Handbuch Enertex® EibPC²

Enertex® EibPC²: Firmware 5.500 oder höher
Enertex® EibStudio: Version 5.500 oder höher

Hinweis

Der Inhalt dieses Dokuments darf ohne vorherige schriftliche Genehmigung durch die Enertex® Bayern GmbH in keiner Form, weder ganz noch teilweise, vervielfältigt, weitergegeben, verbreitet oder gespeichert werden.

Enertex® ist eine eingetragene Marke der Enertex® Bayern GmbH. Andere in diesem Handbuch erwähnte Produkt- und Firmennamen können Marken oder Handelsnamen ihrer jeweiligen Eigentümer sein.

Dieses Handbuch kann ohne Benachrichtigung oder Ankündigung geändert werden und erhebt keinen Anspruch auf Vollständigkeit oder Korrektheit.

Enertex® Bayern GmbH – Ebermannstädter Straße 8 - 91301 Forchheim - mail@enertex.de

Inhalt

Sicherheitshinweise **5**
Lizenzhinweise **5**
Hilfe **5**
Updates **5**
Enertex® EibPC² **6**
Übersicht *6*
Inbetriebnahme *8*
Anschlüsse und Bedienelemente 8
Installation 9
Gerätestart *10*
Firmwareupdates *10*
Werkseinstellungen *10*
Kurzanleitung EibStudio **11**
EibStudio **12**
Installation *12*
Fenstermenü *12*
Projekte *12*
Projektverzeichnis 12
Projekt aus EibStudio 3 importieren 12
Einstellungen *12*
Konfigurationsverzeichnis 12
Bedienelemente *13*
Übersicht *15*
Objekte *15*
Gruppenadressimport 15
Topologie 15
Interne Variablen 15
Konstanten 15
Logik *16*
Definitionen 16
Debug-Modus 17
Visualisierungsobjekte 17
Visu *18*
Elemente 18
Funktionen 18
Eigene Vorlagen 18
Vorlagen 18
Verknüpfung mit Logik und Experte 18
Experte *19*
Autovervollständigung 19
Makros 19
Eigene Visualisierungsseiten 19
Zugriff auf Visualisierungselemente 19
Syntax 20
Online Debuggen zur Laufzeit 21
Projekteinstellungen *22*
Suche 22
KNX-Verbindung 22
Netzwerkadresse 22

Namensauflösung....22
Ports....22
Projekt-Passwort....22
Verbindungsverschlüsselung....22
KNX Secure Sequenzzähler....22
Datum und Uhrzeit....23
Position....23
SHUTDOWN-Variable....23
FTP....23
E-Mail....23
Sicherung....23
Dateien....24
HTTPS....24
VPN....24
IDs....24
Export und Import....25
Debugger....25
Gruppenmonitor....25
Telegrammpuffer....25
Ereignisspeicher....25
Simulation....25
Objekte....26
Datentypen....26
Zahlen (Konstanten)....27
Zeichenketten....28
IP Adresse....28
Pyhsikalische Adresse....28
Übersicht....29
Variablen....30
Gruppenadressen....30
Manuelle Gruppenadresse....30
Initialisieren von Gruppenadressen....31
KNX Data Secure....31
Grundlagen....31
Gruppenadressen für EibPC freigeben....31
Sequenzzähler-Fehler....32
Validierungsschema....33
Visualisierung....41
Passwortschutz....43
Elemente....43
Funktionen....44
Vorlagen....44
Icons....45
Zustände....45
Symbole....46
Logik....69
Beispiele....69
Treppenhausautomat....69
IoT-Geräte via MQTT....70
Experte....76
Visualisierung....76

Seiten....76
Elemente....85
IDs....89
Elementdefinitionen....90
Funktionen....108
Logische Verknüpfungen....108
Zeitfunktionen....112
Datumssteuerung....119
Beschattung und Sonnenstand....121
Zeitschaltuhren....124
Vergleichszeitschaltuhren....126
Konfigurierbare Zeitschaltuhren....129
Spezielle Zeitfunktionen....130
Remanentspeicher....135
Rechnen....138
Sonderfunktionen....146
Szenen....154
Stringfunktionen....157
Parser....169
KNX-Telegramme....170
Netzwerkfunktionen....182
Visualisierung....209
Makros....231
Definition....231
Sonderzeichen....232
Laufzeit- und Syntaxfehler....232
Makroasisstent....232
Lokale Variablen....232
Rückgabewerte....233
Beispiele....234
Ereignisse....258
Fehlermeldungen des Compilers....263
Lizenzen....264
Versionsgeschichte....281

Wir freuen uns, dass Sie sich für einen EibPC² entschieden haben.

Sicherheitshinweise

Bitte beachten Sie folgende Punkte:

- Einbau und Montage elektrischer Geräte dürfen nur durch Elektrofachkräfte erfolgen.
- Bei Anschluss und Inbetriebnahme von KNX-Geräten werden Fachkenntnisse durch Schulungen vorausgesetzt.
- Bei Nichtbeachtung der Anleitung können Schäden am Gerät, Brand oder andere Gefahren auftreten.
- Diese Anleitung ist Bestandteil des Produkts und muss beim Endanwender verbleiben.
- Das Gerät darf nicht für Anwendungen mit Gefährdungspotential eingesetzt werden (Fehlfunktionen, möglicher Ausfall der Zeitschaltuhren, etc.).

Lizenzhinweise

- Mit dem Erwerb des EibPC erwerben Sie eine Lizenz zur Nutzung von EibStudio. Das EibStudio und sämtliche auch eigenständig arbeitenden Komponenten dürfen nur zur Programmierung eines EibPC genutzt werden.
- Der Hersteller haftet nicht für Kosten oder Schäden, die dem Benutzer oder Dritten durch den Einsatz diesen Gerätes, Missbrauch oder Störungen des Anschlusses, Störungen des Gerätes oder der Teilnehmergeräte entstehen.
- Eigenmächtige Veränderungen und Umbauten am Gerät führen zum Erlöschen der Gewährleistung!
- Für nicht bestimmungsgemäße Verwendung haftet der Hersteller ebenso nicht.

Hilfe

E-Mail

Bei Fragen und Problemen zu Ihrem EibPC² können Sie sich an support@enertex.de wenden.

Support-Export

Um ihr Anliegen schneller bearbeiten zu können, fügen Sie Ihrer Anfrage bitte das Projekt hinzu. Klicken Sie im Fenstermenü auf **Hilfe → Export für Support**. Senden Sie uns die Datei (*.esp).
Dies ist zip-Archiv und enthält ihr Projekt und alle hinzugefügten Webserver-Dateien, sowie Informationen zum System und die Log-Datei. Vertrauliche Informationen wie FTP-, E-Mail, VPN-Passwörter sind jedoch **nicht** enthalten.

Telefon

Zu den üblichen Öffnungdszeiten steht Ihnen auch der kostenlose Telefonsupport unter der Rufnummer +49 9191 733950 zur Verfügung.

KNX-User-Forum

Unter http://www.knx-user-forum.de/eibpc ist ein eigener Bereich für den Support des EibPC eingerichtet. Hier finden Sie auch direkten Rat von versierten Anwendern und Profis.

Lehrvideos

Unter http://videos.eibpc.com/ finden Sie Videos zur EibPC-Programmierung.

Updates

Auf der Seite www.eibpc.com finden Sie Updates zum EibPC².

Enertex® EibPC²

Übersicht



Abbildung 1: Der Enertex® EibPC²

Zusammenfassung

Der EibPC² stellt eine Steuerung für die Hutschienenmontage (4 TE) für KNX-Bussysteme dar. Er ermöglicht mit ca. 1,8 W Leistungsaufnahme stromsparend und umweltschonend vollständige Kontrolle über das KNX-Bussystem.

Der EibPC² ist Szenenaktor, Berechnungscomputer, Logikzentrale, SPS, Schaltzeituhr, LAN-und Internet-Anbindung, Webserver und E-Mail-Client in einem Gerät.

Er verfügt über alle Funktionen des EibPC. Über den integrierten Busankoppler stellt der EibPC² der ETS einen KNX-Tunnel zur Programmierung zur Verfügung.

Mit der Software EibStudio ist eine Parametrierung des EibPCs und Zugriff auf Gruppenadressen ohne ETS möglich. Sie können diese Software unter www.enertex.de gratis herunterladen.

KNX-Funktionen

Dabei übernimmt der EibPC im KNX-Netz die Funktionalität von

- Szenenaktoren
- bedingten Anweisungen (wenn-dann)
- Schaltzeituhren
- Zeitgeber (Uhrzeit und Datum) (per LAN oder KNX oder EibStudio synchronisiert)
- genauesten Timern (im ms-Bereich)
- Reglern beliebiger Struktur
- Auswertungen mathematischer Ausdrücke
- Verzögerungsgliedern
- Verknüpfungen von KNX-Objekten (Tor, Multiplexer, ...)
- Überwachung von Aktoren (z.B. zyklische Leseanforderungen).

Diese Funktionen können beliebig oft genutzt werden. Sie könnten also beispielsweise 65000 Szenenaktoren definieren. Der EibPC verarbeitet die gesamte maximal mögliche Anzahl von Objekten einer KNX-Vernetzung.

LAN-Funktionen

Der EibPC besitzt eine LAN-Schnittstelle, so dass zusätzlich

- die Überwachung des Busverkehrs (ohne ETS [und PC])
- das Senden und Verarbeiten von beliebigen KNX -Telegrammen (ohne ETS)
- die Synchronisierung der Bus-Zeit via Internet (ohne ETS)
- das Senden, Empfangen und Verarbeiten von UDP-Telegrammen (Zusatz-Option NP) z.B. zur Steuerung von Multimediasystemen
- das Senden, Empfangen und Verarbeiten von TCP-Telegrammen als TCP Server und/oder Client (Zusatz-Option NP)
- das Senden von E-Mails (Zusatz-Option NP)
- ein integrierter Webserver (Zusatz-Option NP)
- eine VPN Verbindung über KNX Verbindung zu konfigurieren (Zusatz-Option NP)

möglich ist.

Datenlogger

Speicher - Der EibPC speichert sämtliche Bustelegramme. Bis zu 500.000 Telegramme werden in einem Ringspeicher vorgehalten, wenn kein PC mit dem EibPC verbunden ist.

Zeit - Mit Hilfe der Zeitstempel, die der EibPC automatisch generiert, kann der Busverkehr jederzeit analysiert werden.

Online - Daneben besteht die Möglichkeit, die Daten online anzuzeigen und nach Absender und Gruppenadressen zu filtern.

Filter - Die Telegramme können bereits nach Geräteadressen und Gruppenadressen vorgefiltert werden.

Autolog - Das EibStudio ermöglicht das zyklische Abspeichern von (ggf. gefilterten) Telegrammen in Dateien.

FTP - Der EibPC kann Telegrammdaten an einen beliebigen FTP Server schicken. Das EibStudio ermöglicht das Auswerten dieser Binärdaten und exportiert diese in lesbare CSV- Textdateien.

Software

Mit dem EibStudio als Konfigurationsprogramm wird über die LAN-Schnittstelle des EibPC eine Hausautomatisierung erstellt. Damit lässt sich der EibPC ohne ETS auf einfache Weise programmieren.

Programmierung - Die Programmierung erfolgt wahlweise durch Verbinden von vorgefertigten Funktionsblöcken in einem grafischen Editor oder durch Programme in einer einfachen Programmiersprache, für die keine aufwändigen Lehrgänge notwendig sind.

ETS - Das EibStudio importiert die Adressen und Einstellungen der ETS. Es kann aber auch gänzlich ohne ETS-Import gearbeitet werden.

