file name: *eibpc2-patch-x.xxxx.ptc*). The update takes ~5 minutes. Make sure that the power supply is not interrupted during an update.

If the device does not behave correctly after starting an update (e.g., both LEDs stayy off, display not activated by Control-button), wait at least 20 minutes and force a reboot by disconnecting the device from the power supply.

Please contact our support if the device cannot be reactivated.

## Factory Reset

### *Reset on start*

During system boot, the Power-LED is on. After ~1.5 minutes, the Info-LED blinks (1s on, 1s off) for 5 seconds. Press Control to issue a factory reset.

The following settings are reset/deleted:

1. Change network-configuration to DHCP
2. Delete User program
3. Delete Sun data
4. Delete VPN settings
5. Delete HTTPS user
6. Delete scenes, variables

After reset, the Info-LED blinks and the device is restarted.

### *Reset while running*

If the device is already operating, a factory reset is issued by holding the Control-button for ~25 s. The display shows a confirmation, and the Info-LED blinks. The device is restarted.

# Enertex® EibPC

## Overview

*Figure 5: Enertex® EibPC*

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.

*KNX-Functions*

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

*All functions of the KNXBus can be addressed*

- 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.

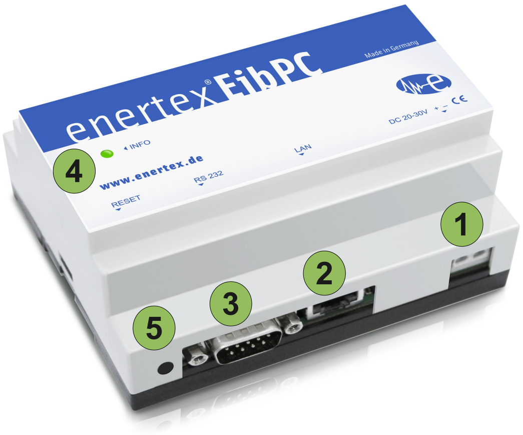
*Software*

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



### Connectors and Control Elements

*Figure 6: Enertex® EibPC Connectors and Control Elements*

See Figure 6 for the connectors and control elements:

1. Power supply 20-30V DC
2. Ethernet interface
3. RS232 interface
4. Info LED (green)
5. Reset Button

*Power supply*

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

*KNX bus*

To access the KNX bus, the Enertex® EibPC requires 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*

**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)
- Merten

**The following interfaces do  not work:**

- 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/router*

All certified IP interfaces and router can be used with the Enertex® EibPC.

We recommend one of the following:

- Enertex® KNX IP Secure Router
- Enertex® KNX IP Secure Interface
- Enertex® KNXnet/IP Router
- Enertex® KNXnet/IP Interface

The router has many additional features and may provide the Enertex® EibPC with a valid system time after restart.

## Installation

*Hardware*

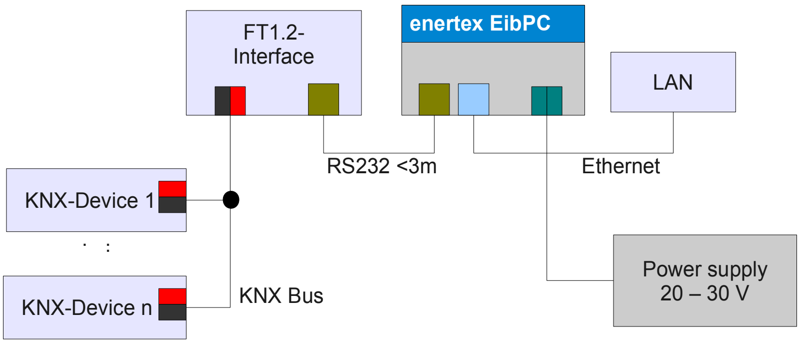


*Figure 7: Connection of the Enertex EibPC to the KNX bus via FT 1.2*

*Using an RS-232 interface*

Figure 7 shows the basic structure of a KNX network which is connected to the Enertex® EibPC using an FT1.2 interface.

To integrate the Enertex® EibPC into the KNX system, proceed as follows

1. Connect the Enertex® EibPC to a 20V - 30V DC power supply (1).
2. Connect the Ethernet port (2) of the Enertex® EibPC to the LAN.
3. Connect the RS-232 interface (3) of the Enertex® EibPC with an FT1.2 KNX interface with an extension cable (“Male“ to “Female“). The pin-out of the RS232 interface of the Enertex® EibPC corresponds to the standard EIA-232.
4. Restart the Enertex® EibPC (via EibStudio or unplug the power supply).

**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.*

When the EibPC is starting, the FT1.2 interface must already be connected to the operational KNX bus.

**So do not** first switch on the Enertex® EibPC and then establish the RS-232 connection. The FT1.2 interface may be switched on simultaneously with the Enertex® EibPC, but not after booting the Enertex® EibPC.

*Installation using an KNXnet/IP interface*

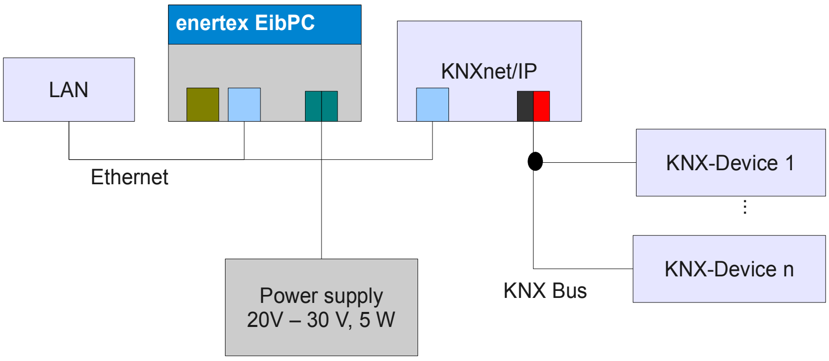


*Figure 8: Installation of the Enertex® EibPC using an KNXnet/IP interface*

Figure 8 shows the connection of the Enertex® EibPC to the KNX bus using a KNXnet/IP interface.

Proceed as explained above, except for the RS232 connection (3).

**Hint:**

If you have problems with the interface, try to reset it first, as they may be in an invalid state.

*Enable the interface in Enertex® EibStudio*

The interface is accessed and activated after the program has been transferred and started.

The Enertex® EibPC uses KNXnet/IP tunnelling. An interface used by the EibPC cannot be used by others (ETS) while the EibPC is connected. The tunnel can be closed and re-opened in Enertex® EibStudio if required.

## Device Start

After the device has been plugged-in or restarted using EibStudio, the start procedure is as follows:

1. The LED is on (power-supply ok) and the boot process is started
2. After ~4 min. the LED blinks once every second. A factory reset can be issued.
3. Initialize bus connection. LED flickers.
4. Normal operation. The LED blinks once every 5s or with KNX bus telegrams.

## Firmware Update

The firmware can be updated using Enertex® EibStudio. Download a firmware file (file name: eibpc1-patch-x.xxx.ptc). It takes ~5 min in total. Make sure that the power supply is operational all the time.

If the device is not ready again after 20 minutes and the LED does not show any activity (constantly off or on), interrupt the power supply to force a restart.

Please contact our support if the device remains unreachable.

## Factory Reset

While the LED blinks once every second after start, press the reset button for factory reset:

1. Reset network configuration to DHCP
2. Delete of the user program
3. Delete sun data

## Install Firmware Version 4

To use Enertex® EibStudio 4, firmware version v4.000 or higher is required. This update is only provided for devices with option V3.

*Enertex® EibStudio 4:*

*Firmware version v4.xxx*

**You cannot use Enertex® EibStudio 4 without option V3. It is possible to purchase this option. You receive an unlock key which you can install by yourself. Please contact our sales department.**

*Enertex® EibStudio 3:*

*Firmware version 3.107*

Before you proceed, download the most recent versions of the following from our website www.enertex.de

- Enertex® EibStudio v3.xxx
- Enertex® EibStudio v4.xxx
- Firmware v4.xxx

**Verify that Enertex® EibStudio 4 runs on your computer. If it dows not work as intended, do not install the firmware.**

To install the firmware:

- Open Enertex® EibStudio 3
- Unzip firmware v4.xxx and verify the file name is *eibpc1-patch-4.xxx.ptc* (with xxx being any number)
- Install the firmware update. It takes ~5 min. The Enertex® EibPC is restarted automatically. The INFO-LED is turned off during the update.The LED blinks as usual when the restart is completed.
- Enertex® EibStudio 3 cannot communicate with the device any more. Close it and open Enertex® EibStudio 4 instead.
- Check the connection settings and connect to the device. Verify that the firmware version is as expected (Overview, p. 28)

### Downgrade to Version 3.107

If you have any issues after installing the update, you can downgrade the device back to version 3.107. Enertex® EibStudio 4 must be running and connected to the Enertex® EibPC.

Downgrade as follows:

- Download and unzip firmware version v3.107. The file *eibpc-patch-3.107.ptc* must be present.
- OPen Enertex® EibStudio 4 and check the connection to the Enertex® EibPC
- Install the firmware update.
- Install the firmware update. It takes ~5 min. The Enertex® EibPC is restarted automatically. The INFO-LED is turned off during the update.The LED blinks as usual when the restart is completed.
- Enertex® EibStudio 4 cannot communicate with the device any more. Close it and open Enertex® EibStudio 3 instead.
- Check the connection settings and connect to the device. Verify that the firmware version is as expected.

# EibStudio 4 Quick Start Guide

The device is connected to the LAN and started. In default configuration, DHCP is used to get an IP address. This can be changed in the PROJECT SETTINGS later.

Enertex® EibStudio 4 or above is used as programming and configuration tool.

Enertex® EibStudio 4 has to be uncompressed. No installation procedure is required.

**Important: A firewall may prevent Enertex® EibStudio 4 to communicate with the Enertex® EibPC. Please verify that the connection is not blocked.**

*Open Enertex® EibStudio 4*

On first start, Enertex® EibStudio 4 shows a configuration dialog to set the Projects Directory (p. 25).

*Project directory*

*EibStudio 4 does not change or delete files outside of the projects directory and the Configuration Directory (p. 25).*

*When a project is imported, the project files are copied here.*

You can change the projects directory in the Settings (p. 25). An open project is closed and all projects in the new directory are listed.

*Project-independent settings*

Project-independent settings can be changed via EDIT → SETTINGS.

*Project*

Enertex® EibStudio 4 opens with the projects list. You can create new projects, import existing projects or delete projects. Only the files associated with the specific project are deleted from the projects directory.

A project contains all information to configure and run a device.

*Project menu*

When a project is opened, the PROJECT MENU provides access to the functions:

- OVERVIEW: Device info, program statistics, project log
- OBJECTS: All group addresses and variables
- LOGIC: Editor to create logical connections of objects
- VISU: Editor for Web visualization
- EXPERT: Code editor for programs
- SETTINGS: Project-specific configuration of the Enertex® EibPC

*Bus connection*

To start the first program, configure the connection to the Enertex® EibPC. Open the project menu and navigate to PROJECT SETTINGS → CONNECTION. If the device is in the same network segment, the automatic search will find it.

The connection to the KNX bus can also be configured according to your installation.

*Program*

Compilation of the project is started by selecting PROJECT → COMPILE from the title menu. The program is a combination of the separate configurations. This includes logic, visualization, expert programs, settings.

To run the program, select COMPILE AND RUN from the same menu.

*Objects*

To add group addresses to the project. select OBJECTS → ETS IMPORT from the project menu. You can use *.esf* and *.knxproj*-Files, to get names and data types of the group addresses. Both can be modified later in OBJECTS → GROUP ADDRESSES if necessary.

Data types are required when using the Debugger and the Group Monitor.

The list on variables is regenerated on compilation and cannot be modified.

# From Enertex® EibStudio 3 to 4

This section summarizes, where settings were moved and how programming the Enertex® EibPC changed with Enertex® EibStudio 4.

If you are not familiar with Enertex® EibStudio 3, you can safely skip this section.

*Projects and structure*

A project is no more a set of files with a master file including the others, but is managed by the functions from the projects menu. It is not recommended to change any file in the projects directory by hand.

The sections [VPN], [Performance], [FTP],... are not to be used any more. All settings are set in the project settings Project Settings (S. 36) zu finden.

*ETS import*

The *.esf*-file is not part of the project but the imported group addresses are stored internally.

*Initialization of group addresses*

To select a group address for initialization, select it in **Objects** from the project menu instead of using the section [InitGA]. Group addresses used in the visualization are selected automatically. The list is updated on compilation.

*Visualization*

Visualizations are created in **Visu** in the project menu. Number, type and position of the elements is completely free. The elements are configured independently and can be copied to different pages. Functions often required can be configured directly on the element without having to write a single line of code.

*Programs*

Use **Expert** from the project menu to write programs which used to be organized in the section [EibPC]. large programs can be split up. They are combined on compilation. Variables are global, i.e., variables defined in a one program are valid in every other program (and must be globally unique).

Enertex® EibStudio 4 creates internal variables, starting with INT_. Do not create your own variables with that prefix to avoid collisions.

*Custom Visualization*

Existing visualization pages can be used together with the new **Visu** pages. It is important that the page IDs do not collide with the internally used IDs.

Set the start ID accordingly in Project Settings → IDs (p. 37). This is required for pages and global elements (*webinput*, *weboutput*, *timebuffer)* (p. 33).

*Macros*

Macros are still available for **Expert** programs. The macro library is integrated in Enertex® EibStudio 4. Macros are added to the program by `double-clicking` the name. Macro function calls can be edited again by `double-clicking`.

Integrated macros cannot be modified. Copy the code to a new library to change the macro. Internal macro libraries are disabled when a user macro with the same name exists.

To define a custom macro library, it is required to use the correct syntax. Otherwise the dialog to insert a macro is not created correctly.

*Predefined Visualization*

Predefined visualization functions are available as “Functions”. They are a combination of elements, which cannot be modified independently, but a function has its own property dialog.

You can use predefined page templates or add pages to your own set of templates, available in every project.

# Enertex® EibStudio 4

This section introduces the basic structure of Enertex® EibStudio 4 and the user interface.

If not made explicit, EibPC refers to all device generations in the following sections while EibStudio (without version number) means version 4.

## Installation

Enertex® EibStudio 4 does not require any installation procedure (like Enertex® EibStudio 3) but only has to be extracted. Check that you have permissions on that directory, especially if you move the EibStudio into a shared directory, e.g., into *Programs* on Windows.

The file *eibparser.exe in the subdirectory bin* must be executable.

## Title Menu

The Title menu bar contains central functions, which do not refer to a specific project (e.g., Settings, Help). With an active project, often-used functions (e.g., compile the project, execute the program), are added to the title menu.

## Projects List

Add new projects or import existing projects from Enertex® EibStudio 3 or Enertex® EibStudio 4. A project manages all information required by one Enertex® EibPC (configuration and program). All projects are stored in the projects directory.

**Do not change any file within the projects directory!**

### Projects Directory

On first start, a dialog asks for the location of the projects directory. Make sure that you have the necessary permissions (read, write) on that directory.

EibStudio 4 does not change or delete files outside of the projects directory and the Configuration Directory (p. 25). When a project is imported, the project files are copied here.

The projects directory can be changed in the **Settings** (p. 25).

### Import Enertex® EibStudio 3 Project

Enertex® EibStudio 3 projects consist of one or more source files (*.epc*). Supplementary source files are are imported by the main file using the #include directive.

To import an Enertex® EibStudio 3 project, click the respective button and select the main program. In the dialog, select the directory of the Enertex® EibStudio 3 program executable. This directory is used if the main program uses relative paths with the #include directives.

A new project is created with the name of the main program file. If an included file is not found, the import process is canceled and a message shows, which file could not be found. Check the path and change the #include if necessary. Restart the import process.

The following is imported into the new project if the process has been successful:

[ETS-ESF]: The group addresses from the *.esf* file are imported into **Objects**

[InitGA]: Initialization is activated for all group addresses

[FTP], [MailConf], [Performance], [VPN], [HTTPS], [Location]: Settings are set in **Settings** → **EibPC** and **Project Settings** → **Remote Access**

[MacroLibs]: The source files are imported as **User Macros** in **Expert**. Most of the Enertex® EibStudio 3 libraries are already integrated into Enertex® EibStudio 4. If a user macro and an internal macro have the same name, the library containing the user macro is disabled.

The program is added as new program in **Expert**. The sections listed above are converted into comments, the sections [EibPC], [Macros], [Webserver] are renamed into #addto [EibPC], ...

