Manual Enertex® EibPC²

Prerequisites

Enertex® EibPC²:	**Firmware 5.000 or newer**
Enertex® EibStudio:	**Version 5.000 or newer**

Contents

Thank you for buying an Enertex® 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 EibStudio. The EibStudio and all independently running components may only be used for the 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

Support-Export

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

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.

Telephone

You can also use our support via telephone 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 http://videos.eibpc.com/

Updates

Please find updates for the EibPC² on our website www.eibpc.com.

Enertex® EibPC²

Overview

Figure 1: 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 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 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

Data logging

Logging of up to 500,000 telegrams is possible

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)

Software

Memory The EibPC stores all bus telegrams. Up to 500,000 frames are held in a ring buffer, even if no PC is connected to the 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 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 EibStudio allows the cyclic saving of (possibly filtered) telegrams in files.

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

By means of the EibStudio as a configuration program a home automation is provided via the LAN interface of the EibPC to a Windows®, Mac® OS X or Linux® PC. This ensures that the 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 EibStudio imports the addresses and settings of the ets. It can also be used entirely without ETS import.

Commissioning

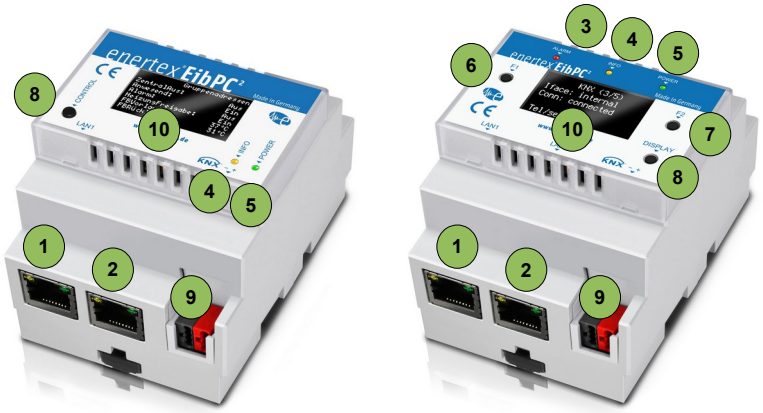


Figure 2: Connectors and Control Elements – one-button and three-button device versions

Connectors and Control Elements

See 2 for the connectors and control elements:

1. LAN1
2. LAN2
3. Alarm-LED (red)
4. Info-LED (orange)
5. Power-LED (green)
6. F1-button
7. F2-button
8. Control-button (one-button version) / Display-button (three-button version)
9. KNX
10. Display

The 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).

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

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

We recommend one of the following:

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

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

Installation

Integrated KNX interface

3

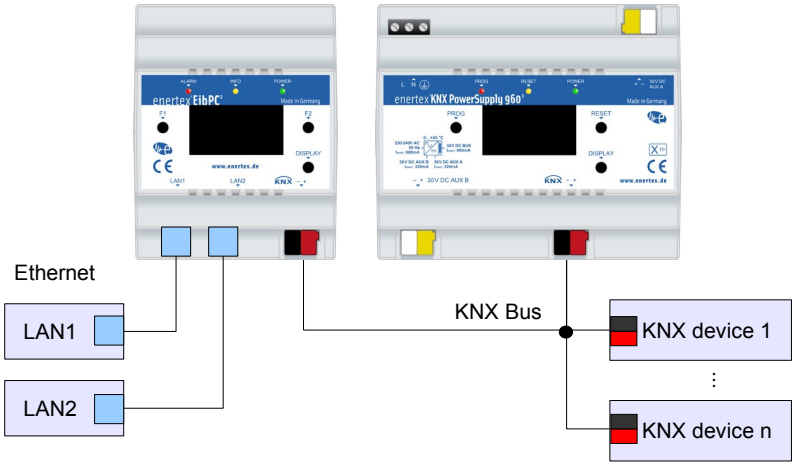


Figure 4: Connection of the Enertex EibPC² to the KNX Bus

4 shows how the installation of the EibPC². Figure

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 EibPC² with a (KNX) power supply.

Integrated Ethernet-Switch

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

When the 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 5.

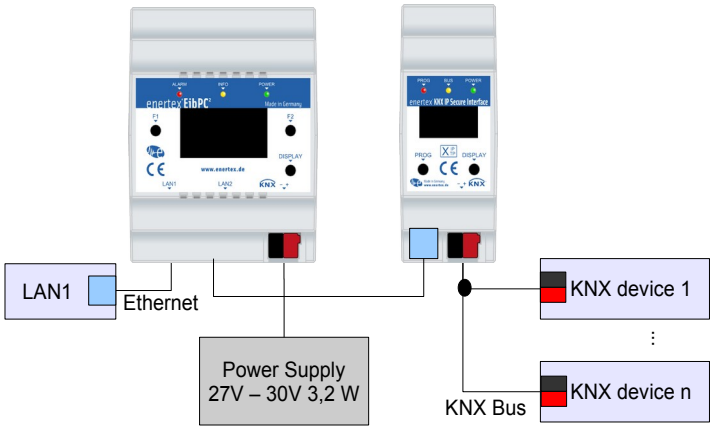


Figure 5: Using a dedicated KNXnet/IP interface

Device Start

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

One-button version:

1. Info- and Power-LED are both on during system boot.
2. After system boot, the Power-LED starts to blink.
3. ~2 min after power-on, 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 address. The display stays on for 30 s. By pressing the Control button, the display can be reactivated.
6. Normal operation. The Power-LED blinks continuously, the Info-LED blinks when KNX telegrams are received.

Three-button version:

1. After power-on, all LEDs are on with medium brightness.
2. After ~5 s, only the Power-LED is on with full brightness.
3. After system boot, the Power-LED starts to blink.
4. ~2 min after power on, the Info-LED blinks once every second. A factory reset can be issued (see below).
5. Initialize bus connection. The Info-LED flickers.
6. After booting, the display shows system information, including the IP address. The display stays on for 30 s. By pressing the Display-button, the display can be reactivated.
7. Normal operation. The Power-LED blinks continuously, the Info-LED blinks when KNX telegrams are received.

Firmware Update

Firmware updates are installed using 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 Display button/Control button for at least 20 s. The display shows a confirmation, and the Info-LED blinks. The device is restarted.

EibStudio 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.

EibStudio or above is used as programming and configuration tool.

EibStudio has to be uncompressed. No installation procedure is required.

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

Open EibStudio

On first start, EibStudio shows a configuration dialog to set the Projects Directory (p. 12).

Project directory

EibStudio does not change or delete files outside of the projects directory and the Configuration Directory (p. 12).

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

You can change the projects directory in the Settings (p. 12). 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

EibStudio 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 EibPC

Bus connection

To start the first program, configure the connection to the 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.

EibStudio

This section introduces the basic structure of EibStudio 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

EibStudio does not require any installation procedure (like 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 EibStudio 3 or EibStudio. A project manages all information required by one 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 does not change or delete files outside of the projects directory and the Configuration Directory (p. 12). When a project is imported, the project files are copied here.

The projects directory can be changed in the **Settings** (p. 12).

Import EibStudio 3 Project

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 EibStudio 3 project, click the respective button and select the main program. In the dialog, select the directory of the 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 EibStudio 3 libraries are already integrated into EibStudio. 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

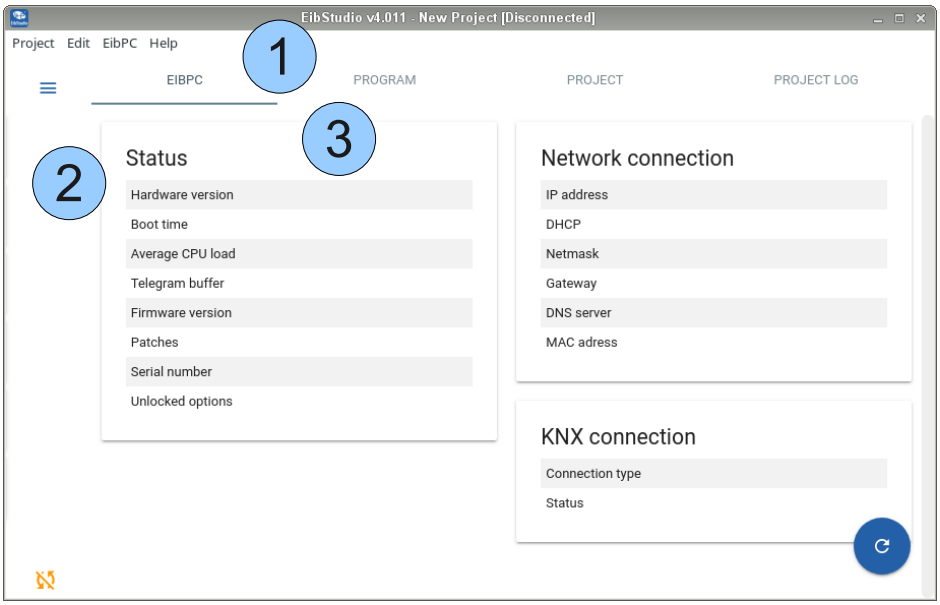


Figure 1: Overview

1 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).

Extended Navigation

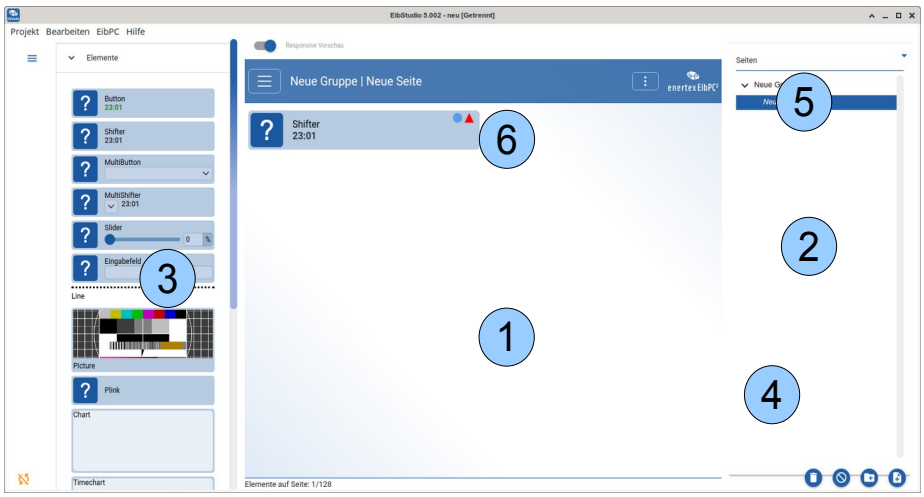


Figure 2: Extended Navigation

The following refers to 2. **Logic**, **Visu** and **Expert** use additional navigation elements.

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 to/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

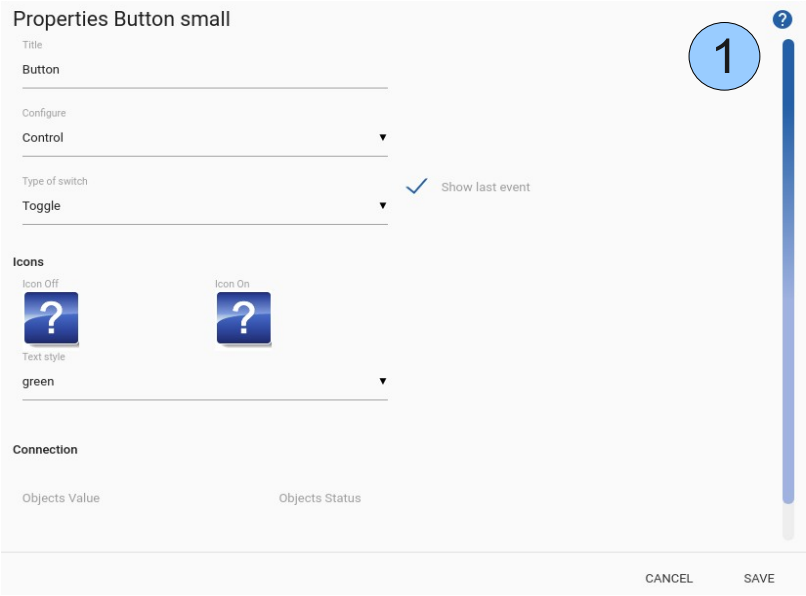


Figure 3: Property dialog

The property dialog (see 3) 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 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. 26). 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

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

Import .knxproj or .esf from ETS4/ETS5

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

EibStudio still supports *.esf* imports (used in 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 (p. 25).

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

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.

Debug-Mode

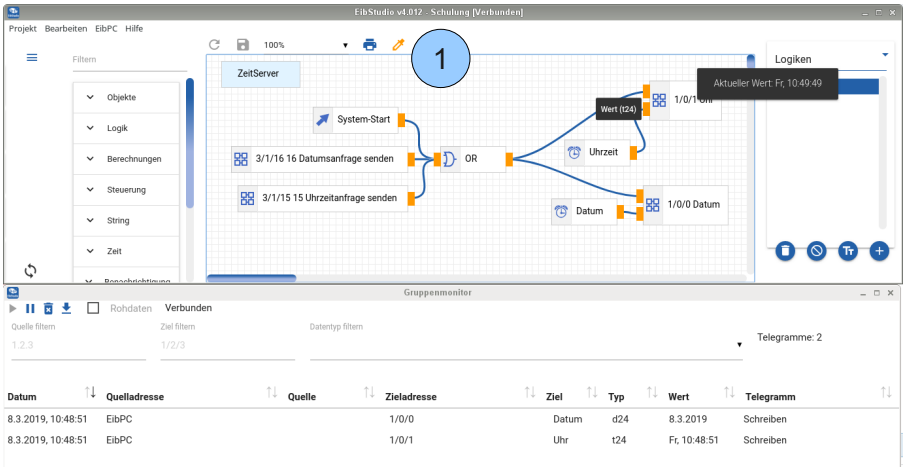


Figure 4: Debug-Mode in Logic

To implement the Logic, internal variables are created for every Input and Output. They are usually hidden (p. 15). 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 EinPC. `Ctrl+Click` can be used to directly set a new value.

It is recommended to use Simulation for advances tests (p. 25).

The Logic in 4 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 EibStudio.

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, EibStudio provides some templates, e.g., for the 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. 19).

Using an Element from within your Logic, is simple. You can switch between the basic appearance and its “active” state (p. 17).

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. 19) for details.

Expert

The Expert provides access to every feature of the Exertex® EibPC by writing programs. For a function reference, see Expert Functions (pp. 112).

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 EibStudio.

Custom Visualization

You can use the expert to “program” your visualization. Use the directive #addto [Webserver] before starting with webserver definitions (Visualization in Expert, p. 44).

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 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. 24).

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. 30).

If the ID of the Element is relative to the page (see below), 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 Visualization (p. 207):

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

If you use custom visualization pages, you have to define the start-IDs for Visu (p. 24).

Page-dependent Visualization-elements:

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

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. 30 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. 30). 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 EibPC, i.e., a single installation.

Changed must be sent to the EibPCs, either by pressing a button (S) 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 address

(S)

The EibPC is automatically restarted when the network configuration is changed. If the device is un-reachable, perform a Factory Reset to activate DHCP (p. 10).

Name resolving

(S)

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

Ports

(P)

TCP- und UDP-Ports für eingehende und ausgehende Verbindungen.

Date and Time

(S)

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 EibPC can use time information from the bus to synchronize the internal clock. If no reliable time source is available, the EibPC can be the time master, and regularly send its internal clock to the bus.

The 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 EibStudio or the KNX bus is overwritten. Before the actual EibPC program starts, it tries (at most 5 minutes) to synchronize its clock.

The 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.

Location

(P)

SHUTDOWN Variable

(S)

Before the program is stopped (when a new program is transferred or the EibPC is restarted using 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 EibPC can forward all telegrams received from the KNX bus to an FTP server. It uses port 21

E-Mail

(P)

Configure the server connection to send emails. (P)

Backup

(S)

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

(S)

To use a custom image in the visualization, it must be sent to the 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.

Only use regular letters and numbers, no symbols or umlauts.

HTTPS

(S)

The 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 EibPC can open an OpenVPN server. You must generate a certificate before the OpenVPN server can be started.

IDs

(P)

The firmware manages internal resources by unique numbers (IDs). To prevent collisions between self-assigned IDs and automatically assigned IDs modify the start IDs.

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 EibPC has been purchased, it can be applied using EibStudio.

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 500.000 telegrams. 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. 227 for an explanation of the Events.

Simulation

To implement and verify complex control logic, simulation may be helpful. Select the KNX connection type “Simulation” from the Project Settings (p. 22). 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 5.

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.

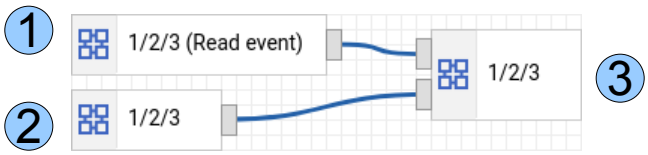


Figure 5: Answer Read Request

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 EibPC starts, which creates a huge and unnecessary delay. The initialization can be disabled in the Project Settings (p. 22).

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 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 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.

Data types

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:

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 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

Character strings

- 14 characters — 14 for DPT 16
- 1400 — no digits, default length
- custom length — Length between 1 and 65534 characters not 14
In the following referred to as c

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 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 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 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 150.

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)

Character strings

Character strings have a custom length between 1 and 65534 characters, e.g.,ac1, ac65534. If the length is omitted, a default length of 1400 characters is used. $String$ reserves memory for 1400 characters. To save memory, short phrases can be defined, e.g., offc3.

A length of 14 is handled differently and represents the DPT 16 which is encoded in ISO 8859 and used e.g., to show text on KNX devices like displays.

The two types of character strings, c14 and custom-length character strings can be transformed into each other by using the convert-function (see page 150) but not used interchangeable.

IP Address

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

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

Individual Address

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

- 1.12.230. This address is of data type u16.

An overview of the data types

Type	Data type	Example of a constant	Usage	Range	DPT	EIS data type
Binary	b01	1b01	Switch actuator, sun-blind actuator	0, 1	1	EIS1/EIS7
2 bit	b02	2b02	Lock objects	0,1,2,3	2	EIS8
4 bit	b04	10b04	Dimming	0,1 ... 15	3	EIS2
Percentage	%	85.3%	Heating regulators, actuators	0,1.1 ... 100.0	5	EIS6/EIS14.001
8 bit integer without sign	u08	255	Simple numbers, programmable thermostats, etc.	0, ... 255	5	EIS6/EIS14.001
8 bit integer without sign	u08	255u8	Optional types		5	EIS6/EIS14.001
8 bit integer with sign	s08	-45s08	Temperature sensors	-128... 127	6	EIS14.000
16 bit integer without sign	u16	45u16		0 ... 65535	7	EIS10.000
16 bit integer with sign	s16	-450s16		-32768 ... 32767	8	EIS10.001
24 bit integer without sign	u24	292235u24		0 .. 16777216	232.600	EIS11.000
24 bit integer with sign	s24	-92999s24		-8388608 .. 8388607		EIS11.001
32 bit integer without sign	u32	92235u32		0 .. 4294967295	12	EIS11.000
IP address	(u32)	192.168.22.100	IP address: sendudp etc.	0.0.0.0 .. 255.255.255.255		EIS11.000
32 bit integer with sign	s32	-9999s32		-2147483648 .. 2147483647	13	EIS11.001
64 bit integer without sign	u64	92235u64		0 .. 18446744073709551615		n.a.
64 bit integer with sign	s64	-9999s64		-9223372036854775808 .9223372036854775807		n.a.
Short float	f16	4.0	Wind sensors	-671088.64 .. 670760.96	9	EIS5
Short float	f16	4.0f16		-671088.64 .. 670760.96		EIS5
Float 32 bit	f32	4.0e01f32		-3.40282e+38 .. 3.40282e+38	14	EIS9
String	c14	$HelloWorld$c14	Display panels	14 characters		EIS15
String	(c1400)	$HelloWorld$	LAN telegrams	1400 characters		n.a.
String	(c1400)	$HelloWorld$	LAN telegrams	1 – 65534 characters		n.a.

Table 1: Data types

Note: The data types d24, t24, Y64 are KNX DTP types handled properly by their definition in 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
```

Not permissible here...

In contrast, it is permissible to program a counter using variables in this way:

```
//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. 15) 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 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 EibPC starts processing the user program, the user might want to initialize the images of the group addresses. The 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 Addresses**.

Important

- Before the actual program starts, the 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

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

Object tree

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. 26). Program Objects include all Objects, but also all Expert Functions (p. 112) 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 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.

Program start

Every program object, e.g., variable, group address, … is initialized to zero (OFF, 0, 0.0 …) and has the state “valid”.

The following examples can be added as new EXPERT program.

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 1).

Assignments

The equal sign is used to assign the value of a constant, a variable or a function on the right to a variable on the left. Both sides are equal after the assignment (p. 20). An assignment is only possible of the data types f both sides are the same. Otherwise use the function convert (p. 150) to convert the type of the right side.

With character strings the whole memory content is copied instead of stopping at the first 0-byte. This allows to combine assignments and stringset (p. 162). If the character string on the left is wider than the right side, the remaining memory content is overwritten with zeros. Please mind the difference between c14 and every other character string type.

Variables

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.

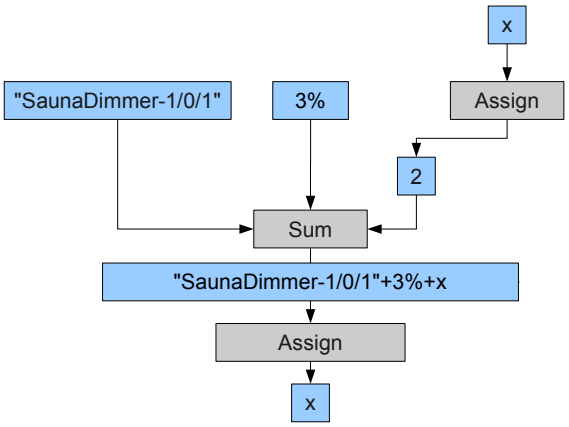


Figure 1: Program Objects Tree for y="SaunaDimmer-1/0/1"+3%+x and x=2

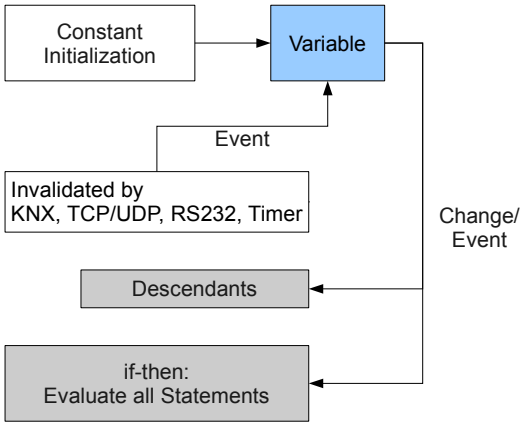


Figure 2: Evaluation of Variables

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 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
- startvpn
- stopvpn
- openvpnuser
- closevpnuser
- ping

Examples:

```
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.

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 on 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 evaluation 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
```

Definitions are global

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 3):

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 EibPC catches circular dependencies, stopps the evaluation and generates an Event (PROC_REPITIONS, p. 227).

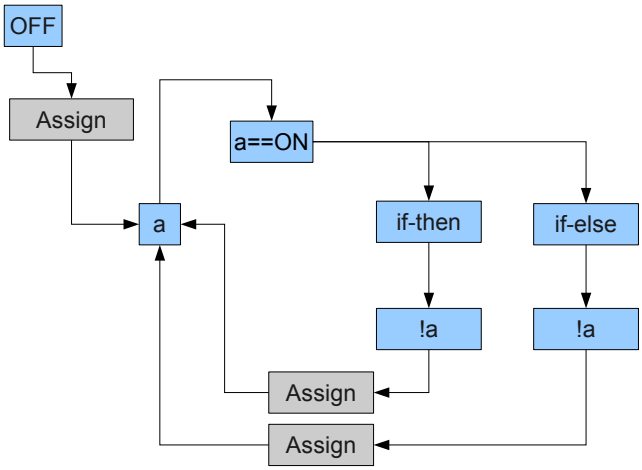


Figure 3: Program Object Tree Structure for a=OFF; if a==ON then a=!a else a=!a endif

The Program Object Evaluation guarantees that

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

Visualization

The EibPC² offers a web based visualization which can be displayed on all modern browsers independent of the operating system. When values change, the visualization website is updated immediately. In EibSTudio, the visualization can be created in Visu and/or in the expert.

The visualization is separated into groups of pages on which different elements are placed. Groups are used only for clarity, but do not have any other properties.

Elements are distinguished between global and page-dependent elements. Global elements can be used more often, i.e. they can be inserted several times on one or different pages. All these elements are addressed with a function via the user program. In addition, there are page-related elements that can only be used on one page. For addressing via the user program, the page must also be specified in each case. This addressing takes place in the form of unique numbers, the IDs. These are assigned when the elements are created and are used for access by the user program.

When creating your own visualization pages, you must ensure that the IDs between Visu and Expert do not overlap (see IDs, p. 23). All pages and elements must have unique IDs. Pages and global elements each have their own number ranges. All page- dependent elements on a specific page share the same ID range. Global elements have a separate ID range for every Element type.