Inbetriebnahme

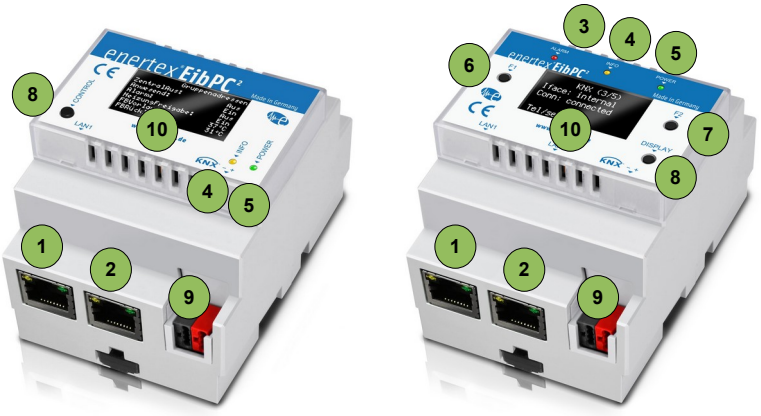


Abbildung 2: Anschlüsse und Bedienelemente – Ein-Tasten und Drei-Tasten-Variante

Anschlüsse und Bedienelemente

In Abbildung 2 sind die folgenden Anschlüsse und Bedienelemente des EibPC² dargestellt:

1. LAN1
2. LAN2
3. Alarm-LED (rot)
4. Info-LED (orange)
5. Power-LED (grün)
6. F1-Taste
7. F2-Taste
8. Control-Taste (Ein-Tasten-Variante) / Display-Taste (Drei-Tasten-Variante)
9. KNX
10. Display

Der EibPC² wird direkt vom KNX-Bus gespeist und benötigt eine Spannung von 27V bis 30V. Prüfen Sie unbedingt die Spannung am Installationsort, falls der EibPC² nicht in unmittelbarer Nähe zur KNX-Spannungsversorgung installiert wird.

Es ist auch möglich, den EibPC² mit einem normalen Netzteil zu betreiben, falls der Buszugriff über eine KNXnet/IP-Schnittstelle erfolgt.

Die Spannungsversorgung muss mindestens 3,2 Watt (110 mA bei 29 V Busspannung) Ausgangsleistung zur Verfügung stellen.

Der EibPC² verfügt über einen integrierten Busankoppler. Alternativ kann im EibStudio eine externe KNXnet/IP-Schnittstelle ausgewählt werden. Dadurch haben Sie die Möglichkeit, den EibPC² unabhängig vom KNX-Bus zu installieren.

Grundsätzlich sind alle KNX-zertifizierte IP Schnittstellen mit dem EibPC² kompatibel.

Besonders zu empfehlen sind

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

Die Router biete viele Zusatz- und Diagnosefunktionen und können den EibPC nach einem Reset ohne Internetverbindung direkt mit der Uhrzeit synchronisieren.

Der EibPC² nutzt das sog. Tunneling der KNXnet/IP-Schnittstelle. Diese wird dadurch exklusiv belegt, d.h. Sie können die Schnittstelle in diesem Modus dann nicht von der ETS aus ansprechen.

Installation

Integrierter Busankoppler

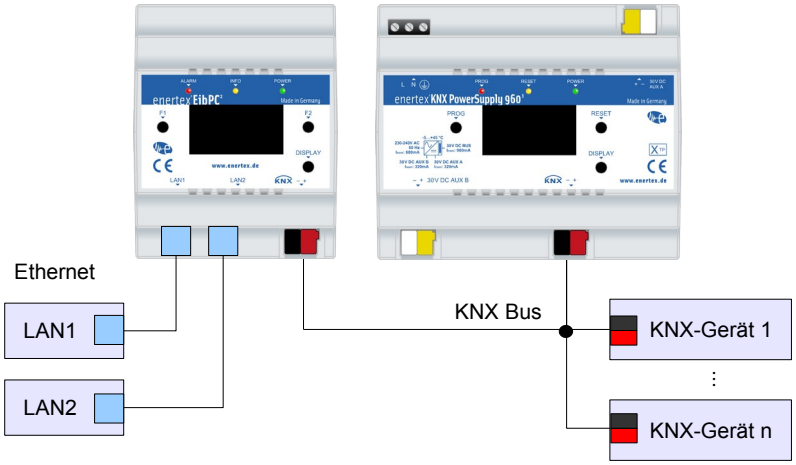


Abbildung 3: Anschluss des Enertex® EibPC an den KNX-Bus

Abbildung 3 zeigt den Anschluss des EibPC². Gehen Sie wie folgt vor:

1. Über den LAN 1 oder LAN 2-Anschluss (1,2) mit dem Netzwerk verbinden.
2. Der freie LAN-Anschluss (2) kann verwendet werden, um weitere Geräte ins gleiche Netz einzubinden.
3. Den EibPC² an eine KNX-Spannungsversorgung anschließen.
   Sie können auch ein anderes Netzteil entsprechender Spannung und Leistung verwenden, falls Sie eine KNXnet/IP-Schnittstelle statt der integrierten verwenden.

Integrierter Switch

Achtung: Kommunikation zwischen LAN1 und LAN2 ist nur möglich, wenn der EibPC² gestartet und betriebsbereit ist.

Anderenfalls ist keine Kommunikation zwischen LAN1 und LAN2 möglich. Dies ist auch der Fall, wenn der EibPC² neu gestartet wird.

Wird das Anwendungsprogramm neu gestartet oder ein neues übertragen, bleibt die Verbindung bestehen.

KNXnet/IP-Schnittstelle

Abbildung 4 zeigt den Anschluss bei Verwendung einer KNXnet/IP-Schnittstelle.

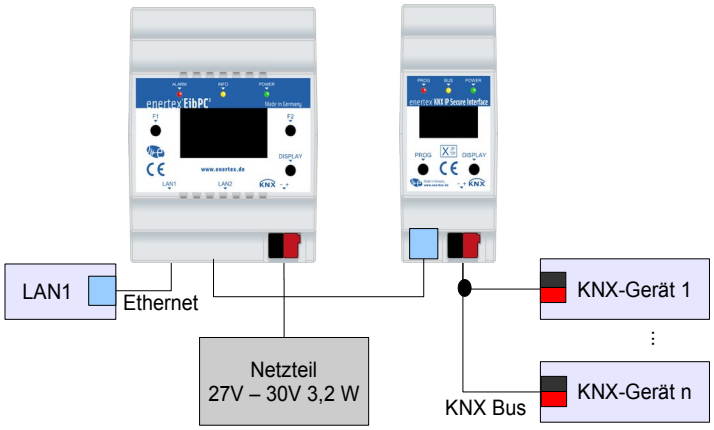


Abbildung 4: Anschluss des Enertex® EibPC² an den KNX-Bus über eine KNXnet/IP- Schnittstelle

Gerätestart

Beim Gerätestart des EibPC² können Sie folgendes beobachten:

Ein-Tasten-Variante:

1. Info- und Power-LED leuchten, der Bootvorgang läuft.
2. Nach dem Start beginnt die Power-LED zu blinken.
3. ~ 2 min nach Anstecken blinkt die Info-LED im Sekundentakt. Während dieser Phase ist ein Zurücksetzen auf Werkseinstellungen möglich (s.u.).
4. Danach wird die Bus-Schnittstelle initialisiert. Die Info-LED blinkt.
5. Nach erfolgreichem Start der Netzwerkschnittstelle wird die aktuelle IP-Adresse im Display angezeigt. Die Anzeige erlischt automatisch nach 30 s. Sie kann durch Drücken der Control-Taste erneut aktiviert werden.
6. Das Gerät arbeitet nun regulär. Die Power-LED blinkt durchgängig, die Info-LED blinkt bei eintreffenden KNX-Telegrammen.

Drei-Tasten-Variante:

1. Nach dem Anstecken leuchten alle LEDs halb-hell.
2. Nach ~5 s leuchtet nur noch die Power-LED in voller Helligkeit
3. Nach dem Booten beginnt die Power-LED zu blinken.
4. ~ 2 min nach Anstecken blinkt die Info-LED im Sekundentakt. Während dieser Phase ist ein Zurücksetzen auf Werkseinstellungen möglich (s.u.).
5. Danach wird die Bus-Schnittstelle initialisiert. Die Info-LED blinkt.
6. Nach erfolgreichem Start der Netzwerkschnittstelle wird die aktuelle IP-Adresse im Display angezeigt. Die Anzeige erlischt automatisch nach 30 s. Sie kann durch Drücken der Control-Taste erneut aktiviert werden.
7. Das Gerät arbeitet nun regulär. Die Power-LED blinkt durchgängig, die Info-LED blinkt bei eintreffenden KNX-Telegrammen.

Firmwareupdates

Aktualisierungen der Firmware können Sie selbst durchführen. Dazu benötigen Sie eine Aktualisierungsdatei (Dateiname: *eibpc2-patch-x.xxxx.ptc*) die Sie über das EibStudio installieren können. Der Vorgang dauert insgesamt etwa 5 Minuten. Während dieser Zeit muss eine stabile Spannungsversorgung garantiert sein.

Sollte das Gerät 20 Minuten nach Start der Aktualisierung noch nicht wieder im EibStudio ansprechbar sein und keine Aktivität signalisieren (beide LEDs dauerhaft aus oder ein, Display ohne Anzeige bei Drücken der Control-Taste), trennen Sie das Gerät kurz von der Spannungsversorgung, um es neu zu starten.

Sollte es danach weiterhin nicht erreichbar sein, wenden Sie sich bitte an unseren Support.

Werkseinstellungen

Zurücksetzen beim Start

Während des Bootvorgangs leuchtet die grüne Power-LED durchgehend. Danach blinkt die Info-LED im Sekundentakt für 5 Sekunden.

Halten Sie in dieser Zeit die Display-Taste/Control-Taste gedrückt, um das Gerät zurückzusetzen. Nach dem Zurücksetzen blinkt die Info-LED mehrmals kurz auf und das Gerät wird neu gestartet.

Es werden die folgenden Einstellungen zurückgesetzt/gelöscht:

1. Netzwerkeinstellungen auf DHCP zurücksetzen
2. Anwenderprogramm löschen
3. Daten zum Sonnenstand löschen
4. VPN-Benutzer und IP-Filter löschen
5. HTTPS-Benutzer löschen
6. Zertifikate löschen
7. Szenen, TimeBuffer und gespeicherte Variablen löschen
8. Hochgeladene Webserver-Dateien löschen

Zurücksetzen während des Betriebs

Wurde das Gerät bereits gestartet, kann es auf Werkseinstellungen zurückgesetzt werden, indem die Display-Taste/Control-Taste für mindestens 20 Sekunden gedrückt wird. Auf dem Display erscheint eine Bestätigung, die Info-LED blinkt mehrmals kurz auf und das Gerät startet neu.

Kurzanleitung EibStudio

Das Gerät wurde wie oben beschrieben mit dem Netzwerk verbunden und gestartet. Im Auslieferungszustand sucht es nach einen DHCP-Server, um eine IP-Adresse zugewiesen zu bekommen. Dieses Verhalten können Sie später in den Einstellungen ändern.

Zur Programmierung wird das EibStudio ab Version 4 verwendet. Es dient sowohl zur Änderung der Einstellungen als auch der Erstellung eines Anwenderprogramms.

Das EibStudio muss lediglich entpackt werden, es erfolgt keine Installation.

Wichtig: Eine aktive Firewall verhindert u.U. die Kommunikation zwischen EibStudio und EibPC und muss entsprechend konfiguriert werden. Schalten Sie ggf. die TCP-Ports 4910, 4911 frei. Die Gerätesuche erfolgt mittels Broadcast an die Adresse 255.255.255.255, UDP, Port 4805.