## Settings

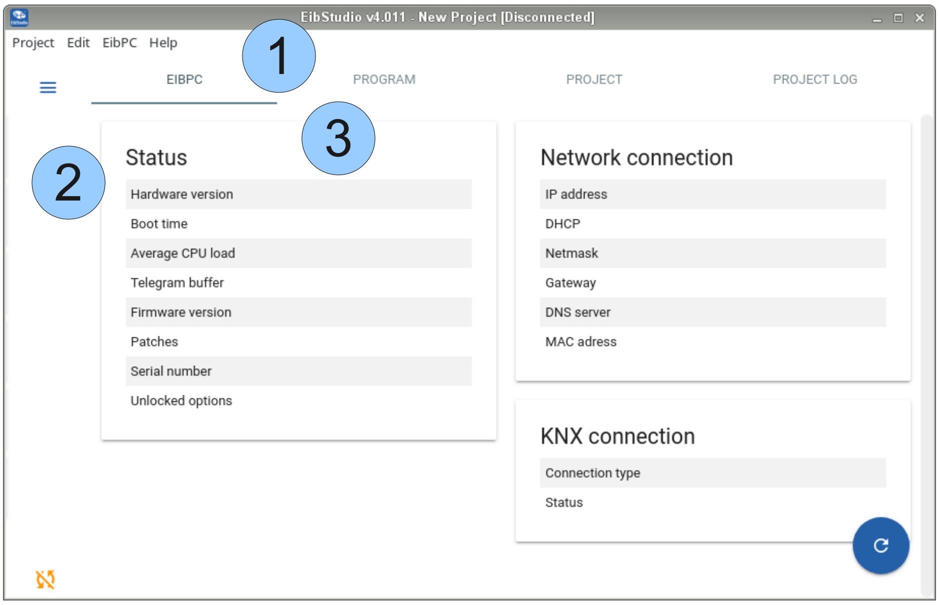
### Configuration Directory

Project-independent settings are located in the title menu **Edit** → **Settings**. They are used for all projects and stored in the configuration directory, in the file *eibstudio.json.* The path of this directory depends on the operating system used:

- Windows 10: *%LOCALAPPDATA%\eibstudio\User Data\Default*
- Linux ~/.config/eibstudio/Default/
- OSX: ~/Library/Application Support/eibstudio/Default

## User Interface

*Navigation*



Status

Hardware version

Boot time

Average CPU load

Telegram buffer

Firmware version

Patches

Serial number

Unlocked options

Network connection

IP address

DHCP

Netmask

Gateway

DNS server

MAC adress

KNX connection

Connection type

Status

*Figure 9: Overview*

Figure 9 shows the main navigation elements. With an active project, the title menu (1) is extended by functions often used. The project menu can be made visible with the project menu button (2). This menu is used to navigate between the different components of the project. Some of the components use tabs (3).

*Extenden Navigation*

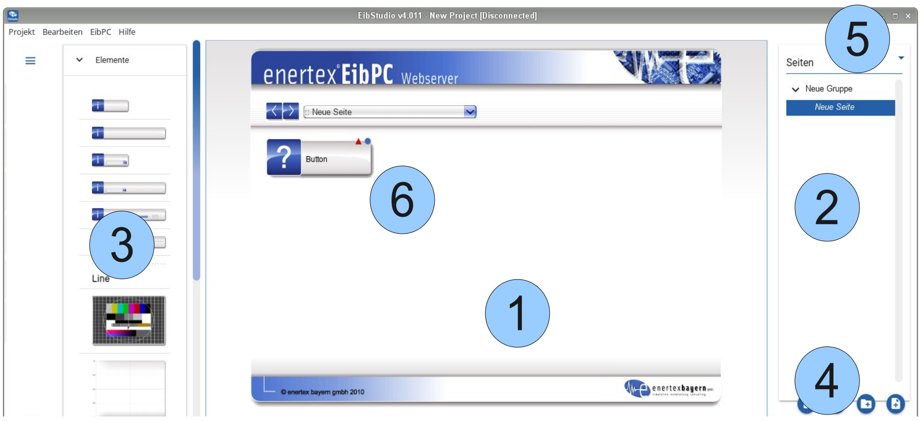


*Figure 10: Extended Navigation*

The following refers to Figure 10. **Logic**, **Visu** and **Expert** use additional navigation.

The main area (1) shows the currently selected entry (2). Entries from (3) can be clicked or dragged into (1). To remove elements from (1), select them by `click` and press `Del`. Hold `Shift` or `Ctrl` to add/remove elements from the selection.

Entries in (2) are added/modified/removed by clicking buttons (4).

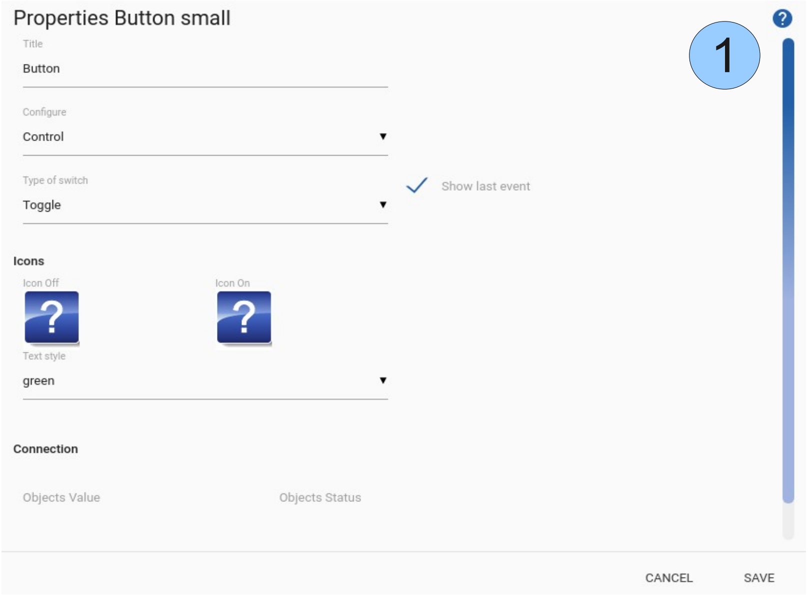
The arrow (5) hides (2) to enlarge the main area.

`Double-click` elements from (1) and (2) to open the property dialog.

The red triangle nearby (6) shows that the internal configuration of the element is incorrect or incomplete. The program will not work as expected.

The blue circle indicates a modification since the project has been saved.

*Property Dialog*



Properties Button small

Title: Button

Configure: Control

Type of switch: Toggle

✓ Show last event

Icons

Icon Off

Icon On

Text style: green

Connection

Objects Value

Objects Status

The property dialog is used to change the internal configuration. Most dialogs provide an integrated help function (1).

The following sections explain the components of the project menu.

## Overview

Overview shows information on the configured Enertex® EibPC and on the compiled program. Similar to the ETS, project-specific information can be set and a project log allows documenting project changes. Log entries are not related to an internal state but only used for documentation.

## Objects

Objects lists all known group addresses (”Manual” Group Addresses are not included), variables and predefined constant values. For a detailed explanation of these objects see Objects (p. 39). When a project is created, these lists are initially empty. On compilation of the project, they are updated. **If the compilation fails, the issues have to be resolved before the lists can reflect changes.**

The group address- and variables lists can be used to fetch the object's state from the EibPC. Select a specific object and click on the respective button in the upper right corner. A `double-click` fetches the current state, `Ctrl+click` to send a bus telegram or change the internal variable state.

Use the Debugger for extended features like sending read requests or watch multiple objects.

### Import Group Addresses

*Import .knxproj or .esf from ETS4/ETS5*

Group addresses cannot be created to avoid inconsistency on the KNX bus. Instead, group addresses must be imported from the ETS. Enertex® EibStudio 4 can read ETS 4/5 project files (*.knxproj*). Export the project in the ETS project list to create it.

**The project must not be password-protected and must use 3-level group addresses.**

For all imported group addresses, EibStudio tries to find the associated Data types. If neither the group address nor the connections have a DPT, an unsigned integer type with the bit length of the communication object is assigned. Unconnected group addresses remain without type information **and cannot be used until a type is assigned**.

*Infer Data Types*

Enertex® EibStudio still supports *.esf* imports (used in Enertex® EibStudio 3). This file however only includes connected group addresses and type information are less detailed. Only use this import type of importing a *.knxproj* file is not an option. Create the *.esf* file from ETS by using “OPC-Export“.

After import, the type of any group address can be modified.

**An incorrect type leads to an incorrect interpretation of bus telegrams!**

### Topology

The *.knxproj* import also reads the bus topology. This information is used to map individual addresses to devices in the Group Monitor (S. 38).

### Internal Variables

Variables can be created by the user to store any kind of internal state without having to send it no the bus.

Variables are also defined automatically by Logic, Visu and Expert macros. These internal macros are hidden by default, but can be made visible in **Objects → Variables** and in the **Debugger**.

### Constants

Enertex® EibStudio defines constants to simplify Expert programs, listed in **Objects → Constants**. Constants cannot be changed or redefined.

A new project has to be compiled once before the list of constants is loaded.

## Logic

The Logic is a simple way to combine objects and operations.

### Definitions

The following definitions:

Node
: Represents an object or operation. Has a type.

Input
: Handle on the left of a Node. Can be connected to one or more Outputs via Edges, except for Outputs of its Node.

Output
: Handle on the right of a Node. Can be connected to one or more Inputs via Edges, except for Inputs of its Node.

Port
: Input or Output

Edge
: Connects exactly one Input with one Output.

Trigger
: Port which starts an operation when the value changes from 0b01 to 1b01. The function is not triggered again while the Port is 1b01.

*Aggregated inputs*

If multiple edges can be conntected to a single Port, their order is not relevant. If the order of a Node's parameters is not relevant (e.g., ADDITION), only a single Input is used for simplicity. Connect all Outputs to this Input.

*Types*

Every Port has a type. Only Porty with compatible types can be connected. The following type combinations are possible:

*: All types
: Any type

b, u, f: Type class
: Any type of the same class

b01, u08, f16: Specific type
: Exactly this type

Examples:

An Input of type b01 may be connected to Outputs *, b, b01.

An Output of type u,s may be connected to Inputs *, u, uXX, s, sXX with XX being any size.

*Delete edges*

Please mind that a specific type must be known at compile time. The allowed types of the affected nodes are are updated with every new edge, but they remain when edges are removed. **It may be necessary to replace a node with a new instance to reset the allowed types.**

*Convert*

If nodes with incompatible types are to be connected, use the special node type CONVERT. It converts every type in every other type, but data may be lost if the new type can store less information.

*Multiple logics*

Logics can be split into multiple ones. Each Logic has the same priority, If a single object is written by multiple Logics, the object keeps the lastly written value. If an object is written multiple times in the same cycle, the result is undefined.

## Example: Automatic light

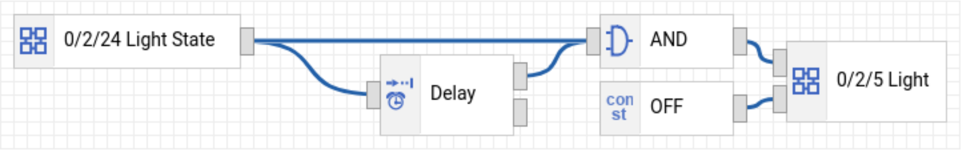


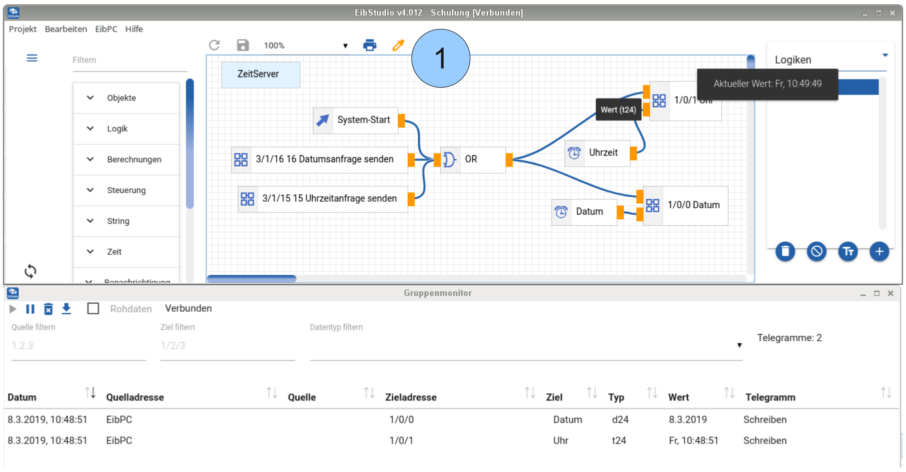
*Figure 11: Automatic Light*

A simple automatically switched-off light. Turn the light off 10 minutes after the last “on”-event.

- Start with an empty project, import your group addresses and compile the project to update predefined constants.
- Create a new Logic.
- Add the following node types:
  Objects/Group Address
  Objects/Group Address
  Objects/Constant
  Logic/AND
  Time/Delay
- Configure the first Group Address node to return the current object value
- The second writes on reception of an external trigger
- Select the constant “OFF”, which represents the 0b01 for the Constant node
- Configure the Delay to trigger after 10 minutes
- Connect the nodes according to Figure 11
- Compile and run the project

The Logic nodes are evaluated when objects change. For details, see Evaluation (p. 45). When the light's state changes from 0b01 to 1b01, the timer is started. Once it is over, its output is 1b01. If the light is still on (1b01), it is turned OFF (0b01) by sending a bus telegram.

## Debug-Mode



| Datum | Quelladresse | Quelle | Zieladresse | Ziel | Typ | Wert | Telegramm |
|---|---|---|---|---|---|---|---|
| 8.3.2019, 10:48:51 | EibPC | | 1/0/0 | Datum | d24 | 8.3.2019 | Schreiben |
| 8.3.2019, 10:48:51 | EibPC | | 1/0/1 | Uhr | t24 | Fr, 10:48:51 | Schreiben |

*Figure 12: Debug-Mode in Logic*

To implement the Logic, internal variables are created for every Input and Output. They are usually hidden (p. 28). To get the current state of each Node, turn on the Debug mode (1).

When active, all Ports are highlighted. On `click`, the internal state is fetched from the Enertex® EinPC. `Ctrl+Click` can be used to directly set a new value.

It is recommended to use Simulation for advances tests (p. 38).

*The Logic in Figure 12 shows how to use the EibPC as a time master for the KNX bus. Every time the EibPC starts its program, it sends date and time to the bus, using appropriate DPTs. If NTP is used, the EibPC waits for the time to be synchronized before starting the actual program. Additionally, time information can be fetched by sending a request to the group addresses.*
*The Group Monitor shows both telegrams, date and time.*

## Visualization Objects

If the predefined Visu elements do not fit your needs, it is easy to use the Logic to evaluate Visualization events and change elements. Open **Visu**, add the element and select “Connect to logic” from its property dialog.

This makes the element usable for your **Logic**. Open your Logic, add the respective type of visualization element, depending on what you added in **Visu**. Open its properties and select the element.

Hint: If you have complex Logics using both, return value and setting the element's status, you simply can add the same node twice (copy `Ctrl+c`, paste `Ctrl+v`), to the left and to the right. Add edged only to the outputs and inputs respectively. Like that, crossing edges can be circumvented.

## Visu

It is simple and fast to create a Visualization in Enertex® EibStudio 4.

Each visualization is split into groups and pages. Each page can have an individual size and design. The order of the sections and pages is also used on the webserver. It can be changed by dragging items to the right place.

### Elements

Elements are individual items of the visualization, e.g., buttons, charts. One's behavior can be changed in its property dialog. Most of the functionality needed for a elaborate visualization can be directly configured on an element, like a button to toggle a group address or a slider to dim the light.

### Functions

Functions on the other hand are predefined Elements or groups of Elements with a custom property dialog. To use a Function, all Elements must be placed on the same page. Otherwise the Function cannot be added to the page.

Placement of Elements (either individual or Function Elements) can be changed by dragging them to an empty space. The preview directly shows how the real visualization will look like.

### User Templates

The currently active page can be stored as a user template, which then can be added to any other project. Created templates cannot be modified. Instead, simply add it to your current project, modify it and save it as a new template. All connections to objects are preserved by the template. If you have similar structure for multiple projects, this saves much of your configuration time.

### Templates

Additionally, Enertex® EibStudio provides some templates, e.g., for the Enertex® SmartMeter.

### Access from Logic and Expert

To implement more complex functionality, it is possible to connect Elements to the **Logic** or your **Expert** programs. This was, you still can use the graphics visualization editor without losing flexibility compared to a “programmed” visualization (Custom Visualization, p. 33).

Using an Element from within your Logic, is simple. You can switch between the basic appearance and its “active” state (p. 31).

With the **Expert**, you are not limited in any way. A unique Variable is defined to access the element, without having to know its ID (nor the ID of the page). See Access Visu Elements (p. 33) for details.

## Expert

The Expert provides access to every feature of the Exertex® EibPC by writing programs. For a function reference, see Instructions and Functions (pp. 52).

Number of Expert programs is not limited. All programs are compiled in the same “global” context without special ordering. A variable defined in one program can be used in any other program (but also must be unique!)

### Auto-completion

Auto-completion is used to functions, macro and objects. The lists are updated on compilation. If you define a new variable, you have to compile the project for the auto-completion to include this variable.

To simplify entering a group address, start it with double-quotes and enter significant parts of its name or number in the correct order: ‟ followed by `123` completes to “1/2/3 Light” and “1/0/23 OtherLight”.

### Macros

Macros are similar to functions in other programming languages. They are used to structure the program and avoid code duplication. An large collection of macros is provided with Enertex® EibStudio.

### Custom Visualization

You can use the expert to “program” your visualization. Use the directive #addto [Webserver] before starting with webserver definitions (Configuration of the  Webserver, p. 151).

Every webserver element uses an individual ID for definition and as a reference for other functions referring to the element. Visualization defined in **Visu** automatically generates such IDs. It is necessary that these IDs do not alias with the IDs used for custom visualization.

If an Enertex® EibStudio 3-project im imported, this is especially important if it includes visualization (custom or defined with the Visu-assistant).