Viszalization editor

Elements of a page are arranged in a rectangular grid (cf. 1). For each page, the number of rows and columns of this grid can be defined. There can be only one element in a cell of this grid. Most elements have a fixed size, i.e. a fixed number of rows and columns they need to be displayed. Overlapping of elements is not possible.

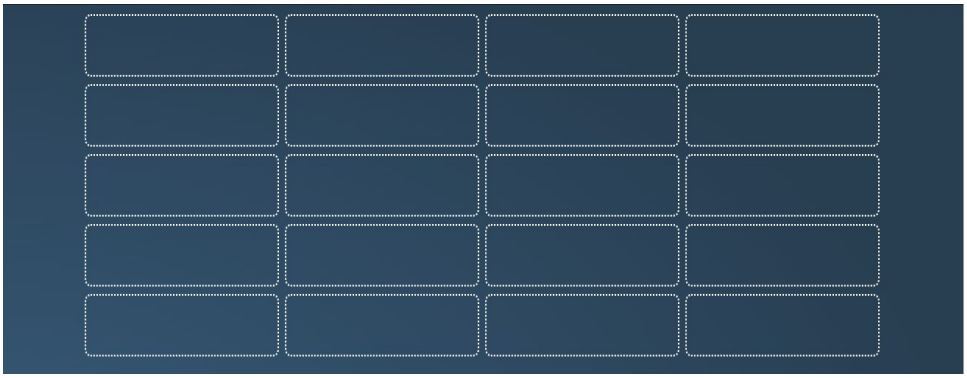


Figure 1: Page grid

For better readability on smaller displays, the number of columns is automatically adjusted (Responsive Design). For example, on smartphones, the visualization is displayed in a single column, regardless of how many columns have been configured for the page. The arrangement is row-based, referring to the upper left corner of an element.

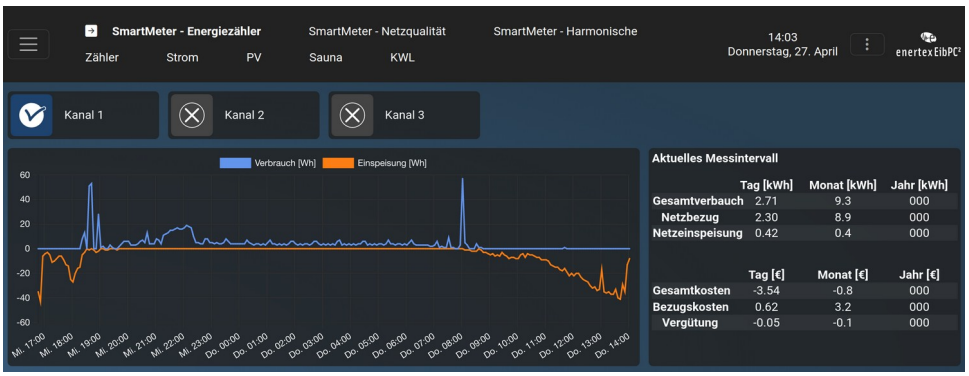


Figure 2: Visualization on Desktop Computer

Figure 3: Visualization on Smartphone

The page navigation is generated automatically (see 4)

Figure 4: Page navigation

For pages, there is a blue display variant in addition to the dark one (see 5). The selection is made in the page properties in EibStudio, or with the corresponding command in the expert program (p. 49).

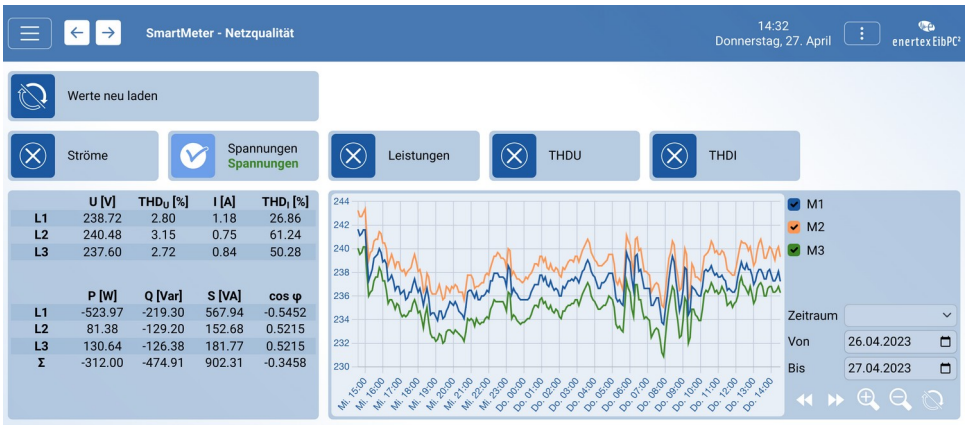


Figure 5: Blue design

Password protection

Pages can be individually protected with a user name/password in their properties dialog. The combination of user name and password must be identical across all pages. These pages are hidden in the navigation until the user logs in on the page in the browser. After that, the page can be accessed normally. The login data can be saved in the browser so that no new login is necessary when the page is visited again.

Elements

6 shows an overview of the available elements.

Buttons of different width and icon count as well as multiple selection are used to switch e.g. lights or blinds.

Sliders and color input can be used for dimming. For timers there is date and time selection. General graphics (of any web address) are displayed with the Picture element. By means of Plink it is possible to jump to visualization pages in addition to the main navigation.

Measured values are displayed either in the chart, without further storage and with any x and y value, or stored as a time series in a TimeBuffer and displayed by the TimeChart.

External web pages, e.g. camera images, can be displayed directly in the visualization using the frame element.

Separation lines can be used to divide a page into sections.

In the visualization editor, the page-related variants of the elements are used, if available. For access in the expert program, the page-related functions (e.g. pdisplay) must therefore also be used (see p. 19).

Figure 6: Elements

Functions

In addition to the self-configurable basic elements, elements with already stored functions are available, which usually comprise several elements.

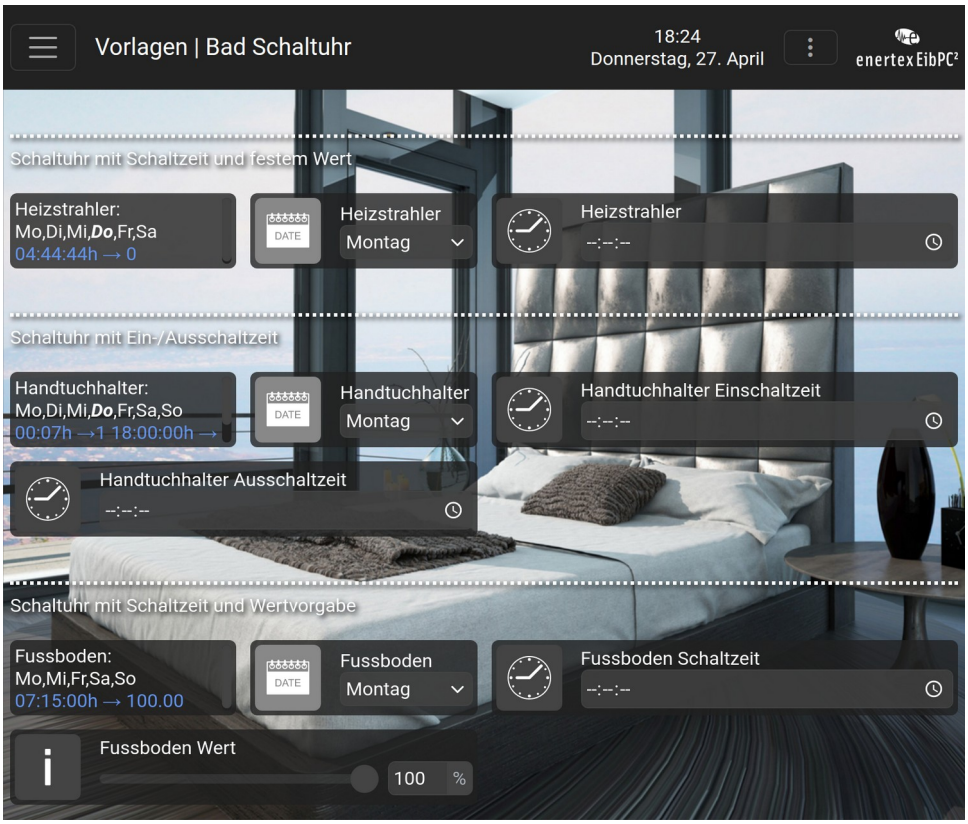


Figure 7: Predefined functions in Visu

Templates

In Templates you can find complete pages that contain elements and functions already arranged. You can also create your own page templates, for example, to quickly create similar visualization pages in different projects.

Visualization in Expert

This section is only relevant if you want to define your own pages within an expert program. As an alternative to creating entire visualization pages in the expert, you can access individual visualization elements within expert programs by assigning them an ID variable (see p. 19).

The elements from the application program are accessed using the visualization functions (from p. 207).

To add pages to the visualization in the Expert, add the following directive in an Expert program

#addto [WebServer]

After that, the commands below can be used to create pages, as well as add elements. Whether pages defined in the expert should appear before or after the pages from the Visu can be changed in the project settings.

To leave the web definitions section, insert

#addto [EibPC]

after the definitions. You can continue with the normal EibPC program.

Pages

Pages can be grouped together in the definition. A maximum of 128 pages are possible, with a maximum of 128 elements per page (each ID 0-127). All elements in a line are separated by one or more spaces or tabs. The compiler detects the number of elements per line and automatically configures the grid (1). Each element must have an ID so that it can be accessed by the user program using the appropriate functions.

Definition

- page(*ID*)[$*Group*$,$*Name*$]

Arguments

- *ID*: Value between 1 and 100 as a 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

Placement

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]
$Multi$
```

a clearance is created in the representation as shown in 8.

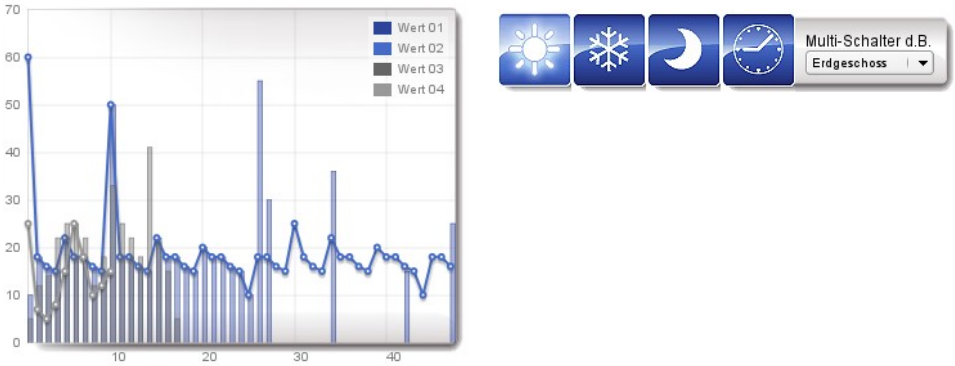


Figure 8: Clearance

When configuring the Web server, each line of the text configuration represents a web server display line. In the "switched off" (compact (off)) mode, the elements of different heights are always arranged in one line, that is, the actual line height of the representation is indicated by the max. Height of all elements in the respective line. This creates the clearance in the web server. In other words, in the representation additional non-visible elements are placed under the elements. 9 shows this "allocation" of the unit sizes (shown in blue) of the above web configuration.

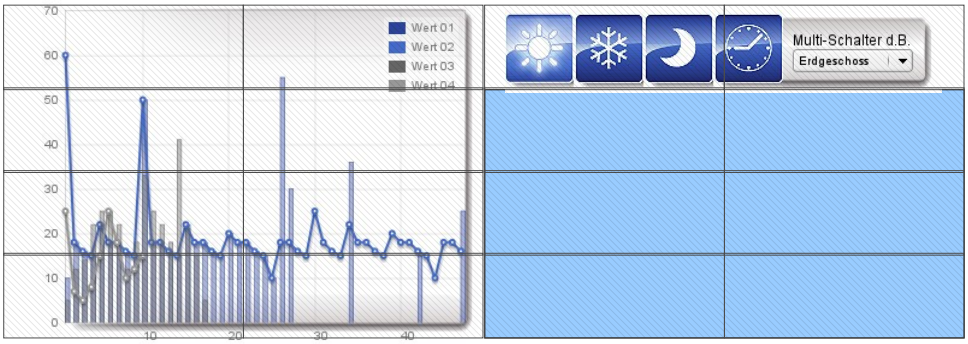


Figure 9: Illustration of the unit sizes

The eibparser already displays the configuration in the Messages window:

```
====== 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 9 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.

```
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] $Multi$
```

The first line is as before. Now the clearances of 8 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 10, which is now output by the web server:

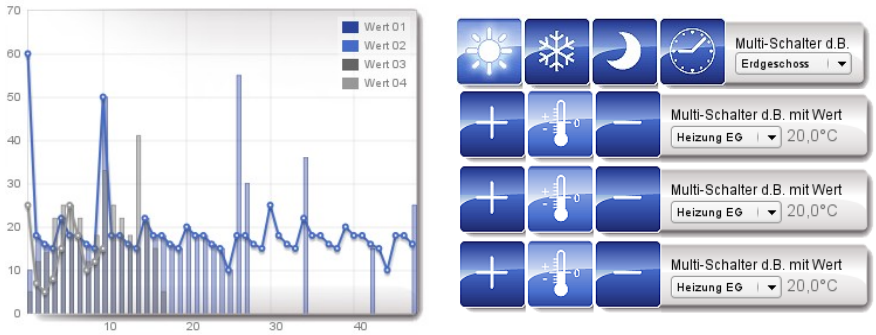


Figure 10: Compact mode

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

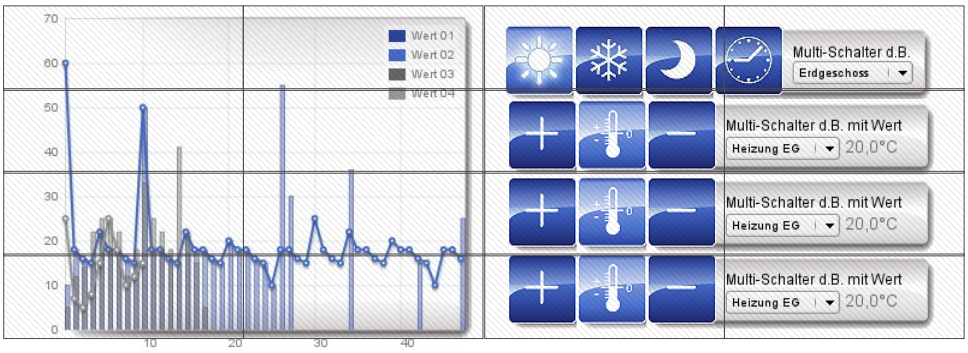


Figure 11: „compact“ with grid (for illustrative purposes)

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 12:

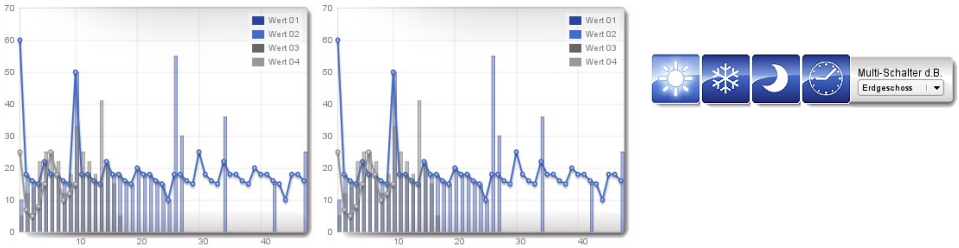


Figure 12: Representation example for line feed

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
```

New Page

Compact mode

Definition

- compact (*State*)

Arguments

- *State* 0 / 1 or ON/OFF

Password protection

Definition

- *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

The user password is not transmitted in plain text, even if the page is accessed via http instead of https. Nevertheless, it is recommended to always open the visualization via https locally as well.

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] $page 5$
```

Color scheme

Definition

- design $*DESIGNSTRING*$ [$*Link/Path*$] [$*CSS-Style*$]

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. 23) 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.
- $*CSS-Style*$ definines an optional CSS „style“ attribute for the background container. It can be used to customize the page background:
  Example:
  design $*black*$ [$*/upload/livingroom.jpg*$] [$*background-position:center;filter:blur(4px)*$]
  (added in EibStudio 4.113, Firmware 4.114).

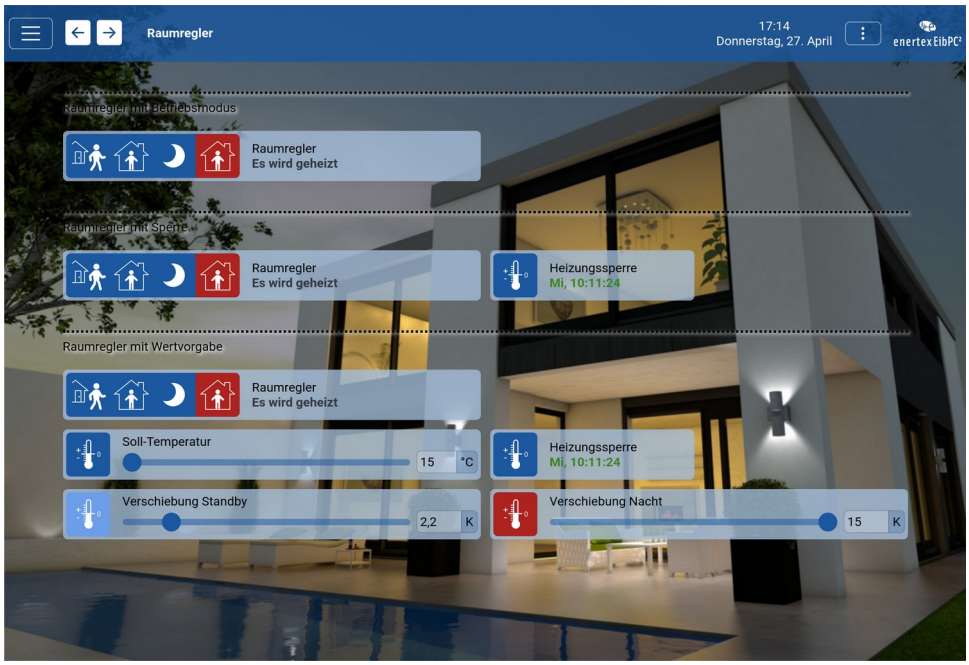


Figure 13: background graphics

Placeholder (Compact mode)

Definition

- empty

Insert an empty row also in compact mode

Placeholder

Definition

- none

Arguments

- None. An empty element of single width is inserted into the web server.

Access to the user program

- none

Separator

Definition

- 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

The following configuration options have no effect on Responsive Visu from firmware 5.000 and are for documentation purposes only.

Header

Definition

- 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

Definition

- 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

Zoom

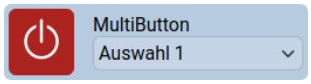
Definition

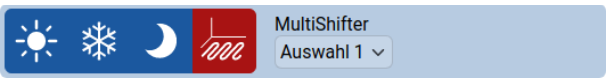
- mobilezoom*(Factor)*

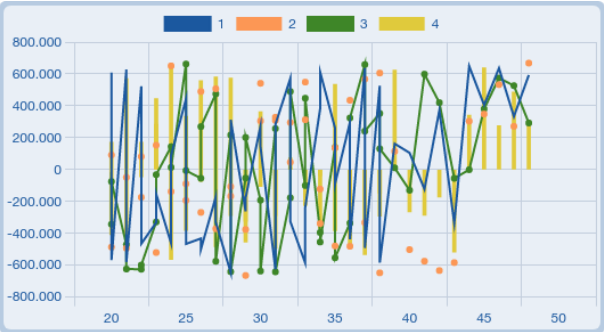
Arguments

- *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

Elements

Group	Element	Description
button		
	button, pbutton	 The graphic constituting the actual control panel can be modified by the user program. The first line of text is static (only changeable at the configuration). The second line can be modified by the user program, e.g. to display variables.
	shifter, pshifter	 The graphic can be modified by the user program. The first line of text is static (only changeable at the configuration). The second line can be modified by the user program.
	shifter, pshifter	 The **right** graphic can be modified by the user program. The **left** graphic can be modified only at the configuration. The first line of text is static (only changeable at the configuration). The second line can be modified by the user program.
	shifter, pshifter	 The **middle** graphic can be modified by the user program. The **outer** graphics can be modified only at the configuration. The first line of text is static (only changeable at the configuration). The second line can be modified by the user program.
	shifter	 The **right** graphic can be modified by the user program. The other graphics can be modified only at the configuration. The first line of text is static (only changeable at the configuration). The second line can be modified by the user program.
mbutton		
	mbutton, mpbutton	 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 208) and mpbutton (page 208), respectively, is sent to the application program.

Group	Element	Description
	mshifter, mp-shifter	 The graphic constituting the actual control panel can be modified by the user program. The first line of text is static (only changeable at the configuration). The second line can be modified by the user program. The listbox can administer a maximum of 4 entries. By operating the listbox, a signal which can be queried by the functions mbutton (page 208) and mpbutton (page 208), respectively, is sent to the application program.
	mshifter, mp-shifter	 The **right** graphic can be modified by the user program. The **left** graphic can be modified only at the configuration. The first line of text is static (only changeable at the configuration). The second line can be modified by the user program. The listbox can administer a maximum of 4 entries. By operating the listbox, a signal which can be queried by the functions mbutton (page 208) and mpbutton (page 208), respectively, is sent to the application program.
	mshifter, mp-shifter	 The **middle** graphic can be modified by the user program. The **outer** graphics can be modified only at the configuration. The first line of text is static (only changeable at the configuration). The second line can be modified by the user program. The listbox can administer a maximum of 4 entries. By operating the listbox, a signal which can be queried by the functions mbutton (page 208) and mpbutton (page 208), respectively, is sent to the application program.
	mshifter, mp-shifter	 The **right** graphic can be modified by the user program. The other graphics can be modified only at the configuration. The first line of text is static (only changeable at the configuration). No text can be displayed in the second line. The listbox can administer a maximum of 4 entries. By operating the listbox, a signal which can be queried by the functions mbutton (page 208) and mpbutton (page 208), respectively, is sent to the application program.
	slider pslider	 The image and the position of the sliders can be set in the application porgramm with the functions setslider and setpslider. Clicking the button element triggers the functions mbutton (page 208) and mpbutton (page 208), respectively.
	eslider peslider	The image and the position of the sliders can be set in the application porgramm with the functions setslider and setpslider. Clicking the button element triggers the functions mbutton (page 208) and mpbutton (page 208), respectively. The mininum, the maximum value and the increment can be parametrized.

Group	Element	Description
chart		
	chart, pchart	 This element serves the purpose of visualizing a time series. The labeling of the y-axis is defined at the configuration. The labeling of the x-axis can be modified by the user program. When calling the function webdisplay, the XY diagram is activated. Values from the field 1...30 can be represented. 0 means no representation. The values are displayed starting from the left. When the end is reached after 47 calls, the values are shifted to the left.
	mchart mpchart	 The pairs of variates are addressed by the application program via the function mchart. One element mchart administers up to 4 XY charts that can be supplied with data via the identical function mchart in the application program. A maximum of 4 diagrams can be defined, each having a labeling of its own (inserted in the top right corner). Up to 47 floating-point values are displayed. The scale is generated automatically.
	mchart mpchart	like above, though double height.
	picture	 Vorhersage An external link to a graphic is integrated. The graphic can be left-justified, centered or right-justified.

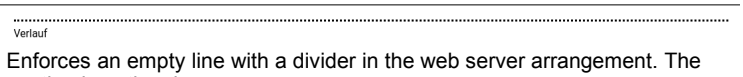
Group	Element	Description
Link		
	frame dframe	 Embedding an external website
	pLink	 Link to an internal page (simple button)
	Link	Link to an external page (simple button)
Decora-tions		
	line	 Enforces an empty line with a divider in the web server arrangement. The caption is optional.
	none	An empty field of single width.

Table 1: Overview of web elements.

Element Definitions

Switch of single width (global)

Definition

- button(*ID*)[*Image*] $*Text*$

Arguments

- *ID*: Value between 0 and 127 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 68).
- *Text*: A static labeling text (first line).

Access by the user program

- The image and the text are accessed by the function display (page 209).
- 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 207).

Switch of single width (page-dependent)

Definition

- pbutton(*ID*)[*Image*] $*Text*$

Arguments

- *ID*: Value between 0 and 127 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 68).
- *Text*: A static labeling text (first line).

Access by the user program

- The image and the text are accessed by the function pdisplay (page 210).
- The element is assigned to only one side
- Activation of the buttons has to be evaluated by the function pbutton (page pbutton).

Switch with selection of single width (global)

Definition

- mbutton(*ID*)[$*Text1*$,$*Text2*$,... $*Text254*$][*Image*] $*Label*$

Arguments

- *ID*: Value between 0 and 127 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 68).
- *Label*: A static labeling text (first line).