EibStudio starten

Beim ersten Start fragt EibStudio nach einem Verzeichnis, in dem die Projekte gespeichert werden sollen.

Projektverzeichnis

Außerhalb dieses Verzeichnisses und dem Konfigurationsverzeichnis (S. 12) werden keine Daten verändert.

Beim Import von bestehenden Projekten werden diese automatisch ins Projektverzeichnis kopiert.

Sie können das Projektverzeichnis über die Benutzer-Einstellungen ändern. Dabei wird ein eventuell geöffnetes Projekt geschlossen und alle vorhandenen Projekte im geänderten Verzeichnis werden angezeigt.

Projektübergreifende Einstellungen

Einstellungen, die für alle Projekte gelten, erreichen Sie über das Fenstermenü unter **Bearbeiten → Einstellungen**.

Projekt

EibStudio startet immer in der Projektübersicht. Darin können Sie neue Projekte anlegen, Projekte importieren oder löschen. Dabei werden nur die für das jeweilige Projekt gespeicherten Daten im aktuellen Projektverzeichnis gelöscht.

Ein Projekt beinhaltet alle Informationen, die für die Konfiguration und den Betrieb eines Geräts notwendig sind.

Projektmenü

Innerhalb eines geöffneten Projekts wird das Projektmenü verwendet. Es bietet Zugriff auf:

- **Übersicht**: Gerätezustand, Programminformationen, Logbuch
- **Objekte**: Alle Gruppenadressen und Variablen
- **Logik**: Logikeditor zur grafischen Erstellung von Programmen
- **Visu**: Erstellen der Web-Visualisierung
- **Experte**: Texteditor zur Erstellung von Programmen
- **Projekteinstellungen**: Projektspezifische Konfiguration des EibPC

Bus-Verbindung

Um das erste Programm ausführen zu können muss die Verbindung zum EibPC im Projektmenu unter **Einstellungen → Verbindung** konfiguriert werden. Befindet sich das Gerät im gleichen Netz wird er über die Suche gefunden.

Je nach Gerät und Installation kann hier auch die Verbindung zum KNX-Bus eingestellt werden.

Programm

Im Fenstermenü **Projekt** kann die Kompilierung des Programms gestartet werden. Das Programm wird dabei aus den einzelnen Konfigurationen zusammengesetzt, d.h. es beinhaltet Logik, Visualisierung, sowie alle Expertenprogramme und Einstellungen.

Im gleichen Menü kann die Programmübertragung auf den EibPC gestartet werden.

Objekte

Im Projektmenü **Objekte → ETS-Import** können *.esf* und *.knxproj-Dateien* importiert werden, um Typinformationen der verfügbaren Gruppenadressen zu erhalten. Diese können bei Bedarf direkt unter **Objekte → Gruppenadressen** geändert werden.

Typinformationen sind insbesondere für die Verwendung des Debuggers und des Gruppenmonitors notwendig.

Die Liste der verwendeten Variablen wird bei jeder Programmübersetzung neu erstellt und kann nicht selbst geändert werden.

EibStudio

Dieses Kapitel gibt einen Überblick über den Programmaufbau und die grundlegende Bedienung.

Sofern nicht ausdrücklich unterschieden, bezieht sich die Bezeichnung EibPC im Folgenden auf alle Generationen. Die Bezeichnung EibStudio ohne explizite Versionsangabe bezieht sich auf Version 4 oder höher.

Installation

EibStudio, ebenso wie EibStudio 3, benötigt keine Installation sondern muss lediglich entpackt werden. Prüfen Sie in jedem Fall, dass Sie die nötigen Rechte in dem Verzeichnis besitzen. Dies gilt insbesondere, wenn Sie das EibStudio an einen zentralen Ort (z.B. das allgemeine Programmverzeichnis) verschieben möchten.

Beachten Sie auch, dass die Datei *eibparser.exe* im Unterverzeichnis *bin* ausgeführt werden können muss.

Fenstermenü

Das Fenstermenü beinhaltet alle zentralen Funktionen, die unabhängig von einen Projekt aufgerufen werden können, wie die Einstellungen oder Hilfe.

Ist ein Projekt geöffnet, befinden sich dort zusätzlich häufig verwendete Funktionen, wie das Kompilieren und Übertragen des Programms.

Projekte

Beim Start von EibStudio wird die Projektübersicht angezeigt. Hier haben Sie die Möglichkeit, neue Projekte anzulegen oder bestehende Projekte aus EibStudio 3 oder EibStudio zu importieren. Projekte kapseln alle Informationen, die einen bestimmten EibPC bzw. eine Installation betreffen. Alle Projekte werden im Projektverzeichnis gespeichert.

Verändern Sie keine Dateien innerhalb des Projektverzeichnisses!

Projektverzeichnis

Beim ersten Start müssen Sie den Speicherort des Projektverzeichnisses wählen. Stellen Sie sicher, dass sie die nötigen Berechtigungen (lesen, schreiben) für das Verzeichnis besitzen.

EibStudio greift nur auf Dateien im Projektverzeichnis und dem Konfigurationsverzeichnis zu. Sie können das Projektverzeichnis in der Übersicht ändern.

Projekt aus EibStudio 3 importieren

EibStudio 3-Projekte bestehen aus einer oder mehreren Quellcodedateien mit der Endung .epc. Dabei werden alle weiteren Quellcodedateien vom Hauptprogramm über #include-Anweisungen eingebunden.

Starten Sie den Importvorgang und wählen Sie zunächst das Hauptprogramm, das bisher mit EibStudio 3 bearbeitet wurde. Wählen Sie im nächsten Dialog das Verzeichnis, in dem sich die Programmdatei von EibStudio 3 befindet. Dies ist nötig, falls das Hauptprogramm #include-Anweisungen mit relativen Pfadangaben enthält.

Daraufhin wird ein neues Projekt angelegt. Der Name entspricht dem Dateinamen des Hauptprogramms. Werden referenzierte Dateien nicht gefunden, so bricht der Import-Vorgang mit einem entsprechenden Hinweis ab, um welche Datei es sich handelt. Prüfen Sie deren Pfad und editieren Sie falls nötig das zu importierende Programm. Starten Sie anschließend den Vorgang neu.

Wurde der Import erfolgreich abgeschlossen, so befinden sich folgende Einstellungen bzw. Einträge im neu angelegten Projekt:

[ETS-ESF]: Die .esf-Datei wird importiert und die Gruppenadressen unter **Objekte** angelegt

[InitGA]: Die Gruppenadressen unter **Objekte** werden mit dem Initialisierungsflag versehen

[FTP], [MailConf], [Performance], [VPN], [HTTPS], [Location]: Die Einstellungen werden unter **Projekteinstellungen** → **EibPC** bzw. **Projekteinstellungen** → **Fernzugriff** übernommen

[MacroLibs]: Die Quelldateien der Bibliotheken werden als **Eigene Makros** unter **Experte** angelegt. Die meisten Makrobibliotheken aus EibStudio 3 sind bereits als **Interne Makros** vorhanden. Diese werden deaktiviert, falls es Namensüberschneidungen mit den eigenen Makros gibt.

Das gesamte Programm wird als neues Programm unter **Experte** angelegt. Die o.g. Sektionen werden auskommentiert, und die Sektionen [EibPC], [Macros] und [WebServer] werden durch #addto [EibPC] usw ersetzt.

Einstellungen

Konfigurationsverzeichnis

Die projektübergreifenden Einstellungen werde über das Fenstermenü **Bearbeiten** → **Einstellungen** geöffnet. Sie gelten für alle Projekte und werden als Datei *eibstudio.json* im Konfigurationsverzeichnis gespeichert. Je nach Betriebssystem befindet sich dies unter:

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

Bedienelemente

Navigationselemente

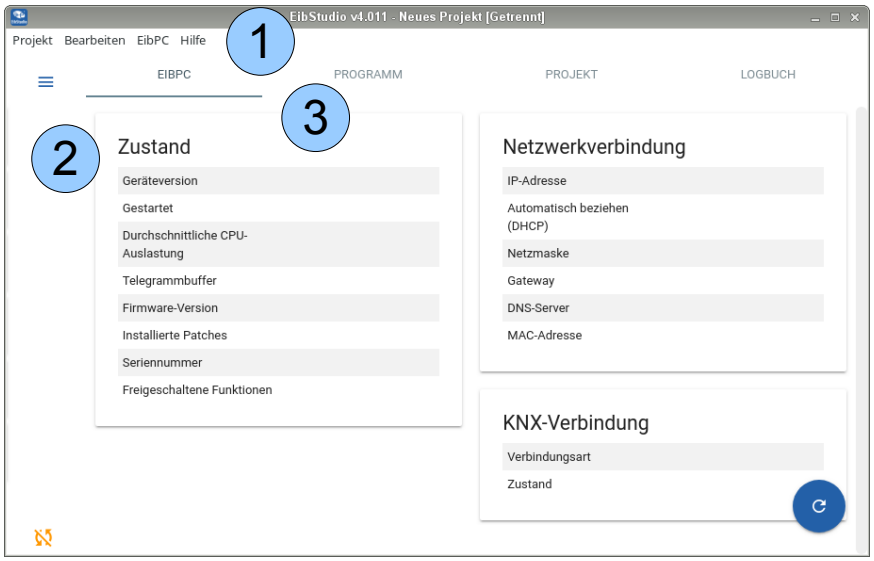


Abbildung 5: Navigationselemente

Erweiterte Navigation

Ist ein Projekt geöffnet so beinhaltet das Fenstermenü (1) zusätzlich häufig verwendete Projekt-Funktionen. Das Projektmenü wird über den Projektmenü-Knopf (2) eingeblendet. Diese wird verwendet um zwischen den verschiedenen Aufgaben innerhalb eines Projekts zu wechseln. Zur weiteren Strukturierung werden Registerkarten (3) verwendet.

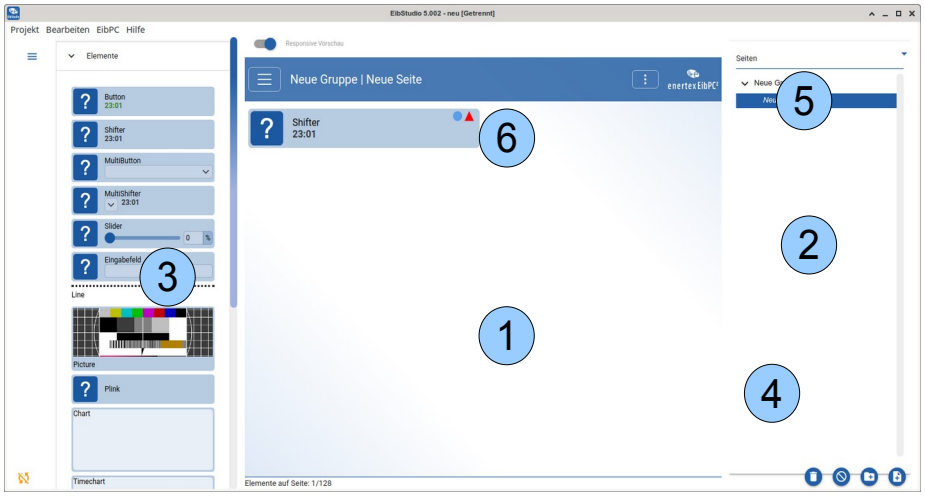


Abbildung 6: Erweiterte Navigation

Für Logik, Visualisierung und Experte wird eine erweiterte Navigation wie in Abbildung 6 dargestellt verwendet.

Der zentrale Editor-Bereich (1) stellt den aktuell in (2) ausgewählten Eintrag dar.

Einträge in der Liste links (3) können in (1) durch `Klicken` hinzugefügt werden. Um sie aus (1) zu entfernen, mit einem `Klick` markieren und `Entf` drücken. Bei gedrückter `Shift` oder `Strg`-Taste können mehrere Elemente markiert werden.