To avoid conflicts, please check the code, which IDs are used, and enter the first free IDs into **Project Settings → IDs** (p. 37).

### Access Visu Elements

It is also possible to combine an Expert program with visualization elements defined in **Visu**. Element-IDs used by the webserver change, depending on page order and placement of the Elements. Instead of the numerical ID, you can assign a unique name to an Element. On compilation, the internal ID is assigned to this Variable. Do not forget to compile the project for the Variables list to be updated, so the name is available for auto-completion.

The name must be a valid Variable name (p. 43).

If the ID of the Element is relative to the page (see below), Enertex® EibStudio automatically defines a Variable for the page's ID. Its name is the Variable's name with the additional suffix “_P”.

Example:

The unique Variable for a Button element is *ButtonVar*. A Button is relative to the page (function pbutton), so the Variable to refer to the page is *ButtonVar_P*. After compilation, both Variables can be used by the Webserver Funktions (p. 135):

```
pdisplay(ButtonVar, $MyButton$, INFO, ACTIVE, GREEN, ButtonVar_P)
```

**If you use custom visualization pages, you have to define the start-IDs for Visu (p. 37).**

Page-dependent Visualization-elements:

*Button*, *Shifter*, *Multibutton*, *Multishifter*, *Slider*, *Picture*, *Value Chart*, *Time Chart*.

Global IDs:

*Webinput* and *Weboutput*.

## Syntax

*Define Variables*

Define a Variable by assigning an initial value and type. The name must be unique. See p. 43 for a detailed explanation of Variables.

```
Var=1b01
```

*Group Addresses*

The last known (internal) value of a group address can be assigned to a Variable. Use the name shown in **Objects → Group Addresses**, consisting of the name of the group address defined in the ETS, followed by the numerical group address (main-, middle-, sub group), separated by a dash "-" (see p. 43). The Value of Var changes whenever the state of the group address changes.

```
Var="GA-1/2/3"
```

*if-Clause*

The most simple form of an if-statement is convenient for simple if-then rules.

```
if "GA-1/2/3" then Var=EIN endif
```

The general definition of th if-clause is

if (Condition) then {Block}Statement1 else {Block}Statement2 endif

The condition must be of type 1b01.

A statement is an assignment, a function call or a macro instantiation. Multiple statements are split by ";" (semicolon).

If the statements span multiple lines, they must be enclosed by "{}":

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

*Comments*

You can add comments to your programs::

1. Line comments starting with „//"
2. Block-Comments "/* ... */": used instead of a statement. When used inside of a block, a semicolon required at the end.

```
/* This is a comment */
// Another comment
u=5;/* And the last comment. Don't forget the semicolon */; u4=5
```

## Online-Debugging

Online debugging helps by getting notified when values change at runtime. A simple way is so emit UDP datagrams with the new value. They can be received by a simple listening program, e.g., netcat (see https://de.wikipedia.org/wiki/Netcat).

A simple Debug-Macro could look like the following. The datagrams are sent to IP 192.168.1.18, port 9000 ( `netcat -ul 9000`).

*Send a string to a remote host*

*Empty macro*

```
#define DEBUG
#ifdef DEBUG
// Send datagrams to 192.168.1.118, 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
```

If Debugging is enabled by `#define DEBUG`, a UDP datagram is sent every time the statement is evaluated. If `#define DEBUG` is not active by adding a comment to the line, nothing is done. Note the statement __EMPTY(). If prevents the macro from being instantiated, and no code is generated at all.

```
x=3
If x>5 then {
    x=x*2;
    vmDebugUDP($x is $+convert(x,$$));
} endif
```

If `#define DEBUG` is defined, a datagram is sent when *x* changes. Otherwise, the statement vmDebugUDP(*$x is $*+convert*(x,$$)*); does nor generate any overhead.

If a statement is used only then debugging is active, keep in mind that even with an empty then-clause, objects are created:

```
x=3
If x>5 then {
    vmDebugUDP($x is $+convert(x,$$));
} endif
```

The compiler does not create anything for the debug statement, but for the if-statement if x>5. A more efficient way is to disable the whole block:

```
x=3
#ifdef DEBUG
If x>5 then {
    vmDebugUDP($x is $+convert(x,$$));
} endif
#endif
```

## Project Settings

The project settings are used to configure a single Enertex® EibPC, i.e., a single installation.

Changed must be sent to the Enertex® EibPCs, either by pressing a button (B) or together with the program (P).

### Search EibPC

The search packet for EibPCs on the local network is sent from every Ethernet device.

### Connection to KNX

(P)

Select the right connection type, depending on your configuration.

### Network

(B)

The EibPC is automatically restarted when the network configuration is changed. If the device is unreachable, perform a Factory Reset to activate DHCP (p. 21).

### Name resolving

(B)

Some functions rely on the network name resolution via one or more DNS server (sendmail, resolve).

### Date and Time

For the time functions, a correctly set internal time is inevitable. It is highly recommended, to use the same time source for each devince connected to the KNX bus. The Enertex® EibPC can use time information from the bus to synchronize the internal clock. If no reliable time source is available, the Enertex® EibPC can be the time master, and regularly send its internal clock to the bus.

The Enertex® EibPC can keep its clock synchronized to a server its internal clock using the NTP protocol.

If NTP synchronization is active, it has the highest priority. A manually set time (either via Enertex® EibStudio or the KNX bus is overwritten. Before the actual EibPC program starts, it tries (at most 5 minutes) to synchronize its clock.

### Location

(P)

The Enertex® EibPC computes a lookup table for each 5-minute interval for the current year, to “know” the sun's position in any cycle. Updating the sun-data takes ~5 min.

### SHUTDOWN Variable

Before the program is stopped (when a new program is transferred or the EibPC is restarted using Enertex® EibStudio) the variable *SHUTDOWN* can be set to 1b01 to allow function to store values on the flash memory. A delay of 5s is recommended.

### FTP

(P)

The Enertex® EibPC can forward all telegrams received from the KNX bus to an FTP server. It uses port 21

### E-Mail

(P)

Um E-Mails senden zu können, müssen Sie hier die Verbindungseinstellungen festlegen. (P)

### Backup

Be fore a new program is transferred to the EibPC, the currenty open project can be exported and sent to the EibPC. The synchronization can also be triggered manually, and the backup can be fetched at any time.

### Files

To use a custom image in the visualization, it must be sent to the Enertex® EibPC. The image is also stored in the projects directory and automatically sent again if another EibPC is used with the same project. Images on the EibPC not yet added to the project are also synchronized.

### HTTPS

(S)

The Enertex® EibPC can provide an encrypted access to the visualization using HTTPS. A certificate has to be generated and user credentials must be set before.

For access from outside of the network, TCP port 443 must be forwarded to the EibPC.

### VPN

(S)

To access your network, the Enertex® EibPC can open an OpenVPN server. You must generate a certificate before the OpenVPN server can be started.

### IDs

The firmware uses unique numerical IDs to access internal objects. They are set when an object is defined and must be used to access the object.

### Activation codes

If a new activation code to unlock features of the Enertex® EibPC has been purchased, it can be applied using Enertex® EbnStudio.

## Export and Import

To export a project, select **Project → Export** from the title menu. All project data is copied into a *.zip*-archive with the file ending *.esp*. In contrast to **Help → Export For Support**, this includes private data (e.g., e-mail password).

## Debugger

To open the Debugger, select **EibPC → Debugger** from the title menu. Add group addresses and variables to the list of watched objects. You can use the Debugger to fetch the internal state of all objects on the watch list, send group telegrams, read requests, and change the internal state of objects, which triggers the evaluation of depending objects just like any other "regular" change.

## Group Monitor

Select **EibPC → Group Monitor** from the title menu to watch telegrams. If the project contains topological information from an .knxproj import, the Group Monitor shows the device name assiciated to the individual address of the sender of group telegrams.

The list is limited to 100 last entries. The list can be stored in a *.csv* file.

## Long Term Buffer

The Long Term Buffer automatically kepps a list of the last telegrams. It is limited to 150.000 telegrams if the visualization is active, and 500.000 otherwise. Old telegrams are removed if the buffer is filled. To fetch the buffered telegrams, select **EibPC → Fetch long term buffer** from the title menu to store a .csv file.

## Events

Whenever something unexpected happens, an Event is logged and buffered until the Event log is read by selecting **EibPC → Events** from the title menu. See p. 225 for an explanation of the Events.

## Simulation

To implement and verify complex control logic, simulation may be helpful. Select the KNX connection type "FT1.2 Bus Monitor" from the Project Settings (p. 36) and disconnect the interface if it is actually used. The Group Monitor still shows all telegrams sent by the EibPC, without affecting other devices.

To simulate other devices' behavior, send status updates to the respective group addresses and answer read requests. A basic simulation is shown in Figure 13.

Add three Group Address nodes and configure the them as follows:

1. Generate a trigger on reception of a read request
2. The currently stored internal value
3. The write node uses an external trigger and marks the telegram as answer.

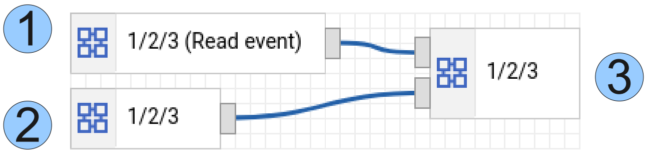


*Figure 13: Answer Read Request*

Use this method to create test environments instead of forcing 10s of values within the Debugger.

Without access to the KNX bus, read requests cannot be answered and have to time-out. Each request takes 1.5 s when the Enertex® EibPC starts, which creates a huge and unnecessary delay. The initialization can be disabled in the Project Settings (p. 36).

**Do not forget to enable the initialization after simulation!**

# Objects

Objects represent internal states, and they can trigger state transitions. Basically, EibPC programs contain a set of rules: if s.th. then do s.th. else. Objects are both, condition as well as result.

The Enertex® EibPC knows of two types of objects: group addresses and variables.

### *Group addresses*

Group addresses are objects with a state known to the knx bus devices. Each device must update its internal state of relevant objects when it receives a bus telegram and react accordingly if configured.

Apart from thes public object states, each devices has internal states, which are only used by the device itself. Those objects are called variables.

### *Variables*

Example: A switching actuator watches a group address connected to its communication object *Toggle channel 1*. The actuator knows its internal switching state used to turn on or off. It also sends its new internal state to inform the other devices of the change.

When switching, the group addresses of the actuator's channel and its status, as well as the internal state of the switch are relevant.

The basic principle of the Enertex® EibPC, being a universal logic machine, is pretty much the same, apart from the fact that the set of rules is defined by the program (and thus by you) instead of the device manufacturer.

Every object can be combined with every other object by using one of many different internal functions.

The ETS uses Datapoint types (DPTs) to organizes the type of group address telegrams. They define size and (optionally) its interpretation. An object of size 1-Bit (DPT 1) may be interpreted as DPT 1.001 On/Off or DPT 1.008 Up/Down.

DPTs are mapped to internal types on import, which only contain data type and size:

## Data types

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 89.

*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: 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 88).

IP addresses (add on Option NP) have the following syntax

*Special type: IP Adress*

- 192.168.22.100. An IP address is of data type u32.

Physikal KNX - addresses are defined as followed in the programm code

*Special type: Individual Address*

- 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<br>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.

## Variables

**Variables** start with letters, followed by any number and combination of letters or numbers, and the "_" character. Variables must be defined in global context (outside of an if-statement) and initialized to 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
```

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:

*Not permissible here...*

```
//Declaration
a=0
//Counting
if (sun()) then a=a+1 endif
```

*... but here*

Umlauts are not allowed in variable names. Therefore, the following expression **is invalid**

*No special characters in variable names*

```
KitchenLightOn=1b01
```

## Group addresses

Use the ETS import (p. 28) to add group addresses.

### "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.

## Initialize Group Addresses

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).

In order to synchronize with the KNX bus, some Group addresses have to be read by the EibPC. You can achieve this by selecting the initialization check-box group address in **Objects → Group Adresses**.

**Important**

- Before the actual program starts, the Enertex® EibPC sends a read request and waits for the reply (no longer than 1.5 s).
- The actual program starts after the last group address has been initialized.
- All statements and functions depending on an initialized group address are marked as invalid and processed in the first cycle, even if the request failed.
- An event is logged when a read request fails.

## Evaluation

*Object tree*

This section explains, how statements are evaluated. When the project is compiled, a program is generated, which is executed by the firmware of the Enertex® EibPC.

In contrast to a program for a microprocessor, this program is not a sequential list of instructions but a dependency tree. The nodes of the tree are called Program Objects (not to be confused with Objects p. 39). Program Objects include all Objects, but also all Instructions and Functions (p. 52) are Program Objects.

Instead of execution one instruction after the other, time is split into logical steps (cycles). Evaluation of objects (logically) happens in parallel within a single cycle., each change has the same priority. To minimize the work in each cycle, only changed Program Objects are evaluated.

Each Program Object knows

- if its value has changed since the last cycle,
- if it is still has a constant value,
- if an event occurred,
- if its descendants must be updated when its value changes.

If its value changed, the state is now “invalid” and is must be evaluated and all descendants must be notified. After that, it is “valid” again.

Example: When the function “write” is evaluated, a telegram is sent to the KNX bus.

Each cycle consists of the following steps, until no object is invalid any more:

Invalidate

If a Program Object is invalid, it has to be re-evaluated. In the first program cycle, every object is invalid. In any other cycle, an event must have invalidated the Program Object, e.g., a bus telegram. Only Program Objects depending on a Group Address, Timer, TCP/UDP, RS232 or an if-clause can become invalid.

Evaluate

Update the value using the new input values. If the value changed, execute next step to notify descendants.

Conditional Invalidation

Invalidate all Program Objects in dependency list.

The exact behavior depends on the type of the Program Object.

**The following examples can be added as new Expert program.**

*Variables*

Example:

```
x=2
y="SaunaDimmer-1/0/1"+3%+x
z='1/2/3'b01 or '1/2/4'b01
```

The compiler generates the Program Object Tree Figure 4.

x is initialized to the value 2, y to the value of the group address plus 3% plus x. The following cycles to not change x since 2 is a constant. Instead, y is re-evaluated with every telegram on the KNX bus, if the value differs from the last one received. Y depends on an expression which became invalid. The same would be valid for x if x would change.

Invalidation propagates down the tree until the a Program Object does not change.

The Variable z indirectly depends on a group address. If “1/2/3” becomes ON (1b01), the logical OR becomes ON and invalidates z if it was OFF in the last cycle. If “1/2/4” becomes ON in the next cycle, OR is invalidated, re-evaluated but does not change. z is thus not invalidated.

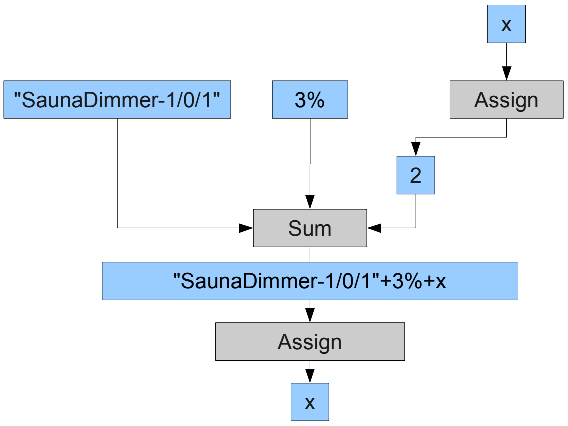


*Figure 14: Program Objects Tree for y="SaunaDimmer-1/0/1"+3%+x and x=2*

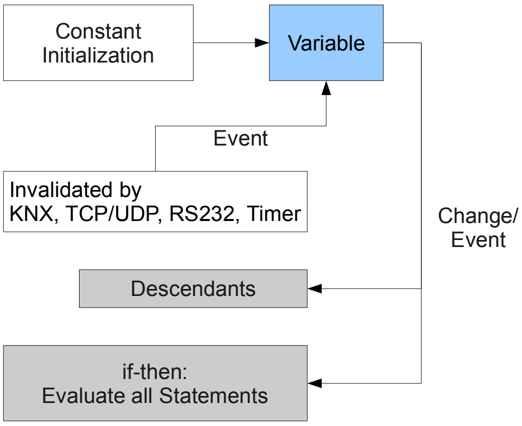


*Figure 15: Evaluation of Variables*

*Functions*

A Function becomes invalid with its arguments. If an argument changes, the function es evaluated. If the result differes from the current value, all descendants become invalid.

```
x=sin(3.14f32)
tan(2.0f32)
y=cos("Temperature-1/0/1")
z=event("Temperature-1/0/1")
```

*Side effects*

Functions with side-effects are handled differently. When they are evaluated, they do not only change their internal state but have some kind of externally visible behavior. To make sure that such functions are only “actively” triggered, their arguments never invalidate the function, but they can only be triggered by an if-statement (to be more precise, by the condition of the if-statement, see below).

```
write("Temperature-1/2/1",22.3)
write("Switch-1/2/10",!"Switch-1/2/10")
read("Temperature-1/2/1")
```

*This program never writes to the KXN bus. If evaluated like a regular function, it would write to the bus in each and every cycle.*

*Timer*

Timers are handled similarily. Only the system time of the Enertex® EibPC invalidates a timer.