Access by the user program

- The image and the text are accessed by the function display (page 209).
- 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 mbutton (page mbutton).
- Switching of the listbox (providing the active listbox element) is arranged by the function display (page display)

Switch with selection of single width (page-dependent)

Definition

- mpbutton(*ID*) [$*Text1*$,$*Text2*$,...$*Text254*$][*Image*] $*Label*$

Arguments

- *ID*: Value between 0 and 127 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 68).
- *Label*: A static labeling text (first line).

Access by the user program

- The image and the text are accessed by the function pdisplay (page 210). 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 208).

Switch of double width (global)

Definition

- shifter(*ID*)[*Image1*, *Image2*, *Image3*, *Image4*]$*Text*$

Arguments

- *ID*: Value between 0 and 127 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 68).
- *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 display (page display).
- The operation of the buttons has to be evaluated by the function button (page 207).

Switch of double width (page-dependent)

Definition

- pshifter(*ID*)[*Image1*, *Image2*, *Image3*, *Image4*]$*Text*$

Arguments

- *ID*: Value between 0 and 127 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 68).
- *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 (page 210).
- The operation of the buttons has to be evaluated by the function pbutton (page 207).

Switch with selection of double width (global)

Definition

- mshifter(*ID*)[$*Text1*$,$*Text2*$,...,$*Text254*$][*Image1*, *Image2*, *Image3*, *Image4*] $*Label*$

Arguments

- *ID*: Value between 0 and 127 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 68).
- *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 display (page 209). 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 208).

Switch with selection of double width (page-dependent)

Definition

- mpshifter(*ID*)[$*Text1*$,$*Text2*$,...,$*Text254*$][*Image1*, *Image2*, *Image3*, *Image4*] $*Label*$

Arguments

- *ID*: Value between 0 and 127 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 68).
- *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 210). 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 mpbutton).

Simple Chart (global)

Definition

- 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 217).
- 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.

Simple Chart (page-dependent)

Definition

- 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 pchart (page 217).
- 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.

Chart with multiple graphs (global)

Definition

- mchart(*ID*) [*Size*,*Type*]($*Label1*$,*Style1*, $*Label2*$,*Style2*, $*Label3*$,*Style3*, $*Label4*$,*Style4*)

Arguments

- *ID*: Value between 0 and 127 as an index for programming and the access to this element.
- *Size*: SINGLE (2x2), DOUBLE (4x2), HALF (2x1), LONG (4x4)
- *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.

Chart with multiple graphs (page-dependant)

Definiition

- mpchart(ID) [*Height,Type*]($Label1$,Style1, $Label2$,Style2, $Label3$,Style3, $Label4$,Style4)

Arguments

- *ID*: Value between 0 and 127 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 218) 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 218.

TimeChart (global)

Definition

- mtimechart (ID) [size, type, length, YLMIN, YLMAX, YRMIN, YRMAX] ($Description1$, ChartPos1, Buffer1, $Description2$, ChartPos2, BUFFER2, $Description3$, ChartPos3, buffer3, $Description4$, ChartPos4, Buffer4)
- $Description1$, CHARTPOS1, Buffer1, $Description2$,...(up to 4 graphs)

Arguments

- ID: A value between 0 and 127 as an index for programming and access to this element.
- Size: 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.

  To ensure proper operation, the buffer and arts must be dimensioned so that the memory of EibPC is not overloaded. See here under timebufferconfig (p. 219) 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 timebufferadd (p. 219) and timebufferconfig (p. 219) 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. 219) and timebufferconfig (p. 219)
- mtimecharts are always global.

Color of graphs (page-dependant)

Definition

- *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 (page-dependant)

Definition

- picture(*ID*)[*Height*,*Type*]($*Label*$,$*www-Link*$)

Arguments

- *ID*: Value between 0 and 127 as an index for programming and the access to this element.
- *Height*: Value 0 or 1 (or constant SINGLE and DOUBLE) or Width x Height: any number for height and width as factor of the unit size of the elements of the web server.)
- *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

Access by the user program

- Label and link can be changed during runtime with the function picture (p. 214).

Simple Slider (global)

Definition

- slider(*ID*)[*Image*]$*Label*$

Arguments

- *ID*: Value between 0 and 127 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 68).
- *Label*: A static labeling text (first line).

Access by the user program

- The image and the text are accessed by the function display (page 209).
- Activation of the slider has to be evaluated by the function getslider (page 211).
- Changing the slider level has to be done by the function setslider (page 212).
- Activation of the button has to be evaluated by the function button (page 207).
- The input field can be used to directly manipulate the slider value in the web interface.

Simple Slider (page-dependant)

Definition

- pslider(*ID*)[*Image*]$*Label*$

Arguments

- *ID*: Value between 0 and 127 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 68).
- *Label*: A static labeling text (first line).

Access by the user program

- The image and the text are accessed by the function pdisplay (page 210).
- Activation of the slider has to be evaluated by the function getslider (page 211).
- Changing the slider level has to be done by the function setslider (page 212).
- Activation of the button has to be evaluated by the function pbutton (page 207).
- The input field can be used to directly manipulate the slider value in the web interface.

Extended Slider (global)

Definition

- eslider(*ID*)[*Image*] (*Min*,*Increment*, *Max*) $*Description*$ $*Label*$

Arguments

- *ID*: Value between 0 and 127 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 68).
- *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 209).
- Activation of the slider has to be evaluated by the function getslider (page 211).
- Changing the slider level has to be done by the function setslider (page 212).
- Activation of the button has to be evaluated by the function button (page 207).
- The input field can be used to directly manipulate the slider value in the web interface.

Extended Slider (page-dependant)

Definition

- peslider(*ID*)[*Image*] (*Min*,*Increment*, *Max*) $*Description*$ $*Label*$

Arguments

- *ID*: Value between 0 and 127 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 68).
- *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 210).
- Activation of the slider has to be evaluated by the function getslider (page 211).
- Changing the slider level has to be done by the function setslider (page 212).
- Activation of the button has to be evaluated by the function pbutton (page 207).
- The input field can be used to directly manipulate the slider value in the web interface.

Input of text, date, time, color (global)

Definition

- webinput(*ID*)[*Graphic*] $*Label*$

Arguments

- *ID*: Value between 0 until 127 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 68).
- *Label*: A static text below the picture
- *Style* is optional. Possible characteristics are
  - *none*: The output of webinput is a regular string.
  - PASSWORD: In this case, the input is hidden with asterisks or characters specified by the web browser. The output of webinput is a regular string.
  - DATEPICK: Enter a date using a standard dialog (depending on the web browser). The output of webinput is a string in the representation $ YYYY-MM-DD $
  - TIMEPICK: Enter a time using a standard dialog (depending on the web browser). The output of webinput 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 of webinput (p. 222) is a 24-bit string.

Access to the user program

- The element is addressed via function webinput (p. webinput).
- The image and the text are accessed by the function display (page 209).
- Elements of web input are always global.

Versatile output area (global)

Definition

- weboutput(*ID*)[*Dimension*,*style*]

Arguments

- *ID*: Value between 0 until 127 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, or Width x 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. 222).
- Elements of weboutput are always global.

Internal link (page-dependant)

Definition

- plink(*ID*)[*Image*] [*PageID*] $*Text*$

Arguments

- *ID*: Value between 0 and 127 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 68).
- *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 210).
- With the function plink (page 215) link, icon and text can be changed dynamically at run time.

External link (page-dependant)

Definition

- link(*ID*)[*Image*][$*Website*$] $*Text*$

Arguments

- *ID*: Value between 0 and 127 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 68).
- *Label*: A dynamically labeling text (first line).

Access to the user program

- With the function link (page 67) the web site, icon and text can be changed dynamically at run time.

Embed external Website (global)

Definition

- 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

Definition

- dframe [$*Text*$]

Arguments

- *Text*: A WWW link (incl. path and leading http://) to an external HTML site, which is integrated in the webserver. The embedded window is twice as high as this from the *frame* element.

Access to the user program

- none

Icons

The EibPC has a built-in set of icons.

See The icons listed in 3. In the visualization editor they are selected directly when configuring the element. In the [WebServer] section as well as in the user program they are selected by name or numerical index. Each symbol can be displayed in different forms. The states listed in Table 3 exist for this purpose.

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 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 2: Overview of states.

Symbol	**Index**	**DARKRED** 0u08	**INACTIVE** 1u08	**ACTIVE** 2u08	**DISPLAY** 3u08	**STATE4** 4u08	**STATE5** 5u08	**STATE6** 6u08	**STATE7** 7u08	**STATE8** 8u08	**BRIGHTRE D** 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	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										
HALT	27u08										
EJECT	28u08										
NEXT	29u08										
PREVIOUS	30u08										

LEFT	31u08										
RIGHT	32u08										
CROSSCIRCLE	33u08										
OKCIRCLE	34u08										
STATESWITCH	35u08										
PLUG	36u08										
METER	37u08	7102,2 kWh	7102,2 kWh	7102,2 kWh	7102,2 kWh						7102,2 kWh
PVSOLAR	38u08										

THERMSOLAR	39u08										
PUMP	40u08										
HEATINGUNIT	41u08										
HEATPUMP	42u08										
FLOORHEATING	43u08										
WALLHEATING	44u08										
COOLER	45u08										
MICRO	46u08										

SPEAKER	47u08										
RGB	48u08	RGB	RGB	RGB	RGB	RGB	R	G	B		RGB
LUX	49u08	k Lux	k Lux	k Lux	k Lux						k Lux
RAIN	50u08	%	%	%	%						%
KEY	51u08										
WASTE	52u08										
ASK	53u08	?	?	?	?						?
WARN	54u08										

NEAR	55u08										
CAMERA	56u08										
SIGNAL	57u08					FIRE	OIL	WATER	GAS		
DOOR	58u08										
GARAGE	59u08										
CURTAIN	60u08										
ANGLE	61u08										
ROLLER	62u08										

EMAIL	63u08					MAIL IN	MAIL OUT				
PETS	64u08										
PHONE	65u08										
PERSON	66u08										
TV	67u08										
BEAMER	68u08										
RADIO	69u08										
RECIEVER	70u08										

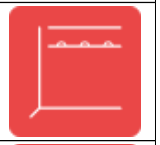
MEDIA	71u08										
STOVE	72u08										
FRIDGE	73u08										
WASHER	74u08										
DISHWASHER	75u08										
HOLIDAY	76u08										
SLEEP	77u08										
REFRESH	78u08										

EV	79u08										
TIMER	80u08										
DELAY	81u08										
SCHEDULE	82u08										
ALARMCLOCK	83u08										
RESET	84u08										
MAN	85u08										
WOMAN	86u08										

CLEANING	87u08										
BEER	88u08										
BATHING	89u08										
WATCHINGTV	90u08										
LOCK	91u08										
SETTINGS	92u08										
GEARS	93u08										
COLORTEMPERA-TURE	94u08	2200K 6000K	2200K 6000K	2200K 6000K	2200K 6000K						2200K 6000K

CHARTS	95u08										
CARBATTERY	96u08										
BATTERYSTORAGE	97u08										
HEATPUMPVENTI-LATION	98u08										
FLUIDMETER	99u08	2306,8 m³	2306,8 m³	2306,8 m³	2306,8 m³						2306,8 m³
WATERMETER	100u08	0124.1 m³	0124.1 m³	0124.1 m³	0124.1 m³						0124.1 m³
HEATMETER	101u08	1501,9 kWh	1501,9 kWh	1501,9 kWh	1501,9 kWh						1501,9 kWh
ENERGYMANAGE-MENT	102u08										

HEATINGROD	103u08										
HOMEVENTILATION	104u08	AIR	AIR	AIR	AIR	AIR	AIR	AIR	AIR	AIR	AIR
WATERING	105u08										
AIRCONDITION	106u08										
AIRCONDITION-HEATING	107u08										
CHRISTMAS	108u08										
STAIRSLIGHT	109u08										
SPOTLIGHT	110u08										

PENDANTLIGHT	111u08										
EXTERIORLIGHT	112u08										
HALLLIGHT	113u08										
LEDSTRIPESCEIL-ING	114u08										
LEDSTRIPESFLOOR	115u08										
MIRRORLIGHT	116u08										
FLOORLIGHT	117u08										
DESKLIGHT	118u08										

CEILINGLIGHT	119u08										
BATHROOM	120u08										
TOILET	121u08										
DININGROOM	122u08										
LIVINGROOM	123u08										
DRESSINGROOM	124u08										
KIDSROOM	125u08										
KITCHEN	126u08										

GARAGEFILLED	127u08										
BASEMENT	128u08										
OFFICE	129u08										
POOL	130u08										
SAUNA	131u08										
MAGNIFIERMINUS	132u08										
MAGNIFIERPLUS	133u08										
SMALLMINUS	134u08										

SMALLPLUS	135u08										
POWERGRID	136u08										
TOGGLE	137u08										
FILLEDDOT	138u08										
VOLTAGE	139u08										
RGBSLIDER	140u08										
WINDSOCK	141u08										

Table 3: Overview icons

Examples

Logic

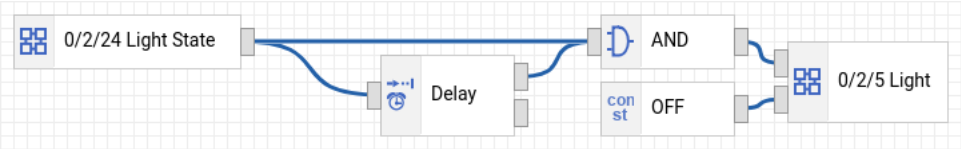


Figure 1: Automatic Light

Example 1: 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 1
- Compile and run the project

The Logic nodes are evaluated when objects change. For details, see Evaluation (p. 32). 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.

Expert

Send group telegrams

Example 2: A switch and two telegrams

If the switch is pressed "ON", turn on a lamp and set a dimming value to 80%.
If it goes to "OFF", turn both lights off.

Background
The switch can only send a single telegram with a single type. The switching actuator requires a binary value, while the dimming actuator needs a percentage (1 byte).

Telegrams can be sent to arbitrary group addresses by giving the address and type in single quotes, without having to import group addresses from ETS before (p. 30).

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

Instead of the “manual” group address, a group address from the ETS project can also be used if a project is imported (p. 15).

```
if ("Schalter-1/0/0"==ON) then write("Lampe-1/1/1",ON); write("Dimmer-1/1/2",80%) endif
if ("Schalter-1/0/0"==OFF) then write("Lampe-1/1/1",OFF); write("Dimmer-1/1/2",0%) endif
```

Example 3: Program start

Background
When the program starts, every program object is initialized to zero (p. 32). If the state of the switch 1/0/0 (or the status of the actuator) in the example above is already ON, the switch sends OFF with the next activation. However, the internal state of the group address object is already OFF, and no telegrams are sent by the EibPC. With the next activation, the switch becomes ON again, the internal state changes and the telegrams are sent.

Request the current state of group address "Schalter-1/0/0" when starting.

To execute an operation once when the program is started, the function systemstart changes from 0b01 to 1b01 and updates (invalidates) its dependencies. To get the current state of a group address, the function read sends a read request to the address when invalidated.

Important: For the actuator to answer the request, the read flag has to be set within ETS.

```
if (systemstart()) then read("Schalter-1/0/0") endif
if ("Schalter-1/0/0"==ON) then write("Lampe-1/1/1",ON); write("Dimmer-1/1/2",80%) endif
if ("Schalter-1/0/0"==OFF) then write("Lampe-1/1/1",OFF); write("Dimmer-1/1/2",0%) endif
```

To send a read request on program start, the function initga can be used as a convenient alternative.

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

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

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

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

The function htime implements the time switch (p. 129).

```
if (systemstart()) then                                          \\
                    MotionDetector=AUS;      \\
                    read("Switch-1/0/0") ;          \\
                    write("Lamp-1/1/1",AUS);      \\
                    write("Dimmer-1/1/2"u08,0%)   \\
              endif
// Variables
Switch="Switch-1/0/0"
MotionDetector=“MotionDetector-1/2/0“
Dimmer=100%
// The switch
if (Switch==ON)  then                          \\
        write("Lamp-1/1/1",EIN);           \\
        write("Dimmer-1/1/1",EIN);          \\
        write("DimmerValue-1/1/2",100%) \\
        endif
if (Switch==OFF) then                          \\
        write("Lamp-1/1/1",AUS);      \\
        write("Dimmer-1/1/2"u08,0%)   \\
        endif
// Motion detector
if (htime(20,00,00)) and (Switch==OFF) then Dimmer=50% endif
if (htime(23,00,00)) and (Switch==OFF) then Dimmer=30% endif
if (htime(03,00,00)) and (Switch==OFF) then Dimmer=10% endif
if (htime(07,30,00)) and (Switch==OFF) then Dimmer=100% endif

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

Example 5: A staircase lighting

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

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

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

Option 1:

```
if systemstart() then write('1/1/1'b01,OFF) endif
SwitchingOperation=OFF
if event('1/0/0'b01) then {
        SwitchingOperation=ON;
        write('1/1/1'b01,ON);
} endif
if (after( event('1/0/0'b01), 180000u64)) then {
        write('1/1/1'b01,AUS);
        SwitchingOperation=OFF;
} endif
```

The function event (p. 176) indicates, when a message is received on the bus by the given group address. It does not check whether the message has changed, its value or type. Once a message arrives, the function object's value becomes ON for a single cycle of EibPC. Thus, the condition of the if statement is true and the body is executed.

The delay function after expects a variable or an expression of type b01 as the first argument. The function after delays the input (ON and OFF), for the time specified in the second argument. The return value is also ON or OFF. This can be quite clearly represented graphically by 2. The second argument is of type integer, unsigned 64-bit. We therefore need the data type u64. This value specifies the delay time in ms.

You can set delays for decades. If the function after is started once, it processes only one impulse at its input. The result is the dead time being equal to the delay time, see 2. In the example we use a delay of

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

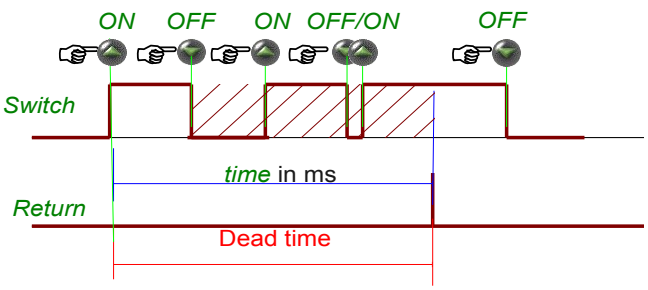


Figure 2: After-Function

The function after can not be triggered again nby the "dead time". In our case (option 1) this is desired. That is, if after has been stored once, any further changes of the input are ignored (see shading in 2).

Option 2. For the light circuit, the timer is to be restarted again if the light switch is pressed again. Therefore we need the function delay (p. 134) which restarts (Re-Triggers) the timer with every rising edge of the first argument.

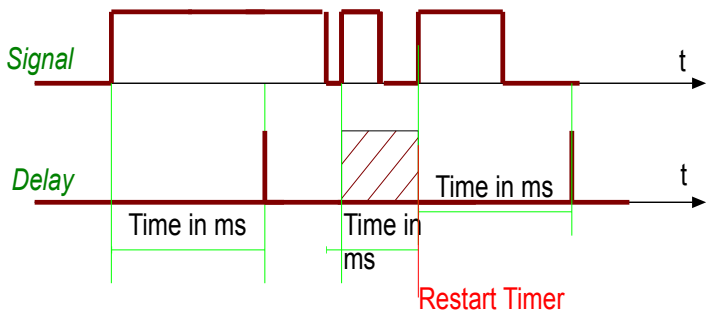


Figure 3: delay-function

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

```
if systemstart() then write('1/1/1'b01,OFF) endif
SwitchingOperation=OFF
if event('1/0/0'b01) then {
            SwitchingOperation=ON;
            write('1/1/1'b01,ON);
} endif
if (delay( event('1/0/0'b01), 180000u64)) then {
          write('1/1/1'b01,AUS);
          SwitchingOperation=OFF;
} endif
```

Duration of a cycle

One of the most asked questions of the user is: How much time does the *EibPC in fact need for the processing? In principal it depends on the size of program respectively the kind of programming and occurring events. By "validation" (p.* 32) of the program, only those parts of the program are activated per cycle that actually change. Therefore in the normal case the processing is done in less than 1 ms in more complex programs in a few ms. The time of cycle depending of the program will fluctuate. Therefore the minimal and maximal processing time is interesting.

The delay of up to 250 ms between two consecutive cycles is configured in EibStudio (pp. 22) to execute asynchronous functions, e.g. to send emails, process webserver requests etc.

To calculate the processing time of the EibPCs, the function afterc can be used:

afterc(variable {Typ b01}, max{Typ u64}, remaining time {Typ u64})

This function is triggered as the after-function with a change of *variable* (1. argument) from OFF to on: The return value is after the specified time *max* (2. argument in ms) for one processing cycle to ON. In each cycle from the beginning of the trigger pulse of variable while the remaining time *variable* while the *remaining time* (3. argument) is updated as countdown timer. The initial value of *variable* is *max*. The change of *remaining time* is always at exactly the time at which the processing is active in one cycle. The chance of *remaining time* is thus the sum of the aforementioned deadtime plus the processing time of the preceding cycle. This allows the cycle time calculated by using systemstart triggers a afterc -timer and starts the countdown of *remaining time* e.g.

```
Max=1000000000000000u64
if afterc(systemstart(), max, remaining time) then { ..... } endif
```

Max is here chosen as large as possible to ensure that the end of the countdown is reached not possible.

With the code

```
MaxZyklusZeit=max(StoppZeit-Restzeit-PerformanceZeit,MaxZyklusZeit);
MinZyklusZeit=min(StoppZeit-Restzeit -PerformanceZeit,MinZyklusZeit);
```

can thus be calculated with an accuracy of about ± 1ms (time slice Linux system time) the minimum and maximum cycle time.

A special case is still taken into account: During the initialisation of the very first program run all parts of the program must be run through, then the basis of the validation later "only when neccessary" are evaluated. Therefore the first processing loop may well need serveral hundred ms, when the program reaches a memory usage of about 30. The start of the countdown counter must therefore be delayed if you do not want to take into account the initialisation of the program as a special case in the measurement of cycle times.

Therefore delaying the pulse of systemstart at startup with another timer after timer by a nesting:

```
if afterc(after(systemstart(),10000u64), Max, Restzeit) then { ... } endif
```

In total the calculation of the cycle time as follows:

```
// Berechnet die minimale und maximale Zyklusdauer
// der Verarbeitung. Dabei ist die Performance-Angabe im EibStudio immer
// als Offset dabei.

Max=1000000000000000u64
Restzeit=0u64
StoppZeit=Max
MaxZyklusZeit=0u64
MinZyklusZeit=Max
// Im EibStudio ggf. geändert, Defaultwert ist 20ms
PerformanceZeit=20u64

// Die erste Zyklus kann etwas länger dauern ...
if afterc(after(systemstart(),10000u64), Max, Restzeit) then {
      StoppZeit=0u64;
} endif

MaxZyklusZeit=max(StoppZeit-Restzeit-PerformanceZeit,MaxZyklusZeit);
MinZyklusZeit=min(StoppZeit-Restzeit -PerformanceZeit,MinZyklusZeit);
```

The timer uses the argument afterc remaining time (s.a.) for storing the elapsed time timer. The user must therefore ensure that various afterc timer use different variables to this store:

```
// Zähler 1
RestZeit1=0u64
RestZeit2=0u64

if afterc(systemstart(),10000u64, Restzeit1) then {
      write('1/2/3'c14,$Timer1$c14)
} endif
if afterc(systemstart(),13000u64, Restzeit2) then {
      write('1/2/3'c14,$Timer2$c14)
} endif
```

The same applies to the function

delayc(TriggerVariable {Typ b01}, Max{Typ u64}, RemaingTimest {Typ u64})

whose timer – just like delay – through every change of the TriggerVariable (1. argument) from OFF to ON is triggered again. Again that for the rest of time each with its own variable must be used otherwise disrupt the timer each other.

When the timer expires the value of 3. arguments (remaining time) to 0u64, upon triggering of the timer it is set to the value of Max. If the remaining time is changed during an active phase by the user so the expiration time of the timer ist changed.

```
RestZeit1=0u64
 if afterc(systemstart(),10000u64, remainingtime1) then {
      write('1/2/3'c14,$Timer1$c14)
} endif
if remainingtime1>1000u64 then remainingtime1=500u64 endif
remainingtime2=0u64
if delayc(systemstart(),13000u64, remainingtime2) then {
      write('1/2/3'c14,$Timer2$c14)
} endif
```

In the above example only the afterc timer is changed the rest of the time variable delayc timer remains unchanged.

With this a timer can now be stopped if there is no longer need for e.g. the end of the process and the associated action of the if-statement.