Einträge in (2) werden über die Knöpfe in (4) hinzugefügt, entfernt, usw.

Um den Editor-Bereich (1) zu vergrößern, kann (2) bei Bedarf ausgeblendet werden (5).

Ein Doppelklick auf Einträge in (1) und (2) öffnen den entsprechenden Dialog zum Bearbeiten ihrer Eigenschaften.

Das rote Dreieck bei (6) signalisiert, dass die Konfiguration nicht korrekt oder nicht vollständig ist, und dass dadurch das Programm wahrscheinlich nicht wie erwartet funktioniert.

Der blaue Kreis signalisiert, dass das Element seit der letzten Speicherung geändert wurde.

Eigenschaftsdialog

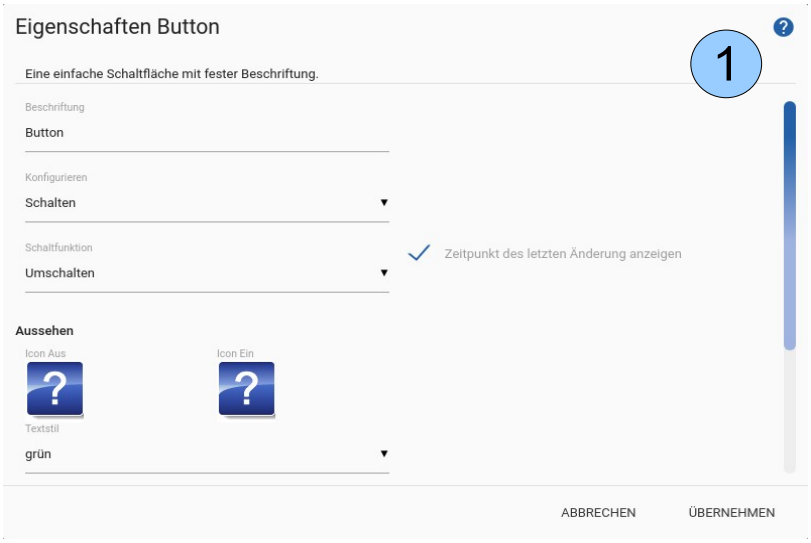


Abbildung 7: Eigenschaftsdialog

Im Eigenschaftsdialog werden die Eigenschaften eines Objekts geändert. Die meisten Dialoge verfügen über eine integrierte Hilfe (1), die bei Bedarf eingeblendet werden kann.

Im Folgenden werden die Funktionen im Projektmenü näher beschrieben.

Übersicht

In der Übersicht werden Informationen zum verbundenen EibPC und zum zuletzt kompilierten Programm dargestellt. In Anlehnung an die ETS können zudem projektbezogene Verwaltungsinformationen und Logbuch-Einträge gespeichert werden. Diese sind nicht mit einem internen Zustand verknüpft sondern dienen lediglich der Dokumentation.

Objekte

Die Objektübersicht listet alle Gruppenadressen, Variablen und vordefinierten Konstanten auf. Eine genaue Erklärung zu den einzelnen Objekttypen finden Sie im Kapitel Objekte. Bei einem neuen Projekt sind diese Listen zunächst leer. Bei jeder erfolgreichen Kompilierung werden die Listen aktualisiert. **Treten also bei der Kompilierung Fehler auf, so müssen diese zunächst behoben werden.**

Die Gruppenadress- und Variablenübersicht bieten zudem die Möglichkeit, den aktuellen Zustand abzufragen. Wählen Sie dazu einen Eintrag und klicken Sie auf das entsprechende Symbol oben rechts. Alternativ: `Doppelklick` zur Wertabfrage, `Strg+Klick` um ein Bustelegramm zu senden, bzw. den Variablenwert zu ändern.

Der Debugger bietet weitere Möglichkeiten wie das Abfragen mehrerer Objekte oder das Senden von Leseanforderungen.

Gruppenadressimport

.knxproj oder .esf aus ETS4/ETS5

Um die Konsistenz mit anderen Busteilnehmern zu gewährleisten, können Gruppenadressen nicht angelegt werden, sondern müssen aus der ETS importiert werden. Dazu stehen grundsätzlich zwei Formate zur Verfügung. Der bevorzugte Weg ist der Import der .knxproj-Datei. Diese erzeugen Sie, indem Sie das ETS-Projekt exportieren.

Aktuell werden ETS4 und ETS5-Projekte unterstützt. Voraussetzung ist, dass das Projekt dreistufige Gruppenadressen verwendet.

Datentyp-Inferenz

Beim Import wird versucht, jeder Gruppenadresse einen Datentypen zuzuordnen. Besitzt weder die Gruppenadresse noch die Verknüpfungen einen DPT, so wird ein vorzeichenloser Ganzzahltyp der entsprechenden Größe des Kommunikationsobjektes angenommen. Bei nicht-verknüpften Gruppenadressen kann kein Typ bestimmt werden. **Typenlose Gruppenadressen können nicht verknüpft werden.**

Alternativ steht der .esf-Import (wie unter EibStudio 3) zur Verfügung. Dieser beinhaltet allerdings nur unzureichende Typinformationen und sollte nur verwendet werden, falls der .knxproj-Import nicht möglich ist. Die .esf-Datei wird in der ETS mit der Funktion „OPC-Export“ erstellt.

Der Typ jeder Gruppenadresse kann beliebig geändert werden.

Falls ein falscher Typ gewählt wird, findet keine korrekte Interpretation der Telegramme statt!

Topologie

Beim .knxproj-Import wird zudem die Topologie des Projekts ausgelesen und als Baum dargestellt. Dies ist insbesondere für die Zuordnung von phsikalischen Adressen im Gruppenmonitor hilfreich (S. 25).

Interne Variablen

Variablen können gezielt vom Benutzer angelegt werden, um einen internen Zustand zu speichern, ohne ihn auf den KNX-Bus senden zu müssen. Außerdem werden interne Variablen für die Logik, die Visualisierung, sowie für Makros in Expertenprogrammen deklariert. Diese Variablen werden normalerweise ausgeblendet, können aber bei Bedarf in der Variablenübersicht sowie im Debugger eingeblendet werden.

Konstanten

Im EibStudio sind Variablen vordefiniert, um das Erstellen eines Anwenderprogramms möglichst einfach zu gestalten. Diese Variablen sind im EibStudio unter **Objekte** → **Konstanten** gelistet. Vordefinierte Variablen können nicht überschrieben werden.

Wird ein neues Projekt angelegt, so muss das (leere) Projekt ein erstes Mal kompiliert werden, um alle Konstanten zu laden.

Logik

Die Logikmodellierung erlaubt es auf einfache Weise, Objekte (Gruppenadressen, Variablen) mit Operationen zu verknüpfen.

Folgende Begrifflichkeiten werden hierbei unterschieden:

Definitionen

Knoten

Hat einen festgelegten Typ und repräsentiert entweder ein Objekt oder eine Operation.

Eingang

„Anfasser“ auf der linken Seite eines Knotens. Kann mit mindestens einem Ausgang durch eine Kante verbunden werden, jedoch nicht mehrfach mit demselben Ausgang.

Ausgang

„Anfasser“ auf der rechten Seite eines Knotens. Kann mit mindestens einem Eingang durch eine Kante verbunden werden, jedoch nicht mehrfach mit demselben Eingang.

Port

Eingang oder Ausgang.

Kante

Verbindet genau einen Ausgang eines Knoten mit genau einem Eingang eines anderen Knotens.

Auslöser

Port, der beim Wechsel von 0b01 auf 1b01 die entsprechende Funktion des Knotens startet. Bleibt der Auslöser auf 1b01, so wird die Funktion nicht erneut ausgeführt.

Sammeleingänge

Besitzt ein Port mehrere Kanten, so sind diese gleichwertig. Knoten, bei denen die Parameterreihenfolge keine Rolle spielt (z.B. ADDITION), besitzen der Übersichtlichkeit halber lediglich einen einzigen Eingang, mit dem alle Parameter verbunden werden.

Typen

Jeder Port besitzt einen Typ (S. 26). Es können nur kompatible Ports miteinander verbunden werden. Folgende Möglichkeiten bestehen bei der Angabe des Typs und der möglichen Verbindung:

*: Alle Typen

Beliebige Typen

b, u, f: Typklasse

Beliebige Typen der gleichen Klasse

b01, u08, f16: Spezifischer Typ

Verbindung mit genau diesem Typ

Beispiele:

Ein Eingang vom Typ b01 darf mit Ausgängen der Typen *, b, b01 verbunden werden.

Ein Ausgang vom Typ u,s darf mit Eingängen der Typen *, u, uXX, s, sXX verbunden werden.

Dabei gilt allerdings: Zur Übersetzung muss ein spezifischer Datentyp feststehen. Mit jeder neuen Knotenverbindung werden die Typen der verbundenen Knoten und ihrer Vorgänger und Nachfolger aktualisiert.

Entfernen von Kanten

Wird eine Verbindung entfernt, bleiben die Typen erhalten. **Gegebenenfalls muss also der Knoten entfernt und neu hinzugefügt werden.**

Konvertieren

Falls inkompatible Typen verbunden werden sollen, so müssen diese explizit konvertiert werden (Knoten TYPKONVERTIERUNG). Grundsätzlich kann jeder Typ in jeden anderen Typ konvertiert werden. Ein eventueller Informationsverlust muss mit bedacht werden.

Mehrere Logiken

Logiken können beliebig aufgeteilt werden. Einzelnen Logiken sind gleichberechtigt. Werden also dem gleichen Objekt in unterschiedlichen Logiken Werte zugewiesen, so hat das Objekt jeweils den Wert der zeitlich zuletzt ausgewerteten Zuweisung. Findet die Zuweisung gleichzeitig statt, so ist das Ergebnis undefiniert.

Debug-Modus

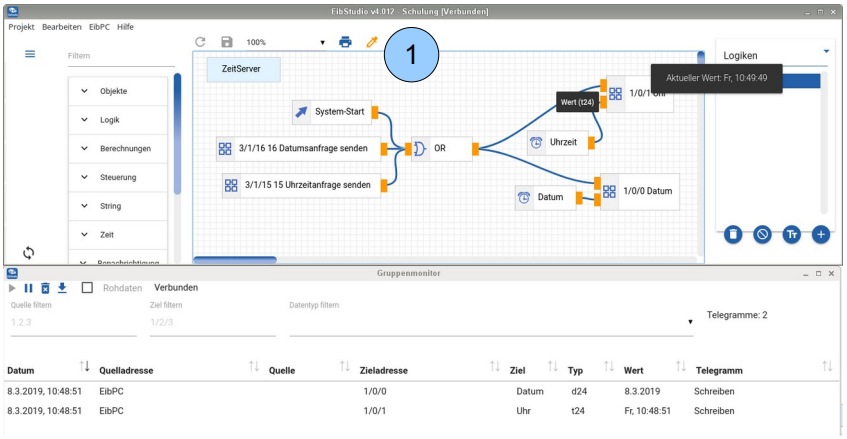


Abbildung 8: Debug-Modus im Logikeditor

Der Logikeditor legt Variablen an, die normalerweise vor dem Anwender verborgen werden (S. 15). Um trotzdem den aktuellen Zustand jedes Knotens innerhalb einer Logik zu erfahren, kann der Debug-Modus aktiviert werden (1).

Es werden alle Ports farbig hervorgehoben. Durch einfaches Klicken auf einen Port wird der aktuelle Objektzustand abgefragt. Mittels `Strg+Klick` kann der Zustand auch gezielt verändert werden.

Um ungewünschte Reaktionen anderer Geräte zu vermeiden, empfiehlt sich die Simulation für umfangreiche Tests (S. 25).