```
o=stime(19)
```

*O is ON (1b01) exactly 19 seconds after the beginning of every minute, and only for a single cycle.*

*if-statements*

The (non-nested) if-clause behaves a like a function with the condition being the single argument. If the condition becomes invalid (any Program Object part of the condition changes), if is evaluated. Not that this is true even if the condition changes to “false” (0b01).

```
a=1
if '1/2/3'b01 then a=3 endif
```

*If a bus telegram for group address '1/2/3' is received and its value is 1b01, a becomes 3. It never changes any more because 1 (from a=1) never invalidates a.*

*Nested if-statements*

Nested if-clauses do not become invalid by their condition (in contrast to non-nested if-clauses) but by the condition of the outer if-clause. This guarantees that the outer condition is evaluated. Thus, the inner then-clause does not require the inner condition to change.

```
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
```

This example demonstrates the changed semantics of nested if-statements:

**Do not try!**

```
if change('0/0/1'b01) then {
   if ON then write('0/0/1'b01, !'0/0/1'b01) endif
} endif
```

*If the inner write statement was not inside of a nested-if, it would never be evaluated and nothing would get written to the KNX bus, because the condition (constantly ON) never changes.*
*Due to being nested, wite becomes invalid with every change of '0/0/1', again invalidating the group address by sending a telegram with the inverted value.*
*The program emits a telegram with every single cycle.*

*Timer in then-clause*

Timer in nested if-statements are only evaluated if the outer if-condition invalidates it.

```
Button='1/2/3'b01
a=OFF
if Button then {
   if htime(12,00,00) then a=ON endif
} endif
```

*a becomes ON if Button becomes ON exactly at 12:00:00 (htime is 1b01 for a single cycle only at the exact time). A more robust implementation uses chtime (its value becomes 1b01 at 12:00:00 and is reset at 24:00:00). If Button is ON at any time after 12:00:00, a is ON (though a is never set to OFF again).*

*else-clause*

The else-caluse of an if-statement is essentially another independent if-statement with an inverted condition.

```
Button='1/2/3'b01
if Button then write('4/5/6'b01, OFF) else write('4/5/6'b01, ON) endif
```

The program is identical to

```
Button='1/2/3'b01
if Button then write('4/5/6'b01, OFF) endif
if !Button then write('4/5/6'b01, ON) endif
```

*Queues*

When a cycle is complete (no Program Object is invalid), the output queues are processed. Function arguments are evaluated with their most recent state, i.e., an Object may have been changed by a function after the queued function. The following functions are queued until the end of a cycle:

- sendudp
- sendudparray
- resolve
- sendmail
- sendhtmlmail
- sendcp
- sendtcparray
- connecttcp
- closetcp
- resetasyncserial
- sendasyncserial
- startvpn
- stopvpn
- openvpnuser
- closevpnuser
- ping
- writeflash
- writeflashvar

Examples:

*Network and RS232*

```
uPing=10
uIp=192.168.1.1
if after(systemstart(),1000u64) then {
   uPing=ping(uIp);
   uIp=192.168.1.100;
} endif
```

*uIP is initialized with 192.168.1.1. One second after system start, the if condition is evaluated, and thus the statements of the then-clause. ping is queued, while uIp=192.168.1.100 is executed without delay. When the cycle ends, ping is executed with the already changed IP.*

```
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
```

*The program send the string $World$ twice as the UDP queue is processed after the assign statements.*

*Flash*

Writes to Flash are also queued:

```
b=1
if systemstart() then {
  writeflash(b,0u16);
  writeflashvar(b);
  b=b+1;
} endif
```

*The variable b is incremented before it is written into the flash.*

```
b=5
if systemstart() then {
  writeflash(b,0u16);
  b=b+1;
  readflash(b,0u16);
} endif
```

*Reading from flash is not queued but executed directly. The variable is read, incremented by one and finally written when the cycle is over.*

*Asynchronous return values*

Some function calls (e.g., connecttcp, sendmail) do not update their return value during the same cycle of their of evaluation. Instead, they change their return value “asynchronously” to their evaluation.

Example:

```
// TCP off == 5
TCP=5
if after(systemstart(),2000u64) then {
  TCP=connecttcp(233u16,192.168.2.100)
} endif
```

*Two seconds after Systemstart is 1b01, connectcp is called. The return value is set to 0 (Connecting). When the connection is established, connecttcp changes TCP to 1 (Connected), without evaluating the if-condition again. All Program Objects, depending on the return value, are evaluated in the next cycle.*

*Macros*

Macros are essentially simple string-replacements.

Example:

```
:begin MyFunction( Message )
    write( '9/2/0'c14, $Display $c14);
    write( '9/2/0'c14, $Message:$c14);
    write( '9/2/0'c14, convert(Message,$$c14))
:return OFF
:end
```

Only those macro statements after :return are relevant to the Program Object evaluation.

The program

```
if sun()  then MyFunction($Light$) endif
```

does not write anything when at sunrise. It is identical to:

```
write( '9/2/0'c14, $Display $c14);
write( '9/2/0'c14, $Message:$c14);
write( '9/2/0'c14, convert($Licht$,$$c14))
if sun() then OFF endif
```

*writes are global!*

The write-instructions do not depend on sun(). With the changed program, evaluation is applied to the writes:

```
:begin MyOutputFunction( Message )
:return {
    write( '9/2/0'c14, $Display $c14);
    write( '9/2/0'c14, $Message:$c14);
    write( '9/2/0'c14, convert(Message,$$c14))
}
:end
```

The same macro call

```
if sun()  then MyOutputFunction($Light$) endif
```

*“Forward dependencies”*

now sends three telegrams to the KNX bus.

The :return expression “forwards” the dependencies of an if-statement to control evauation within macros. With :return, a larger block of statements or single parts of the function code depend on the calling code.

Example:

```
:begin Act_3(Actuator,Now)
   Variable=3
    if Now then write(Actuator,Variable) endif
:return OFF
:endif
```

*When used similar to*

```
if sun() then Act_3('1/2/3'u08,chtime(5,00,00)) endif
```

*only OFF depends on the condition of the if-statement (sun()).*

**:return defines the return value and which part of the macro becomes invalid with the if-condition.**

The macro is expanded to

```
Variable=3
if chtime(5,00,00) then write('1/2/3'u08,Variable) endif
if sun()  then OFF endif
```

*Nur globale Definitionen*

Changing the macro to

```
:begin Act(Actuator,Now)
:return Variable=3; if Now then write(Actuator,Variable) endif
:endif
```

and calling it like

```
Variable=0
if sun()  then Act('1/2/3'u08,chtime(5,00,00)) endif
```

is expanded to

```
Variable=0
if sun()  then Variable=3; if chtime(5,00,00)) then write('1/2/3'u08,Variable) endif
```

*After sunrise, after the system time is 5:00 o'clock or later, Variable becomes 3 and the new value is sent to the group address '1/2/3'.*

Attention: By moving the variable assignment into the then-clause, is is never initialized within the global context and an explicit definition (*Variable*=0) is required.

*Recursion*

The program

```
a=OFF
if a==ON then a=!a else a=!a endif
```

results in a recursive tree (see Figure 16):

*When initialized, the else-clause is evaluated, interverting a. Because it was changed, a (now ON) is invalid, the condition is re-evaluated and the then-clause is evaluated, inverting a again. As it changed again, the condition is re-evaluated, invalidating the else-caluse, inverting a, …*

The firmware of the Enertex® EibPC catches circular dependencies, stopps the evaluation and generates an Event (PROC_REPITIONS, p. 225).

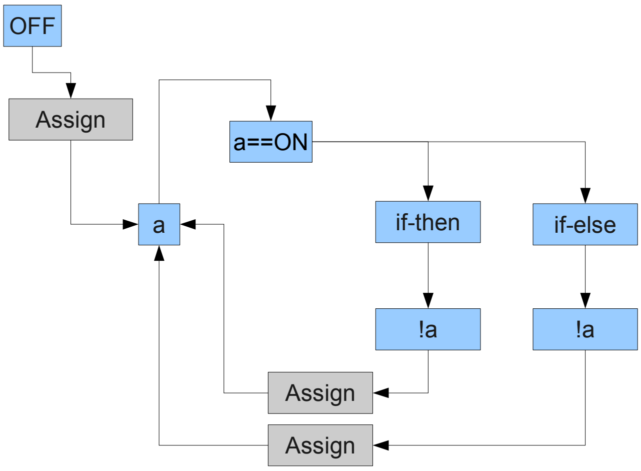


*Figure 16: Program Object Tree Structure for a=OFF; if a==ON then a=!a else a=!a endif*

The Program Object Evaluation guarantees that

- complex programs are executed efficiently by the Enertex® EibPC
- Basic rules (if Button then Light) are easy to program
- *all statements in a single cycle are executed "in parallel".*

# Instructions and Functions

This section is only relevant if you plan to write own expert programs.

***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 40.

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 40.

**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*

## Time

### 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 59*

## 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 59*

## 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 105).

*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.

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.

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.

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($2012-09-03 20:00:17$)
```

# Date

## 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 with the binary value 1b01 and hence can be stated as a comparison operator instead of 1b01.

### Azimuth

**Definition**

- Function azimuth()

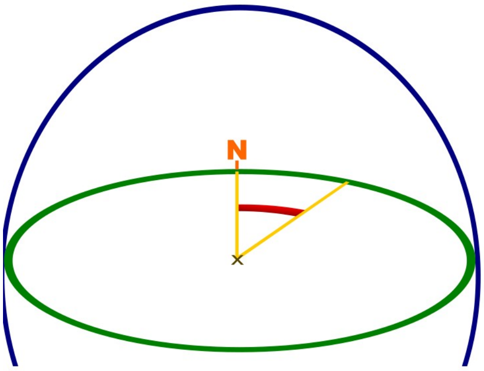
**Arguments**

- None. However, the Enertex® EibPC should know the longitude and latitude of the place.

  These can be entered in Enertex® EibStudio (see page 65).

**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**

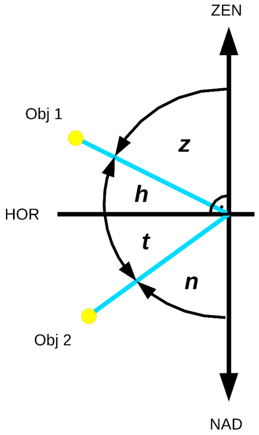
- Function elevation()

**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 65).

**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 b01

```
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
```

# Timer

## 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, 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, 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, 7=Weekdays, 8=Weekends)

**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
```

## Delays

### 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**

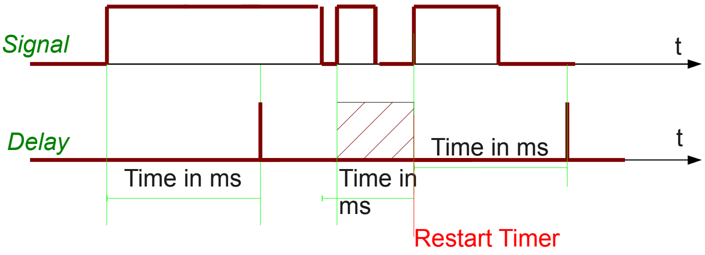
- Function delay(*Signal*, *Time*)

**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. 73).
- 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**

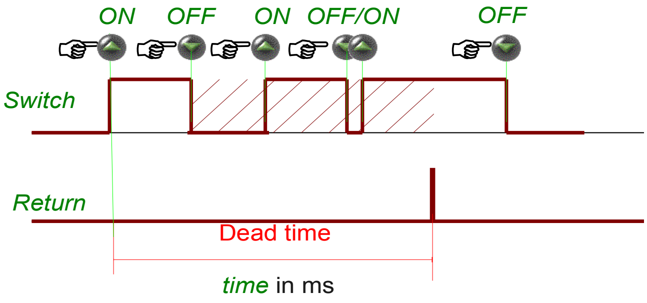
- Function after(*Signal*,*Time*)

**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
```

*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. 74).
- 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

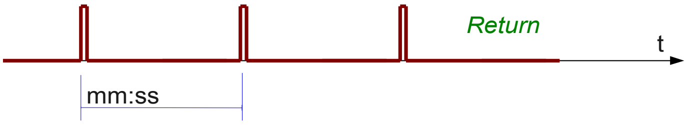
**Definition**

- 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 transferring patches and nor by changing 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 40).

### 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 40). 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 is changed asynchronously to the main development loop)

### 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 changed asynchronously to the main development loop)

***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 40)). 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 changed asynchronously to the main processing loop)

### Writeflashvar

**Definition**

- Function writeflashvar(*Variable)*

**Arguments**

- *Variable* arbitrary data type

**Effect**

- The binary data of the memory content (see bit length, page 40) 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 changed asynchronously to the main processing loop)

***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. 94)

```
ValuePowerK1="K1-Wirkenergiezähler (Verbrauch)-14/2/76"
if htime(0,0,0) or SHUTDOWN then {
                writeflashvar(ValuePowerK1)
} endif
if systemstart() then readflashvar(ValuePowerK1) endif
```

## Arithmetic operations

### 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 40

**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 89)

Data type result (Return)

- Data type of the arguments

Note:

With the same syntax, you can concatenate character strings (see page 101).

### 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 40

**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 40

**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.
- If the  base is negative, no calculation is performed.

Data type result (Return)

- Data type f32

## 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.
- 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 112), 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 type, except for b01.
- 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.
- If a string is converted into a numerical type, the value is parsed. If the string starts with 0x or 0X, the number is converted from hexadecimal.
- 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.91***

## 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(34f32)+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)+tan(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. 80

## 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 1: 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 2: 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 3: 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 **Project Settings → Files**).

**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. 98).

**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. 98.

**Data type result (Return)**

- none

**Note:**

1. It is possible to deactivate inputs differently in each scene number, see presetscene p. 98.
2. You can (like other functions) define an arbitrary number of scene actuators.
3. Each Scene actuator has 64 scenes (1to 64).

***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
```

## Strings

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.

### 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 102.

## 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. 38), 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 40.
- 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 59).***

> 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 59), 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)
```

## 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)
```

## 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)
```

## RS232 interface

If you establish your KNX 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.

### 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 Telegrams

Writing information to the KNX$^{TM}$ bus is realized with the help of the write function.

### write

**Definition**

- write(*GroupAddress, Value*)

**Arguments**

- 2 arguments of the same data type, but otherwise the data types are arbitrary..
- *GroupAddress*: Imported or manual KNX$^{TM}$ 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 39).

### read

Send read request

**Definition**

- read(*GroupAddress*)

**Arguments**

- *GroupAddress*: Imported or manual KNX$^{TM}$ 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

**Note:**

The flag in the ETS program must also be set so that the actuator in the KNX network responds.

***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 4.

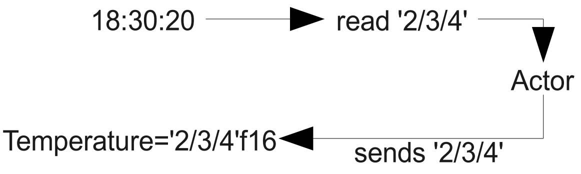


*Figure 4: Operation of read*

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).

## 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, TCP or RS232 telegrams, it reacts to the arrival of packets.

An event function is defined as follows:

**Definition**

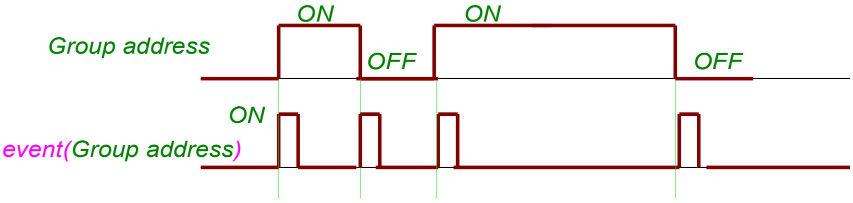
- Function event(*Group address*)

**Arguments**

- *Group address*: Imported or manual KNX™ group address
- The groupaddress can be optionally negated using the !-Sign.
- For UDP, TCP or RS232 telegrams the event function can be applied.

**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.

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

**value)**

- Data type b01

**Definition**

- Function eventresponse(*Group address*)

**Arguments**

- *Group address*: Imported or manual KNX$^{TM}$ 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$^{TM}$ bus, regardless of its content.

### eventresponse

**Data type results (Return value)**

- Data type b01

**Definition**

- Function eventwrite(*Group address*)

**Arguments**

- *Group address*: Imported or manual KNX$^{TM}$ 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$^{TM}$ bus, regardless of its content.

### eventwrite

**Data type results (Return value)**

- Data type b01

**Definition**

- Function writeresponse(*Group address,value*)

**Arguments**

- *Group address*: Imported or manual KNX$^{TM}$ 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$^{TM}$ which writes the

  *value* to the *group address* is sent to the bus. The response flag is set in the telegram.

### writeresponse

**Data type results (Return value)**

- none

Definition

- Function eventread(*Group address*)

**Arguments**

- *Group address*: Imported or manual KNX$^{TM}$ 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$^{TM}$ bus, regardless of its content.

**Data type results (Return**

## initga

**Definition**

- initga(*GroupAddress*)

**Arguments**

- *GroupAddress*: Imported or manual KNX$^{TM}$ 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. 88)

**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.

### eibtelegramm

## 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. 96

### 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 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

### Default Ports

The ports via which the Enertex® EibPC communicates can be changed via **PROJECT SETTINGS → CONNECTION**.

### 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.

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.

*UDP Ports*

*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.

*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*

*TCP ports*

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.