```
MyTrigger=OFF
remainingtime1=0u64
 if afterc(MyTrigger,10000u64, remainingtime1) then {
      write('1/2/3'c14,$Timer1$c14)
} endif
if MyTrigger== OFF then remainingtime1=0u64 endif
```

If in the example *MyTrigger* switches to ON the timer is started, if *MyTrigger* switches to OFF before the expiry of the time, the timer is stopped by setting *remainingtime1=0u64* . The then-branch is not executed.

If you want to stop the timer before but running the then-branch it must *RestZeit1=1u64* be set. In this case the execution is performed in the next processing cycle.

Queue

The event-based processing in EibPC requires the programming of socalled “state machine”. The (abstract) basic principle of a “state machine” is that programming is not performed sequentially but that the software assumes a certain state depending on events.

When exchanging data with another device e.g. via TCP/IP telegrams, you can define the following states:

1. Receive data from the other participants
2. Send data to the other participants
3. Cache data of the other participants
4. Evaluate the data of the other participants
5. Perform various KNX actions on the bus

Each of these conditions is at least in principle independently of the other i.e. the EibPC has to accept data while e.g. KNX telegrams arrive. In addition various states can “triggering” each other respectively the arrival of a KNX telegram encourage the data processing.

Presence state machine

A user wants to use the macro `At_Sunset_Capped_withRelease` to send a group telegram at sunset, but at latest at a given time.

In the same way the macro is used: At`_Sunset_Capped_withRelease` at sunset.

```
Bei_Sonnenuntergang_Gedeckelt_mitFreigabe(Sued,FreigabeVar,"Licht Wohnen-2/2/3",AUS,22060000,22,31,00)
Bei_Sonnenaufgang_Gedeckelt_mitFreigabe(Sonnenaufgang1,FreigabeVar,"Rolläden Ost-5/2/0",RAUF,7200000,07,28,00)
```

The macros are parameterized with the release-variable *FreigabeVar* .

For this purpose the release is divided into the following observation periods:

- Day mode: Sunrise to sunset
- Early mode: Period after 0:00 clock and before sunrise
- Late mode: After sunset and not after 0:00 clock

The user presses a group address "Presence-8/1/1" (Typ b01, ON==present).

The release-variable *FreigabeVar* should be switched dependent on the following states.

State 1:

Description:

Early mode

Target:

It should not be run through a macro regardless of whether "Presence-8/1/1" is ON or OFF. *FreigabeVar* **has to be set to OFF respectively has to remain in the (OFF-)-condition.**

State 2:

Description:

Day mode

Target:

If "Presence-8/1/1" is set to ON, *FreigabeVar* has to be set to ON, the macros will be activated, if "Presence-8/1/1" is set to OFF. *FreigabeVar* should set to OFF the macros will be deactivated.

If the group address "Presence-8/1/1" is changed (bus telegram/user) should the *FreigabeVar* immediately accept its value.

State 3:

Description:

Late mode

Target:

If"Anwesenheit-8/1/1" is set to ON, *ReleaseVar* should be set to ON, the macros so are activated, if "Presence-8/1/1" is set to OFF. *FreigabeVar* should be set to OFF the macros will deactivated.

This can now directly be converted into a program:

```
FreigabeVar=AUS
TFrueh=chtime(00,00,01) and !chtime(12,00,00)
// Zustand 1: Frühmodus
if TFrueh and !sun() then FreigabeVar=AUS endif
// Zustand 2: TagModus
if sun() and change("Anwesenheit-8/1/1") then FreigabeVar="Anwesenheit-8/1/1" endif
// Zustand3   Spätmodus
if !TFrueh and !sun() then FreigabeVar="Anwesenheit-8/1/1" endif
```

Especially here is the use of variable *TFrueh.* This is realized via a link from one timer at midnight and a second at noon. This is ensures that *TFrueh* is set at 0:00 clock to ON and from the afternoon to OFF.

Presence simulation

The macro collection includes macros for presence simulation. The basis concept of these macros is to be explained in the following.

With a presence simulation two states can be differentiated.

1. Record

During this phase selected group addresses are recorded before. Group telegrams are often triggered by residents e.g. upon actuation of switches. The recording is usually performed over a 2-week interval in which the recording continuously overwrites the old values.

2. Play

If the resident of a property e.g. goes on vacation the group telegrams will now be triggered by the EibPC so that outsiders will have the impression of presence of the residents. There the play has to take place same day and time, so that e.g. the recording of Saturday is played on a Saturday again too.

As above mentioned conditions the following is needed:

- Determination of raw data of the telegrams
- Determination of sending group address
- Determination of telegrams arrival time
- Recording of data
- Sending of raw data time-shifted to the bus

Determination of sending group address

For this task you need the function readrawknx:

readrawknx(*Sim_Control {u08}, Sim_Sender{u08}, Sim_GA{u08}, Sim_IsGA{b01}, Sim_RoutingCnt {u08}, Sim_Len{u08}, Sim_Data{c1400}*)

If any KNX telegram is observed on the bus the function readrawknx updated its arguments. In this case the arguments of the function are “filled” with data. The received user data are then copied to the argument *Sim_Data* , the amount of data (bit length) can be queried with the variable *Sim_Len* .

Upon receipt of a telegram the argument *Sim_IsGA* is set accordingly, i.e. is it an ordinary group telegram so this argument is set by readrawknx to ON and *Sim_GA* contains the address itself. The function readrawknx can be linked to event in order to process the arrival of a telegram

With the selected definitions

```
Sim_GA=0u16
Sim_IsGa=OFF
Sim_RoutingCnt=0
Sim_Len=0
Sim_Data=$$c4000
Recorder=$$c4000
Timestamp=$$c4000
```

you can now process the arrival of a telegram as follows:

```
if event(readrawknx(Sim_Kontroll,Sim_Sender,Sim_GA,Sim_IsGa,Sim_RoutingCnt,Sim_Len,Sim_Data))  then ....
```

It should be noted that the group address *Sim_GA* is calculated as 16-bit value. In order to compare this address with the usual spelling is the function getaddress at your disposal. In the following example

```
MeinGA=getaddress("Licht-1/2/3")
```

there is now MeinGA the 16-bit value which represents the group address and how this is also copied *Sim_GA.* Now it is determined out of which group address the arrived telegram has been sent.

With the help of variables

```
Sim_GA=OFF
```

should the recording of an incoming message be triggered as follows. For each recorded group address are if-queries deposited. *Sim_GA* is determined as above mentioned by readrawknx .

Code-part 1

```
if Sim_GA==getaddress("Heizvorlauf-0/0/1") then Sim_MyGA=ON else Sim_MyGA=OFF endif
if Sim_GA==getaddress("Temperatur-3/5/0") then Sim_MyGA=ON else Sim_MyGA=OFF endif
if Sim_GA==getaddress("Licht-1/0/29"u16) then Sim_MyGA=ON else Sim_MyGA=OFF endif
```

The both modes Record/Play are realised via

```
Sim_Play=OFF
```

At *Sim_Play* = ON the existing recording should be played and at OFF the recording should be started.

Determination of raw data of the telegrams

Now it is necessary how the raw data of the telegrams on the bus can be determined. For this purpose

Code-part 2

```
if event(readrawknx(Sim_Kontroll,Sim_Sender,Sim_GA,Sim_IsGa,Sim_RoutingCnt,Sim_Len,Sim_Data)) and
Sim_Len!=0 and Sim_IsGa and !Sim_Play then {
  if !Sim_MyGA then Sim_Next=OFF endif;
  if Sim_MyGA then {
     if Sim_Len==1 then Sim_RawData=convert(stringcast(Sim_Data,0u08,1u16) and 0x7F,0u32) endif;
     if Sim_Len==2 then Sim_RawData=convert(stringcast(Sim_Data,0u08,2u16),0u32) endif;
     // Byte Order has to be considered
      if Sim_Len==3 then Sim_RawData=convert(stringcast(Sim_Data,0u08,2u16),0u32)*256u32
+convert(stringcast(Sim_Data,0u08,3u16),0u32) endif;
     if Sim_Len==5 then Sim_RawData=convert(stringcast(Sim_Data,0u08,2u16),0u32)*16777216u32
+convert(stringcast(Sim_Data,0u08,3u16),0u32)*65536u32+convert(stringcast(Sim_Data,0u08,4u16),0u32)*2
56u32+convert(stringcast(Sim_Data,0u08,5u16),0u32)  endif;
     Sim_Next=ON;
  } endif;
}endif
```

Sim_RawData are raw data in u32 format. If only one bit has been sent, so 31 bits are “unused”. Die incoming data are written from readrawknx in *Sim_Data* string variable. These are basically regarded as raw data and then be converted into u32 bit value.The arrangement of data in 4 bytes (32bits) unifies the saving of the telegrams data and simplifies the method (how to show yet).

For processing these raw data on string *Sim_RawData* now the single bytes have to be interpreted as 1-byte integer values. This happens with the help of function stringcast.

X=stringcast(*src{cxxxx}, dest, Pos{u16}*)

This function start to look at the bytes on string *src* from the byte-position *Pos. dest* on there gives the target data type conversion on, which specifies the number of bytes and defines the conversion to the result *X*. Based on 3 it is explained: The graphic shows the string as byte arrangement. At position 3{u16} the value is hexadecimal 0x74.

			3u16	4u16	5u16	6u16				
0xXX	0xXX	0xXX	0x74	0xA0	0xE1	0x01	0xXX	0xXX	0xXX	0xX

Figure 4: String src as arrayfield.

A statement Z1=stringcast(*src,0,3u16*) will define a variable Z1 from the data type u08 (argument „0“) . The value is obtained from *src* (4) on position 3{u16} and is thus in this case 0x74 (dezcmal 116). A statement Z2=stringcast(*src,10u32,3u16*) however defines die number 0x74a0e101 (decimal 1956700417). This number of bytes, which are extracted from the string is obtained by the argument 10u32: The data type u32 is 32 bits long and consists of 4 bytes. The value 10 of „10u32“ itself is ignored, here. The order of bytes remains unchanged in the stringcast function.

Back to the example: *Sim_RawData* contains the data of the incoming telegrams in the first 4 bytes. The order of the bytes on the bus is different to the order of the bytes of the Linuxsystem of the EibPCs. In order to use these data the byte order has to be reversed i.e. the last bit has to be in the first place etc. This rearrangement is realised by the help of multiplication by 256 and 65536 and 16777216.

The present processing of raw data is limited to max. 32 bit telegrams. Longer data telegrams can not be recorded, on the other hand bytes will be surely wasted by recording 1 bit elements, because all telegrams are treated equally. Nevertheless this approach to some extent an optimal compromise because the processing is easier later.

The code-part 2 calculates now the data of the u32 – variable *Sim_RawData* .

Determination of telegrams arrival time

The points of transmission time of the telegrams have to be determined relative, because a previously recorded simulation relative (time-shifted) to the starting point of the simulation have to take place.

Code-part 3

```
// Die Uhr wird gestartet (Countdowntimer)
if Sim_Start then {
        Position=0u16;
        Sim_MyGA=OFF;
        if !Sim_Play then {
                stringset(TimeStamp,convert(Interval,0u32),Position);
        } endif;
} endif

// Die Uhr wird gestoppt nach dem Intervall
if afterc(Sim_Start,Interval,Timer) then {
        Position=0u16;
} endif
```

When changing from *Sim_Start* to ON the first if-statement initialises the string timestamp. In addtion a afterc-timer (a.m.) is started. *Interval* determines how the duration of the recording is, e.g. 1 day = 86400000ms. This function updates at each loop run as a countdown-timer die variable *Timer*. This function relatively counts down from the starting point the elapsed time in ms. In the string *Timestamp* the start is written on position zero but in order to simplify the maximum recording duration is limited on 32 bit (49 days).

Recording of data

if with code-part 1 is set, that the incoming GA is to be recorded (*Sim_MyGA* at ON), thus the data in the string *Data* and die group address in the string *Recoder* are saved. As the group addresses are only 16 bits wide, the bit length can saved in the same array at the same time. For storing the raw data in one string stringset is used.

stringset(*dest{cxxxx}, src, pos{u16}*)

This function writes into the target string *dest* on its position of location *Pos* the (binary) contents of *src*.

Code-part 4

```
if  !Sim_Play and Sim_Next then {
        stringset(TimeStamp,convert(Timer,0u32),Postion);
        //ggf. alten Zeitstempel löschen
                stringset(TimeStamp,convert(Timer,0u32),Postion+4u16);
        // GA abspeichern
        stringset(Recorder,Sim_GA,Postion);
        // Die Länge speichern
        stringset(Recorder,Sim_GA,Postion+2u16);
        // Den Wert speichern
        stringset(Data,Sim_RawData,Postion);
        Sim_MyGA=OFF;
        Sim_Next=OFF;
        Sim_GA=65365u16;
        Postion=Postion+4u16;
        // Überlauf?
            if Postion>capacity(TimeStamp) then Sim_Start=OFF endif;
} endif
```

The fact that the timestamp, data- and group addresses are stored 32 bits wide, the position of a telegram is equal in thess strings, which simplifies the processing. In a c1400 string are recorded up to 350 telegrams. With the help of 65k strings are recorded up to 16341 telegrams. In the present case the telegram memory was with c4000 determinated by 1000. The function capacity shows how many bytes the string can maximum save.

After the preset time the recoding will restart in code-part 3. The first stored values are overwritten , the old values are preserved, which can disturb. Therefore in the above code-part 4 a possibly existing timestamp out of a previous recording is deleted.

Playing of a recording

The playing of a recording is relatively simple. For this purpose there are only the group address and the raw data is “loaded” (strings) and these are written to the bus. In this case the timer from code-part 3 has to be restarted. The present countdown time on *Timer* is compared with the timestamp in *Timestamp* and initialize a letter when falling below of the time:

Code-part 5

```
if Sim_Play and Timer<convert(stringcast(TimeStamp,1u32,Position),0u64) then {
    SimGA_Out=stringcast(Recorder,0u16,Position);
    SimGA_Len=stringcast(Recorder,0,Position+2u16);
    SimGA_Val=stringcast(Data,0u32,Position);
    if SimGA_Len==1 then write(address(SimGA_Out),convert(SimGA_Val,EIN)) endif;
    if SimGA_Len==2 then write(address(SimGA_Out),convert(SimGA_Val,0)) endif;
    if SimGA_Len==3 then write(address(SimGA_Out),convert(SimGA_Val,0u16)) endif;
    if SimGA_Len==4 then write(address(SimGA_Out),SimGA_Val) endif;
    Position=Position+4u16;
} endif
```

The data types due to the use of the raw data need not be observed. Only the length of telegrams is to be evaluated so that they correspond to those of the recording.

The macro-library EnertexPresence.lib is realised in this manner.

In the library the recording will be broken down into smaller day intervals and assembled later when playing. The recording then starts each to the next day interval.

Encoding of c14

The KNX™ standard requires that devices with 14-byte messages („c14“ types) have to implement only the ASCII code, and optionally allows ISO8859-1, which itself only uses 1-byte characters (see http://de.wikipedia.org/wiki/ISO_8859-1).

EibStudio uses UTF-8 as internal character encoding. When the EibPC program is compiled, c14-strings are re-encoded in ISO8815-1 automatically.

String concatenation with different length

In string processing is often resorted to the concatenation i.e. the “concateantion” of strings.

Thus e.g. in the code

```
s1=$Hallo $c1000
s2=$Welt$c1000
s3=s1+s2
```

the string *s3* will have the content *Hello World.* The data type control in the EibParser ensures that *s3* is of type c1000. The EibParser ensures that the concatenation can record the size of the longest string , in the present case are for *s1+s2* 1000 Bytes. *s3* are assigned as a result of the concatenation *s1+s2* 1000 Bytes.

If 950 bytes of data already available in *s2* and in *s1* in turn is 90 bytes then 40 bytes are in the concatenation “lost” because only *s3* can max. hold 1000 Bytes.

The following code is to be sonsidered as well:

```
s1=$Hallo $c1000
s2=$Welt$c1000
s3=$$c2000
if htime(10,00,00) then s3=s1+s2 endif
```

Again the concatenation is *s1+s2* the length of 1000 Bytes, as they are composed out of two 1000 byte-strings. The assignment to the 2000 bytes long *s3* ovvurs only after the concatenation. However as already the concatenation operation has limited the length up to 1000 bytes here bytes can get “lost”.

This is in the following code different:

```
s1=$Hallo $c1000
s2=$Welt$c1000
s3=$$c200
if htime(10,00,00) then s3=s1+s2 endif
```

Again the concatenation is *s1+s2* the length of 1000 bytes, as they are composed out of two 1000 byte strings. The assignment of the 200 bytes long *s3* occurs only as a result of the concatenation: First the concatenation operation *s1+s2* limited the length up to 1000 bytes, allocating limited to *s3* its length to 200 bytes, so assuming, where 800 bytes of data „lost".

If the concatenation *s1+s2* in no case lose data, a dummy variable has to be introduced:

```
s1=$Hallo $c1000
s2=$Welt$c1000
s3=$$c2000
dummy=$$c2000
if htime(10,00,00) then s3=s1+s2+dummy endif
```

This ensures that *s1+s2+dummy* 2000 bytes can hold as a result. Therefore the concatenation will deliver 2000 bytes to *s3* as a result.

FTP Data streams

Four data streams

With the help of configurable FTP transfers any ASCII ("plaintext") files can be written to an external FTP server. The maximum file size is 64 kB.

For this purpose, four different handles (= ID number of transfers) are created, which - by itself buffered queue - create these files on the server. The files are via timeout earlier (and then fewer bytes if necessary) written or initiated by flushftp () by the user. The file names are assigned automatically by the firmware by date and time.

In the following, the procedure must be described in detail when creating and applying these FTP outsourcing.

First, the stream and its handle must be defined in the program. For this purpose, the function

ftpconfig(server,user,password,path,timeout)

is needed (P. 195). A handle refers to a unique number (ID) for a transfer and is about tantamount to a name.

Configuration of the transfer

The first three arguments are used to configure the Tranfers: IP address, user name and password, then follows the target directory on the server and a timeout parameter. Use this statement to reserve a 64 Kbyte buffer in Enertex ® EibPC. The transfer of the buffer occurs when either the buffer was completely filled (more on this below) or the number *timeout* seconds have elapsed since the last transfer.

```
// ServerDaten
server=$ftp.enertex.de$
user=$enertex$
password=$enertex$
path=$KNX/Telegramme$

// Timeout in Sekunden
timeout=900u32

// FTP Queue anlegen
// Wenn Handle ungleich Null, dann ist das fehlerfrei gelungen
Handle=ftpConfig(server,user,password,path,timeout)
```

Several strings are summarised in a line of text

During operation, the data must now be written into the buffer. Therefore

sendftp(handle,data1,[data2],[...])

is needed. The function allows arbitrary strings as arguments, because the target file is also just a text file.Any data in the form of numerical values must be converted using the Convert function. In this case an LF CR (newline suitable for Windows) is inserted at the end of the data transmission of sendftp. All call to sendftp can pass more than one substring, but no more than 1400 bytes assume total. Accordingly, the maximum length is 1400 bytes:

```
// Daten in die Queue schreiben
Data1=$Daten Nr. $
Data2=$ des internen Zählers - $
Nr=0u16
status=3
// minütlich werden die Daten Data1 in den internen Buffer geschrieben
// nach 15 Minuten (timeout) werden die Daten am FTP-Server ausgelagert
if stime(60) then {
  status=sendftp(Handle, Data1,convert(Nr,$$),Data2,convert(settime(),$$));
  Nr=Nr+1u16;
} endif
```

If the variable *status* to 1, writing to the buffer of the transfer was successful. However, this has nothing to do with the fact that the data have arrived on the FTP server.

For this, the status of the FTP data stream must be queried.

Therefore is

ftpstate(handle)

available.

With

```
ftpstatus=ftpstate(Handle)
if ftpstatus==5 then write('1/2/3'c14,$FTP Overflow$c14) endif
```

the following status can be obtained:

- Configures / error-free = 0
- the last transmission was error-free = 1
- the FTP server was not reachable = 2
- the password / user is not allowed = 3
- The target directory does not exist and it could not be created = 4
- The queue has an overflow (= 5), this can only occur if the transmission was not successful before.
- Handle is not defined = 6

If it is for the processing of importance to determine the level of the stream buffer, this can be learned with the aid of

ftpbuffer(handle)

ftptimeout(handle).

The first function returns the number of unused bytes in the buffer, the second function describes the elapsed time since the last transfer.

```
if mtime(0,0) then {
        //Füllstand des FTP Buffers
        buffer=ftpbuffer(Handle)+1u16
        //Bereits verstrichene Zeit seit dem letzten Transfer in Sekunden.
        timeout=ftptimeout(Handle)
} endif
```

In addition to the automatic writing of the data to the FTP server, the buffer can also be manually emptied (“flushed”) with the use of the function

flushftp(handle)

while you are uploading the data to the FTP server “manually”.

```
// Daten "manuell" flushen (nur dann wird die Übertragung aktiv)
// täglich um 00:00:00 Uhr
if htime(00,00,00) then {
  status=flushftp(Handle);
} endif
```

If no manual flushing or writing is done, the EibPC is going to initiate the transfer independently. The transfer takes place when the buffer is full or the configured timeout elapsed (in seconds) since the last transfer.

Use of own Html code and graphics on the Web server

With the weboutput field of the web server, the user can show his own HTML code on the visu. In the output field a simple text can be represented, but it is also possible to represent dynamically a complex HTML code.

Incorrect or invalid HTML code in weboutput may interfere with the page layout. Such errors are not corrected by the free support. Please work here with tools as shown on the link http://www.quackit.com/html/online-html-editor/ **to test the HTML code.**

Thereto you have to define the output field in the web server:

```
[WebServer]
page (2) [$Haus$,$Energie$]
weboutput(Out1)[QUAD,ICON] weboutput(Out2)[QUAD,NOICON]
[EibPC]
Out1=2
Out2=3
```

You can note that the weboutput field can only be set globally. The element can be displayed with or without an icon (ICON or NOICON). The width is set to 2 unit width, the height can be set single (SINGLE), double (DOUBLE) or quadruple (QUAD).

The restriction to global elements arises from the possibility that the Weboutput-box can absorb 65 Kbytes of data. For 40 global elements, which make 2 MB, you have to keep free space in RAM for these items.

With the function

weboutput(ID,Data)

is the data written of the field. In this case is Data a string with a maximum length of 65534 bytes (type c65534). A special feature is that this string can be a valid html code. This makes it possible to dynamic formatting and display.

We are going to describe the both at the outset specified fields so that a website as in 5 is created:

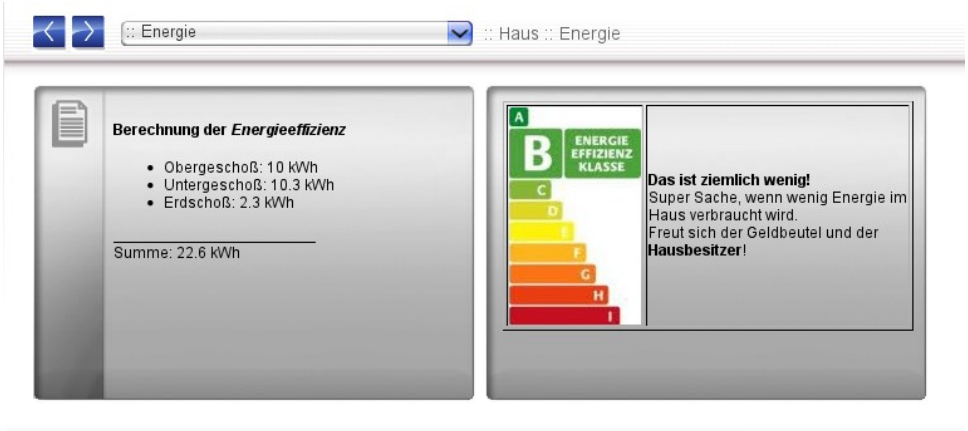


Figure 5: Weboutput

For the creation of the actual HTML code, please refer to http://de.selfhtml.org. The Html code can be preset using the website as the following:

```
if systemstart() then {
        weboutput(Out1,$<h4>Berechnung der <i>Energieeffizienz</i></h4> <ul style="list-style-type:disc">
<li>Obergescho&szlig;:  10 kWh </li> <li> Untergescho&szlig;: 10.3 kWh </li> <li> Erdscho&szlig;:      2.3 kWh
</li> </ul> ___________________________ <br>        Summe:  22.6 kWh  $c10000)
} endif
```

You can note, that the code inside the $-Sign can't be wrapped. In the development it's recommended to create and test the HTML code seperately.

With the help of an other dependency as the if systemstart() the text and the formatting can be changed the whole time even during the term of the program.

The second weboutput field should also have its own graphic. At first a .png, .jpg or .gif file has to be uploaded at the EibPC using EibStudio (p. 23). The path of the graphic for the weboutput is `/upload/` + file name. Thereby the graph and some text and the HTML formatting will be initialize with the following statement:

```
if systemstart() then {
   weboutput(Out2,$  <table border="1"><tr> <td class="oben"> <img src="/upload/effb.jpg" alt="Bild
fehl"></td> <td class="mittig"><b>Das ist ziemlich wenig! </b><br> Super Sache, wenn wenig Energie im
<br> Haus verbraucht wird. <br>Freut sich der Geldbeutel und der <br> <b>
Hausbesitzer</b>!</td></tr></table>$)
} endif
```

The output can be made depended of current values e.g. meter readings of an KNX device, which is shown in the following.

An engery meter sends via the GA "Energy-2/3/5“,"Energy-2/3/6“ "Energy-2/3/7“ of type u32 the consumption in Wh. We first define the variables in kWh as a string (c.1400).

Convert the consumption in kWh