Die Logik in Abbildung 8 zeigt den EibPC als Zeitgeber für den KNX-Bus. Beim Systemstart (also nachdem der EibPC sich mit einem NTP-Server synchronisiert hat) wird Datum und Uhrzeit mit den entsprechenden DPTs auf den Bus gesendet.Durch die verknüpften Gruppenadressen kann die aktuelle Zeit auch manuell gesendet werden. Die aktuelle Uhrzeit wurde gerade abgefragt und wird oben rechts eingeblendet. Der Gruppenmonitor zeigt die beiden Telegramme.

Visualisierungsobjekte

Falls die vordefinierten Visualisierungselemente und Funktionen nicht ausreichen, ist es mit dem Logikeditor sehr leicht, auf Elemente zuzugreifen. Dazu muss in der **VISU** in den Eigenschaften des Elements nur die Konfigurationsoption „Mit Logik verknüpfen“ gewählt werden.

Dadurch steht das Element in der **LOGIK** zur Verfügung. Fügen Sie den Visualisierungsknoten des entsprechenden Typs hinzu. In dessen Eigenschaften können Sie nun das Visualisierungselement auswählen.

Hinweis: Falls Sie bei komplexeren Logiken auch eine Rückmeldung auf das Visualisierungselement verwenden, können Sie den Visualisierungsknoten auch duplizieren (Kopieren `Strg+c`, Einfügen `Strg+v`) und auf beiden Seiten Ihrer Logik platzieren. Es genügt dabei, jeweils nur die Eingänge oder Ausgänge zu verbinden.

Visu

Visualisierungen können in EibStudio bequem und schnell erstellt werden.

Die Visualisierung ist aufgeteilt in Gruppen von Seiten. Jede Seite ist in Größe und Design konfigurierbar. Die Reihenfolge der Seiten entspricht der späteren Darstellung im Webserver und kann durch Ziehen mit der Maus verändert werden.

Grundsätzlich wird zwischen Elementen und Funktionen unterschieden.

Elemente

Elemente sind alle darstellbaren Schaltflächen, Graphen usw. Das Verhalten jedes Elements kann in seinen Eigenschaften konfiguriert werden. Die meisten Funktionen lassen sich direkt im Element realisieren, z.B. ein einfaches Schaltelement, das eine Gruppenadresse umschaltet, oder einen Slider für einen Prozentwert.

Funktionen

Funktionen sind vorkonfigurierte Elemente oder Gruppen von Elementen mit eigenen Eigenschaften. Soll eine Funktion verwendet werden, müssen alle Elemente, die zu dieser Funktion gehören, auf der Seite Platz finden. Anderenfalls wird die gesamte Funktion nicht eingefügt.

Die Platzierung der Elemente in der Vorschau kann durch Ziehen mit der Maus geändert werden und entspricht der späteren Darstellung im Browser.

Eigene Vorlagen

Die aktuell angezeigte Seite kann als eigene Vorlage in den Benutzereinstellungen gespeichert werden. Dadurch ist es möglich, die gleichen Seiten in mehreren Projekten zu nutzen. Diese Vorlagen können nicht editiert werden, sondern müssen bei Bedarf als neue Vorlage gespeichert werden. Es werden alle Objektverknüpfungen kopiert. Unterscheiden sich die Gruppenadressen, müssen sie vor der übersetzung angepasst werden.

Vorlagen

Zusätzlich stehen vordefinierte Vorlagen, z.B. für den KNX SmartMeter bereit.

Verknüpfung mit Logik und Experte

Falls die vordefinierten Funktionen nicht ausreichen, ist es über eine Verknüpfung des Visualisierungselements mit der **Logik** oder dem **Experten** möglich, das Verhalten selbst zu definieren.

Hierbei ist die Verwendung in der Logik sehr einfach, mit bewusst reduzierter Flexibilität in der Veränderung des Aussehens. Es kann zwischen dem Grundzustand und dem Aktiv-Zustand gewechselt werden. Es ist also z.B. nicht möglich, zwischen rotem und grünem Text für einen Button umzuschalten, der Grundzustand des Textes ist stets grau (siehe S. 17).

Eine Verknüpfung mit dem Experten bietet die meiste Flexibilität. Hier wird der Grundzustand festgelegt und eine eindeutige Variable definiert, über die im Experten auf das Visualisierungselement zugegriffen wird (siehe S. 19).

Experte

Der Experte bietet vollen Zugriff auf alle Funktionen des EibPCs. Eine detaillierte Beschreibung der Funktionen finden Sie im Kapitel Experte (S. 76).

Es können beliebig viele Expertenprogramme angelegt werden, die alle im globalen Kontext kompiliert werden. Das bedeutet auch, dass eine Variable, die in einem Expertenprogramm angelegt wurde, in allen anderen verwendet werden kann (und nicht mehr neu definiert!).

Autovervollständigung

Der Texteditor unterstützt die Programmierung, indem automatisch passende Funktionen oder Objekte bei der Eingabe vorgeschlagen werden. Diese Listen werden beim Kompilieren generiert. Legen Sie also eine neue Variable an, so wird diese erst nach dem ersten Kompiliervorgang für die weitere Verwendung vorgeschlagen.

Für den Zugriff auf Gruppenadressen genügt es, doppelte Anführungszeichen für den Beginn einer Gruppenadresse, gefolgt von aussagekräftigen Namensbestandteilen einzugeben, z.B. "123“ um Vorschläge für die Gruppenadresse 1/2/3 zu erhalten.

Makros

Makros sind Funktionsblöcke, mit denen Expertenprogramme zusätzlich strukturiert werden können. Zudem wird eine umfangreiche Sammlung an Makros in Form von internen Makrobibliotheken mitgeliefert. Wie Sie eigene Makros definieren können, wird im Kapitel Makros (S. 231) erklärt.

Eigene Visualisierungsseiten

Innerhalb von Expertenprogrammen können Sie auf den Webserver zugreifen, indem Sie die Direktive #addto [WebServer] verwenden.

Möchten Sie eigene Visualisierungsseiten oder globale Visualisierungselemente im Experten nutzen, so muss gewährleistet werden, dass diese nicht mit der **Visu** kollidieren.

Eine korrekte Vergabe der IDs ist insbesondere wichtig, wenn ein EibStudio 3-Projekt importiert wurde, und zu den bereits vorhandenen „programmierten“ Visualisierungsseiten über die **Visu** generierte hinzugefügt werden sollen.

Prüfen Sie dazu, welche IDs bereits verwendet wurden, und tragen Sie die erste freie ID in den **Projekteinstellungen → IDs** (S. 24) ein.

Zugriff auf Visualisierungselemente

Für den Zugriff auf einzelne Visualisierungselemente innerhalb eines Expertenprogramms haben Sie auch die Möglichkeit, diesem einen eindeutigen Variablennamen zuzuordnen. Wählen Sie dazu den passenden Konfigurationseintrag des Elements innerhalb der **Visu** und übersetzten Sie das Projekt, um die Variablenliste zu aktualisieren.

Dieser Name muss den Anforderungen an Variablennamen (S. 30) genügen.

Beim Kompilieren der Visualisierung wird diesem Variablennamen die ID des Visualisierungselements zugewiesen. Handelt es sich um ein seitenbezogenes Element (z.B. Button, Shifter), so wird außerdem eine Variable mit dem gleichen Namen und der Endung „_P“ mit der Seiten-ID angelegt.

Beispiel:

Als eindeutige Variable wird einem Button *ButtonVar* zugewiesen. Da es sich um ein seitenbezogenes Element handelt, wird außerdem die Variable *ButtonVar_P* angelegt. Nach dem nächsten Kompilieren existieren beide Variablen.

Diese können nun von den Visualisierungsfunktionen (S. 209) verwendet werden:

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

Falls Sie eigene Visualisierungsseiten in Expertenprogrammen definiert haben, müssen Sie unbedingt die Start-IDs für die Visu entsprechend setzen (S. 24).

Seitenbezogene Visualisierungselemente:

Button, *Shifter*, *Multibutton*, *Multishifter*, *Slider*, *Picture*, *Plink*, *Wertediagramm*, *Zeitdiagramm*.

Nicht-seitenbezogen:

Webinput und *Weboutput*.

Enertex® Bayern GmbH – Ebermannstädter Straße 8 - 91301 Forchheim - mail@enertex.de

Syntax

Variablendefinition

Variablen werden angelegt, indem ihnen ein Initialwert zugewiesen wird. Die Definition muss dabei eindeutig sein. Variablen sind auf Seite 30 detailliert beschrieben.

```
Var=1b01
```

Gruppenadressen

Um auf den aktuellen (interne) Wert einer Gruppenadresse zuzugreifen, wird der Objektname verwendet. Dieser wird als Name unter **Objekte → Gruppenadressen** angezeigt und setzt sich aus dem in der ETS vergebenen Namen und der Gruppenadresse (Haupt-, Mittel-, Untergruppe), getrennt durch ein – zusammen (siehe S. 30). Der Wert der Variablen Var ändert sich mit dem Zustand der Gruppenadresse.

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

if-Anweisungen

Die einfachste Form der if-Anweisung

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

eignet sich besonders für einfache wenn-dann-Fälle.

Allgemein ist die if-Anweisung definiert als

if (Abfragebedingung) then Anweisung{sblock}1 else Anweisung{sblock}2 endif

Die Abfragebedingung muss ein Ausdruck vom Typ b01 sein.

Eine Anweisung entspricht dabei einer Zuweisung, einem Funktionsaufruf oder einem Makroaufruf. Diese werden durch „;“ (Semikolon) getrennt.

Falls zwischen den Anweisungen zur besseren Lesbarkeit Zeilenumbrüche eingefügt werden, muss der Anweisungsblock in {} stehen:

```
if ("Schalter-1/0/0"==EIN) then {
        write("Lampe-1/1/1",EIN);
        write("Dimmer-1/1/2"u08,80%);
} else {
        write("Lampe-1/1/1",AUS);
        write("Dimmer-1/1/2"u08,0%);
} endif
```

Kommentare

Sie können ihre Programmierung kommentieren. Dazu gibt es zwei Möglichkeiten:

1. Kommentare beginnen mit „//“ und stehen am Anfang einer Zeile.
2. **Anstelle** einer Anweisung können Kommentare an beliebiger Stelle stehen (Strichpunkt beachten), die durch /* */ eingerahmt werden. Z.B.

```
/* Dies ist ein Kommentar */
// Das ist noch ein Kommentar
u=5;/* Und dies ist noch Kommentar */; u4=5
```

Online Debuggen zur Laufzeit

Wenn zur Laufzeit die Zustandsänderung beobachtet werden soll, so empfiehlt es sich mit Hilfe von UDP Datagrammen und einem netcat Client (vgl. https://de.wikipedia.org/wiki/Netcat) zu debuggen.

Als Debug-Makro wird folgender Code verwendet, der davon ausgeht, dass die Gegenstelle 192.168.1.18 auf Port 9000 hört, z.B. mit dem Unix-Tool `netcat -ul 9000` konfiguriert:

Einen String mit CR auf einen UDP Client schicken

Leermakro

```
#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
```

Je nachdem, ob das Debugging mit dem `#define DEBUG` eingeschaltet ist, wird über UDP eine Nachricht verschickt. Im Falle, dass das `#define DEBUG` auskommentiert wird, werden keine Nachrichten verschickt. Eine Besonderheit stellt dabei die Verwendung von __EMPTY() dar. Diese Anweisung sorgt dafür, dass das Makro nichts expandiert, also auch keinen Code erzeugt.