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 changes asynchronously to the main development loop).

*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
```

*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)

***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 changes asynchronously to the main development loop)

***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 p. 36).

## 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 changes asynchronously to the main development loop)

***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 130)

## Sendhtmlmail

Before the function sendhtmlmail can be used, the basic e-mail configuration has to be done (see p. 36).

**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 129).

## VPN Server

*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**

*Getvpnusers*

**Definition**

- Function getvpnusers()

**Arguments**

- none

**Effect**

- Get a list of active VPN user

**Data type result (Return)**

- **none**

**Hint: The Macro Library EnertexVPN.lib implements functions to simplify VPN usage.**

*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
```

## FTP-Functions

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. 133 is written 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. This is not to be confused with the periodic outsourcing of the KNX telegramms.

*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

# Webserver Funktions

## 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 151 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

## 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 175 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 176)
- 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 177).
- 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 175 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 176).
- 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 177).
- 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 151 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 151 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.

### 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

### 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

### 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 175 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 176).
- 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

## 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 151 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).

## 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 175 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 176).
- 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 177).
- 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

**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).**

## 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 175 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 176).
- 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

### 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
```

### 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 175 and are referenced with predefined values (u08). Further predefined constants make the choice easier. Table 2 (page 176) 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 177) 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 175 and are referenced with predefined values (u08). Further predefined constants make the choice easier. Table 2 (page 176) 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 177) 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 175 and are referenced with predefined values (u08). Further predefined constants make the choice easier. Table 2 (page 176) 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 177) 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**

Timebufferconfig

- 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 175 and are referenced with predefined values (u08). Further predefined constants make the choice easier. Table 2 (page 176) 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 177) provides an overview over all possibel states.

**Data type result (Return)**

- none

**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.

Timebufferadd

- 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 147 and timebufferread 147.

**Data type result (Return)**

- Values: 0 success, 1 Error: exceeded maximum number of time buffers, 2 Error: time buffer already defined.

**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
- Append a new value to the time buffer with the current time

**Data type result (Return)**

- 0 success, 1 error

### 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

### 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

### 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

## 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.

***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

```
$Webserver$]
webinput(1)[INFO] $Eingabe hier -> Ausgabe in Outputfeldern$
weboutput(2)[SINGLE,ICON]

[EibPC]
inputstring=webinput(1)
if change(inputstring) then weboutput(2,inputstring) endif
```

### Weboutput

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

**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

```
WebServer]
page(1)[$Enertex$,
```

# Configuration of the Webserver

The built-in web server allows the visualization of data and automation processes. For this, you only need a web browser.

## The design of the web server

The integrated web server arranges its elements which have either a predefined single or double length (cf. Figure 17) like a checkerboard pattern. There is basically the option not to use some fields and to insert dividers.

*The design of the buttons is predefined*

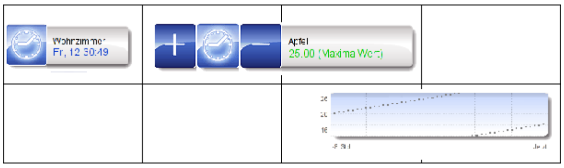


*Figure 17: Scheme of the web server*

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 175) 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 160).

When using multiple pages, you can assign every page to a group. Via the list box of the page navigation (cf. Figure 18), you can select the grouped pages.

*Navigation with multiple pages*

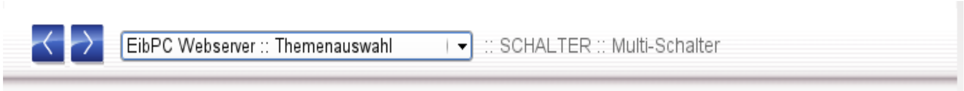

EibPC Webserver :: Themenauswahl
:: SCHALTER :: Multi-Schalter

*Figure 18: 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 18) 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.

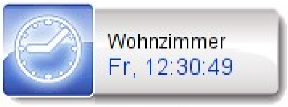
***Example:***

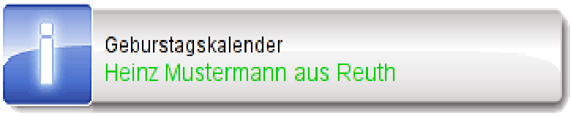
```
[WebServer]
page(1) [$User administration$,$page 1$]
user $Michael$ [PasswordM]
user $Florian$ [PasswordF]
button(1) [INFO] $page 1$

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] $page 5$
```

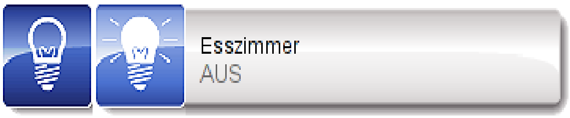
## Elements of the web server

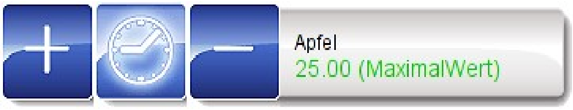
There a two groups of elements for displays, the *Webbutton* and the *Webdisplay*. Of these, the element *button* (sub-group of W*ebbutton*) is the only element which exhibits the single width. At last, there are also design elements to mention: The header and footer (*header*) and the divider (*line*). The following pictures are screen shots of the standard blue design (see also p. 160).

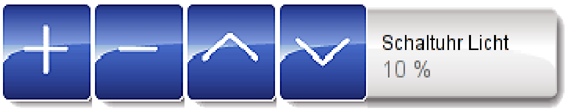
*Every web button can modify a graphic and a line of text dynamically at the runtime of the program.*

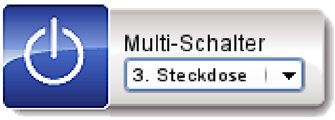
<table>
<tr><th><em>Group</em></th><th><em>Element</em></th><th><em>Description</em></th></tr>
<tr><td><strong>button</strong></td><td></td><td></td></tr>
<tr><td></td><td>button,<br>pbutton</td><td>


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.
</td></tr>
<tr><td></td><td>shifter,<br>pshifter</td><td>


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.
</td></tr>
<tr><td></td><td>shifter,<br>pshifter</td><td>


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.
</td></tr>
<tr><td></td><td>shifter,<br>pshifter</td><td>


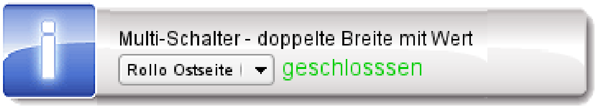
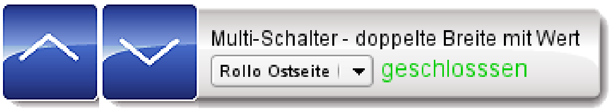
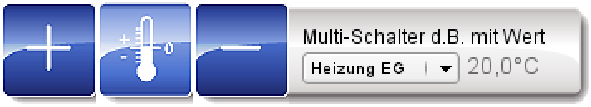
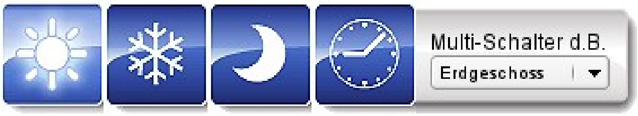
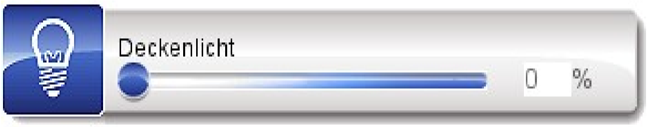
The **middle** graphic can be modified by the user program. The **outer** graphics can be modified only at the configuration. The first line of text is static (only changeable at the configuration). The second line can be modified by the user program.
</td></tr>
<tr><td></td><td>shifter</td><td>


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.
</td></tr>
<tr><td><strong>mbutton</strong></td><td></td><td></td></tr>
<tr><td></td><td>mbutton,<br>mpbutton</td><td>


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 137) and mpbutton (page 140), respectively, is sent to the application program.
</td></tr>
</table>

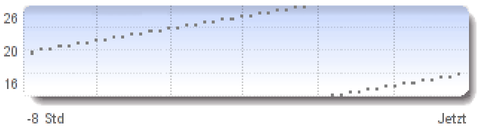
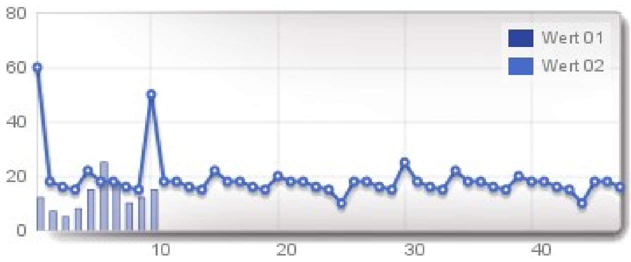
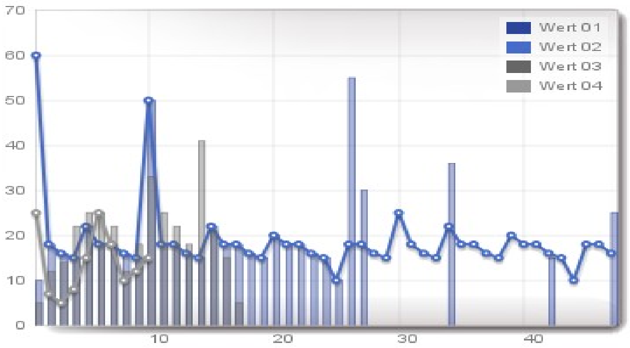
| ***Group*** | ***Element*** | ***Description*** |
|---|---|---|
| | mshifter,<br>mpshifter |   Multi-Schalter - doppelte Breite mit Wert<br>Rollo Ostseite<br>geschlosssen<br>The graphic constituting the actual control panel can be modified by the user program. The first line of text is static (only changeable at the configuration). The second line can be modified by the user program.<br>The listbox can administer a maximum of 4 entries. By operating the listbox, a signal which can be queried by the functions mbutton (page 137) and mpbutton (page 140), respectively, is sent to the application program. |
| | mshifter,<br>mpshifter |   Multi-Schalter - doppelte Breite mit Wert<br>Rollo Ostseite<br>geschlosssen<br>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.<br>The listbox can administer a maximum of 4 entries. By operating the listbox, a signal which can be queried by the functions mbutton (page 137) and mpbutton (page 140), respectively, is sent to the application program. |
| | mshifter,<br>mpshifter |  <br>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.<br>The listbox can administer a maximum of 4 entries. By operating the listbox, a signal which can be queried by the functions mbutton (page 137) and mpbutton (page 140), respectively, is sent to the application program. |
| | mshifter,<br>mpshifter |  <br>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.<br>The listbox can administer a maximum of 4 entries. By operating the listbox, a signal which can be queried by the functions mbutton (page 137) and mpbutton (page 140), respectively, is sent to the application program. |
| | *slider*<br>*pslider* |  <br>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 137) and mpbutton (page 140), respectively. |
| | *eslider*<br>*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 137) and mpbutton (page 140), 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,<br>pchart | <br> <br>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<br>mpchart | <br> <br>The pairs of variates are addressed by the application program via the function mchart. 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<br>mpchart | <br> <br>like above, though double height. |
| | picture | <br> °C<br>Vorhersage für 07/05/10<br>13<br>5<br>Frankfurt<br>15<br>3<br>Nürnberg<br>14<br>4<br>Stuttgart<br>16<br>6<br>München<br>06 May 2010 05:26 GMT<br>The Weather Channel<br>weather.com<br>Anzeige eines externen Bildes z. B. Wetterkarte für Deutschland<br>An external link to a graphic is integrated. The graphic can be left-justified, centered or right-justified. |

| | | |
|---|---|---|
| | picture | °C<br>Vorhersage für 07/05/10<br>10 7 Hamburg<br>11 6 Hannover<br>Berlin<br>10 6 Essen<br>14 6 Leipzig<br>13 5 Frankfurt<br>15 3 Nürnberg<br>14 4 Stuttgart<br>16 6 München<br>06 May 2010 05:26 GMT<br>The Weather Channel<br>weather.com<br>Anzeige eines externen Bildes z. B. Wetterkarte für Deutschland<br>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<br>dframe | Embedding an external website |
| | pLink | Link to an internal page (simple button) |
| | Link | Link to an external page (simple button) |
| **Decorations** | | |
| | line | Überschrift<br>Enforces an empty line with a divider in the web server arrangement. The caption is optional. |
| | header | <br>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 | <br>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 Figure 19 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. Each element must be indexed so that it can be accessed by the user program via the respective functions.

### Element *Compact*

*compact*

- *compact* (STATE)

**Arguments**

- State value of 0 / 1 or ON/OFF

*An example without „compact“ mode*

The web server is built in unit sizes. All elements fit into this grid or are integer multiples thereof. Therefore, when a four-fold height element (e.g., mpchart) is configured next to a simple-height element,