```
ConsumptionOG_kWh=convert(convert("Energy-2/3/5",1f32)/1000f32,$$)
ConsumptionEG_kWh=convert(convert("Energy-2/3/6"1f32)/1000f32,$$)
ConsumptionUG_kWh=convert(convert("Energy-2/3/7",1f32)/1000f32,$$)
Sum_kWh= convert(convert("Energy-2/3/7“+"Energy-2/3/6"+"Energy-2/3/5“,1f32)/1000f32,$$)
```

At twelve o'clock the values should be displayed daily:

and on the webserver as a string – link (note: the “+” sign)

```
if htime(12,0,0)  then {
        weboutput(Out1,$<h4>Berechnung der <i>Energieeffizienz</i></h4> <ul style="list-style-type:disc">
<li>Obergescho&szlig;: $+ VerbrauchOG_kWh +$ kWh</li> <li> Untergescho&szlig;: $+VerbrauchUG_kWh+$
kWh </li> <li> Erdscho&szlig;: $+VerbrauchEG_kWh+$ kWh </li> </ul> ___________________________ <br>
Summe:$+Summe_kWh+$ kWh  $c10000)
} endif
```

Depending on the actual transmitted values, the display will be on the web server (compare with 6):

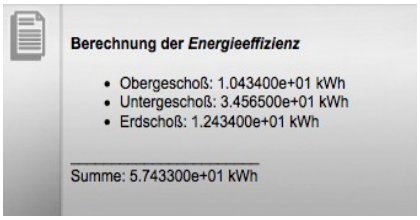


Figure 6: Dynamic Output

In the code section the HTML string is made of substrings by the use of concatenation (“+”-Signs). It is important to ensure, that the concatention produces the matching string length. The function weboutput can transfer up to 65564 bytes to weboutput-element. The concatenation consists only of $$ (=c1400) and one c10000 string. The string concatenation reserves for the result the number of bytes, such as the “longest” string-argument is predented.In this case it makes 10.000 bytes, which are given through the one c10000 string in the code (shown above).

At this point it should be said, that special signs could be composed of multiple bytes, as already described on P. 99. The concatenation could bring theoretically more than 10000 bytes as a result, if the strings exhaust the full length of their definition. In this case the “overlaying” signs cannot respect the concatenation function and accordingly the concatenation function is going to cut the signs of the string before copying into the result. It is up to the User if he respects it. (compare with p. 99).

Back to the example:

The most users don't like the output representation of the exponential floating-point representation. Therefore the representation of values should be more readable with the function stringformat. This function changes a number into a string - whereupon leading zeros and the indicated accuracy and floating-point representation can be parameterized.

Arguments:

1. Value (her f32)

2. Conversion of F32 in floating-point represenation: 4

3. Representaion with leading zeros: 4

4. Maximum length: 8

5. Accuracy: 1 point

```
VerbrauchOG_kWh=stringformat(convert("Energie-2/3/5",1f32)/1000f32,4,4,8,1)
VerbrauchEG_kWh=stringformat(convert("Energie-2/3/6",1f32)/1000f32,4,4,8,1)
VerbrauchUG_kWh=stringformat(convert("Energie-2/3/7" ,1f32)/1000f32, 4,4,8,1)
Summe_kWh=stringformat(convert("Energie-2/3/5"+"Energie-2/3/5"+"Energie-2/3/5",1f32)/1000f32,4,4,8,1)
```

Figure 7: Output format

Visualisation of time series

With the EibPC time series can be easily added, pemanently stored and visualised. For this purpose a diagram element art (p. 221) is available on the webserver.

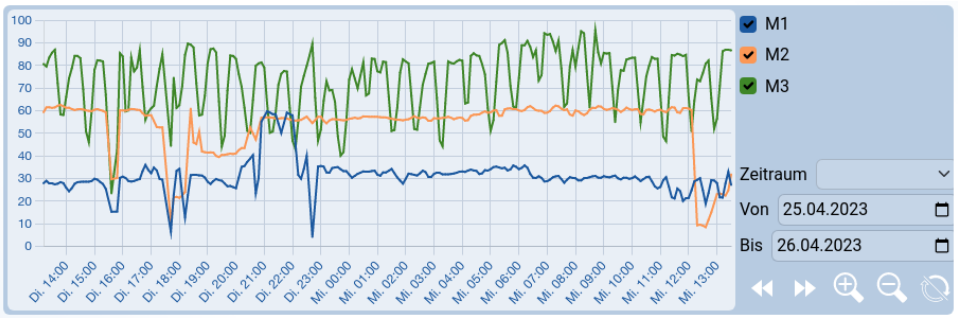


Figure 8: TimeChart webelement EXTLONG

8 shows a TimeChart with three of a total of four possible graphs and the action buttons. The user can scroll left and right in the TimeBuffer, as well as zoom. In addition, the time range to be displayed can be selected. The action buttons are part of the TimeCharts, so there is no further effort in the web element programming. The operations are applied to all graphs and displayed by all open visualizations. If the displayed area of the TimeCharts has been changed, this is indicated by highlighting the Reset button. Changes made to the TimeCharts by the EibPC program will no longer be automatically displayed in this state until the TimeChart has been reset by pressing the button (see 9).

Figure 9: Interactive TimeChart

Now consider the following definition (comp. 219):

timebufferconfig(*ChartBufferID, MemTyp, Length, DataTyp*)

This function allows up to 256 (ID 0 to 255) various buffers for recording time series. *MemTyp* indicates whether the memory in the ring (0) or linear (1) is described (more on this below). The length of the max. recording of time series is specified with *Length* (0u16 to 65565u16). Per stored value (see below) time series requires 12 bytes regardless of the stored *DataTyp*. It is recommendable to adjust the size of the memory to the real needs: A time series with the max. length occupies 780 kB RAM.

DataTyp displays a representative number of time series e.g. 0f16 for 16-bits numbers or 3% for u08 values. The number itself is not further processed and serves the compiler to win only the type information. We use the timebuffer with ID 0 for recording the temperature group address 1/2/3 (type f16) and the ID 1 for the adjusting size of the heat-controller 1/2/4 (u08).

```
R1_ID=1
// Timebuffer IDs vergeben:
ChartBuffer1=1
ChartBuffer1=2
// timebufferconfig: Einen Zeitbuffer konfigurieren
 MemTyp=0
Len=35040u16
Datatyp=3.3f16
timebufferconfig(ChartBuffer1, MemTyp, Len, "Temperature-1/2/3")
timebufferconfig(ChartBuffer2, MemTyp, Len, "Controll-1/2/4")
```

The readability of the code is increased, if we specify in the above example as the last argument the to be stored variable or group address. This is not absolutely necessary e.g. timebufferconfig(ChartBuffer1, MemTyp, Len, 2.2f16) or timebufferconfig(ChartBuffer2, MemTyp, Len,2) would also configure the timebuffer correctly.

With the configuration of the timebuffer to the webelement mtimechart the memory of the time series (timebuffer) is submitted for presentation by configuring their ID (=handle, acces of number). In this case the webelement accesses always out the last valid data in the memory.

Now the time series must be “filled” with data. The function

timebufferadd(*ChartBufferID, Daten*)

completes this task. The function writes the current value of the variable or group address (*data*) as well as the timestamp, which is derived from system time of the EibPC, in the memory of the selected time series. So there a time series exists exactly out of a combination value-timestamp. Values can be up to 4 bytes long. Timestamps internally nedd 8 bytes.

1.23 (4 Byte)	2.23 (4 Byte)	45.23 (4 Byte)	1.23 (4 Byte)	2.23 (4 Byte)	45.23 (4 Byte)
2013-11-08 8:00:00.223	2013-11-08 8:00:00.823	2013-11-08 8:03:00.223	2013-11-08 8:04:00.000	2013-11-09 8:00:00.700	2013-11-09 8:03:00.675

Figure 10: Building of time series (timebuffer)

As 10 should suggest, it is not necessarily so that the values in the timebuffer in the same interval have to be included, although this can often be the case when logging of energy data. The webelement mtimechart evaluates correctly the timestamp.

If the argument *MemTyp* from timebufferconfig was defined as a ring[store] so after reaching the last value the memory will be filled again from the beginning. i.e. the oldest value is replaces with the latest. Is *MemTyp* defined as linear[memory] then the recording stops if the memory is full

With a timeseries of linked diagram are automatically updated in the visualization i.e. it can be represented basically the same time series in different diagrams. For example writing every 15 minutes a value in the buffer and indicating the most recent 192 values in our diagram, you only need the following code:

```
// Store values in the time buffer
if mtime(0,0) or mtime(15,0) or mtime(30,0) or mtime(45,0) then {
        timebufferadd(ChartBuffer0,"Temperature-1/2/3");
        timebufferadd(ChartBuffer1,"Controll-1/2/4");
} endif
```

With

timebuffersize(*ChartBufferID*)

the level of buffer can be accessed at any time.

The mtimechart webelement now displays 192 values, which is equivalent to a period of 2 days. Our buffer has space for 35040 values, which corresponds to ¼ hours values one year recording time. 11 shows the option for the user to represent the past values: It an be given a start- and end date. If more than the configured number of values in the web element are stored in the same period in the time series as the diagram adjusts the display so that it hides intermediate values.

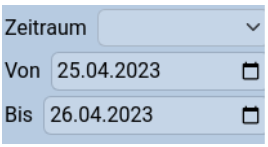


Figure 11: Change time range of TimeChart

Example: The user sets a period of four days (e.g. 2013-07-11 bis 2013-09-13). In the here given configuration in the time buffer (ID 0 und 1) 384 values are stored. The diagram can only display 192 values and shows therefore in presentation each second value, effectively ½ hour values over 4 days will be displayed. Values fluctuations that are present in ¼ hour intervals, are no longer displayed. Th time axis is scaled or adjusted to the time specified. If the user configures the date fields in different time intervals the axis is scaled so that the stored values are displayed from oldest to the newest date.

It is important to note: If the user moves or scales a diagram, he disconnect the diagram from the real-time web server, i.e. further changing of values, which are written in the time series (time buffer) are no longer visible on the web server until a page refresh (usually F5) of the browser is running. This does not affect the other elements of the website.

After the time series was taken over some time in the EibPC it has to be ensured that these are not los even if reloading of program or restarting the values. The functions

timebufferstore(*ChartBufferID*)

timebufferread(*ChartBufferID*)

are created for this task (comp. p. 219).

timebufferstore sets the values of the timebuffer with the *ChartBufferID* permanently into the flash memory of the EibPC , timebufferread reads a stored buffer back. In addition the values with EibStudio as described on page 23 to an external device to ensure data can be downloaded and uploaded.

Thus we store our buffer every 24 h in the following way:

```
// Wert im Flash speichern
if chtime(01,00,00) then {
        timebufferstore(ChartBuffer0);
        timebufferstore(ChartBuffer1);
} endif
```

The values we save back at startup as follows:

```
if systemstart() then {
        timebufferread(ChartBuffer0);
        timebufferread(ChartBuffer1);
} endif
```

Less „ease of operation“, especially in the application with a touch panel but more space for the representation is provides by the time charts without interval selection. In this form the diagram is similar to mcharts and mpcharts (comp. p. 218 and p. 218), where also the time axis is automatically scaled and taken out of the time buffer.

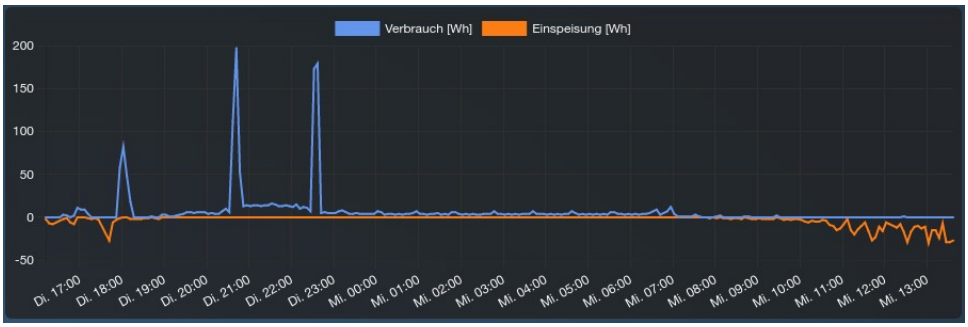


Figure 12: Default format

Also here: If the user moves or scales a diagram he disconnects the diagram from the real time webserver, i.e. further changing of values, which are written in the time series (time buffer) are no longer visible on the web server until a page refresh (usually F5) of the browser is running. This does not affect the other elements of the website.

The functions

mtimechartpos(TimeChartID,ChartIdx,ChartBuffer,StartPos,EndPos)

mtimechart(TimeChartID,ChartIdx,ChartBuffer,StartZeit,EndZeit)

(p. 221) change the visible data range of the chart.

mtimechartpos requires additionally to the ID and the graph index mtimechart the position of the value range of the data in the buffer to which the value is fixed. As indicated in 13 “numbers” the EibPC every space from 0 up to max. configured value n-1. In this case, n is the configured buffer length. Figure 13 shows a buffer with length 4000, start position 0 and end position 3999. With the help of mtimechartpos one can fall back to the specified position in the time buffer where position 0 is always the oldest value in the buffer and position n-1 (in the example, the 3999) is the most recent value in the buffer.

0_{u16}	1_{u16}	2_{u16}	3_{u16}	4_{u16}	5_{u16}		3998_{u16}	3999_{u16}
1.23 (4 Byte)	2.23 (4 Byte)	45.23 (4 Byte)	1.23 (4 Byte)	2.23 (4 Byte)	45.23 (4 Byte)	...	1.23 (4 Byte)	2.23 (4 Byte)
2013-11-08 8:00:00.223	2013-11-08 8:00:00.823	2013-11-08 8:03:00.223	2013-11-08 8:04:00.000	2013-11-09 8:00:00.700	2013-11-09 8:03:00.675		2013-11-18 14:30:00.223	2013-11-18 21:00:00.000

Figure 13: Structure of the timebuffer with index

Selecting the values on the position in the timebuffer with the function mtimechartpos() or the time with mitimechart()

mtimechart does not evaluate the index of the graph but the value of the timestamp itself. Here have to be specified the time statements StartTime,EndTime in the argument as utc-millisecond format. In order to simplify this for the user, you can fall back to the function

utc(*Zeit*)

(comp. 121). This converts a string specifying of the form $2013-01-30 14:00:00$ into the utc-millisecond format.

```
if systemstart() {
    mtimechart(1,0,ChartBuffer0,utc($2013-01-30-14-00-00$),utc($2013-01-30 14:00:00$))
} enduf
```

Change of the displayed buffer of a mtimechart

Of interest is the possibility to “separate” the pre-configured linking in the web element from time series to the graph and to display the graph in another buffer.

Here is another example: As shown in 14 should be taken a selection via a mpshifter-webelement, which is displayed in the recorded timebuffer.

Using the same diagram for different timebuffer: For this purpose the year will be chosen with the selection box at the bottom left. The application program sets up the connection of diagram graph to the destinated timebuffer.

Figure 14: Change of the presentation during running time

In the webserver the three elements shown are defined in which the pshifter is only used to display the current time. At the start of the application program the webelement ist linked to the timebuffer with ID chartbuffer3.

```
[WebServer]
page(PageID)[$Log$,$Room5$]
design $black$
mtimechart(TimeChartID)[LONG,2,255,30,17,256,0] ( $Room1$,LEFTGRAF, ChartBuffer3)
mpshifter(SelectID)[$2011$,$2012$,$2013$][DATE]$Room1$ pshifter(ClockID)[CLOCK]$Aktuelle Uhrzeit$
```

We define three time series (time buffer),

```
MemTyp=1
Len=30640u16
Datatyp=3.3f16
timebufferconfig(ChartBuffer0, MemTyp, Len, "RkWohnzimmerTemp-3/1/28")
timebufferconfig(ChartBuffer1, MemTyp, Len, "RkWohnzimmerTemp-3/1/28")
timebufferconfig(ChartBuffer2, MemTyp, Len, "RkWohnzimmerTemp-3/1/28")
```

which now record data for every 1 year in ¼ time:

```
Y2011=date(1,1,11) and !date(1,1,12)
Y2012=date(1,1,12) and !date(1,1,13)
Y2013=date(1,1,13) and !date(1,1,15)
if (mtime(45,00) or mtime(45,00) or mtime(15,00) or mtime(00,00) ) and Y2011 then {
        timebufferadd(ChartBuffer0,"RkWohnzimmerTemp-3/1/28");
} endif
if (mtime(45,00) or mtime(45,00) or mtime(15,00) or mtime(00,00) ) and Y2012 then {
        timebufferadd(ChartBuffer1,"RkWohnzimmerTemp-3/1/28");
} endif
if (mtime(45,00) or mtime(45,00) or mtime(15,00) or mtime(00,00) ) and Y2013 then {
        timebufferadd(ChartBuffer2,"RkWohnzimmerTemp-3/1/28");
} endif
```

If the user now changes the selection box the corresponding time buffer should be displayed:

Evaluate selection box

```
if mpbutton(SelectID,1,PageID)==255 then {
        mtimechartpos(TimeChartID,0,ChartBuffer0,0u16,30639u16);
        pdisplay(SelectID,$Es wird 2011 dargestellt$,DATE,DISPLAY,GREY,PageID,1)
} endif
if mpbutton(SelectID,2,PageID)==255 then {
        mtimechartpos(TimeChartID,0,ChartBuffer1,0u16,30639u16);
        pdisplay(SelectID,$Es wird 2012 dargestellt$,DATE,DISPLAY,GREY,PageID,2)
} endif
if mpbutton(SelectID,3,PageID)==255 then {
        mtimechartpos(TimeChartID,0,ChartBuffer2,0u16,30639u16);
        pdisplay(SelectID,$Es wird 2013 dargestellt$,DATE,DISPLAY,GREY,PageID,3)
} endif
```

It can be seen how the graph with index 0 of the mtimechart is “diverted” to the different time buffer via ID. We fall back to the function mtimechartpos, which links the year chart buffer each with the graph 0.

Even a small addition to the clock display: This is now shown in the exact seconds in visualization, because the real-time web server adjusts every change of the “second hand”.

Expert Functions

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

For all arguments or functions, the group addresses can also be used directly instead of variables.

Logical operators

AND

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.

Return value

- 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

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.

Return value

- 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

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.

Return value

- 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 compare values, the following operators are defined:

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 27.

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.

Return value

- Data type b01

Hysteresis

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 27.

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.

Return value

- 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

Return value

- 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.

Return value

- as *Operand*

Time

Set system time

Definition

- Function gettime(*address*) with:

Arguments

- 1 Argument of data type t24

Effect

- The system clock of EibPC is overwritten with the time stored in *address* and thus reset.

Return value

- 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 EibPC by the gettime function.

Send system time

Definition

- Function settime()

Arguments

- none

Effect

- The system time is read from the 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) and !day2)) then write(„WallClock24,settime()) endif
if (day(1) and !day(2)) then time=settime() endif
```

Set system date

Definition

- Function getdate(*Address*) with:

Arguments

- 1 Argument of data type d24.

Effect

- The system clock of the EibPC is overwritten with the time stored in *address* and thus reset.

Return value

- 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
```

Send system date

Definition

- Function setdate()

Arguments

- none

Effect

- The system date is read from the EibPC. The return value is the time in the format of type d24

Return value

- 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 EibPC and to be reset.

Implementation in the user program:

```
if (month(1,1)) then write("Date-3/5/3"d24, setdate()) endif
```

Set system time and date

Definition

- Function gettimedate(*address*) with:

Arguments

- 1 argument of data type y64

Effect

- The system clock and the system date of the EibPC are overwritten with the time and the date stored in *address* and thus reset.

Return value

- 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
```

Send system time and date

Definition

- Function settimedate()

Arguments

- none

Effect

- The system time and system date are read from the EibPC and assigned to a variable as a value

Return value

- 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 EibPC and to be reset.

Implementation in the user program:

```
if (month(1,1)) then write("RadioClock-1/2/1"d24, settimedate()) endif
```

Current hour

Definition

- Function hour()

Arguments

- none

Effect

- The system time (hour) is stored in a variable

Return value

- Data type u08

Example:

Stop watch see page 119

Current minute

Definition

- Function minute()

Arguments

- none

Effect

- The system time (minute) is stored in a variable

Return value

- Data type u08

Example:

Stop watch see page 119

Current second

Definition

- Function second()

Arguments

- none

Effect

- The system time (second) is stored in a variable

Return value

- 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 165).

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
```

Change hour

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 EibPC establishes an NTP connection, the time is reset again.

Return value

- none

Change minute

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 EibPC establishes an NTP connection, the time is reset again.

Return value

- none

Change second

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 EibPC establishes an NTP connection, the time is reset again.

Return value

- none

String in Unixtime (UTC)

Definition

- utc(*time*)

Arguments

- *time* (c) with format YYYY-MM-DD HH:MM:SS

Effect

- Time since 00:00:00 UTC on 1 Jan 1970 without leap seconds (Unixtime) until *time* in milliseconds (UTC).

Return value (u64)

Current time Unixtime (UTC)

Definition

- utctime()

Arguments

- none

Effect

- Time since 00:00:00 UTC on 1 Jan 1970 without leap seconds (Unixtime) until now in milliseconds (UTC).

Return value (u64)

Unixtime in String (UTC)

Definition

- utcconvert(*unixtime*)

Arguments

- *unixtime* (u64)

Effect

- Convert *unixtime* (time since 00:00:00 UTC on 1 Jan 1970 without leap seconds) in milliseconds into a String (UTC).

Return value (c1400)

- Format YYYY-MM-DD HH:MM:SS

Example:

```
// Current Unixtime (UTC)
unixtime=utctime()

// Convert specific unixtime (Mo 1. Apr 14:22:02 UTC 2013) in YYYY-MM-DD HH:MM:SS
DateTime=utcconvert(1364826122000u64)

// Convert 2012-09-03 20:00:17 in Unixtime (UTC). Result: 1346702417000
utcZ=utc($2012-09-03 20:00:17$)

// Days of February – leap year?
uDaysFeb2020=(utc($2020-03-01 00:00:00$) - utc($2020-02-01 00:00:00$))/(24u64*3600u64*1000u64)
uDaysFeb2019=(utc($2019-03-01 00:00:00$) - utc($2019-02-01 00:00:00$))/(24u64*3600u64*1000u64)
```

String in Unix time (local time)

Definition

- localtime(*time*)

Arguments

- *time* (c) with format YYYY-MM-DD HH:MM:SS

Effect

- Time since 00:00:00 UTC on 1 Jan 1970 without leap seconds (Unixtime) until *time* in milliseconds (local time).

Return value (u64)

Unix time in String (local time)

Definition

- localtimeconvert(*unixtime*)

Arguments

- *unixtime* (u64)

Effect

- Convert *unixtime* (time since 00:00:00 UTC on 1 Jan 1970 without leap seconds) in milliseconds into a String (local time).

Return value (c1400)

- Format YYYY-MM-DD HH:MM:SS

Example:

```
// Yesterday at the same time
now=utctime()
yesterdayLocal=localtimeconvert(now-(24u64*3600000u64))
```

Offset between local time and UTC

Definition

- difftime()

Arguments

- none

Effect

- Offset between local time and UTC in milliseconds. Represents offset due to the selected timezone and eventually daylight saving time with respect to UTC. For central europe with UTC+1, the function returns +1000s64 (CET) or +2000s64 (CEST).

Return value (s64)

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

Return value

- 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.

Return value

- 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.

Return value

- 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
```

Day of week

Definition

- Function dayofweek() with:

Arguments

- none

Effect

- The output returns the current day of the week [0{Sunday}..6{Saturday}.

Return value

- 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
```

Day (relative to) Easter Sunday

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.

Return value

- Data type u08

Month (relative to) Easter Sunday

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..

Return value

- 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

Day or night

The function sun returns whether it is day or night. It requires the EibPC's knowledge of the longitude and latitude of the concerned location.

These can be entered in EibStudio.

Definition

- Function sun()

Effect

- Return Value: The return value is 1 binary, if it is day and 0 binary, if it is night.

Return value

- 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

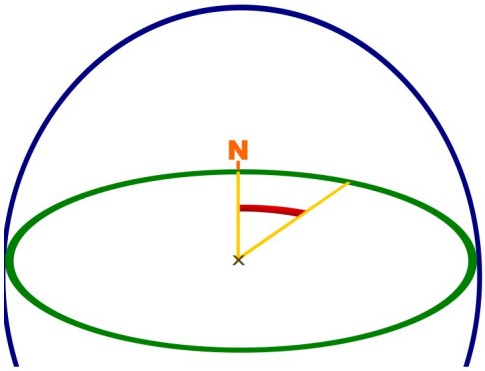
- Function azimuth()

Arguments

- None. However, the EibPC should know the longitude and latitude of the place. These can be entered in EibStudio (see page 126).

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 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

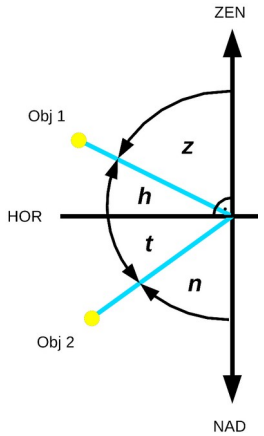
- Function elevation()

Arguments

- None. However, the EibPC should know the longitude and latitude of the concerned location. These can be entered in EibStudio (see page 126).

Effect

- This function cyclically (time frame: 5 minutes) calculates the elevation angle of the sun in degrees.



(Source: Wikipedia)

Return value

- Data type f32

Example 4: elevation

At 5:00, calculate the elevation angle of the sun at the location of the 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.

Time relative to sunrise/sunset

Definition

- Function presun(*hh,mm*)
  hh: hours (0... 23)
  mm: minutes (0... 59)

Arguments

- two arguments of data type u08

Effect

- Changes from 0b01 to 1b01 at the specified time before sunrise, and from 1b01 to 0b01 at the specified time before chancing from day to night.
- The program has to know the geographic coordinates.

Return value (b01)

- Sun position, 1b01= Day, 0b01 = Night

```
s=$$
if presun(1,30) then s=$Eine Stunde vor Sonnenaufgang$ endif
if !presun(0,20) then s=$20 Minuten vor Sonnenuntergang$ endif
```

Hour of sunrise

Definition

- Function sunrisehour()

Arguments

- none

Effect

- The hour (0 to 23) at sunrise is returned.

Return value

- Data type u08

Minute of sunrise

Definition

- Function sunriseminute()

Arguments

- none

Effect

- The minute (0 to 59) at sunrise is returned.

Return value

- 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
```

Hour of sunset

Definition

- Function sunsethour()

Arguments

- none

Effect

- The hour (0 to 23) at sunset is returned.