Effizient bei inaktivem #define von DEBUG

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

Nun wird bei aktiven `#define DEBUG` über UDP der Wert an den Empfänger zur Laufzeit des Programms automatisch übertragen. Wenn `//#define DEBUG` auskommentiert wird, so erzeugt die Zeile vmDebugUDP(*$x ist nun $*+convert*(x,$$)*); keinen Overhead.

Wenn hingegen eine If-Anweisung **nur** zu Debuggzwecken eingerichtet wird, etwa:

Ineffizient bei inaktivem #define von DEBUG – if Abfrage, die nur dem Debuggen dient.

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

so erzeugt der Compiler zwar für vmDebugUDP keinerlei Objekte, allerdings wird ein „verweistes“ if x>5 Objekt angelegt. Diese Art des automatischen Debuggings sollte daher vermieden werden oder eben mit `#define` im Code komplett inaktiviert werden:

... dann lieber so.

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

Projekteinstellungen

In den Projekteinstellungen befinden sich alle Einstellungen, die einen konkreten EibPC bzw. eine Installation betreffen.

Änderungen an den Einstellungen des EibPCs müssen gezielt auf diesen übertragen werden. Dies geschieht, je nach Einstellungen, entweder beim Übertragen des Programms (P), oder durch Betätigen einer Schaltfläche (S).

Suche

Sie können über alle installierten Netzwerkkarten nach vorhandenen EibPCs suchen.

KNX-Verbindung

(P)

Wählen Sie die passende Verbindungsart. Eine Änderung dieser Einstellung wird mit der nächsten Programmübertragung aktiv. Sie können das Senden von Leseanforderungen beim Start des Gerätes global deaktivieren. Dies ist insbesondere beim Entwickeln von Vorteil, um den Start zu beschleunigen

Netzwerkadresse

(S)

Bei Änderungen an der Netzwerkverbindung wird der EibPC neu gestartet. Sollten Sie den EibPC nicht mehr erreichen, können Sie ihn auf Werkseinstellungen (S. 10) zurücksetzen.

Namensauflösung

Für einige der Netzwerkfunktionen (sendmail, resolve) benötigt der EibPC einen Zugang zu einem DNS Server.

Ports

(P)

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

Projekt-Passwort

EibStudio > 5.300

Projekte können verschlüsselt gespeichert werden. Dies ist notwendig, falls KNX Data Secure verwendet wird. Das Projektpasswort kann später jederzeit geändert oder entfernt werden. Falls Secure-Gruppenadressen importiert wurden, muss das Projekt verschlüsselt gespeichert werden. Exportiere Projekte werden ebenfalls durch dieses Passwort geschützt (Ausnahme: Support-Export).

Verbindungsverschlüsselung

EibStudio > 5.300

Um den EibPC vor Veränderungen zu schützen, können Sie die Verbindungsverschlüsselung zwischen EibPC und EibStudio aktivieren. Dadurch kann nur mit dem passenden Projekt ein neues Anwenderprogramm übertragen oder Änderungen an der Konfiguration durchgeführt werden. Falls Sie das Projekt verlieren, muss das Gerät auf Werkseinstellungen zurückgesetzt werden.

Um die Verbindungsverschlüsselung zu aktivieren, benötigen Sie physikalischen Zugriff auf das Gerät. Es muss außerdem ein Projekt-Passwort vergeben werden. Drücken Sie die Display-Taste am Gerät, bis das "Initial Password" angezeigt wird und geben Sie dieses im EibStudio ein. Ist die Aktivierung erfolgreich, zeigt das Display "Encryption enabled" und ein Zugriff ist nur noch mit dem passenden Projekt möglich.

KNX Secure Sequenzzähler

EibStudio > 5.300

Falls Sie im Projekt durch KNX Data Secure geschützte Gruppenkommunikation verwenden, können Sie die hier die Sequenzzähler der Secure-Geräte und des EibPCs anzeigen und verändern.

Achtung: Falls Sie den Sequenzzähler des EibPCs selbst erhöhen, müssen alle zukünftigen Telegramme, die vom EibPC gesendet werden, ebenfalls einen höheren Sequenzzähler verwenden. Anderenfalls werden seine Telegramme von anderen Geräten ignoriert.

Datum und Uhrzeit

(S)

Für das korrekte Funktionieren der Zeitfunktionen ist es wichtig, dass der EibPC eine korrekte Uhrzeit besitzt. Es bietet sich hier an, diese Zeit mit anderen Busteilnehmern synchron zu halten. Dazu kann der EibPC Zeitinformationen vom KNX-Bus verwenden, um seine eigene Uhr zu synchronisieren. Ist in der KNX-Installation keine zuverlässige Zeitquelle vorhanden, kann dies auch der EibPC übernehmen, und seine Systemzeit als Telegramm auf den Bus senden.

Zusätzlich kann der EibPC seine Systemuhr über das NTP-Protokoll mit einem Server im lokalen Netzwerk oder im Internet synchronisieren.

Falls die Verwendung eines NTP-Servers konfiguriert ist, wird dieser zum Setzten der Zeit verwendet, auch wenn zwischenzeitlich die Uhr manuell verstellt wurde. Die Synchronisierung erfolgt initial beim Programmstart. Wenn nach fünf Minuten kein NTP-Server erreicht werden konnte, beginnt das Anwendungsprogramm trotzdem.

Position

(P)

Der EibPC berechnet für das aktuelle Jahr den Sonnenstand voraus, so dass zu jedem 5-Minuten-Intervall die Sonnenposition zur Verfügung steht. Dazu werden die Koordinaten des Aufstellortes benötigt. Müssen die Sonnendaten neu berechnet werden, dauert dies ca 5 Minuten.

SHUTDOWN-Variable

(S)

Es besteht die Möglichkeit, bei jedem Neustart des EibPC-Programms die Variable *SHUTDOWN* auf 1b01 zu setzen. Dies ermöglicht das persistente Speichern von Daten auf dem Flash. Eine Verzögerung von 5 Sekunden ist hierbei eine sinnvolle Konfiguration.

FTP

(P)

Der EibPC kann alle Telegramme vom KNX-Bus zur Protokollierung auf einen FTP-Server hochladen. Dazu muss der Server auf Port 21 auf eingehende Verbindungen warten. (P)

E-Mail

(P)

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

Sicherung

(S)

Es ist möglich, bei jeder Programmübertragung das aktuelle Projekt zur Sicherung abzulegen. Über die entsprechende Schaltfläche kann das Projekt heruntergeladen werden. Sie können es bei Bedarf als neues Projekt importieren.

Dateien

(S)

Um eigene Grafiken in der Visualisierung verwenden zu können, müssen diese auf den EibPC übertragen werden. Die Grafikdateien werden außerdem im Projektverzeichnis gespeichert. Wird das Projekt auf einen anderen EibPC übertragen, so werden automatisch die hinzugefügten Dateien synchronisiert. Gleiches gilt in die Rückrichtung. Sind also Dateien auf dem EibPC gespeichert, die noch nicht dem Projekt hinzugefügt wurden, werden diese automatisch heruntergeladen. Die so hochgeladenen Dateien können in der Visualisierung mit dem Pfad */upload/<Dateiname>* verwendet werden, z.B. mit dem picture-Element und dem Pfad */upload/enertex.jpg*.

Verwenden Sie keine Umlaute oder Sonderzeichen als Dateiname.

HTTPS

(S)

Mit dem EibPC ist eine sichere Kommunikation via HTTPS zwischen WebServer und Browser möglich. Dazu muss lediglich ein SSL-Zertifikat auf dem EiBPC vorhanden sein und ein Benutzer mit Passwort eingetragen werden.

Um diese Funktionalität nutzen zu können, benötigen Sie einen gültigen Freischaltcode für die Option NP. Zusätzlich muss in Ihrem Router Port Forwarding für den Port 443 eingerichtet sein.

VPN

(S)

Der EibPC verfügt über einen OpenVPN-Server. Auch hierfür muss vorher ein Zertifikat generiert werden. Legen Sie Benutzer mit einem jeweiligen Passwort an. Der VPN-Server muss mit dem entsprechenden Befehl startvpn oder dem passenden Logikknoten gestartet werden. Dies kann in Verbindung mit dem Befehl systemstart auch automatisch bei jedem Start erfolgen. Um von „außen“, d.h. außerhalb des aktuellen Netzes auf das Gerät zugreifen zu können, mussd er UDP-Port 1194 an die IP des EibPCs weitergeleitet werden.

IDs

(P)

Die Firmware des EibPCs organisiert den Zugriff auf interne Objekte über eindeutige Nummern, sog. IDs. Diese werden bei der Definition angegeben und müssen beim Zugriff verwendet werden, um das entsprechende interne Objekt anzusprechen. Verändern Sie die Start-IDs, falls Sie eigene IDs verwenden, um Kollisionen mit automatisch vergebenen IDs zu vermeiden.

Export und Import

Projekte werden über das Fenstermenü **Projekt → Exportieren** gepackt. Im Gegensatz zum **Hilfe → Export für Support** beinhaltet der Export auch alle „sensiblen" Daten wie das E-Mail-Passwort. Wird ein Projekt-Passwort verwendet, ist das exportierte Projekt ebenfalls mit dem gleichen Passwort verschlüsselt. Der **Export für Support** ist hingegen unverschlüsselt, beinhaltet aber keine Verschlüsselungsinformationen für KNX Data Secure.

Debugger

Der Debugger kann über das Fenstermenü **EibPC → Debugger** geöffnet werden. Er bietet die Möglichkeit, den internen Status von Verarbeitungsobjekten zu verändern, Gruppentelegramme zu senden und aktuelle Werte abzufragen.

Gruppenmonitor

Der Gruppenmonitor kann über das Fenstermenü **EibPC → Gruppenmonitor** geöffnet werden. Er zeigt alle Telegramme, die der EibPC empfängt. Wurde eine .knxproj-Datei importiert, so werden zu den physikalischen Adressen die Gerätenamen angezeigt.

Es werden stets die letzten 100 Telegramme angezeigt. Diese können als .csv-Datei exportiert werden.

Telegrammpuffer

In Ergänzung zum Gruppenmonitor, mit dem stets nur die letzten Telegramme ausgewertet werden können, Speichert der EibPC die letzten 500.000 Telegramme im Telegrammpuffer. Dieser kann über das Fenstermenü **EibPC → KNX-Telegramme abholen** ausgelesen und als .csv-Datei gespeichert werden.

Ereignisspeicher

Sobald bei der Verarbeitung im EibPC etwas unvorhergesehenes passiert, wird das Ereignis protokolliert. Das Ereignisprotokoll kann über das Fenstermenü **EibPC → Ereignisse** ausgelesen werden. Zur Erklärung der Ereignisse siehe S. 258.

Simulation

Bei komplexeren Logiken ist es für die Erstellung hilfreich, diese vorab simulieren zu können. Stellen Sie dazu die Verbindung zum KNX-Bus auf „Simulation" (S. 22). Im Gruppenmonitor werden nun weiterhin alle Telegramme angezeigt, die der EibPC sendet, ohne andere Geräte zu beeinflussen.

Um die Kommunikation mit anderen Geräten nachzubilden, können Sie z.B. in der Logik auf Leseanfragen reagieren und gezielt Testwerte senden. Eine einfache Simulation ist in Abbildung 5 dargestellt.

Fügen Sie drei Gruppenadress-Knoten hinzu und konfigurieren Sie sie wie folgt:

1. Ereignis erzeugen, sobald eine Leseanfrage an die Gruppenadresse gesendet wird
2. Den aktuellen Wert am Ausgang ausgeben
3. Ein Antwort-Telegramm senden, mit der Leseanfrage als externem Trigger

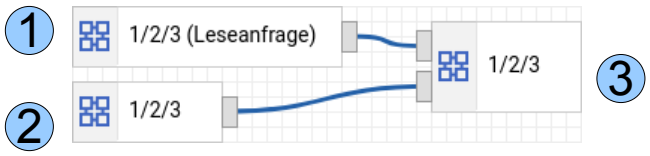


Figure 9: Simulation Leseanfrage

Auf diese Weise können Sie schnell komplette Testumgebungen schaffen, z.B. für eine automatische Beschattung. Dies ist deutlich einfacher, als im Debugger immer wieder manuell Gruppenadressen zu beschreiben.