```
[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]
$Multi$
```

*generates a clearance under the 4-way multi button*

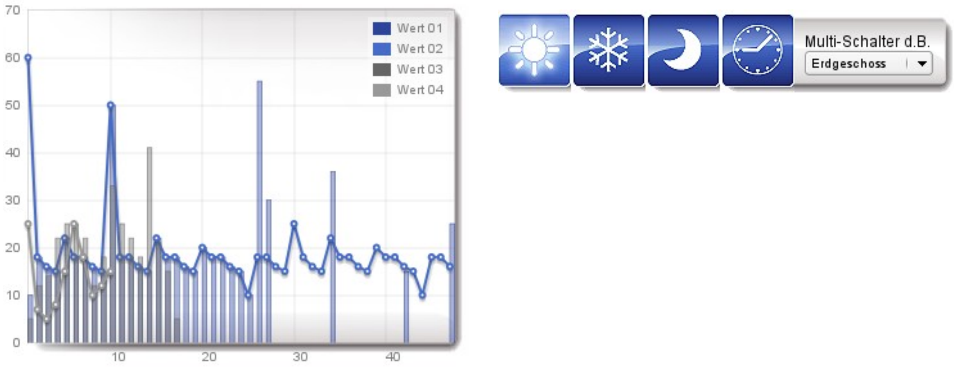


*Figure 19: Clearance*

a clearance is created in the representation as shown in 157.

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 20 shows this "allocation" of the unit sizes (shown in blue) of the above web configuration.

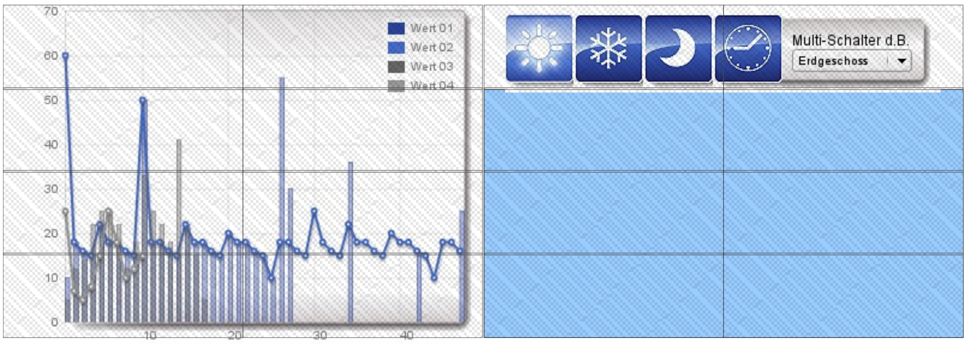


*Figure 20:  Illustration of the unit sizes*

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 20 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.

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.

*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 21 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 21, which is now output by the web server:

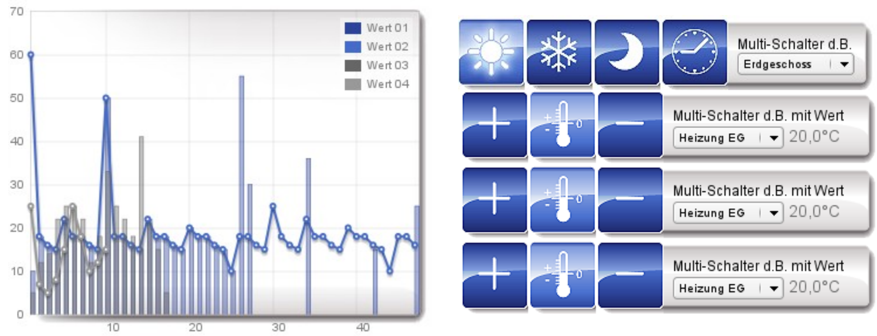


*Figure 21: Compact mode*

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 22 shows again schematically the arrangement of the elements, as is already output in the eibparser

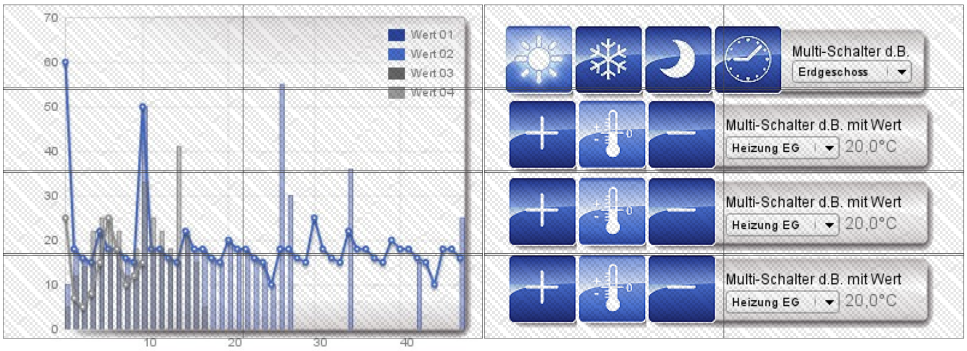


*Figure 22: „compact“ with grid (for illustrative purposes)*

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 23:

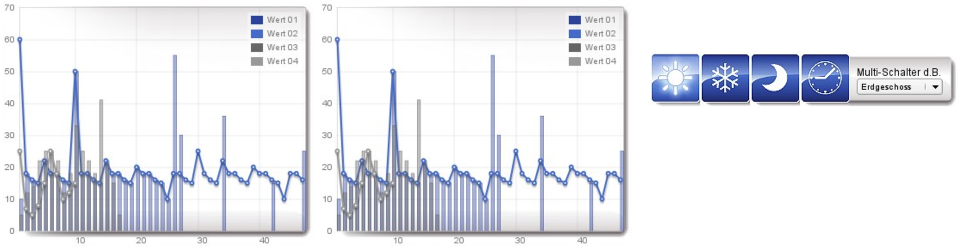


*Figure 23: Representation example for line feed*

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 175).
- 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 136).
- 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 135).

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. 36) 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.

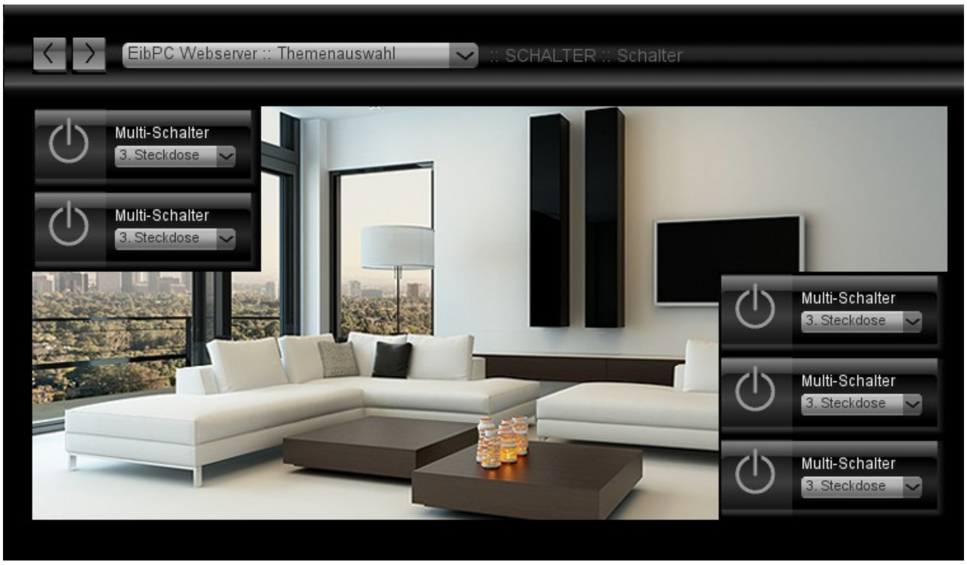


*Figure 1: background graphics*

*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 175).
- 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 136).
- 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 135).
- Switching of the listbox (providing the active listbox element) is arranged by the function Display (Webdisplay) (page 136)

*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 175).
- Text: A static labeling text (first line).

**Access by the user program**

- The image and the text are accessed by the function pdisplay (page 150).
- The element is assigned to only one side
- Activation of the buttons has to be evaluated by the function pbutton (page 150).

*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 175).
- Label: A static labeling text (first line).

**Access by the user program**

- The image and the text are accessed by the function pdisplay (page 150). 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 140).

*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 175).
- 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. 150).
- The operation of the buttons has to be evaluated by the function button (page 135).

*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 175).
- 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. 150).
- The operation of the buttons has to be evaluated by the function pbutton (page 150).

*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 175).
- 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 150). 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 135).

*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 175).
- 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 150). 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 135).

*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 135).
- 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 in the user program. 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().

*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)

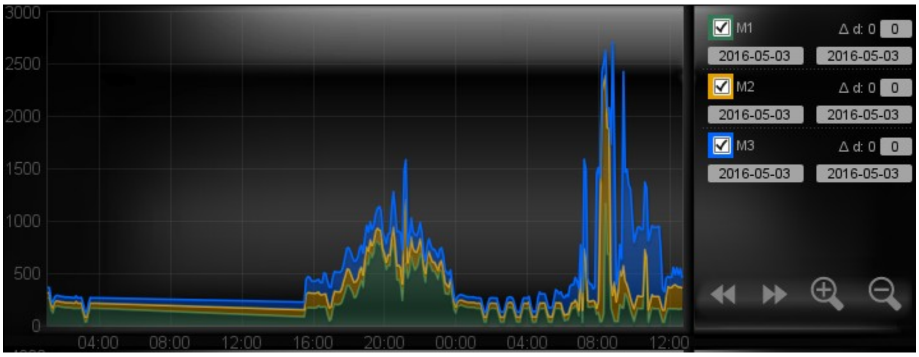
**Arguments**

- 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. 145) 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.145 and timebufferconf p.145 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. 145 and timebufferconf p. 145
- mtimecharts are always global.

*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.

*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 140) in the user program. 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 140.

*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 135).
- 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 175).
- Label: A static labeling text (first line).

**Access by the user program**

- The image and the text are accessed by the function display (page 136).
- Activation of the slider has to be evaluated by the function getslider (page 137).
- Changing the slider level has to be done by the function setslider (page 144).
- Activation of the button has to be evaluated by the function Button (page 135).
- 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 175).
- Label: A static labeling text (first line).

**Access by the user program**

- The image and the text are accessed by the function pdisplay (page 150).
- Activation of the slider has to be evaluated by the function getslider (page 137).
- Changing the slider level has to be done by the function setslider (page 144).
- Activation of the button has to be evaluated by the function Button (page 135).
- 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 175).
- 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 136).
- 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 135).
- The input field can be used to directly manipulate the slider value in the web interface.

*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 175).
- *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. 150) 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. 150) 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. 150) is a 24-bit string.

**Access to the user program**

- The element is addressed via function web input (p. 150).
- 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. 150).
- Elements of weboutput are always global.

*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 175).
- 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 150).
- 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 135).
- 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) 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 175).
- 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 150).
- With the function plink (page 143) 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 175).
- Label: A dynamically labeling text (first line).

**Access to the user program**

- With the function link (page 138) 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.

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 175) 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.

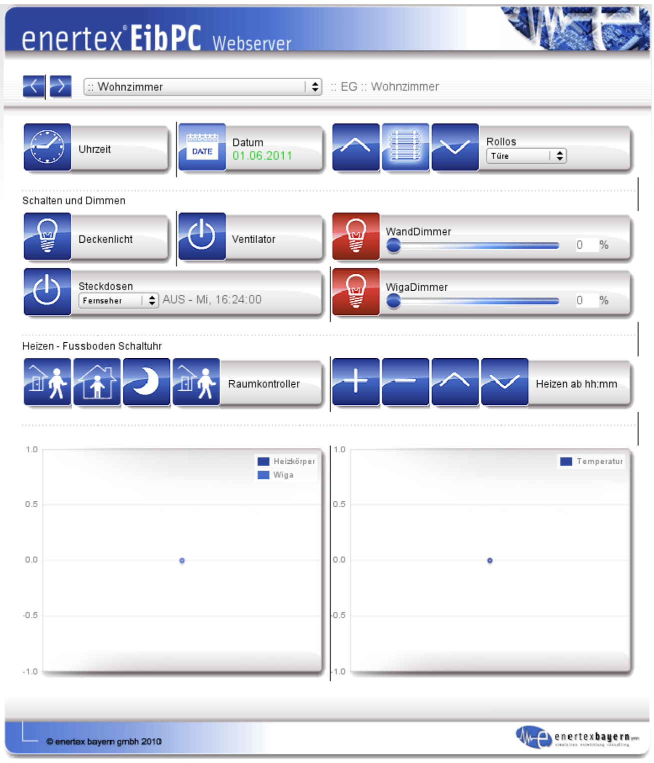


*Figure 2: The web server – multipage-version, blue-Design*

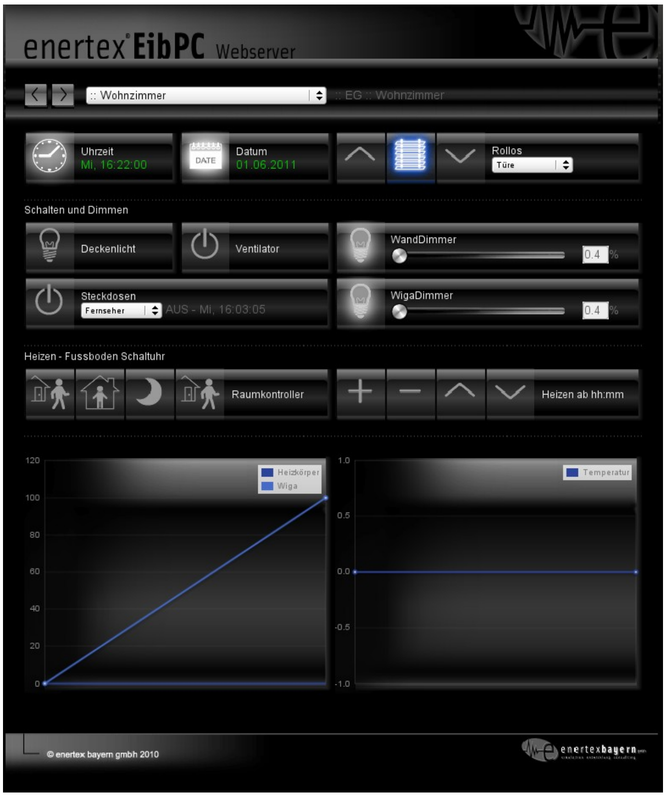


*Figure 3: The web server – multipage-version, black-Design*

*Figure 4: 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 175) can be set by the function Display (Webdisplay) (page 136).

Each icon can have different decorations, symbol variations etc. In the configuration the compiler sets automatically the state1 (INACTIVE). More states (see page 175) can be set with the function Display (Webdisplay) (page 136).

## 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 |
| 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 |
| LUX | 49u08 |
| RAIN | 50u08 |
| KEY | 51u08 |
| WASTE | 52u08 |
| ASK | 53u08 |
| WARN | 54u08 |
| NEAR | 55u08 |
| CAMERA | 56u08 |
| SIGNAL | 57u08 |
| DOOR | 58u08 |
| GARAGE | 59u08 |
| CURTAIN | 60u08 |
| ANGLE | 61u08 |
| ROLLER | 62u08 |
| EMAIL | 63u08 |
| PETS | 64u08 |
| PERSON | 65u08 |
| PHONE | 66u08 |
| TV | 67u08 |
| BEAMER | 68u08 |
| RADIO | 69u08 |
| RECIEVER | 70u08 |
| MEDIA | 71u08 |
| STOVE | 72u08 |
| FRIDGE | 73u08 |
| WASHER | 74u08 |
| DISHWASHER | 75u08 |
| HOLIDAY | 76u08 |
| SLEEP | 77u08 |

*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*.

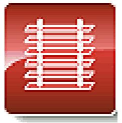
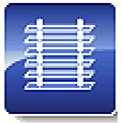
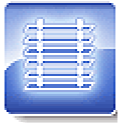
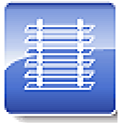
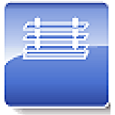
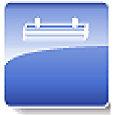
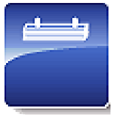
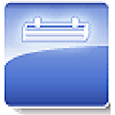
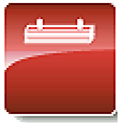
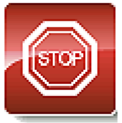
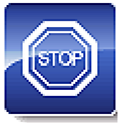
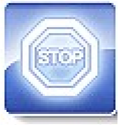
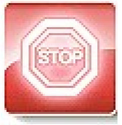
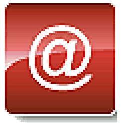
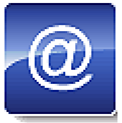
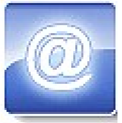
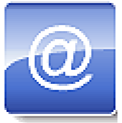
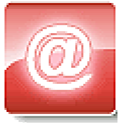
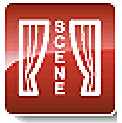
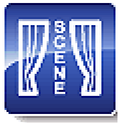
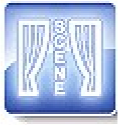
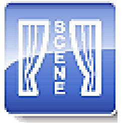
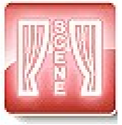
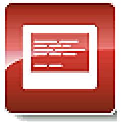
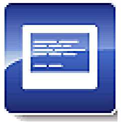
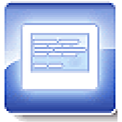
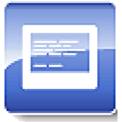
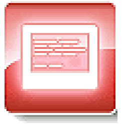
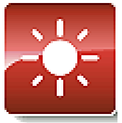
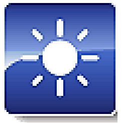
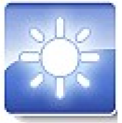
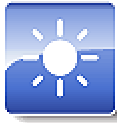
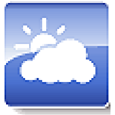
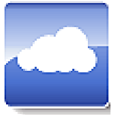
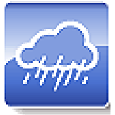
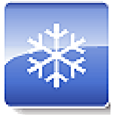
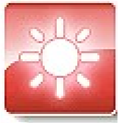
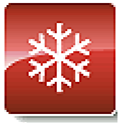
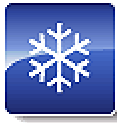
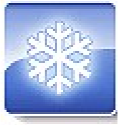
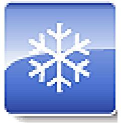
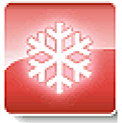
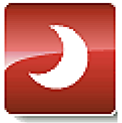
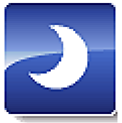
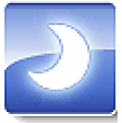
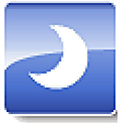
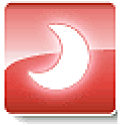
### Behavior of the web server at user interaction

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 in Project Settings (p. 36).

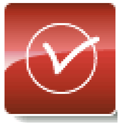
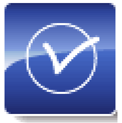
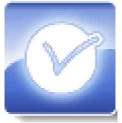
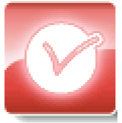
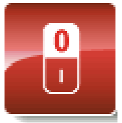
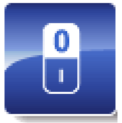
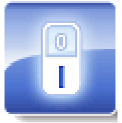
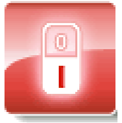
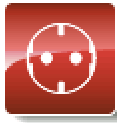
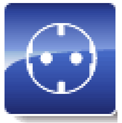
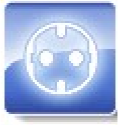
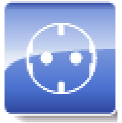
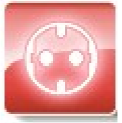
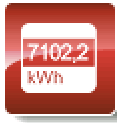
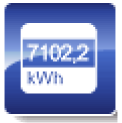
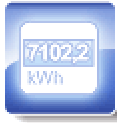
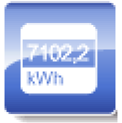
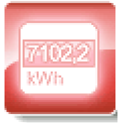
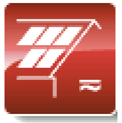
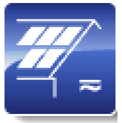
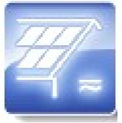
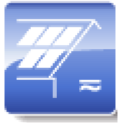
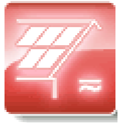
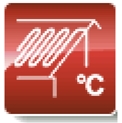
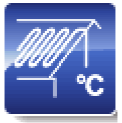
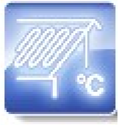
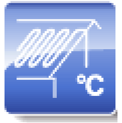
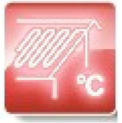
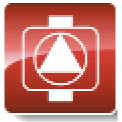
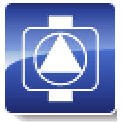
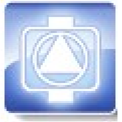
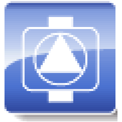
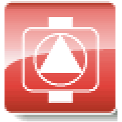
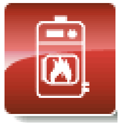
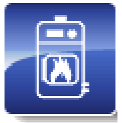
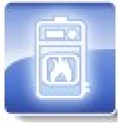
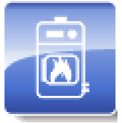
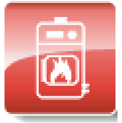
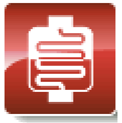
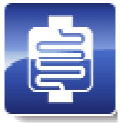
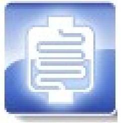
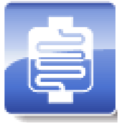
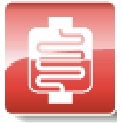
Again, here all symbols in an overview:

| **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 | 🌡 | 🌡 | 🌡 | 🌡 | | | | | | 🌡 |

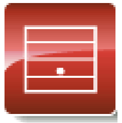
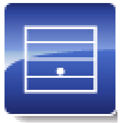
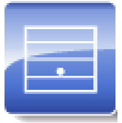
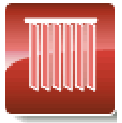
| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **BLIND** | 8u08 |  |  |  |  |  |  |  |  | |  |
| **STOP** | 9u08 |  |  |  | | | | | | |  |
| **MAIL** | 10u08 |  |  |  |  | | | | | |  |
| **SCENES** | 11u08 |  |  |  |  | | | | | |  |
| **MONITOR** | 12u08 |  |  |  |  | | | | | |  |
| **WEATHER** | 13u08 |  |  |  |  |  |  |  |  | |  |
| **ICE** | 14u08 |  |  |  |  | | | | | |  |
| **NIGHT** | 15u08 |  |  |  |  | | | | | |  |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **CLOCK** | 16u08 | 🕒 | 🕒 | 🕒 | 🕒 | | | | | | 🕒 |
| **WIND** | 17u08 | N W O S | N W O S | N W O S | N W O S | | | | | | N W O S |
| **WINDOW** | 18u08 | ▯ | ▯ | ▯ | ▯ | ▯ | ▯ | ▯ | ▯ | | ▯ |
| **DATE** | 19u08 | DATE | DATE | DATE | DATE | | | | | | DATE |
| **PRESENT** | 20u08 | 🏠 | 🏠 | 🏠 | 🏠 | | | | | | 🏠 |
| **ABSENT** | 21u08 | 🚶 | 🚶 | 🚶 | 🚶 | | | | | | 🚶 |
| **REWIND** | 22u08 | ⏪ | ⏪ | ⏪ | | | | | | | ⏪ |
| **PLAY** | 23u08 | ▶ | ▶ | ▶ | | | | | | | ▶ |
| **PAUSE** | 24u08 | ⏸ | ⏸ | ⏸ | | | | | | | ⏸ |

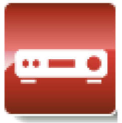
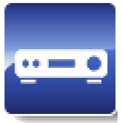
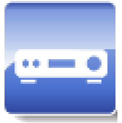
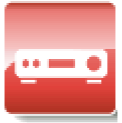
| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **FORWARD** | 25u08 | ⏩ | ⏩ | ⏩ | | | | | | | ⏩ |
| **RECORD** | 26u08 | ⏺ | ⏺ | ⏺ | | | | | | | ⏺ |
| **STOP** | 27u08 | ⏹ | ⏹ | ⏹ | | | | | | | ⏹ |