Return value

- Data type u08

Hour of sunset

Definition

- Function sunsetminute()

Arguments

- none

Effect

- The minute (0 to 59) at sunset is returned.

Return value

- 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

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 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 EibPC is used, which is given the EibPC either by the Internet or via a KNX system device.

Weekly timer

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 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).

Return value

- 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 timer

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 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).

Return value

- 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 timer

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 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).

Return value

- 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 timer

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 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).

Return value

- 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

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 EibPC is used, which is given the 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 timer

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 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.

Return value

- 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 timer

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 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).

Return value

- 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 timer

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 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.

Return value

- 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 timer

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 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.

Return value

- 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

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 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

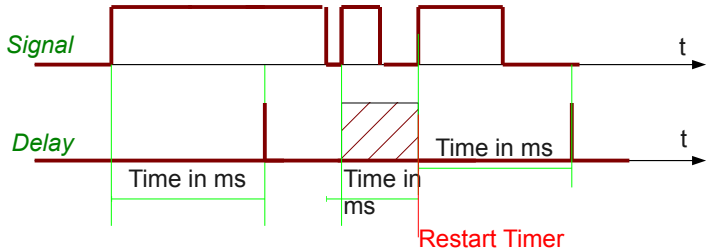
- Function delay(*Signal*, *Time*)

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.

Return value

- 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. 134).
- 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.

Return value

- 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

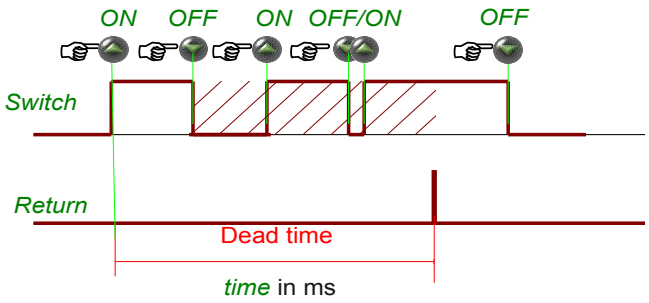
- Function after(*Signal*,*Time*)

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.

Return value

- 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. 135).
- 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.

Return value

- 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).

Return value

- 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 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 27).

Read from index

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 27). The return value is 0b01, when the read process was successful. If the read process failed, the function returns 1b01.

Return value

- Data type b01

Write at index

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.

Return value

- Data type b01

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
```

Read variable

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 27)). The return value is 0b01 when reading was successful, otherwise 1b01 is returned.
- The reading or de-referencing is performed via the variable name.

Return value

- Data type b01

Write variable

Definition

- Function writeflashvar(*Variable)*

Arguments

- *Variable* arbitrary data type

Effect

- The binary data of the memory content (see bit length, page 27) 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.

Return value

- Data type b01

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 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. 154)

```
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

Not only (logical and temporal) processes can be programmed by 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

Effect

- Return value: Absolute of *variable*

Return value

- 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 27

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 150)

Return value

- Data type of the arguments

Note:

With the same syntax, you can concatenate character strings (see page 161).

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

Return value

- 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

Return value

- 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

Return value

- 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

Return value

- 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)
```

Ceil

Definition

- Function ceil(*variable*)

Arguments

- *variable* is of data type f16, f32

Effect

- Smallest integer ≥ *variable*

Return value

- **Data type f32**

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 27

Effect

- Calculation of the quotient of Variable1 and Variable2

Return value

- 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

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.

Return value

- 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*

Return value

- 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)
```

Floor

Definition

- Function floor(*variable)*

Argumente

- *Variable* of data type f16, f32

Effect

- Biggest integer ≤ *variable*

Return value

- Data type f32

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.

Return value

- 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

Effect

- Return value: The maximum value of the given variables which must all be of the same data type

Return value

- 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 27

Effect

- Return value: The minimum value of the given variables which must all be of the same data type

Return value

- 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")
```

Mod

Definition

- Function mod(*variable1, variable2*)

Arguments

- all arguments are of the same data type
- Data type: uXX, sXX with XX arbitrary bit length

Effect

- *variable1* modulo *variable2*

Return value

- Data type of arguments

Multiplication

Definition

- *variable1 * variable2 [...]*

Arguments

- all arguments are of the same data type
- Data type: uXX, sXX and fXX, with XX arbitrary bit length

Effect

- The values of the variables are multiplied.

Return value

- 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.

Return value

- 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.

Return value

- 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.

Return value

- 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

Effect

- *variable1* is subtracted from *variable2*

Return value

- Data type of arguments

Tangent

Definition

- Function tan(*variable*)

Arguments

- 1 argument of data type f32

Effect

- Tangent of *variable*

Return value

- 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 EibPC.

Return value

- 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 176), 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

Effect

- Return value: The value of the variable which has changed most recently.

Return value

- 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.

Return value

- 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
```

Serial number

Definition

- Function devicenr()

Arguments

- none

Effect

- Serial number inquery of EibPC

Return value

- data type u32

Example: devicenr

The serial number should be assigned to the variable SNR.

Implementation in the user program:

```
SNR=devicenr()
```

Message log

Definition

- Function elog()

Arguments

- none

Effect

- Reading the oldest event stored item.
- After reading the log the entry is deleted.

Return value

- data type c1400 string

Example: see example elognum p.151

Elognum

Definition

- Function elognum()

Arguments

- none

Effect

- Returns the number of entries returned in the error memory.

Return value

- 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.

Return value

- Data type of argument

Example: Counter

You want to program a counter which increases a variable by 1 with every processing pass of the 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 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 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.

Return value

- 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(234f3
2)+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 EibPC, this function changes from ON to OFF during the first processing pass.

Return value

- 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 EibPC. An EibPC program is terminated by either disconnecting the power supply or by the user entering a new program. In the latter case, EibStudio sets the built-in variable SHUTDOWN ON so that the appropriate program can be executed in the user program. EibStudio then waits 5 seconds before the application program is stopped. Ongoing running of the Flash is still running properly.

Example see p. 141

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.

Return value

- 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
```

Passive Mode

Definition

- Function sleep(status)

Arguments

- 1 argument status of data type b01.

Effect

- If the input's value is OFF, the 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.

Return value

- none

Example: Put the EibPC to passive mode

You want to put an 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 EibPC, new programs can be tested (the web server can be accessed in the usual way). If the EibPC is in passive mode, its internal program runs normally, i.e. variables are being calculated, states changed, the web server adjusted, etc.

Create KNX telegram

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 EibPC internally works in the group message mode and therefore only logs group telegrams sent to a group address.

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:	15 .. 12	11 .. 8	7 .. 0
Address	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 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).

Return value

- 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

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 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).

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.

Return value

- none

Note:

1. It is possible to deactivate inputs differently in each scene number, see presetscene.
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“)
```

Preset scene

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.

Return value

- 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 scene

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*.

Return value

- 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
```

Call scene

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*.

Return value

- 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.

Return value

- 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..65534u16). It returns 65535u16 (constant EOS) if the character string has not been found

Return value

- 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.

Return value

- 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 162.

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*. If *data* is of type c, the terminating Zero byte of *data* is omitted.
- *pos*: The position of the bytes in *string* to be replaced. The number of bytes arises from the data type of *data.*

Return value

- 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. 25), 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 27.
- 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 (man 3 printf) (only permitted if no conversion into decimal notation)
  - 10: Alternative notation (man 3 printf) and a blank before a positive number (only permitted if *data* is of data type fXX)
  - 11: Alternative notation (man 3 printf) and a sign before a positive number (only permitted if *data* is of data type fXX)
  - 12: Alternative notation (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 (man 3 printf), zero-padded and a blank before a positive number (only permitted if *data* is of data type fXX)
  - 14: Alternative notation (man 3 printf), zero-padded and a sign before a positive number (only permitted if *data* is of data type fXX)
  - 15: Alternative notation (man 3 printf) and left-justified (only permitted if no conversion into decimal notation)
  - 16: Alternative notation (man 3 printf), left-justified and a blank before a positive number (only permitted if *data* is of data type fXX)

- 17: Alternative notation (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

Return value

- Data type c1400

Example: Stop watch V2 (Cf. Example:Stop watch, page 119).

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 119), the time difference is counted by means of after.

```
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
```

Typical configurations:

Value	Function arguments	Result	Meaning
pi=3.1415926535f32	stringformat(pi, 4, 1, 0, 2)	*$ 3.14$*	Space or minus sign, two digitals
	stringformat(-pi, 4, 1, 0, 1)	*-3.1*	one decimal
	stringformat(pi, 4, 6, 0, 2)	*3.14*	left-aligned, two decimals
	stringformat(pi, 4, 1, 0, 4)	*$ 3.1416$*	space or minus sign, four decimals
	stringformat(pi, 4, 1, 10, 4)	*$ 3.1416$*	10 chars incl. ".", fill left w/ spaces
e=.00000000000000000016f32	stringformat(e, 5, 6, 0, 2)	*$1.60e-19$*	Sci. notation
nowH=5u32	stringformat(nowH, 0, 3, 2, 2)	*05*	Fill left w/ 0, two digits
	stringformat(nowH, 0, 3, 4, 2)	*$ 05$*	Leading zero for two digits, fill with spaces for four characters
rgb1=0x0000ffu24 rgb2=255u24	stringformat(rgb1, 2, 18, 0, 6) stringformat(rgb2, 2, 18, 0, 6)	$0x0000ff$ $0x0000ff$	Convert to lower-case Hex w/ leading 0x and fill with 0 up to 6 digits

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.

Return value

- 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, encoding*)

Arguments

- *string* (c)
- *encoding* (c14) optional

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.
- If *encoding* is omitted, ASCII is used.
- See encode (p. 169) for values of *encoding*.

Return value

- 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

Return value

- 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

Return value

- 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.

Return value

- 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.

Return value

- 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

- urlencocode(*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.

Return value

- 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)
```

MD5

Definition

- 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)
```

Hash

Definition

- hash(*Algorithm, String, Length*)

Arguments

- *Algorithm* (u08)
- *String* (c)
- *Length* (u16) optional

Effect

- Return hash value as string of *String* with given *Algorithm*
- *Algorithm* must be one of:
  HASH_MD5=0u08,
  HASH_SHA1=1u08,
  HASH_SHA256=2u08,
  HASH_SHA512=3u08
- *Length* Bytes are hashed. Default: size(*String*)

Return value (c)

- Hexs string of hash in ASCII encoding (c1400)

Example

Get SHA1-Hash of string $Enertex$

```
sha1sum=sha1(HASH_SHA1, $Enertex$)
// Result: $1e00fa0ed981756b1fd4344a1467e8b6c52e476f$
```

Lower case

Definition

- tolower(*String*)

Arguments

- *String* (c)

Effect

- Convert all ASCII characters to lowercase

Return value (c)

- String length of *String*

Example

Convert $Enertex$ into lowercase

```
input1=$AlLeSgRosS$
lower_ascii=tolower(input1)
// Result: $allesgross$
```

Upper case

Definition

- toupper(*String*)

Arguments

- *String* (c)

Effect

- Convert all ASCII characters to uppercase

Return value (c)

- String length of *String*

Beispiel

Convert $Enertex$ into uppercase

```
input1=$AlLeSgRosS$
upper_ascii=toupper(input1)
// Result: $ALLESGROSS$
```

Base64 encode

Definition

- base64encode(*String, Length*)

Arguments

- *String* (c)
- *Length* (u16) (optional) length of the string to convert. Default: size(*String*).

Effect

- All characters of *String* (up to *Length*) are Base64-encoded. If *Length* is omitted, encoding of strings stops with the first 0-Byte. The 0-byte is not encoded.
- Please mind: Base64-encoding requires more bytes than the input. The data type of *String* must be large enough for the result.

Return value (c)

- String with the same size as *String*

Example

Encode the string $Enertex$ in base64

```
base64=base64encode($Enertex$)
// base64 is $RW5lcnRleA==$
```

Base64 decode

Definition

- base64decode(*String*)

Arguments

- *String* (c)

Effect

- All characters in *String* are decoded. Control characters are also decoded.

Return value (c)

- String with the same size as *String*

Example

Decode the base64 encoded string $RW5lcnRleA==$

```
plain=base64decode($RW5lcnRleA==$)
// plain is $Enertex$
```

TLS certificates, private keys, root certificates/CA certificates

Definition

- pem(*String*)

Arguments

- *String* (c)

Effect

- *String* is formatted into PEM format for functions, which require certificates.
- Required because strings cannot be defined with line breaks.
- To bundle multiple certificates, concatenate the single pem()-return values with CR.
- Please mind: certificates often require more than the default string length of 1400 characters.

Return value (c)

- String with the same size as *String*

Example

Accept the self signed certificate of a local web server

```
cert=pem($-----BEGIN CERTIFICATE-----
MIIDUDCCAjigAwIBAgIJALvECSjcmOhXMA0GCSqGSIb3DQEBCwUAMB8xHTAbBgNVBAMMFEVuZXJ0Z
XggRU5BIFNOMTExIENBMB4XDTIyMDgzMTEwNDgxOVoXDTM4MDExNzEwNDgxOVowHzEdMBsGA1UE
AwwURW5lcnRleCBFTkEgU04xMTEgQ0EwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDIyL
1tsDMp8d98yDHQPvWRUYZD5nyrHTmkdyiz4nckHvm9H8wx1bO8EjXn+m7AXdgIlRulf6Ni48alvnb77Ld9XgjI
LeHJUeuiX651OIDwR8BBYsQfLp5qzp/L5gwSDKo2Or1Hs+GISqedaLNN3+h/tit2d/
g04j9vjK5qE97HIKoRfJv0wVuuGtyy6azHwXGjbKYIFjbIDH+FXHpL5WTZScxyOyISVFCjXcYvuyWVGhQKSW
+vpOUA3S3IAWj7YB+yvINeEXYAZgZ5kcawa9dvVM/zdgoPe42cL8wuVRsBzng9XQjAcCqibv/
ComRCm4l6jhbJL2dWZCYcAtkZwQQ1AgMBAAGjgY4wgYswHQYDVR0OBBYEFMpsNzzdS9s7/
JfA2LIKn2z2m7m3ME8GA1UdIwRIMEaAFMpsNzzdS9s7/
JfA2LIKn2z2m7m3oSOkITAfMR0wGwYDVQQDDBRFbmVydGV4IEVOQSBTTjExMSBDQYIJALvECSjcmOh
XMAwGA1UdEwQFMAMBAf8wCwYDVR0PBAQDAgEGMA0GCSqGSIb3DQEBCwUAA4IBAQAJyPComoQF
ZrLG8rdd0yXEP3OuNsVjYxU4ZswZ56qWyrMk6aEHH2FghbEzERxjkdJGgNm7ZWpAhhlb0ZMfh0qUc9toQcN
vT7fRV7YXSRQ/dhkQFBeVVd0Dx75GFhqpDBf3GSwVZGM799nPPj3rPmxiXy9S6OQXyyKVrhoJyQ/
vTm3HX/URZ/
+05m8hdgcK6TZ6SNVCWPs07pUZgsMyZzf1Vzya3uOwaBHQ0C7alU+2PGPGUE3ld3uDzfyLnmt9NPvYFD
BHoqGiV3p82N1HUQfoJOh/
PkBLG9UqdTNVbraW+SE8ZHpeHyDcOLa3HKjgsmW4GoKryz6MUzuOxud8PvgC-----END
CERTIFICATE-----$c1400)

// cert is $-----BEGIN CERTIFICATE-----
MIIDUDCCAjigAwIBAgIJALvECSjcmOhXMA0GCSqGSIb3DQEBCwUAMB8xHTAbBgNV
BAMMFEVuZXJ0ZXggRU5BIFNOMTExIENBMB4XDTIyMDgzMTEwNDgxOVoXDTM4MDEx
NzEwNDgxOVowHzEdMBsGA1UEAwwURW5lcnRleCBFTkEgU04xMTEgQ0EwggEiMA0G
CSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDIyL1tsDMp8d98yDHQPvWRUYZD5nyr
HTmkdyiz4nckHvm9H8wx1bO8EjXn+m7AXdgIlRulf6Ni48alvnb77Ld9XgjlLeHJ
UeuiX651OIDwR8BBYsQfLp5qzp/L5gwSDKo2Or1Hs+GISqedaLNN3+h/tit2d/g0
4j9vjK5qE97HIKoRfJv0wVuuGtyy6azHwXGjbKYIFjbIDH+FXHpL5WTZScxyOyIS
VFCjXcYvuyWVGhQKSW+vpOUA3S3IAWj7YB+yvINeEXYAZgZ5kcawa9dvVM/zdgoP
e42cL8wuVRsBzng9XQjAcCqibv/ComRCm4l6jhbJL2dWZCYcAtkZwQQ1AgMBAAGj
gY4wgYswHQYDVR0OBBYEFMpsNzzdS9s7/JfA2LIKn2z2m7m3ME8GA1UdIwRIMEaA
FMpsNzzdS9s7/JfA2LIKn2z2m7m3oSOkITAfMR0wGwYDVQQDDBRFbmVydGV4IEVO
QSBTTjExMSBDQYIJALvECSjcmOhXMAwGA1UdEwQFMAMBAf8wCwYDVR0PBAQDAgEG
MA0GCSqGSIb3DQEBCwUAA4IBAQAJyPComoQFZrLG8rdd0yXEP3OuNsVjYxU4ZswZ
56qWyrMk6aEHH2FghbEzERxjkdJGgNm7ZWpAhhlb0ZMfh0qUc9toQcNvT7fRV7YX
SRQ/dhkQFBeVVd0Dx75GFhqpDBf3GSwVZGM799nPPj3rPmxiXy9S6OQXyyKVrhoJ
yQ/vTm3HX/URZ/+05m8hdgcK6TZ6SNVCWPs07pUZgsMyZzf1Vzya3uOwaBHQ0C7a
lU+2PGPGUE3ld3uDzfyLnmt9NPvYFDBHoqGiV3p82N1HUQfoJOh/PkBLG9UqdTNV
braW+SE8ZHpeHyDcOLa3HKjgsmW4GoKryz6MUzuOxud8PvgC
-----END CERTIFICATE-----$c1400
```

Parser

The following functions are useful to process the result of HTTP-Requests.

XML

Definition

- parsexml(*String, XPath, Return-Length*)

Arguments

- *String* (c)
- *XPath* (c)
- *Return-Length* (c)

Effect

- Parse the XML string *String* and return the XML nodes references with *XPath.*
  See https://www.w3schools.com/xml/xml_xpath.asp for a detailed description of XPath.
- Selected nodes can be single attributes, values and sub-trees. When multiple attributes are selected, only the last attribute is returned.
- If multiple nodes are selected, they are returned as child nodes of a new <root/> node converted into a string which can be parsed again.
- If nothing matches *XPath* the result is empty
- The argument *Return-Length* only defines the length of the returned value. Its value is never used.
- If *String or XPath* are empty, the result is empty

Return value (c)

- String length of *Return-Length*

Hint

- Array indices start with 1

Beispiel

Select an attribute from a non-empty node:

```
xml=$<root><node></node><node></node><node attr="attribute">content</node></root>$
attr=parsexml(xml, $//node[string-length() > 0]/@attr$, $$c9)
// Result: attr=$attribute$c9
```

JSON

Definition

- parsejson(*String, JSONPointer, Rückgabelänge*)

Arguments

- *String* (c)
- *JSONPointer* (c)
- *Rückgabelänge* (c)

Effect

- Parse the JSON string *String* and return the property references by *JSONPointer*.
  See https://tools.ietf.org/html/rfc6901 for a detailed description of JSONPointer.
- Selected properties can be single values (number, string) and object properties. Only a single property can be selected. Objects are returned as new JSON object which can be parsed again.
- If nothing matches *JSONPointer* the result is empty
- The argument *Return-Length* only defines the length of the returned value. Its value is never used.

Return value (c)

- String length of *Return-Length*

Hint

- Array indices start with 0

Beispiel

Select a property from a JSON object string

```
json=${"number": 5, "array": ["x","y"]}$
number=parsejson(json, $/number$, $$c1)
// Result: number=$5$c1

arrayElement=parsejson(json, $/array/1$,$$c1)
// Result:arrayElement=$y$, first element at index 0!
```

KNX Telegrams

Writing information to the KNX™ 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™ 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 26).

read

Send read request

Definition

- read(*GroupAddress*)

Arguments

- *GroupAddress*: Imported or manual KNX™ group address
- The groupaddress can be optionally negated using the !-Sign.

Effect

- A valid KNX telegram with the "read-flag" set is sent to the bus. Confirm, that the actors are parameterized properly (set read flag).

Return value

- 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 4.

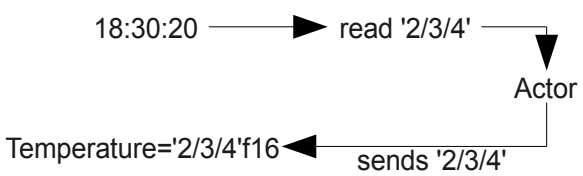


Figure 4: Operation of read

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

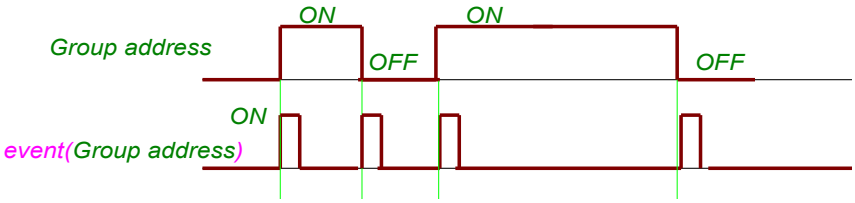
- Function event(*Group address*)

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

Definition

- Function eventread(*Group address*)

Arguments

- *Group address*: Imported or manual KNX™ group address
- The group address can be optionally negated using the !-Sign.

Effect

- Return value: 1b01 (ON pulse) when a Read-telegram with the group address has been written on the KNX™ bus, regardless of its content.

Data type results (Return value)

- Data type b01

eventresponse

Definition

- Function eventresponse(*Group address*)

Arguments

- *Group address*: Imported or manual KNX™ group address
- The group address can be optionally negated using the !-Sign.

Effect

- Return value: 1b01 (ON pulse) when an answer to a Read-telegram with the group address has been written on the KNX™ bus, regardless of its content.

Data type results (Return value)

- Data type b01

eventwrite

Definition

- Function eventwrite(*Group address*)

Arguments

- *Group address*: Imported or manual KNX™ group address
- The groupaddress can be optionally negated using the !-Sign.

Effect

- Return value: 1b01 (ON pulse) when an write-telegram with the group address has been written on the KNX™ bus, regardless of its content.

Data type results (Return value)

- Data type b01

writeresponse

Definition

- Function writeresponse(*Group address,value*)

Arguments

- *Group address*: Imported or manual KNX™ group address
- *Value:* The value which is to be written to the KNX™ group address (via the KNX™ bus)

Effect

- Responds to a read request by a valid telegram generated by KNX™ which writes the *value* to the *group address* is sent to the bus. The response flag is set in the telegram.

Data type results (Return value)

- none

Init group address

Definition

- initga(*GroupAddress*)

Arguments

- *GroupAddress*: Imported or manual KNX™ group address
- The groupaddress can be optionally negated using the !-Sign.

Effect

- The effect of this function is same as if the *GroupAddress* was listed in the [InitGA]-section.
- The function can be used top-level only, which means, that it can not be used in a then or else branch of an if-query.
- The function can also be used in related to the function comobject (p. 149)

Return value

- 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.

KNX-Telegram-Routing

With help of the functions address and readknx the EibPC can used as an free programmable router for KNX telegrams. If e.g. the group address is sent (as number) to the 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..

Return value

- 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.

Return value

- 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, routing-Counter, 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. 156

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 address.
- 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.

Return value

- 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.

Return value

- 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 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.

Return value

- 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 EibPC format when this group address has been imported into the application program (ESF import)

Return value

- 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

The ports via which the EibPC communicates can be changed via **Project Settings → Connection**.

UDP

The EibPC sends the data of a UDP transfer always from its port 4807, whereas the receiver's port can be chosen arbitrarily.

The 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 EibPC.

Receive UDP datagrams

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 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 EibPC always receives from port 4806. This port cannot be changed and must be taken into consideration by a UDP transmitter.

Return value

- 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 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.

Send UDP datagrams

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 EibPC.
- The 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.

Return value

- none

Example: Send UDP telegrams

Every 2 minutes, a UDP telegram shall be sent by the 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 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.

Return value

- none

Example: Send UDP telegrams

Every 2 minutes, a UDP telegram shall be sent by the 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 EibPC functions both as a server and as a client. Every 100 ms, it responds to a new connection request. If the EibPC is connected, it answer the requests with the cycle time of the processing cycle.

The TCP/IP server of the 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 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 EibPC disconnects automatically

Return value

- 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 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

Return value

- 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 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).

Return value

- 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 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.

Return value

- none

Example: Send TCP telegrams

Every 2 minutes, a TCP telegram shall be sent by the 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 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.

Return value

- none

Example: Send TCP telegrams

Every 2 minutes, a TCP telegram shall be sent by the 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
```

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

Return value

- 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 Hostname

Definition

- 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.