Beachten Sie, dass ohne Busverbindung Leseanforderungen beim Start nicht beantwortet werden können. Pro Anfrage wird 1,5 Sekunden gewartet, was den Systemstart unnötig verzögert. Um diese unnötige Wartezeit zu vermeiden, können Sie in den Einstellungen der Bus-Verbindung die Initialisierung komplett deaktivieren.

Achten Sie aber darauf, diese wieder zu aktivieren, wenn Sie die Simulation beenden.

Objekte

Objekte repräsentieren Zustände, auf die der EibPC mit der Erzeugung neuer Zustände reagieren kann. Vereinfacht ausgedrückt lassen sich die meisten Aufgaben auf *Wenn dies, dann tue jenes* abbilden. Objekte sind dabei sowohl die Bedingung als auch das Ziel.

Es gibt im EibPC zwei verschiedene Arten von Objekten: Gruppenadressen und Variablen.

Gruppenadressen

Gruppenadressen sind Objekte, die für andere Busteilnehmer relevant sind. Jeder Teilnehmer muss die für ihn relevanten Zustandsänderungen, die als Bustelegramme an Gruppenadressen gesendet werden, mit seinem intern gespeicherten abgleichen und bei Bedarf auswerten.

Variablen

Darüber hinaus speichern Busteilnehmer eigene interne Zustände, die zunächst auch nur intern verarbeitet werden. Der EibPC kennt diese als Variablen.

Beispiel: Ein Schaltaktor reagiert auf eine Gruppenadresse, die mit seinem Kommunikationsobjekt *Schalte Kanal 1 um* verknüpft ist. Der Schaltaktor weiß dabei seinen internen Zustand, den er benötigt, um entweder ein oder aus zu schalten. Außerdem informiert er andere Busteilnehmer über die Änderung, indem er ein Status-Telegramm auf den Bus sendet.

Bei diesem Schaltvorgang sind die Gruppenadressen für Aktor und Status, aber auch der interne Schaltzustand beteiligt.

Der EibPC als universelle Logikmaschine funktioniert nach dem gleichen Prinzip, nur dass das Regelwerk durch das Anwenderprogramm und damit Sie selbst vorgegeben wird. Es kann also jedes Objekt mit jedem anderen Objekt über eine Vielzahl interner Funktionen verknüpft werden.

Datentypen

In der ETS liegen Datentypen als Datenpunkttyp (DPT) vor. Dieser spezifiziert die Größe eines Objekts und legt ggf. auch seine Interpretation fest. Ein 1-Bit Objekt vom Typ DPT 1 kann z.B. als DPT 1.001 Ein/Aus oder DPT 1.008 Hoch/Runter interpretiert werden.

Beim Import aus der ETS werden diese Typen auf interne abgebildet, die ausschließlich ihre Größe beschreiben. Diese setzen sich aus dem Typ und der Bitlänge wie folgt zusammen:

Mögliche Typen sind (angelehnt an übliche Programmiersprachen):

- vorzeichenlose (pos.) ganze Zahlen — Buchstabe u („unsigned“)
- vorzeichenbehaftete ganze Zahlen — Buchstabe s („signed“)
- Fließkommazahlen — Buchstabe f („float“)
- Zeichenketten — Buchstabe c („char“)
- Datum und Zeit — Buchstabe t bzw. d bzw. y („time“, „day“, „year“)

Folgende Längen sind möglich

- 1 Bit — Ziffern 01
- 4 Bit — Ziffern 04
- 8 Bit — Ziffern 08
- 16 Bit — Ziffern 16
- 24 Bit — Ziffern 24
- 32 Bit — Ziffern 32
- 64 Bit — Ziffern 64

Zeichenketten

- 14 Zeichen — Ziffern 14 für DPT 16
- 1400 Zeichen — keine Ziffern, Standardlänge
- freie Länge — Länge zwischen 1 und 65534 Zeichen jedoch nicht 14 im Folgenden ohne Längenangabe als c angegeben

Demnach ist u08 eine Datentyp der Länge 8 Bit und stellt eine vorzeichenlose (positive) ganze Zahl dar.

Zahlen (Konstanten)

Mit Hilfe des Datentyps können Zahlen und Konstanten im EibStudio deklariert werden.

Bei Zahlen wird die Zahl dem Datentyp vorangestellt, also z.B.

- 2u08 Unsigned 8-Bit-Integer: 2
- 2.0f16 Fließkommazahl 2.0
- -6s32 Ganze Zahl mit Vorzeichen -6
- 33.2% Prozentwert 33.2 (entspricht 84)
- 35 Unsigned 8-Bit-Integer: 35

Ungültige Syntax erkennt der EibParser (integrierter Compiler im EibStudio) und erzeugt eine Fehlermeldung.

Bei ganzen positiven Zahlen der Länge 8 Bit und Fließkommazahlen der Länge 16 Bit kann auf die Angabe des Datentypen verzichtet werden, d.h. Werte in der Form

- 0 ... 255 sind vom Datentyp u08.
- 2.0 (Dezimalpunkt in der Zahl) sind vom Datentyp f16.

Bei diese beiden Zahlentypen ist die Angabe von Datentypen optional.

Sondertyp: % (Prozentwert)

In der ETS Programmierung werden Prozentwerte „%“ verwendet. Diese sind kompatibel zum Datentyp „u08“ und werden von den KNX-Aktoren durch Skalierung intern angepasst. Um dem Anwender hier die Programmierung zu vereinfachen, haben wir den Prozentwert für Konstanten definiert. Dabei kann der Prozentwert mit einer Kommastelle genau angegeben werden, also z.B. 2.3%. Aufgrund der Skalierung entspricht 100% einem Wert von 255u08 oder allgemein lautet die Umrechnung einer Größe Y% wie folgt:

$$X[u08]=\frac{Y[\%]}{100}\cdot 255 \qquad \text{bei Abschneiden der Kommastellen}$$

Der integrierte Compiler im EibStudio nimmt diese Anpassung für Sie vor, so dass Sie hier Aktoren wie gewohnt ansprechen können.

Wenn Sie unterschiedliche Datentypen in ihrem Anwenderprogramm miteinander verknüpfen, z.B. als die Summe von 2u08 und 2u32, so wird vom integrierten Compiler im EibStudio ein Fehler gemeldet, so dass nicht unbeabsichtigte Überläufe, numerische Probleme etc. auftreten können. Um dennoch solche Zahlen in einander umzuwandeln und damit verarbeiten zu können, benutzen Sie die convert-Funktion. Damit sind auch Umwandlungen von Zahlen in Zeichenketten möglich. Weiteres hierzu finden Sie auf Seite 146.

Hexadezimaldarstellung

Vorzeichenlose Ganzzahlen (Typ „u“) können auch in Hexadezimaldarstellung mit Präfix „0x“ angegeben werden. Der Compiler wandelt diese Darstellung in die entsprechende Zahl um.

- Datentyp u08: Es müssen 2 Stellen angegebenen werden: 0xF1 (= 241) oder 0xF1u08 (= 241)
- Datentyp u16: Es müssen mindestens zwei Stellen und der Datentyp „u16“ angegebenen werden: 0xF1A3u16 (= 61859u16)
- Datentyp u24: Es müssen mindestens zwei Stellen und der Datentyp „u24“ angegebenen 0x03A186u24 hexadezimale Schreibweise (=237958 bzw. Byte 1 0x03, Byte 2 0xA1 Byte 3 0x86)
- Datentyp u32: Es müssen mindestens zwei Stellen und der Datentyp „u32“ angegebenen werden: 0xF1A3u32 (= 61859u32)
- Datentyp u64: Es müssen mindestens zwei Stellen und der Datentyp „u64“ angegebenen werden: 0xF1A3u64 (= 61859u64)

Zeichenketten

Zeichenketten haben eine frei definierbare Länge zwischen 1 und 65534 Zeichen, z.B. ac1, ac65534. Ohne Längenangabe beträgt diese 1400 Zeichen. Um Speicher zu sparen, können kurze Texte z.B. als ausc3 definiert werden. Soll die Länge einer Zeichenkette automatisch bestimmt werden, kann die Typangabe c0 verwendet werden. Dies ist insbesondere für Konstanten hilfreich.

Einen Sonderfall stellt die $Zeichenkette$c14 dar. Dabei steht Zeichenkette für einen beliebigen Text, der aber nicht mehr als aus 14 Zeichen bestehen darf. Dieser Typ ist kompatibel zur KNX Zeichenkette DPT 16 (z.B: Anzeigeelemente), dessen Codierung auf ISO 8859-1 festgelegt ist.

Die beiden Zeichenketten-Typen c14 und Zeichenketten mit freier Länge können mit Hilfe der convert-Funktion ineinander umgewandelt werden (s. Seite 146).

Die Angabe von cXXXXX als Datentyp im Handbuch erlaubt jede beliebige Stringlänge, abgesehen von obiger Einschränkung zu c14.

IP Adresse

IP Adressen (Zusatzpaket Option NP) können Sie in der üblichen Syntax eingeben

- 192.168.22.100 kennzeichnet die gleichlautende Adresse. Intern ist diese Adresse identisch zum Datentyp u32.

Pyhsikalische Adresse

Physikalische Adressen können Sie in der folgenden Syntax eingeben

- 1.12.230 kennzeichnet die gleichlautende physikalische Adresse eines KNX Busteilnehmers. Diese Adresse identisch zum Datentyp u16.

Übersicht

Typ	Datentyp	Beispiel für Konstante	Verwendung	Wertebereich	DPT	EIS Datentyp
Binär	b01	1b01	Schaltaktor, Jalousieaktor	0, 1	1	EIS1/EIS7
2 Bit	b02	2b02	Sperrobjekte	0,1 ... 3	2	EIS8
4 Bit	bb04	10b04	Dimmen	0,1 ... 15	3	EIS2
Prozentwert	%	85.3%	Heizregler, Stellglieder	0,0.1 ... 100.0	5	EIS6/EIS14.001
8 Bit Integer ohne Vorzeichen	u08	255	Einfache Zahlen, Raumtemperaturregler etc.	0, ... 255	5	EIS6/EIS14.001
8 Bit Integer ohne Vorzeichen	u08	255u8	Optional mit Datentypen		5	EIS6/EIS14.001
8 Bit Integer mit Vorzeichen	s08	-45s08	Temperatursensor	-128... 127	6	EIS14.000
16 Bit Integer ohne Vorzeichen	u16	45u16		0 ... 65535	7	EIS10.000
16 Bit Integer mit Vorzeichen	s16	-450s16		-32768 ... 32767	8	EIS10.001
24 Bit Integer ohne Vorzeichen	u24	92235u24		0 .. 16777215	232.600	n.V.
32 Bit Integer ohne Vorzeichen	u32	92235u32		0 .. 4294967295		EIS11.000
IP Adresse	(u32)	192.168.22.100	Feste IP Adressen bei sendudp etc.			EIS11.000
32 Bit Integer mit Vorzeichen	s32	-9999s32		-2147483648 .. 2147483647	12	EIS11.001
64 Bit Integer ohne Vorzeichen	u64	92235u64		0 .. 18446744073709551615	13	n.V.
64 Bit Integer mit Vorzeichen	s64	-9999s64		-9223372036854775808 .. 9223372036854775807		n.V.
Short Float	f16	4.0	Windsensor	-671088.64 .. 670760.96		EIS5
Short Float	f16	4.0f16		-671088.64 .. 670760.96		EIS5
Float 32 Bit	f32	4.0e01f32		-3.40282e+38 .. 3.40282e+38	9	EIS9
Zeichenkette	c14	$HalloWelt$c14	Anzeigepanels	14 Zeichen	14	EIS15
Zeichenkette	(c1400)	$HalloWelt$	LAN Telegramme	1400 Zeichen		n.v.
Zeichenketten können von1 bis 65534 definiert werden	c65534	$HalloWelt$c65534		65534 Zeichen		n.v.