| **EJECT** | 28u08 | ⏏ | ⏏ | ⏏ | | | | | | | ⏏ |
| **NEXT** | 29u08 | ⏭ | ⏭ | ⏭ | | | | | | | ⏭ |
| **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 | 🔑 | 🔑 | 🔑 | 🔑 | | | | | | 🔑 |

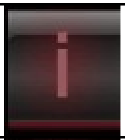
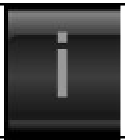
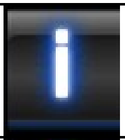
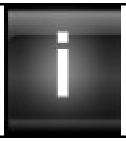
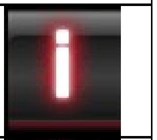
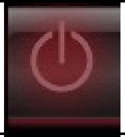
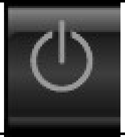
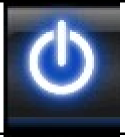
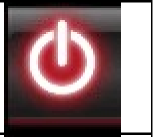
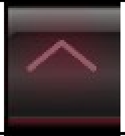
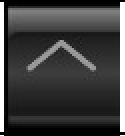
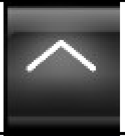
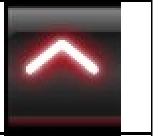
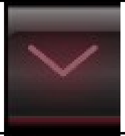
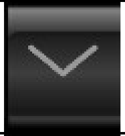
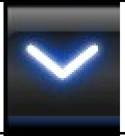
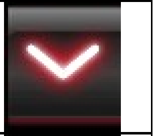
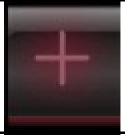
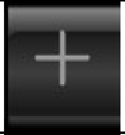
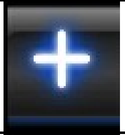
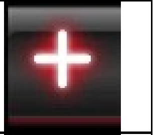
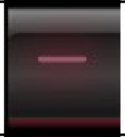
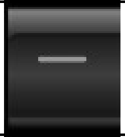
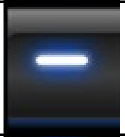
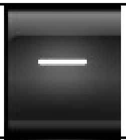
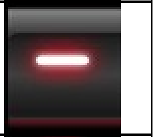
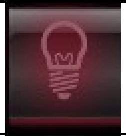
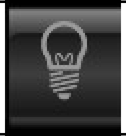
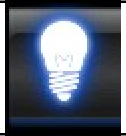
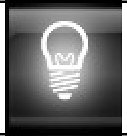
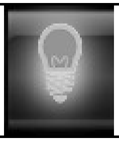
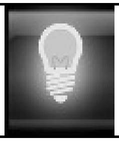
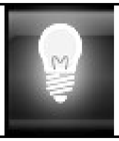
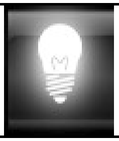
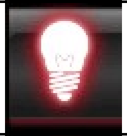
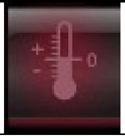
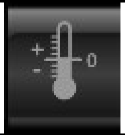
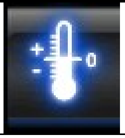
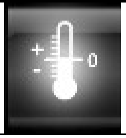
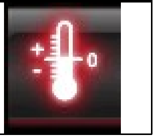
| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **WASTE** | 52u08 | | | | | | | | | | |
| **ASK** | 53u08 | | | | | | | | | | |
| **WARN** | 54u08 | | | | | | | | | | |
| **NEAR** | 55u08 | | | | | | | | | | |
| **CAMERA** | 56u08 | | | | | | | | | | |
| **SIGNAL** | 57u08 | | | | | | | | | | |
| **DOOR** | 58u08 | | | | | | | | | | |
| **GARAGE** | 59u08 |  |  | |  | | | | | | |
| **CURTAIN** | 60u08 |  | | | | | | | | | |

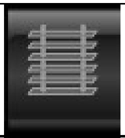
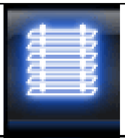
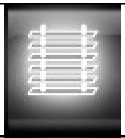
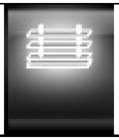
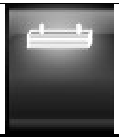
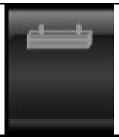
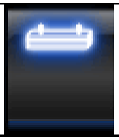
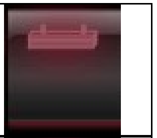
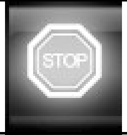
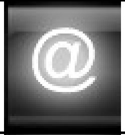
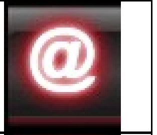
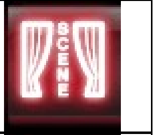
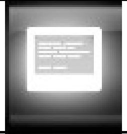
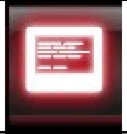
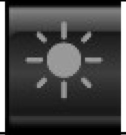
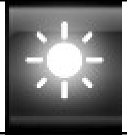
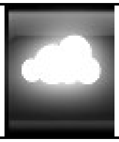
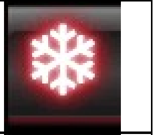
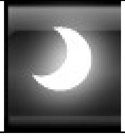
| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **ANGLE** | 61u08 | ↶ | ↶ | ↶ | ↶ | ☰ | ↷ | ↷ | ↷ | | ↷ |
| **ROLLER** | 62u08 | ▤ | ▤ | ▤ | ▤ | ▤ | ▤ | ▤ | ▤ | | ▤ |
| **EMAIL** | 63u08 | ✉ | ✉ | ✉ | ✉ MAIL IN | | | | | | |
| **PETS** | 64u08 | 🚫🐱 | 🚫🐱 | 🚫🐱 | 🚫🐱 | 🐱 | | | | | 🚫🐱 |
| **PHONE** | 65u08 | ☎ | ☎ | ☎ | ☎ | | | | | | ☎ |
| **PERSON** | 66u08 | 👪 | 👪 | 👪 | 👪 | | | | | | 👪 |
| **TV** | 67u08 | 📺 | 📺 | 📺 | 📺 | | | | | | 📺 |
| **BEAMER** | 68u08 | 📽 | 📽 | 📽 | 📽 | | | | | | 📽 |
| **RADIO** | 69u08 | 📻 | 📻 | 📻 | 📻 | | | | | | 📻 |

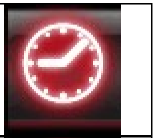
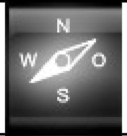
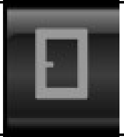
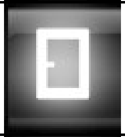
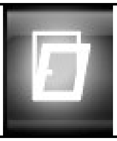
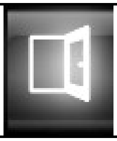
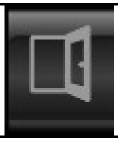
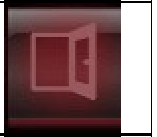
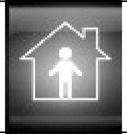
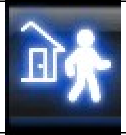
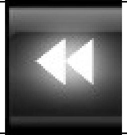
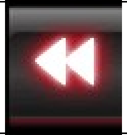
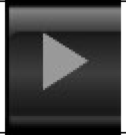
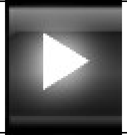
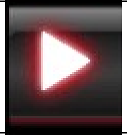
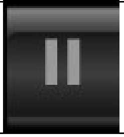
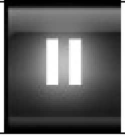
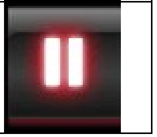
| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **RECIEVER** | 70u08 |  |  | |  | | | | | |  |
| **MEDIA** | 71u08 | | | | | | | | | | |
| **STOVE** | 72u08 | | | | | | | | | | |
| **FRIDGE** | 73u08 | | | | | | | | | | |
| **WASHER** | 74u08 | | | | | | | | | | |
| **DISHWASHER** | 75u08 | | | | | | | | | | |
| **HOLIDAY** | 76u08 | | | | | | | | | | |
| **SLEEP** | 77u08 | | | | | | | | | | |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **UPDATE** | 78u08 | | | | | | | | | | |

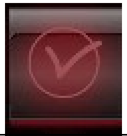
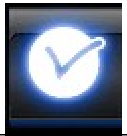
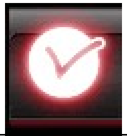
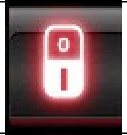
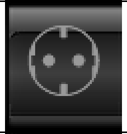
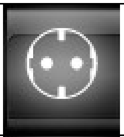
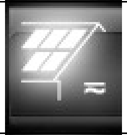
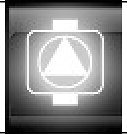
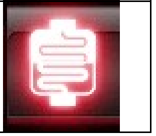
*Table 5: Overview icons – blue design*

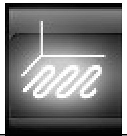
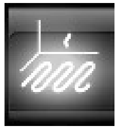
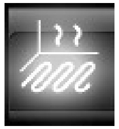
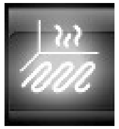
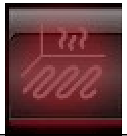
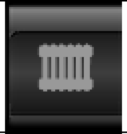
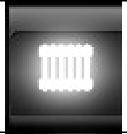
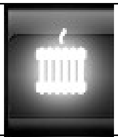
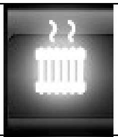
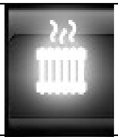
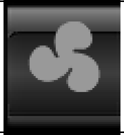
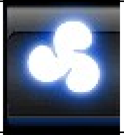
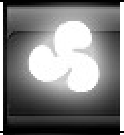
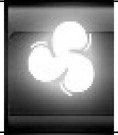
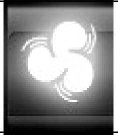
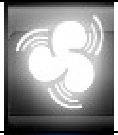
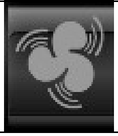
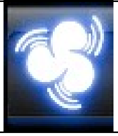
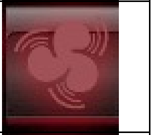
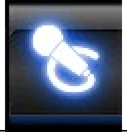
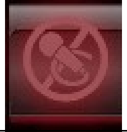
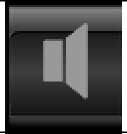
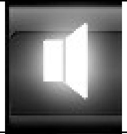
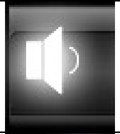
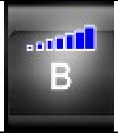
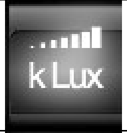
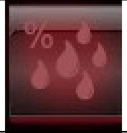
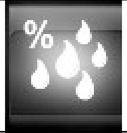
| Symbol | Index | DARKRED<br>0u08 | INACTIVE<br>1u08 | ACTIVE<br>2u08 | DISPLAY<br>3u08 | STATE4<br>4u08 | STATE5<br>5u08 | STATE6<br>6u08 | STATE7<br>7u08 | STATE8<br>8u08 | BRIGHTRED 9u08 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **INFO** | 0u08 |  |  |  |  | | | | | |  |
| **SWITCH** | 1u08 |  |  |  | | | | | | |  |
| **UP** | 2u08 |  |  |  | | | | | | |  |
| **DOWN** | 3u08 |  |  |  | | | | | | |  |
| **PLUS** | 4u08 |  |  |  | | | | | | |  |
| **MINUS** | 5u08 |  |  |  |  | | | | | |  |
| **LIGHT** | 6u08 |  |  |  |  |  |  |  |  | |  |
| **TEMPERATURE** | 7u08 |  |  |  |  | | | | | |  |

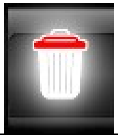
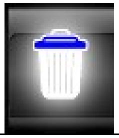
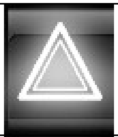
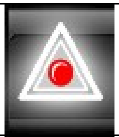
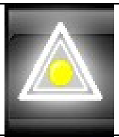
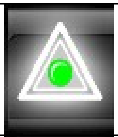
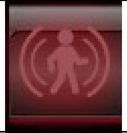
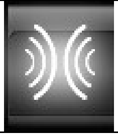
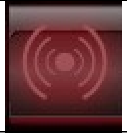
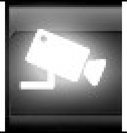
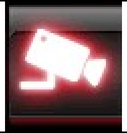
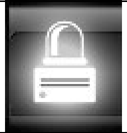
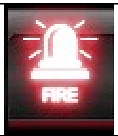
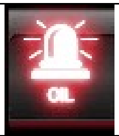
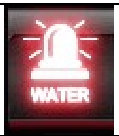
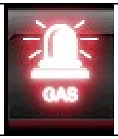
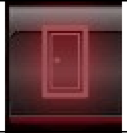
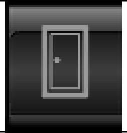
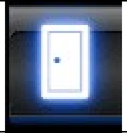
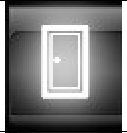
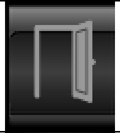
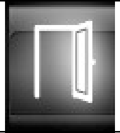
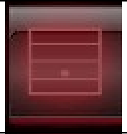
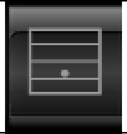
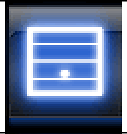
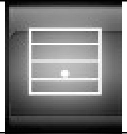
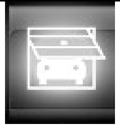
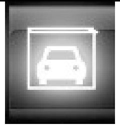
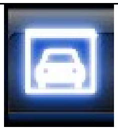
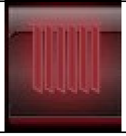
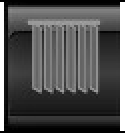
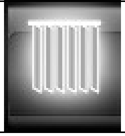
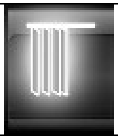
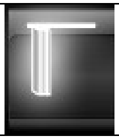
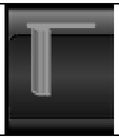
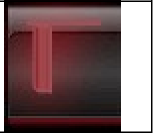
| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **BLIND** | 8u08 | |  |  |  |  |  |  |  | |  |
| **STOP** | 9u08 | | | |  | | | | | | |
| **MAIL** | 10u08 | | | |  | | | | | |  |
| **SCENES** | 11u08 | | | | | | | | | |  |
| **MONITOR** | 12u08 | | | |  | | | | | |  |
| **WEATHER** | 13u08 | |  | |  | |  | | | | |
| **ICE** | 14u08 | | | | | | | | | |  |
| **NIGHT** | 15u08 | | | |  | | | | | | |

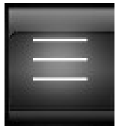
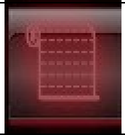
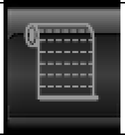
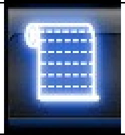
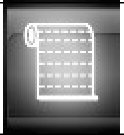
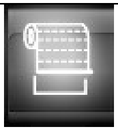
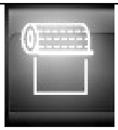
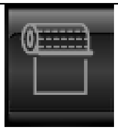
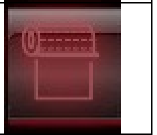
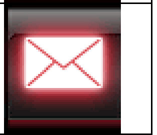
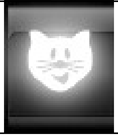
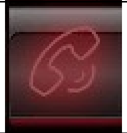
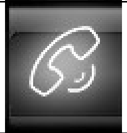
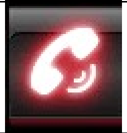
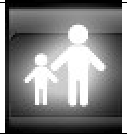
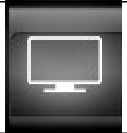
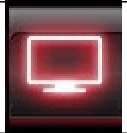
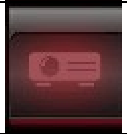
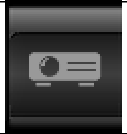
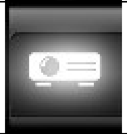
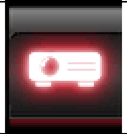
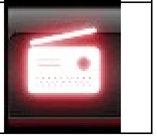
| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **CLOCK** | 16u08 | | | | | | | | | |  |
| **WIND** | 17u08 | | | |  | | | | | | |
| **WINDOW** | 18u08 | |  | |  |  |  |  | | |  |
| **DATE** | 19u08 | | | | | | | | | | |
| **PRESENT** | 20u08 | | | |  | | | | | | |
| **ABSENT** | 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 | | | | | | | |  | | |
| **LUX** | 49u08 | | | |  | | | | | | |
| **RAIN** | 50u08 |  | | |  | | | | | | |
| **KEY** | 51u08 | | | | | | | | | | |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **WASTE** | 52u08 | | | | |  | |  | | | |
| **ASK** | 53u08 | | | | | | | | | | |
| **WARN** | 54u08 | | | | |  |  |  |  | | |
| **NEAR** | 55u08 |  | | | |  | | | | |  |
| **CAMERA** | 56u08 | | | |  | | | | | |  |
| **SIGNAL** | 57u08 | | | |  |  |  |  |  | | |
| **DOOR** | 58u08 |  |  |  |  |  | |  | | | |
| **GARAGE** | 59u08 |  |  |  |  |  |  | |  | | |
| **CURTAIN** | 60u08 |  |  | |  |  |  |  | | |  |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **ANGLE** | 61u08 | | | | |  | | | | | |
| **ROLLER** | 62u08 |  |  |  |  |  |  |  | | |  |
| **EMAIL** | 63u08 | | | | | | | | | |  |
| **PETS** | 64u08 | | | | |  | | | | | |
| **PHONE** | 65u08 |  | | |  | | | | | |  |
| **PERSON** | 66u08 | | | |  | | | | | | |
| **TV** | 67u08 | | | |  | | | | | |  |
| **BEAMER** | 68u08 |  |  | |  | | | | | |  |
| **RADIO** | 69u08 | | | | | | | | | |  |

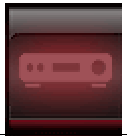
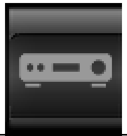
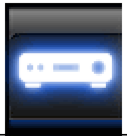
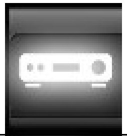
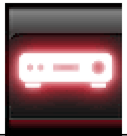
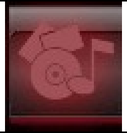
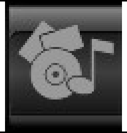
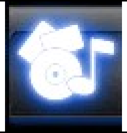
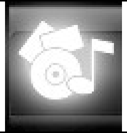
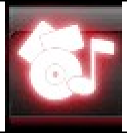
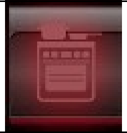
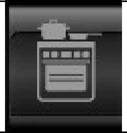
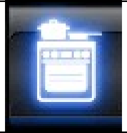
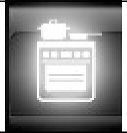
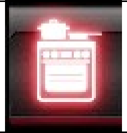
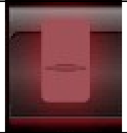
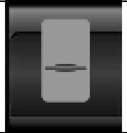
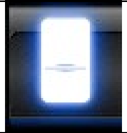
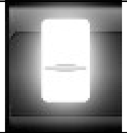
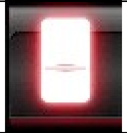
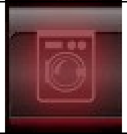
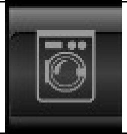
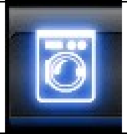
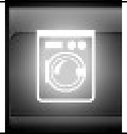
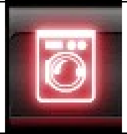
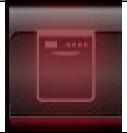
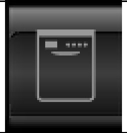
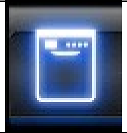
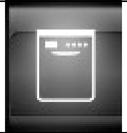
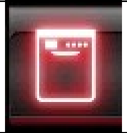
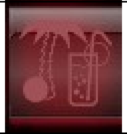
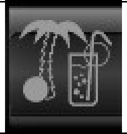
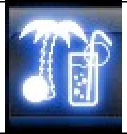
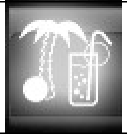
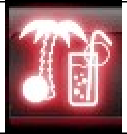
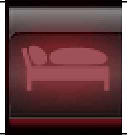
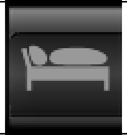
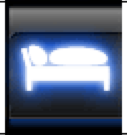
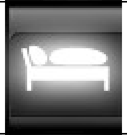
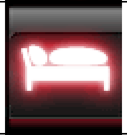
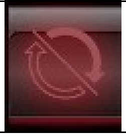
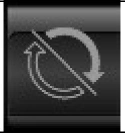
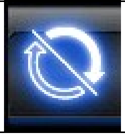
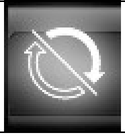
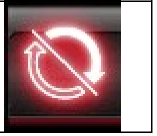
| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **RECIEVER** | 70u08 |  |  |  |  | | | | | |  |
| **MEDIA** | 71u08 |  |  |  |  | | | | | |  |
| **STOVE** | 72u08 |  |  |  |  | | | | | |  |
| **FRIDGE** | 73u08 |  |  |  |  | | | | | |  |
| **WASHER** | 74u08 |  |  |  |  | | | | | |  |
| **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.

## 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*

*... ends with :end*