Return value

- 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)
```

Email

Before the function sendmail can be used, the basic e-mail configuration has to be done (see p. 22).

Plain-text email

Definition

- 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
- Return value Firmware > 4.113:
  0 = e-mail successfully sent
  1 = in progress
  2 = No system memory
  3 = Invalid server address
  4 = Authentication failed
  5 = TLS failed
  6 = Send failed, e.g, PLAIN oder STARTTLS not supported
  7 = Unexpected server response
  8 = Timeout after 5 s

Return value

- 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 192)

HTML mail

Before the function sendhtmlmail can be used, the basic e-mail configuration has to be done (see p. 22).

Definition

- 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
- Return value Firmware > 4.113:
  0 = e-mail successfully sent
  1 = in progress
  2 = No system memory
  3 = Invalid server address
  4 = Authentication failed
  5 = TLS failed
  6 = Send failed, e.g, PLAIN oder STARTTLS not supported
  7 = Unexpected server response
  8 = Timeout after 5 s

Return value

- 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 191).

VPN Server

Startvpn

Definition

- startvpn()

Arguments

- none

Effect

- Starts the VPN Service on the EibPC. The VPN must be configured with 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.

Return value

- none

Stopvpn

Definition

- Function stopvpn()

Arguments

- none

Effect

- Stops the VPN Service on the 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.

Return value

- **none**

Getvpnusers

Definition

- Function getvpnusers()

Arguments

- none

Effect

- Get a list of active VPN user

Return value

- **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.

Return value

- **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.

Return value

- **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

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. 195 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.

Return value

- 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.

Return value

- 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.

Return value

- 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

Return value

- 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

Return value

- 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

HTTP-Requests

Definition

- httprequest(*Type*, *URL*, *Query*, *Header*, *Body*, *TLS*, *Timeout*, *Priority*, *HTTP-Status*, *Reply-Header, Reply-Body*)

Arguments

- *Type* (u08)
  GET=0u08, POST=1u08, PUT=2u08, DELETE=3u08, PATCH=4u08
- *URL* (c) at most 256 characters
  Format:
  `http[s]://[user:password@]enertex.de[:Port]/complete/path`
- *Query* (c)
- *Header* (c)
- *Body* (c)
- *TLS* (b01)
  TLS_VERIFY_CERT=1b01, TLS_IGNORE_CERT=0b01
- *Timeout* (u08)
- *Priority* (u08)
- *HTTP-Status* (u16)
  Returns HTTP after execution (e.g., 200 on success)
- *Reply-Header* (c)
  Returns Header of server reply
- *Reply-Body* (c)
  Returns Body of server reply

Effect

- Send a HTTP request to the specified *URL*
- Use `https` instead of `http` in *URL* for encryption
- If *TLS* has the value TLS_IGNORE_CERT the server certificate is ignored
- If authentication is needed, pass username and password as part of *URL*
- Specify the remote port after the host. If omitted, the default ports 80/443 are used for http/https
- *Query* arguments must be separated by & and URL-encoded, e.g., `arg1=wert1&arg2=wert2`. They are added to the *URL* after ? internally
- The *Body* is transmitted without modification. Set encoding appropriately in the Header (Content-Type) if required.
- *Header* must be a list separated by LF, e.g.,
  `$Content-Type: application/json$+LF+$Accept: text/plain$`
  Default: `User-Agent: Enertex EibPC2`
- After *Timeout* seconds the request is canceled. Passing 0 uses the default timeout of 10 seconds.
- HTTP requests are executed sequentially. By setting a *Priority* urgent HTTP requests can be executed before others, e.g. turn on an IoT device when a telegram is sent has a higher priority than getting weather information. The least urgent priority is 0, the most urgent is 255.
- At most 10 HTTP requests are processed per second (Firmware < 4.105: 2 requests).
- At most 5 HTTP redirects are allowed, if the server answers with 3xx (Firmware < 4.008: no redirection at all).
- With Firmware > 4.110 redirects can be disabled: add 128 to parameter *Type,* e.g., GET without redirect: 128, POST without redirect 129.
- The function asynchronously returns values into its arguments *HTTP-Status, Reply Header, Reply Body*. **Always use unique return variables, never shared variables, e.g. $$!**

Return value

- 0u08: Success
- 1u08: Enqueued
- 2u08: Invalid arguments
- 3u08: Error during execution
- 4u08: Invalid URL or no connection to host
- 5u08: forbidded, e.g. authentication required nötig
- 6u08: server certificate invalid and option TLS_IGNORE_CERT not used
- 7u08: no reply during *Timeout*
- 8u08: too many requests pending (limit: 1000)
- 9u08: too many HTTP redirects
- The return values are updated asynchronously

Example

Daily check if a firmware update is available

```
// Arguments
timeout=5
priority=128
// Return values
status=255
httpstatus=0u16
header=$$
body=$$c65534

if systemstart() or htime(0,0,0) then \\
        status=httprequest(GET, $http://enertex.de/downloads/1159/VersionsLog.json$,\\
                $$,$$,$$,TLS_VERIFY_CERT,timeout,priority,httpstatus,header,body) endif

FirmwareV2=$$
if status == 0 then FirmwareV2=parsejson(body, $/FirmwareV2$, $$c5) endif
```

Modbus TCP

The EibPC² acts as Modbus TCP Master and Slave, i.e., it can read/write resources of other devices and provide its internal objects to be read by others.

Modbus resources are

- MB_COIL: 1 Bit, Addresses 1-9999
- MB_DISCRETE_INPUT: 1 Bit, read only, Addresses 10001-19999
- MB_INPUT_REGISTER: 16 Bit, read only, Addresses 30001-39999
- MB_HOLDING_REGISTER: 16 Bit, Addresses 40001-49999

A 0-based addressing scheme and an explicit selection of the resource type is used. To access the first Holding Register, use MB_HOLDING_REGISTER and index 0.

Modbus resources are 1 Bit or 16 Bit. The functions to read, write, and the Slave definitions map them to EibPC objects. Objects of type b01 are directly mapped to MB_DISCRETE_INPUT or MB_COIL, 16 Bit wide datatypes (e.g., u16) are directly mapped to MB_INPUT_REGISTER or MB_HOLDING_REGISTER.

When accessing multi-byte values, the byte order (Endianess) is important, as it defines the interpretation. Either the most-significant byte (Big Endian) or the least significant byte (Little Endian) is at the lowest address.

Byte-Order

A value of 0x1234 (decimal 4660) has two bytes Bytes 0x12 and 0x34. If the value is stored as 0x3412 (Little Endian) internally by a given device, the argument Byte-Order set to LITTLE_ENDIAN tells the EibPC to change its interpretation accordingly.

Word-Order

If the EibPC datatype is larger than the Modbus resource, neighboring resources are addressed. Separate single 1 Bit register can be read as a single u08. The order of separate data words (scalar values, here separate bits or 16-bit register values) is given by the argument *Word-Order*. A resource with a lower index has a higher significance for the result when using BIG_ENDIAN.

The following Bits 1, 0, 0, 1, 1, 0, 0, 0 starting with index 7 are interpreted as binary value 10011000 or hex 0x98 or decimal 152 when using BIG_ENDIAN, and interpreted as binary value 00011001 or 0x19 or decimal 25 when using LITTLE_ENDIAN.

Master

Similar to FTP functions (p. 195) a Modbus Master handle has to be created first. The handle stored the connection information used by the read and write functions. If the connection is interrupted, it is automatically reestablished.

Definition

- modbusmaster(*Host*, *Port*, *Timeout*, *Slave-Address*)

Arguments

- *Host* (c)
- *Port* (u16)
- *Timeout* (u32)
- *Slave-Address* (u08)

Effect

- Return a Modbus TCP handle to be used by readmodbus, writemodbus
- *Host* is a IP-Address string oder a hostname resolved on program start.
- The Modbus default *Port* is 502u16.
- *Timeout* in seconds defines how long to wait on a single resource.
- At most 10 read or write requests are processed per second (Firmware < 4.106: 2 requests).
- Most devices use a *Slave-Address* of 1u08 or 255u08.

Return value (u08)

- 0u08 Error
- Modbus Master handle to be passed to readmodbus and writemodbus

Read resource

Definition

- readmodbus(*Master-Handle*, *Type*, *Index*, *Return-Object*, *Byte-Order*, *Word-Order*)

Arguments

- *Master-Handle* (u08)
- *Type* (u08)
- *Index* (u16)
- *Return-Object* (b01, b02, b04, u08, s08, u16, s16, f16, u24, s24, u32, s32, f32, u64, s64)
- *Byte-Order* (u08)
- *Word-Order* (u08)

Effect

- Read the current value from a Modbus resource of *Type*, starting at *Index*, and write the result into *Return-Object*
- *Type* must be one of MB_DISCRETE_INPUT, MB_COIL, MB_INPUT_REGISTER, MB_HOLDING_REGISTER
- The Bit or Byte order when mapping the resource to *Return-Object* is defined by *Byte-Order* (u08) and *Word-Order* (u08)
- The function asynchronously returns values into its arguments

Return value (u08)

- 0u08 Success
- 1u08 Executing
- 2u08 Error

Example

```
Every 10 seconds an energy storage shall be queried for effective power and
charge state, and respective variables must be updated. Slave address (unit ID)
is 255, the port 502 (default).
```

1066	active power	R	SINT16	1	W	measured at internal inverter	positive: charge negative: discharge	✔	✔	✔	✔	✔
1067	apparent power	R	SINT16	1	VA	measured at internal inverter	positive: charge negative: discharge	✔	✔	✔	✔	✔
1068	SOC	R	UINT16	1	%	total state of charge		✔	✔	✔	✔	✔

Figure 15: Modbus-Register of energy storage (source: Varta)

```
mm1=modbusmaster($192.168.1.100$, 502u16, 10u32, 255)
activePower=0s16
stateCharged=0u16
status=0
if cycle(0,10) then {
   status=readmodbus(mm1, MB_INPUT_REGISTER, 1066u16, activePower, BIG_ENDIAN, BIG_ENDIAN);
   status=readmodbus(mm1, MB_INPUT_REGISTER, 1068u16, stateCharged, BIG_ENDIAN, BIG_ENDIAN);
} endif

if status == 2 then {
… // Error
} endif
```

Write resource

Definition

- writemodbus(*Master-Handle*, *Type*, *Index*, *Source-Object*, *Byte-Order*, *Word-Order*)

Arguments

- *Master-Handle* (u08)
- *Type* (u08)
- *Index* (u16)
- *Source-Object* (b01, b02, b04, u08, s08, u16, s16, f16, u24, s24, u32, s32, f32, u64, s64)
- *Byte-Order* (u08)
- *Word-Order* (u08)

Effect

- Write the current value of *Source-Object* into the Modbus resource of *Type,* starting from *Index*.
- *Type* must be one of MB_COIL, MB_HOLDING_REGISTER
- The Bit or Byte order when mapping the value of *Source-Object* to the Modbus resource is defined by *Byte-Order* (u08) and *Word-Order* (u08)
- The function asynchronously returns values into its arguments

Return value (u08)

- 0u08 Success
- 1u08 Executing
- 2u08 Error

Example

```
Change the scaling of the effective power for the energy storage above.
```

Figure 16: Modbus-Register of energy storage (Quelle: Varta)

```
mm1=modbusmaster($192.168.1.100$, 502u16, 10u32, 255)
status=0
if cycle(0,10) then {
   status=writemodbus(mm1, MB_HOLDING_REGISTER, 2066u16, -3s16, BIG_ENDIAN, BIG_ENDIAN);
} endif

if status == 2 then {
… // Error
} endif
```

Slave

Acting as Modbus TCP Slave the EibPC² other Modbus TCP Master can read the current status ob internal objects. These values are updated every 5 seconds.

The number of simultaneous Modbus TCP Master connections is limited to 4.

The TCP port can be changed. The default Modbus TCP port is 502. (p. 22)

All Modbus master devices have access the same resources.

Definition

- modbusslave(*Type*, *Index*, *Source-Object*, *Byte-Order, Word-Order*)

Arguments

- *Type* (u08)
- *Index* (u16)
- *Source-Object* (b01, b02, b04, u08, s08, u16, s16, f16, u24, s24, u32, s32, f32, u64, s64)
- *Byte-Order* (u08)
- *Word-Order* (u08)

Effect

- Maps the *Source-Object* to Modbus resources of *Type* at *Index* to be read by other Modbus TCP Master devices
- *Type* must be one of MB_DISCRETE_INPUT, MB_COIL, MB_INPUT_REGISTER, MB_HOLDING_REGISTER
- The Bit or Byte order when mapping the *Source-Object* is defined by *Byte-Order* (u08) and *Word-Order* (u08)
- The function asynchronously returns values into its arguments

Return value (u08)

- 0u08 Modbus resource correctly created
- 1u08 Creating modbus resource
- 2u08 Error

Example

```
The EibPC shall be queried by a Modbus TCP master. Register address 0 maps a 1-
Bit-Value and register addresses 100/101 (two sequential registers, each 16-Bit)
map a 32-Bit value.
```

```
flag=1b01
val=0x12345678u32
modbusslave(MB_COIL, 0u16, flag, BIG_ENDIAN, BIG_ENDIAN);
modbusslave(MB_INPUT_REGISTER, 100u16, val, BIG_ENDIAN, BIG_ENDIAN);
```

MQTT

The EibPC² with Option NP has support for MQTT for simple data exchange with other devices. The integrated broker accepts and distributes messages.

Configure a MQTT client handle to process messages sent by other clients, e.g., to forward them as KNX group adddress telegram. This is also required if the internal MQTT broker is used.

MQTT Broker

Definition

- startmqttbroker(*Port*, *TLS*, *Username*, *Password*)

Arguments

- *Port* (u16) default port 1883u16 unencrypted, 8883 with TLS
- *TLS* (b01) enable encryption
- *Username* (c) Username for authentication
- *Password* (c) Password for authentication

Effect

- Start the integrated MQTT Broker den integrierten MQTT-Broker.
- If *TLS* is enabled (=1b01), the communication is encrypted with TLS. The webserver-certificate is used as server certificate.
- If *Username* and/or *Password* are empty strings, authentication is disabled.
- Up to 100 concurrent Clients are supported.
- If the broker is already running, it is only restarted if *Port* or *TLS* change. Otherwise the user configuration is reloaded.

Return value (u08)

- 0u08: the MQTT Broker is started and running
- 1u08: starting
- 2u08: stopped
- 3u08: start failed, e.g., no server certificate but *TLS* is enabled
- 4u08: configuration error
- 5u08: configuration reloaded
- The return value is updated asynchronously.

Example

```
Start the MQTT broker when the EibPC starts. TLS is disabled, but clients must
authenticate with username and password (eibpc:secret).
```

```
uBrokerStatus=255
if systemstart() then uBrokerStatus=startmqttbroker(1883u16, 0b01, $eibpc$,
$secret$) endif
```

Definition

- stopmqttbroker()

Arguments

- none

Effect

- Stop the running MQTT broker

Return value (u08)

- none

MQTT-Client

Definition

- mqttclient(*Host*, *Port*, *TLS, Username, Password, ValidateServerCert, CACert, ClientCert, ClientKey*)

Arguments

- *Host* (c) Hostname or IP address as string
- *Port* (u16) default port 1883u16 unencrypted, 8883 with TLS
- *TLS* (b01) enable encryptiont
- *Username* (c) Username or empty string
- *Password* (c) Username or empty string
- *ValidateServerCert* (b01) TLS_VERIFY_CERT or TLS_IGNORE_CERT
- *CACert* (c) Root certificate to validate Server certificate, PEM format
- *ClientCert* (c) Client certificate, PEM format
- *ClientKey* (c) Unencrypted private key for Client certificate, PEM format

Effect

- Creates an MQTT client connection handle. Up to 4 handles are supported.
- Connection is opened automatically. If the connection fails, the EibPC tries again after 60 seconds.
- If *Username* or *Password* is empty, authentication is disabled.
- If *ValidateServerCert* is TLS_VERIFY_CERT=1b01, the server address is verified. Only active with TLS. Expired or self-signed certificates are not accepted with TLS_VERIFY_CERT=0b01.
- If *CACert* is empty, the integrated certificates are used to validate the server, if TLS is enabled.
- If *ClientCert* and *ClientKey* are not empty, the client presents the certificate to authenticate the user to the server of TLS is enabled.
- MQTT client ID is fixed to "eibpc-<serial number>-<handle>".

Return value (u08)

- 0u08 Error
- MQTT handle (u08 > 0u08) for the functions subscribemqtt, unsubscribemqtt, publishmqtt.

Definition

- subscribemqtt(*Handle*, *Topic*, *QualityOfService*, *Result*, *[ResultTopic]*)

Arguments

- *Handle* (u08) Connection handle from mqttclient
- *Topic* (c)
- *QualityOfService* (u08) valid values: 0u08 (QoS 0), 1u08 (QoS 1), 2u08 (QoS 2)
- *Result* (Variable of type b01, b02, b04, u08, s08, u16, s16, f16, u24, u32, s32, f32, u64, s64, cXXXXX)
- *ResultTopic* (c) optional

Effect

- Subscribes an MQTT topic.
- The topic can contain wildcards:
  - sensors/+/temp for a single level
  - sensors/# for all topics of all (sub-)levels. # must be the last character.
- The connection to Broker is opened if required.
- *QualityOfService* steuert die Zuverlässigkeit der Zustellung:
  - QoS 0: simple delivery
  - QoS 1: guaranteed delivery
  - QoS 2: exactly-once delivery
- Every message changes the *Result* object if the data differs.
- If *Result* is changed, *ResultTopic* contains the topic of the message if provided. If the subscription topic contains wildcards, it can be used to decide how to parse the message.
- The message is decoded according to the type of *Result*. Many devices however send string messages. *Result* must also be a string. It can then be processed further, e.g, with parsejson or convert.

Return value (u08)

- 0u08 Success
- 1u08 Error
- 2u08 Subscription exists
- 3u08 Max. number of subscriptions reached

Example

```
The integrated MQTT-Broker is enabled. Changes of an MQTT topic shall be mapped
to a group address.
uMqttHandleEibPC = mqttclient($localhost$, 1883u16, AUS, $eibpc$, $secret$,
TLS_IGNORE_CERT, $$,$$,$$ )
zStatus=$$c3
if  uMqttHandleEibPC > 0 then {
iSubscriptionStatus=subscribemqtt(uMqttHandleEibPC, $stat/tv/POWER$, 0,
zStatus);
} endif
if zStatus == $OFF$ then write("Status-13/1/9", 0b01) endif
if zStatus == $ON$ then write("Status-13/1/9", 1b01) endif
```

Definition

- unsubscribemqtt(*Handle*, *Topic*, *Result*)

Arguments

- *Handle* (u08) Connection handle from mqttclient
- *Topic* (c) Topic used by subscribemqtt
- *Result* Object used by subscribemqtt

Effect

- Remove the subscription for the *Result* object. Other subscriptions (also for the same topic) with different result objects remain active.
- *Result* is not changed but only used to identify the subscription.

Return value (u08)

- 0u08 Success
- 1u08 Error

Definition

- publishmqtt(*Handle*, *Topic, QualityOfService, Retain, Object, Size*)

Arguments

- *Handle* (u08) Connection handle from mqttclient
- *Topic* (c) Topic without wildcards
- *QualityOfService* (u08) see subscribemqtt
- *Retain* (b01)
- Objekt (b01, b02, b04, u08, s08, u16, s16, f16, u24, u32, s32, f32, u64, s64, c)
- Size (u16) Anzahl der Bytes, die gesendet werden sollen

Effect

- Send *Object* top the MQTT broker.
- The payload contains the raw data of object, optionally truncated to *Size*.
- *Size* is the number ob bytes to be sent. If *Size* == 0u16, numerical objects are sent in-total, String objects are truncated to the actual length of the string (size*(Object)*).
- *Retain* notifies the broker to store the message and automatically send it to new subscribers of a matching topic.

Return value (u08)

- 0u08 Success
- 1u08 Error

Example

```
The integrated MQTT-Broker is enabled. Group address writes shall be forwarded
to an MQTT topic.
```

```
uMqttHandleEibPC = mqttclient($localhost$, 1883u16, AUS, $eibpc$, $secret$,
TLS_IGNORE_CERT, $$,$$,$$ )

if eventwrite("TV-13/1/8") and "TV-13/1/8"==1b01 then {
    publishmqtt(uMqttHandleEibPC, $cmnd/tv/Power$, 0, 0b01, $ON$, 0u16);
} endif
if eventwrite("TV-13/1/8") and "TV-13/1/8"==0b01 then {
    publishmqtt(uMqttHandleEibPC, $cmnd/tv/Power$, 0, 0b01, $OFF$, 0u16);
} endif
```

Visualization

To be able to use the web visualization of the EibPC, you must activate the NP option in the EibPC. The unlock code is always bound to the serial number of the device and is not transferable to other devices.

The following functions are used to access visualization elements.

Visualization elements are divided into global and page-related elements (see p. 18).

Switches

Visualizations created via Visu always use page-related elements, if available. How to create a web visualization in Expert itself is described in Visualization in Expert (p. 44).

Button pressed (global)

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*, 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).

Return value

- Data type u08, values 0,1,2,3,4

Button pressed (page-dependent)

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* 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).

Button with selection (global)

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,* 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).

Return value

- Data type u08, values 0,1,2,3,4.

Button with selection (global)

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 pressing 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*, the function returns 1 for a single cycle. When the selected entry is changed to *selection*, it returns 255. Otherwise, it returns zero.
- 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).

Return value

- Data type u08, values 0,1,2,3,4.

Change switch (global)

Definition

- display(*id, text, icon, state, style, [mbutton]*)
- 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 listed in 3 (page 86)
- 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 2 (page 86).
- The text to be displayed can be represented in the stylesGREY (==0), GREEN (==1), BLINKRED(==2) and BLINKBLUE (==3).

Return value

- 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 runtime, the index *ClockWebID* will still use its initial value, which is 0.

Change switch (page-dependant)

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 in 3.
- 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 2 (page 86).
- The text to be displayed can be represented in the styles GREY (==0), GREEN (==1), BLINKRED(==2) and BLINKBLUE (==3).

Return value

- none

Slider

Get value (global)

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.

Return value

- Data type u08

Get value (page-dependant)

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.

Return value

- Data type u08

Get value of extended Slider (global)

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.

Return value

- Data type f32

Get value of extended Slider (page-dependant)

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.

Return value

- Data type f32

Set slider value (global)

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 in 3 (page 86) 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*. 2 (page 68) provides an overview over all possible states.

Return value

- none

Set slider value (page-dependant)

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 in 3 (page 86) 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*. 2 (page 68) provides an overview over all possible states.

Return value

- none

Set extended slider value (global)

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 in 3 (page 86) 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*. 2 (page 68) provides an overview over all possible states.

Return value

- none

Set extended slider value (page-dependant)

Definition

- Function setpeslider(*id, value, icon, state page_id*)

Arguments

- All arguments of data type u08

Effect

- The function addresses the *peslider* that refers to a page at the *id* on page *page_id* and sets it to the value *value*. If there are multiple occurrences of *id*, all elements of this id on the web page with *page_id* are addressed.
- A call of the function sets the icon to the symbol with the number *icon*. Possible symbols are shown in 3 (page 86) 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*. 2 (page 68) provides an overview over all possible states.

Return value

- none

Pictures

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 picture element. If there are multiple pictures with *id* on the web page of *page_id*, they all will be addressed.
- 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.

Return value

- none

Links

External link (page-dependant)

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 in 3 (page 68).
- 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 2 (page 68).
- 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.

Return value

- none

Change link to visu page (page-dependant)

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 in 3 (page 86).
- 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

Return value

- 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
```

Value Charts

Chart with single graph (global)

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).