```
: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
```

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 generate the description by yourself with “:info”.*

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.

*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:

```
:begin cosh(x)
:info Calculates the cosh-function
:return (exp(x)-exp(-x))/2f32
:end
```

*Compact design*

Additionally the return command could be left (due to compatibility reasons to older macros), so the code

```
:begin cosh(x)
:info Calculates the cosh-function
(exp(x)-exp(-x))/2f32
:end
```

*Compact design*

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.

```
:begin cosh(x)
:info Calculates the cosh-function
(exp(x)-exp(-x))/2f32

:end
```

*empty line before :end means no return value (if :return is not defined)*

Once defined in a macro-lib and added to the [MacroLibs] section, the macro can be used as a built-in function:

```
[EibPC]
MyVar=cosh(2.3f32)
MyVar2=cosh(cosh('1/3/2'f32)) +cosh('1/3/3'f32) + 32f32
```

*Use it as built-in*

## Online debugging at runtime

*Sending a string with CR to a UDP client*

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:

*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.

```
x=3
If x>5 then {
    x=x*2;
   vmDebugUDP($x ist nun $+convert(x,$$));
} endif
```

*Efficient for inactive #define of DEBUG*

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:

```
x=3
If x>5 then {
   vmDebugUDP($x ist nun $+convert(x,$$));
} endif
```

*Inefficient for inactive #define of DEBUG - if query that is used only for debugging.*

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:

```
    x=3
    #ifdef DEBUG
    If x>5 then {
       vmDebugUDP($x ist nun $+convert(x,$$));
    } endif
#endif
```

*... then rather this way..*

## 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:<br>Var1 = Base<br>Var2 = Argument |
| **pow(Var1, Var2)** | Power:<br>Var1= Base<br>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<br>msTime: Time in ms (<$2^{64}$) |
| **afterc(Control, msTime, xT)** | Control: Binary value<br>msTime: Time in ms (<$2^{64}$)<br>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.<br>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.<br>mm= Minutes (0...255)<br>ss = Seconds (0..59) |
| **wtime(dd,hh,mm,ss)** | Weekly time switch:<br>ss: Seconds (0..59 )<br>mm: Minutes (0..59 )<br>hh: Hours (0..23)<br>dd: Day (0=Sunday, 6=Saturday) |

| | |
|---|---|
| **htime(hh,mm,ss)** | Daily time switch:<br>ss: Seconds (0..59 )<br>mm: Minutes (0..59 )<br>hh: Hours (0..23) |
| **mtime(mm,ss)** | Hourly time switch:<br>ss: Seconds (0..59 )<br>mm: Minutes (0..59 ) |
| **stime(ss)** | Minute time switch:<br>ss: seconds (0..59 ) |
| **cwtime(hh,mm,ss,dd)** | Weekly comparator time switch:<br>ss: Seconds (0..59 )<br>mm: Minutes (0..59 )<br>hh: Hours (0..23)<br>dd: Day (0=Sunday, 6=Saturday) |
| **chtime(hh,mm,ss)** | Daily comparator time switch:<br>ss: Seconds (0..59 )<br>mm: Minutes (0..59 )<br>hh: Hours (0..23) |
| **cmtime(mm,ss)** | Hourly comparator time switch:<br>ss: Seconds (0..59 )<br>mm: Minutes (0..59 ) |
| **cstime(ss)** | Minute comparator time switch:<br>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:<br>yyy: Years (0..255) since the year 2000<br>mm: Month (1=January, 12= December)<br>dd: Day (1..31) |

| | |
|---|---|
| **month(mm,dd)** | Monthly comparison:<br>mm: Month (1=January, 12= December)<br>dd: Day (1..31) |
| **day(dd)** | Daily comparison:<br>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 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 $blue$.<br>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').<br>Group: Assignment of the page to a group.<br>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].<br>The link (incl. path and leading http://) is optional.<br>The URL can access an extern resource. In this case the number must be set to 2.<br>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].<br>The link (incl. path and leading http://) is optional.<br>The URL can access an extern resource. In this case the number must be set to 2.<br>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].<br>Image: A value between 0 and 99.<br>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].<br>Image: A value between 0 and 99.<br>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].<br>Text1, Text2, .. Text254: label texts for mbutton. The second and following elements are optional.<br>Image: A value between 0 and 99. To arrange the application more clearly, constants have been predefined (page 175).<br>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].<br>Text1, Text2, .. Text254: label texts for mbutton. The second and following elements are optional.<br>Image: A value between 0 and 99. To arrange the application more clearly, constants have been predefined (page 175).<br>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].<br>Image1 to Image4: A value between 0 and 99.<br>Image2 to Image4 are optional.<br>If only three images are defined, the element has only three buttons etc..<br>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].<br>Image1 to Image4: A value between 0 and 99.<br>Image2 to Image4 are optional.<br>If only three images are defined, the element has only three buttons etc..<br>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].<br>Image1 to Image4: A value between 0 and 99.<br>Image2 to Image4 are optional.<br>If only three images are defined, the element has only three buttons etc.<br>Text1, Text2, .. Text254: labels for the mshifter. The second and following elements are optional.<br>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].<br>Image1 to Image4: A value between 0 and 99.<br>Image2 to Image4 are optional.<br>If only three images are defined, the element has only three buttons etc.<br>Text1, Text2, .. Text254: labels for the mshifter. The second and following elements are optional.<br>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.<br>$Y0$, $Y1$,$Y2$: 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.<br>$Y0$, $Y1$,$Y2$: 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.<br>Height: Value 0 or 1 (or constant SINGLE and DOUBLE)<br>Type: Value 8 (or constant XY) for plots<br>Type: Value 9 (or constant SXY) for plots with sorted X-Y sets (well suited for time-based plots)<br>$Label1$ .. $Label2$ Legend of the graph<br>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.<br>Height: Value 0 or 1 (or constant SINGLE and DOUBLE)<br>Type: Value 8 (or constant XY) for plots<br>Type: Value 9 (or constant SXY) for plots with sorted X-Y sets (well suited for time-based plots)<br>$Label1$ .. $Label2$ Legend of the graph<br>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].<br>Image: A value between 0 and 99. To arrange the application more clearly, constants have been predefined (page 175).<br>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].<br>Image: A value between 0 and 99. To arrange the application more clearly, constants have been predefined (page 175).<br>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].<br>Image: A value between 0 and 99. To arrange the application more clearly, constants have been predefined (page 175).<br>Min: slider minimum value<br>Increment: slider increment<br>Max: slider maximum value<br>Description: A static labeling text (first line).<br>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].<br>Image: A value between 0 and 99. To arrange the application more clearly, constants have been predefined (page 175).<br>Min: slider minimum value<br>Increment: slider increment<br>Max: slider maximum value<br>Description: A static labeling text (first line).<br>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.<br>Height:Value 0 or 1 (or constant SINGLE and DOUBLE)<br>Typ: Value 0,1,2 (or LEFTGRAF, CENTERGRAF, ZOOMGRAF): left aligned, centered or streched embedding of the image<br>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 175).
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.<br>One mchart displays four different graphs. index (0,1,2,3) defines the graph to be addressed.<br>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.<br>One mpchart displays four different graphs. index (0,1,2,3) defines the graph to be addressed.<br>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. |
| **timebufferadd(ChartBufferID, Daten)** | |
| **timebufferconfig(ChartBufferID, MemTyp, Länge, DataTyp)** | |
| **timebufferstore(ChartBufferID)** | |
| **timebufferread(ChartBufferID)** | |
| **timebuffersize(ChartBufferID)** | |
| **mtimechartpos(TimeChartID,ChartIdx,ChartBuffer,StartPos,EndPos)** | |
| **mtimechart(TimeChartID,ChartIdx,ChartBuffer,StartZeit,EndZeit)** | |

**mtimechartupdate(TimeChartID,ChartIdx,ChartBuffer,Strategy)**

**webinput**

**weboutput**

**Synohr MultiSense Funktionen**

**synohrmaster**

## 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<br>msTime: Time in ms ($<2^{64}$) |
| **afterc(Control, msTime, xT)** | Control: Binary value<br>msTime: Time in ms ($<2^{64}$)<br>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.<br>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].<br>Image: A value between 0 and 99.<br>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.<br>X1, X2: The labeling of the x-axis.<br>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.<br>$Y0$, $Y1$,$Y2$: Labeling of the y-axis. |
| **chtime(hh,mm,ss)** | Daily comparator time switch:<br>ss: Seconds (0..59 )<br>mm: Minutes (0..59 )<br>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.

## Keywords - reference - alphanumeric order

| | |
|---|---|
| **elevation()** | This function cyclically (time frame: 5 minutes) calculates the elevation angle of the sun in degrees. |
| **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].<br>Image: A value between 0 and 99. To arrange the application more clearly, constants have been predefined (page 175).<br>Min: slider minimum value<br>Increment: slider increment<br>Max: slider maximum value<br>Description: A static labeling text (first line).<br>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.<br>String2: Character string to be searched for.<br>Pos1: Ignore the first pos1 incidences of the character string to be searched for.<br>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<br>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].<br>The link (incl. path and leading http://) is optional.<br>The URL can access an extern resource. In this case the number must be set to 2.<br>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 |

## Keywords - reference - alphanumeric order

**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.

**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 175).
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 175).
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 175).
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 175).
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 175).
Label: A static labeling text (first line).

## Keywords - reference - alphanumeric order

| | |
|---|---|
| **split(String, Pos1, Pos2)** | String: Character string a character string shall be extracted from.<br>Pos1: Position of the first character of the character string to be extracted (0...1399u16).<br>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.<br>The variable string must be of data type c1400.<br>Return value: The character string extracted from String. |
| **sqrt(Var1)** | Square root |
| **startvpn()** | Stats the VPN Service |
| **stime(ss)** | Minute time switch:<br>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 |

## Keywords - reference - alphanumeric order

**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.

**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)

# Code

Enertex® EibPC² and Enertex® EibStudio 4 use software distributed under various different licenses. If the respective license demands it, the source code is provided on request. Enertex® Bayern GmbH may charge a handling fee to fulfill the request.

Enertex® EibPC

Operating system: Debian Linux: 2.6.24-rt1.

Enertex® EibPC²

Operating system: Debian Linux 9.6: 4.14.16.

Enertex® EibStudio 4

Open **Help → Licenses** to open the list of libraries.

# Questions and answers

## Events

| 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:<br>Datatypes of parameters are not the same: >Var1+Var2< ! | Var3=convert(Var1,Var2) + Var2 |
| Syntax error in line:[17]<br>>if (("EntireKitchen-1/1/9"==On) and wtime(23,00,00,00)) <<br>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 ' \\'.<br>if ........ and \\<br>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.<br>The predefined variables are listed in the Enertex® EibStudio in the right section of the window.<br>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]<br>>a=4,6e1f32<<br>Valid until position: STOP--> ,6e1f32 | As a decimal point, always "." has to be used. |
| Syntax error in line:[21]<br>>"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]<br>>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 ";".<br>if(a=EIN) then write(b=EIN); write(c=AUS) endif |
| Syntax error in line:[26]<br>>write(on,ON)<<br>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 31 (> Firmware 4.000, EibStudio 4.000)

- Rewritten for Enertex® EibPC² and Enertex® EibStudio 4