Return value

- 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
```

Chart with single graph (global)

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).

Return value

- **Data type u08 (internal state of the webchart).**

Chart with up to four graphs (global)

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.

Return value

- u08 (internal state).

Chart with up to four graphs (page-dependant)

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.

Return value

- u08 (internal state).

TimeCharts

Configure TimeBuiffer

Definition

- Function timebufferconfig(*ChartBufferID, MemTyp, Laenge, DataTyp*)

Arguments

- *ID* of data type u08
- *MemTyp* Memory Type, with "0" ring memory and "1" represents a linear memory.
- *Length* of the data in the puffer. Maximum 65535 records with max. 4 bytes in length. The data type has to be u16.
- The memory is of data type *DataTyp* of the input object.

Effect

- There is a pair of values buffer is created or configured here. It can be set using the memory type, if this becomes full after filling with the values or if the oldest value is discarded.
- CAUTION: The EibPC has a RAM of 64MB, of which about 40 MB can be used by the user maximum.

  To ensure proper operation, the buffer and arts must be sized so that the memory of the EibPC is not overloaded. Using the function to buffer 255 for storing history data can be defined. The following applies for the necessary storage capacity = (number of values) * 12 Thus, for example, has a buffer with 65000 values about 780 kB.
- You can store them in the Flash buffer at any time, so when you restart the values are not lost, see timebufferstore and timebufferread.

Return value

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

Store value

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

Return value

- 0 success, 1 error

Clear all values

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)

Return value

- Level of the time buffer of the data type u16

Example

```
if systemstart() then timebufferclear(2) endif
```

Store TimeBuffer

Definition

- Function timebufferstore(*ChartBufferID*)

Arguments

- *ChartBufferID* of data type u08

Effect

- It is permanently stored in a flash buffer.

Return value

- 0 success, 1 error

Read TimeBuffer from flash

Definition

- Function timebufferread(*ChartBufferID*)

Arguments

- *ChartBufferID* of data type u08

Effect

- A buffer is selected from the Flasch.

Return value

- 0 success, 1 error, 2 ongoing processing, data type u08

Filling level

Definition

- Function timebuffersize(*ChartBufferID*)

Arguments

- *ChartBufferID* of data type u08

Effect

- Show the current level of the time buffer.

Return value

- Level of the time buffer of data type u16

Get value

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.

Return value

- 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
```

Change time range shown in TimeChart

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.

Return value

- none

Change position interval chown in TimeChart

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.

Return value

- no

Inputs

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

Return value

- string c1400 as result

Output

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

Return value

- none

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

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

Macros

With macros, also named functional blocks, programming the EibPC is

- substantially simplified for the beginner and
- faster for the experienced user. The user can extract 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.
- The macro-wizard 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.

Definition

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.

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 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:

```
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 EibStudio.

However, this method possibly may become error-prone.

A macro starts with :begin

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:

... 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 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 compiled together with the other program code. You can look at your macro code generated by the compiler in the file „tmpMacroOut.txt” in the working directory of the EibStudio.

If the above macro is saved e.g. as myMakros.lib, the “double-click” on a KNX button is simplified:

```
DoubleClick(Basement,'0/0/1'b01,ON,'3/4/5'b01,ON,'3/4/6'b01,ON)
```

Now the compiler writes in our example „tmpMacroOut.txt“ (in the working directory of the EibStudio):

```
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.

Arguments are only replaced within strings if they are surrounded by separators. If a macro with argument

```
:begin stringTest(arg)
```

is used like in

```
stringTest(Parameter)
```

the argument is replaced as in the following table:

$ arg $	<space>Parameter<space>
$-arg+$	-Parameter+
$_arg_$	_Parameter_
$^arg^$	Parameter
$Text arg$	Text arg
$Text arg^$	Text Parameter
$Text ^arg^$	Text Parameter

Runtime errors and syntax errors

Runtime errors or syntax errors due to the erroneous use of e.g. group address assignments first occur at the "expansion" of the macro.

Macro wizard

You can document your macros directly in the source code for the application. For this, the keyword :info exists. At the first position after the keyword the description of the function is located, followed by a description of each argument. The descriptions are enclosed by two "$" character.

You can generate the description by yourself with ":info".

Each description of the arguments is enclosed by two $ characters.

```
:info $With this function block, you can realize a double-click on a button:\\
    If you press the button twice within 0.8 seconds, another function is triggered than if you press once.\\
    You can control both actions by this function block macro$\\
  $Name of the button (for the purpose of unambiguousness)$\\
  $Group address to which the button sends values$\\
  $The value sent by the button (e.g. ON or OFF)$\\
  $Group address for a telegram at single-click$\\
  $Value for the telegram at single-click (e.g. ON or OFF or 23%)$\\
  $Group address for a telegram at double-click$\\
  $Value for the telegram at double-click (e.g. ON or OFF or 23%)$
```

In order to use a the wizard or re-parametrize your macros, these have to be coded in the [Macros] section.

Local Variables

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).

Return Values

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
```

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
```

is still equivalent to the code above. If the last line before :end is empty or only spaces, no return value is defined. So it is a good coding style always to use :return. :return can be placed anywhere in the code of the macro.

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

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

:end
```

Use it as built-in

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

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

Online debugging at runtime

If variables are to be monitored at runtime, it is recommended to debug with UDP telegrams and a netcat client (see https://de.wikipedia.org/wiki/Netcat).
The following code is used as a debug macro, assuming that the remote 192.168.1.18 listens on port 9000, e.g. Configured with the Unix tool netcat -ul 9000:

Sending a string with CR to a UDP client

Empty macro

```
#define DEBUG
#ifdef DEBUG
// Debugger an 192.168.1.118 an Port 9000u16
:begin vmDebugUDP(cString)
:return  {
        sendudp(9000u16, 192.168.1.18, cString+tostring(0x0d,0x0a));
}
:end
#endif
#ifndef  DEBUG
:begin vmDebugUDP(cString)
:return __EMPTY()
:end
#endif
```

Depending on whether debugging is enabled with #define DEBUG, a message is sent via UDP. In the event that the #define DEBUG is not commented, no messages will be sent. A special feature is the use of __EMPTY(). This statement ensures that the macro does not expand and does not generate any code.

Efficient for inactive #define of DEBUG

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

Now with active #define DEBUG via UDP the value is automatically transferred to the receiver at runtime of the program. If // #define DEBUG is uncommented, the line vmDebugUDP ($ x is now $ + convert (x, $$)) does not create any overhead.

If, on the other hand, an If statement is **just** set up for debug purposes, for example:

```
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..

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: Datatypes of parameters are not the same: >Var1+Var2< !	Var3=convert(Var1,Var2) + Var2
Syntax error in line:[17] >if (("EntireKitchen-1/1/9"==On) and wtime(23,00,00,00)) < Valid until position: STOP--> and wtime(23,00,00,00))	The instruction must be positioned in one line or the line must be finished with ' \\'. if ..... and \\ then ....
! Predefined variable cannot be re-defined in >EIN=1b01< !	In the EibParser, variables are predefined to make the construction of a user program as simple as possible. The predefined variables are listed in the EibStudio in the right section of the window. They cannot be defined again.
Datatypes of parameters are not the same: >sun()==1< !	The return value of the function is binary. A number without the definition of a data type is always an unsigned 8 bit value. As a relational operator, a binary value must be given. sun()==1b01
Syntax error in line:[13] >a=4,6e1f32< Valid until position: STOP--> ,6e1f32	As a decimal point, always “.” has to be used.
Syntax error in line:[21] >"Akt1-0/0/5"=after(a,5000u64)<	A direct assignment is only possible for variables, not for addresses. Writing information to the KNX bus is realized with the help of the write function. write(„Akt1-0/0/5“, 1b01)
Syntax error in line:[19] >if (a==EIN) then write("Akt1-0/0/5",EIN) write("Akt2-0/0/6",EIN);write("Akt3-0/0/8",EIN); write("Ak4-0/0/7",EIN) endif<	Multiple instructions in an if statement must be separated by “;“. if(a=EIN) then write(b=EIN); write(c=AUS) endif
Syntax error in line:[26] >write(on,ON)< data type is unkown in >write(on<	The write function can only affect group addresses (1st argument), not variables.
Deklaration der Variable muss eindeutig sein in >u=convert(z,r)-r-e<	Every variable may be declared only once. An additional declaration produces this error messages.
Wrong data type in >cycle(0.5,5<	Only integer values may be entered.

Licenses

The EibPC² uses Software under various licenses. If required by the respective license, the source code is provided upon request.

Enertex® EibPC²

Betriebssystem: Debian Linux 9: Kernel 4.14.16

EibStudio

Please see **Help → Licenses** for a complete list.

The following libraries are used:

libcurl

```
COPYRIGHT AND PERMISSION NOTICE

Copyright (c) 1996 - 2020, Daniel Stenberg, <daniel@haxx.se>, and many
contributors, see the THANKS file.

All rights reserved.

Permission to use, copy, modify, and distribute this software for any purpose
with or without fee is hereby granted, provided that the above copyright
notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN
NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE
OR OTHER DEALINGS IN THE SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not
be used in advertising or otherwise to promote the sale, use or other dealings
in this Software without prior written authorization of the copyright holder.
```

zlib

```
(C) 1995-2017 Jean-loup Gailly and Mark Adler

  This software is provided 'as-is', without any express or implied
  warranty.  In no event will the authors be held liable for any damages
  arising from the use of this software.

  Permission is granted to anyone to use this software for any purpose,
  including commercial applications, and to alter it and redistribute it
  freely, subject to the following restrictions:

  1. The origin of this software must not be misrepresented; you must not
     claim that you wrote the original software. If you use this software
     in a product, an acknowledgment in the product documentation would be
     appreciated but is not required.
  2. Altered source versions must be plainly marked as such, and must not be
     misrepresented as being the original software.
  3. This notice may not be removed or altered from any source distribution.

  Jean-loup Gailly        Mark Adler
  jloup@gzip.org          madler@alumni.caltech.edu

If you use the zlib library in a product, we would appreciate *not* receiving
lengthy legal documents to sign.  The sources are provided for free but without
warranty of any kind.  The library has been entirely written by Jean-loup
Gailly and Mark Adler; it does not include third-party code.

If you redistribute modified sources, we would appreciate that you include in
the file ChangeLog history information documenting your changes.  Please read
the FAQ for more information on the distribution of modified source versions.
```

json-c

```
Copyright (c) 2009-2012 Eric Haszlakiewicz

Permission is hereby granted, free of charge, to any person obtaining a
copy of this software and associated documentation files (the "Software"),
```

libmodbus

introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

GNU LESSER GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) The modified work must itself be a software library.

b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME

```
THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

  16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN
WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY
AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU
FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR
CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE
LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING
RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A
FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF
SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH
DAMAGES.

                     END OF TERMS AND CONDITIONS
```

libxml

```
Except where otherwise noted in the source code (e.g. the files hash.c,
list.c and the trio files, which are covered by a similar licence but
with different Copyright notices) all the files are:

 Copyright (C) 1998-2012 Daniel Veillard.  All Rights Reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is fur-
nished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FIT-
NESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.  IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
```

OpenSSL

```
  LICENSE ISSUES
  ==============

  The OpenSSL toolkit stays under a double license, i.e. both the conditions of
  the OpenSSL License and the original SSLeay license apply to the toolkit.
  See below for the actual license texts.

  OpenSSL License
  ---------------

/* ====================================================================
 * Copyright (c) 1998-2019 The OpenSSL Project.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. All advertising materials mentioning features or use of this
 *    software must display the following acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit. (http://www.openssl.org/)"
 *
 * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
 *    endorse or promote products derived from this software without
 *    prior written permission. For written permission, please contact
```

```
 *    openssl-core@openssl.org.
 *
 * 5. Products derived from this software may not be called "OpenSSL"
 *    nor may "OpenSSL" appear in their names without prior written
 *    permission of the OpenSSL Project.
 *
 * 6. Redistributions of any form whatsoever must retain the following
 *    acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit (http://www.openssl.org/)"
 *
 * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
 * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 * ====================================================================
 *
 * This product includes cryptographic software written by Eric Young
 * (eay@cryptsoft.com).  This product includes software written by Tim
 * Hudson (tjh@cryptsoft.com).
 *
 */

 Original SSLeay License
 -----------------------

/* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
 * All rights reserved.
 *
 * This package is an SSL implementation written
 * by Eric Young (eay@cryptsoft.com).
 * The implementation was written so as to conform with Netscapes SSL.
 *
 * This library is free for commercial and non-commercial use as long as
 * the following conditions are aheared to.  The following conditions
 * apply to all code found in this distribution, be it the RC4, RSA,
 * lhash, DES, etc., code; not just the SSL code.  The SSL documentation
 * included with this distribution is covered by the same copyright terms
 * except that the holder is Tim Hudson (tjh@cryptsoft.com).
 *
 * Copyright remains Eric Young's, and as such any Copyright notices in
 * the code are not to be removed.
 * If this package is used in a product, Eric Young should be given attribution
 * as the author of the parts of the library used.
 * This can be in the form of a textual message at program startup or
 * in documentation (online or textual) provided with the package.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *    "This product includes cryptographic software written by
 *     Eric Young (eay@cryptsoft.com)"
 *    The word 'cryptographic' can be left out if the rouines from the library
 *    being used are not cryptographic related :-).
 * 4. If you include any Windows specific code (or a derivative thereof) from
 *    the apps directory (application code) you must include an acknowledgement:
 *    "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
```

```
 *
 * THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 * The licence and distribution terms for any publically available version or
 * derivative of this code cannot be changed.  i.e. this code cannot simply be
 * copied and put under another distribution licence
 * [including the GNU Public Licence.]
 */
```

libical

```
libical is distributed under two licenses.
You may choose the terms of either:

 * The Mozilla Public License (MPL) v2.0

 or

 * The GNU Lesser General Public License (LGPL) v2.1

-----------------------------------------------------------------------

Software distributed under these licenses is distributed on an "AS
IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or
implied. See the License for the specific language governing rights
and limitations under the License.

     The Original Code is libical.

     The Initial Developer of the Original Code is Eric Busboom

     All Rights Reserved.

     Contributor(s): See individual source files.

Mozilla Public License Version 2.0
==================================

1. Definitions
--------------

1.1. "Contributor"
    means each individual or legal entity that creates, contributes to
    the creation of, or owns Covered Software.

1.2. "Contributor Version"
    means the combination of the Contributions of others (if any) used
    by a Contributor and that particular Contributor's Contribution.

1.3. "Contribution"
    means Covered Software of a particular Contributor.

1.4. "Covered Software"
    means Source Code Form to which the initial Contributor has attached
    the notice in Exhibit A, the Executable Form of such Source Code
    Form, and Modifications of such Source Code Form, in each case
    including portions thereof.

1.5. "Incompatible With Secondary Licenses"
    means

    (a) that the initial Contributor has attached the notice described
        in Exhibit B to the Covered Software; or
```

(b) that the Covered Software was made available under the terms of version 1.1 or earlier of the License, but not also under the terms of a Secondary License.

1.6. "Executable Form"
means any form of the work other than Source Code Form.

1.7. "Larger Work"
means a work that combines Covered Software with other material, in a separate file or files, that is not Covered Software.

1.8. "License"
means this document.

1.9. "Licensable"
means having the right to grant, to the maximum extent possible, whether at the time of the initial grant or subsequently, any and all of the rights conveyed by this License.

1.10. "Modifications"
means any of the following:

(a) any file in Source Code Form that results from an addition to, deletion from, or modification of the contents of Covered Software; or

(b) any new file in Source Code Form that contains any Covered Software.

1.11. "Patent Claims" of a Contributor
means any patent claim(s), including without limitation, method, process, and apparatus claims, in any patent Licensable by such Contributor that would be infringed, but for the grant of the License, by the making, using, selling, offering for sale, having made, import, or transfer of either its Contributions or its Contributor Version.

1.12. "Secondary License"
means either the GNU General Public License, Version 2.0, the GNU Lesser General Public License, Version 2.1, the GNU Affero General Public License, Version 3.0, or any later versions of those licenses.

1.13. "Source Code Form"
means the form of the work preferred for making modifications.

1.14. "You" (or "Your")
means an individual or a legal entity exercising rights under this License. For legal entities, "You" includes any entity that controls, is controlled by, or is under common control with You. For purposes of this definition, "control" means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of more than fifty percent (50%) of the outstanding shares or beneficial ownership of such entity.

2. License Grants and Conditions

2.1. Grants

Each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license:

(a) under intellectual property rights (other than patent or trademark) Licensable by such Contributor to use, reproduce, make available, modify, display, perform, distribute, and otherwise exploit its Contributions, either on an unmodified basis, with Modifications, or as part of a Larger Work; and

(b) under Patent Claims of such Contributor to make, use, sell, offer for sale, have made, import, and otherwise transfer either its

Contributions or its Contributor Version.

2.2. Effective Date

The licenses granted in Section 2.1 with respect to any Contribution become effective for each Contribution on the date the Contributor first distributes such Contribution.

2.3. Limitations on Grant Scope

The licenses granted in this Section 2 are the only rights granted under this License. No additional rights or licenses will be implied from the distribution or licensing of Covered Software under this License. Notwithstanding Section 2.1(b) above, no patent license is granted by a Contributor:

(a) for any code that a Contributor has removed from Covered Software; or

(b) for infringements caused by: (i) Your and any other third party's modifications of Covered Software, or (ii) the combination of its Contributions with other software (except as part of its Contributor Version); or

(c) under Patent Claims infringed by Covered Software in the absence of its Contributions.

This License does not grant any rights in the trademarks, service marks, or logos of any Contributor (except as may be necessary to comply with the notice requirements in Section 3.4).

2.4. Subsequent Licenses

No Contributor makes additional grants as a result of Your choice to distribute the Covered Software under a subsequent version of this License (see Section 10.2) or under the terms of a Secondary License (if permitted under the terms of Section 3.3).

2.5. Representation

Each Contributor represents that the Contributor believes its Contributions are its original creation(s) or it has sufficient rights to grant the rights to its Contributions conveyed by this License.

2.6. Fair Use

This License is not intended to limit any rights You have under applicable copyright doctrines of fair use, fair dealing, or other equivalents.

2.7. Conditions

Sections 3.1, 3.2, 3.3, and 3.4 are conditions of the licenses granted in Section 2.1.

3. Responsibilities

3.1. Distribution of Source Form

All distribution of Covered Software in Source Code Form, including any Modifications that You create or to which You contribute, must be under the terms of this License. You must inform recipients that the Source Code Form of the Covered Software is governed by the terms of this License, and how they can obtain a copy of this License. You may not attempt to alter or restrict the recipients' rights in the Source Code Form.

3.2. Distribution of Executable Form

If You distribute Covered Software in Executable Form then:

(a) such Covered Software must also be made available in Source Code Form, as described in Section 3.1, and You must inform recipients of the Executable Form how they can obtain a copy of such Source Code Form by reasonable means in a timely manner, at a charge no more than the cost of distribution to the recipient; and

(b) You may distribute such Executable Form under the terms of this License, or sublicense it under different terms, provided that the license for the Executable Form does not attempt to limit or alter the recipients' rights in the Source Code Form under this License.

3.3. Distribution of a Larger Work

You may create and distribute a Larger Work under terms of Your choice, provided that You also comply with the requirements of this License for the Covered Software. If the Larger Work is a combination of Covered Software with a work governed by one or more Secondary Licenses, and the Covered Software is not Incompatible With Secondary Licenses, this License permits You to additionally distribute such Covered Software under the terms of such Secondary License(s), so that the recipient of the Larger Work may, at their option, further distribute the Covered Software under the terms of either this License or such Secondary License(s).

3.4. Notices

You may not remove or alter the substance of any license notices (including copyright notices, patent notices, disclaimers of warranty, or limitations of liability) contained within the Source Code Form of the Covered Software, except that You may alter any license notices to the extent required to remedy known factual inaccuracies.

3.5. Application of Additional Terms

You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Software. However, You may do so only on Your own behalf, and not on behalf of any Contributor. You must make it absolutely clear that any such warranty, support, indemnity, or liability obligation is offered by You alone, and You hereby agree to indemnify every Contributor for any liability incurred by such Contributor as a result of warranty, support, indemnity or liability terms You offer. You may include additional disclaimers of warranty and limitations of liability specific to any jurisdiction.

4. Inability to Comply Due to Statute or Regulation

If it is impossible for You to comply with any of the terms of this License with respect to some or all of the Covered Software due to statute, judicial order, or regulation then You must: (a) comply with the terms of this License to the maximum extent possible; and (b) describe the limitations and the code they affect. Such description must be placed in a text file included with all distributions of the Covered Software under this License. Except to the extent prohibited by statute or regulation, such description must be sufficiently detailed for a recipient of ordinary skill to be able to understand it.

5. Termination

5.1. The rights granted under this License will terminate automatically if You fail to comply with any of its terms. However, if You become compliant, then the rights granted under this License from a particular Contributor are reinstated (a) provisionally, unless and until such Contributor explicitly and finally terminates Your grants, and (b) on an ongoing basis, if such Contributor fails to notify You of the non-compliance by some reasonable means prior to 60 days after You have come back into compliance. Moreover, Your grants from a particular Contributor are reinstated on an ongoing basis if such Contributor notifies You of the non-compliance by some reasonable means, this is the first time You have received notice of non-compliance with this License

from such Contributor, and You become compliant prior to 30 days after Your receipt of the notice.

5.2. If You initiate litigation against any entity by asserting a patent infringement claim (excluding declaratory judgment actions, counter-claims, and cross-claims) alleging that a Contributor Version directly or indirectly infringes any patent, then the rights granted to You by any and all Contributors for the Covered Software under Section 2.1 of this License shall terminate.

5.3. In the event of termination under Sections 5.1 or 5.2 above, all end user license agreements (excluding distributors and resellers) which have been validly granted by You or Your distributors under this License prior to termination shall survive termination.

```
************************************************************************
*                                                                      *
*  6. Disclaimer of Warranty                                           *
*  -------------------------                                           *
*                                                                      *
*  Covered Software is provided under this License on an "as is"       *
*  basis, without warranty of any kind, either expressed, implied, or  *
*  statutory, including, without limitation, warranties that the       *
*  Covered Software is free of defects, merchantable, fit for a        *
*  particular purpose or non-infringing. The entire risk as to the     *
*  quality and performance of the Covered Software is with You.        *
*  Should any Covered Software prove defective in any respect, You     *
*  (not any Contributor) assume the cost of any necessary servicing,   *
*  repair, or correction. This disclaimer of warranty constitutes an   *
*  essential part of this License. No use of any Covered Software is   *
*  authorized under this License except under this disclaimer.         *
*                                                                      *
************************************************************************

************************************************************************
*                                                                      *
*  7. Limitation of Liability                                          *
*  --------------------------                                          *
*                                                                      *
*  Under no circumstances and under no legal theory, whether tort      *
*  (including negligence), contract, or otherwise, shall any           *
*  Contributor, or anyone who distributes Covered Software as          *
*  permitted above, be liable to You for any direct, indirect,         *
*  special, incidental, or consequential damages of any character      *
*  including, without limitation, damages for lost profits, loss of    *
*  goodwill, work stoppage, computer failure or malfunction, or any    *
*  and all other commercial damages or losses, even if such party      *
*  shall have been informed of the possibility of such damages. This   *
*  limitation of liability shall not apply to liability for death or   *
*  personal injury resulting from such party's negligence to the       *
*  extent applicable law prohibits such limitation. Some               *
*  jurisdictions do not allow the exclusion or limitation of           *
*  incidental or consequential damages, so this exclusion and          *
*  limitation may not apply to You.                                    *
*                                                                      *
************************************************************************
```

8. Litigation

Any litigation relating to this License may be brought only in the courts of a jurisdiction where the defendant maintains its principal place of business and such litigation shall be governed by laws of that jurisdiction, without reference to its conflict-of-law provisions. Nothing in this Section shall prevent a party's ability to bring cross-claims or counter-claims.

9. Miscellaneous

This License represents the complete agreement concerning the subject matter hereof. If any provision of this License is held to be

unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not be used to construe this License against a Contributor.

10. Versions of the License

10.1. New Versions

Mozilla Foundation is the license steward. Except as provided in Section 10.3, no one other than the license steward has the right to modify or publish new versions of this License. Each version will be given a distinguishing version number.

10.2. Effect of New Versions

You may distribute the Covered Software under the terms of the version of the License under which You originally received the Covered Software, or under the terms of any subsequent version published by the license steward.

10.3. Modified Versions

If you create software not governed by this License, and you want to create a new license for such software, you may create and use a modified version of this License if you rename the license and remove any references to the name of the license steward (except to note that such modified license differs from this License).

10.4. Distributing Source Code Form that is Incompatible With Secondary Licenses

If You choose to distribute Source Code Form that is Incompatible With Secondary Licenses under the terms of this version of the License, the notice described in Exhibit B of this License must be attached.

Exhibit A - Source Code Form License Notice

  This Source Code Form is subject to the terms of the Mozilla Public License, v. 2.0. If a copy of the MPL was not distributed with this file, You can obtain one at https://mozilla.org/MPL/2.0/.

If it is not possible or desirable to put the notice in a particular file, then You may include the notice in a location (such as a LICENSE file in a relevant directory) where a recipient would be likely to look for such a notice.

You may add additional accurate notices of copyright ownership.

Exhibit B - "Incompatible With Secondary Licenses" Notice

  This Source Code Form is "Incompatible With Secondary Licenses", as defined by the Mozilla Public License, v. 2.0.