Tabelle 1: Datentypen

Hinweis: Die Datentypen d24, t24, y64 sind KNX DTP Typen, die über ihre Definition im EibPC korrekt verarbeitet werden. Eine Eingabe als Konstante ist nicht notwendig und daher nicht möglich. Diese Datentypen werden nur in Verbindung mit den Funktionen getdate und gettime benötigt.

Variablen

Variablen beginnen mit Buchstaben, gefolgt von einer beliebigen Anzahl und Kombination von Buchstaben oder Zahlen und dem „_“ Zeichen. Variablen werden mit einem Wert oder einer Funktion initialisiert. Groß- und Kleinschreibung wird im Gegensatz zu Schlüsselwörtern und Funktionsnamen beachtet.

Daher sind z.B. adresse und Adresse unterschiedliche Variablen.

Der Compiler „EibParser“ überprüft bei Zuweisung einer Variable und deren Weiterverarbeitung immer die Datentypen und unterbindet durch eine Fehlermeldung beim Generieren des Anwendungsprogamms unzulässige Verknüpfungen von nicht kompatiblen Datentypen, so dass nicht unbeabsichtigte Überläufe, numerische Probleme etc. auftreten können.

Wollen Sie Variablen mit unterschiedlichen Datentypen verknüpfen, benutzen Sie die convert-Funktion (siehe Seite 146).

Jeder Variable muss ein einziges Mal initialisiert werden. Die Deklaration von Variablen muss demnach eindeutig sein.

Ein paar Beispiele

```
a=123
A1=1b01
adresse=A1 or 0b01
Adresse=4%+5%+23u08
Wert=4e4*0.2
w=4e16f32
```

Variablen dürfen nicht von sich selbst abhängig definiert werden („Rekursion“). Daher ist folgender Ausdruck als Definition ungültig:

Nicht zulässig, aber...

```
a=a+1
```

Hingegen ist es zulässig, mit Hilfe von Variablen auf diese Weise einen Zähler zu programmieren:

... hier schon

```
//Deklaration
a=0
//Zählen
if (sun()) then a=a+1 endif
```

Keine Sonderzeichen in Variablennamen

Umlaute sind bei Variablennamen nicht erlaubt. Daher ist folgender Ausdruck **ungültig**

```
KücheLichtEin=1b01
```

Gruppenadressen

Der sinnvollste Weg, Gruppenadressen einem EibStudio-Projekt hinzuzufügen, ist der ETS-Import (S. 15).

Manuelle Gruppenadresse

Gruppenadressen können auch manuell definiert werden. Dazu steht die Gruppenadresse in einfachen Anführungszeichen, gefolgt von Datentyp, z.B. '1/2/3'b01.

Initialisieren von Gruppenadressen

Bevor der EibPC mit der Verarbeitung des Anwenderprogramms beginnt, kann der Benutzer den Speicher der Gruppeadressen initialisieren. Die Enertex ® EibPC speichert immer den aktuellen Stand des Inhaltes der Gruppenadresse als ein Abbild im Speicher. Wenn der EibPC ein neues Programm hochgeladen hat, so sind diese Abbilder im Speicher auf 0 gesetzt. Aber da der KNX Bus bereits vor dem EibPC läuft, werden die Abbilder im Speicher nicht mit den Zuständen der Gruppenadressen übereinstimmen.

Um sich mit dem KNX Bus vor der eigentlichen Verarbeitung zu synchronisieren, kann der Anwender Gruppenadressen mit einem Lesetelegramm vom EibPC abfragen. Dazu kann für jede importierte Gruppenadresse die Initialisierung unter **Objekte → Gruppenadressen** aktiviert werden.

Wichtige Erklärungen

- Initialisierung einer Gruppenadresse bedeutet, dass der EibPC vor dem eigentlichen Beginn der Verarbeitung Leseanforderungen auf den Bus ausgibt. Nach jeder Leseanforderung auf eine Gruppenadresse wird auf die Antwort gewartet, jedoch nicht länger als 1,5 s.
- Nach dem Abfragen der letzten Gruppenadresse wird mit der Verarbeitung begonnen.
- Alle Anweisungen die als Bestandteil einer der zu initialisierenden Gruppenadressen beinhalten, werden ungültig und die Verarbeitung im ersten Durchlauf ausgeführt.
- Wenn eine Gruppenadresse nicht rechtzeitig auf die Leseanforderung antwortet, wird ein Event generiert.

KNX Data Secure

Der EibPC kann mittels KNX Data Secure verschlüsselte Gruppenadresstelegramme am KNX-Bus senden und empfangen (Firmware, EibStudio > 5.300). Dadurch wird gewährleistet, dass nur berechtigte Geräte Werte senden können oder Aktionen im EibPC auslösen. Die Konfiguration erfolgt ausschließlich in der ETS. Anschließend wird das exportierte ETS-Projekt im EibStudio importiert.

Grundlagen

Secure Key

Bei KNX Data Secure werden Gruppenadresstelegramme verschlüsselt übertragen. Zur Verschlüsselung wird für jede Gruppenadresse ein eigener Schlüssel (Secure Key) verwendet, der dem Sender und dem Empfänger eines Telegramms vorab bekannt sein muss. Dieser wird von der ETS automatisch erzeugt, wenn eine Gruppenadresse nur mit KNX Data Secure-Geräten verknüpft ist, und in den Eigenschaften der Gruppenadresse „Sicherheit“ nicht deaktiviert wurde. Der Schlüssel wird im ETS-Projekt für jede Gruppenadresse gespeichert und beim Projektimport vom EibStudio übernommen.

Senderliste

Wird in der ETS ein Kommunikationsobjekt (KO) mit einer Secure-Gruppenadresse verknüpft, wird dem Gerät bei Übertragung der Applikation der Schlüssel für diese Gruppenadresse mitgeteilt. Ebenso wird für die Gruppenadresse gespeichert, welche Geräte Telegramme an diese Gruppenadresse senden dürfen. Deshalb muss bei neuen Sendern die Applikation aller Empfänger ebenfalls neu übertragen werden, so dass die ETS die Liste der erlaubten Sender aktualisiert.

Sequenzzähler

Um zu verhindern, dass mitgelesene Telegramme zu einem späteren Zeitpunkt von einem potentiellen Angreifer einfach wiederholt werden können, besitzt jedes Gerät einen eigenen Sequenzzähler, der mit jedem gesendeten Telegramm erhöht wird. Wird ein Telegramm mit einem niedrigeren Sequenzzähler empfangen als erwartet, wird das Telegramm verworfen. Der Sequenzzähler ist unabhängig davon, an welche Gruppenadresse das Telegramm adressiert ist. Im ETS Projektexport wird der aktuelle Stand des Sequenzzählers jedes Gerätes gespeichert. Beim Übertragen der Applikation wird jedem Secure-Gerät der aktuelle Stand des Sequenzzählers aller Geräte mitgeteilt, die mittels Secure-Gruppenadressen mit dem Gerät kommunizieren können.

Dieser muss jedoch nicht dem aktuellen Stand entsprechen, falls das Gerät zwischenzeitlich Telegramme gesendet hat und die ETS den Sequenzzähler nicht neu ausgelesen hat. Deshalb speichert jedes Gerät selbstständig den zuletzt empfangenen Wert für jedes verknüpfte Gerät zusätzlich ab.

Gruppenadressen für EibPC freigeben

Um vom EibPC mit KNX Data Secure verschlüsselte Telegramme an andere Geräte senden zu können, und Telegramme von anderen Geräten zu akzeptieren, muss allen beteiligten Geräten vorab bekannt sein, über welche Gruppenadressen sie kommunizieren.

Fügen Sie dazu einen Dummy-KNX IP-Tunnel mit der physikalischen Adresse des EibPC ein. Verwenden Sie hier die in den Projekteinstellungen eingegebene Adresse "Tunnel EibPC". Diese kann, muss aber nicht innerhalb Ihrer TP-Linie(n) liegen. Als Gerät können Sie beispielsweise die Applikation des „Enertex KNX IP Secure Interface“ verwenden (Abbildung 10). Überspringen Sie beim Hinzufügen die Eingabe des Gerätezertifikats, dieses ist nicht nötig. Ziehen Sie nun alle Gruppenadressen, die der EibPC senden oder auf die er hören soll auf den Tunnel. Die Gruppenadressen erscheinen nun in der Liste „Assoziationen“, wenn Sie den Tunnel in der Geräteliste öffnen.

Tipp: Sie können auch alle Hauptgruppen markieren und diese auf den Tunnel ziehen, um alle Gruppenadressen auf einmal für den EibPC freizugeben. Denken Sie auch daran, beim Hinzufügen von neuen Gruppenadressen diese wieder dem Tunnel zuzuordnen.

Anschließend müssen Sie die Applikation für alle beteiligten Geräte, abgesehen vom EibPC, neu übertragen. Exportieren Sie das Projekt als .knxproj-Datei und importieren Sie es im EibStudio. Nach dem Projektimport sehen Sie unter Topologie die Adressliste (Abbildung 10). Außerdem wertet das EibStudio aus, welche Geräte mit diesen Gruppenadressen verknüpft sind und damit Telegramme senden dürfen. Dies sehen Sie in der Liste der Gruppenadressen unter **Objekte**.

Sequenzzähler-Fehler

Der EibPC speichert die gesehenen und gültigen Sequenzzähler anderer Geräte automatisch im Flash-Speicher. Beim Programmstart lädt der EibPC alle gespeicherten Sequenzzähler aus dem Flash und übernimmt jeweils den höchsten Wert, entweder aus dem Programm oder aus dem Speicher. Damit ist gewährleistet, dass auch nach einem unerwarteten Neustart keine oder nur wenige Telegramme als gültig akzeptiert werden, die von einem Angreifer erneut auf den Bus gesendet wurden. Den aktuellen Stand des eigenen KNX Secure Sequenzzählers, sowie den aller Geräte können Sie in den Projekteinstellungen abfragen und falls nötig ändern.

Der EibPC generiert seinen Sequenzzähler beim Programmstart jeweils neu aus der aktuellen Uhrzeit. Falls die Synchronisierung mittels NTP aktiviert ist, kann mit dem Programmstart gewartet werden, bis der EibPC eine gültige Uhrzeit besitzt. Dies ist bei Verwendung von KNX Data Secure unbedingt empfohlen. Mit jedem gesendeten, verschlüsselten Telegramm wird der Sequenzzähler erhöht.

Achtung: Falls Sie den Sequenzzähler des EibPCs manuell ändern, oder falls der EibPC mit einer Uhrzeit in der Zukunft gestartet wurde, ist nicht mehr gewährleistet, dass andere Geräte die Telegramme des EibPCs akzeptieren, auch wenn die Uhrzeit wieder korrekt ist, da sie einen zu hohen Sequenzzähler akzeptiert haben und nun erwarten, dass alle künftigen Telegramme einen höheren Zählerstand mitsenden. Die Telegramme des EibPCs werden dann still ignoriert. In diesem Fall müssen Sie die Applikation aller betroffenen Geräte über die ETS entladen und neu laden. Ein Löschen des Geräts aus dem Projekt oder ein Zurücksetzten auf Werkseinstellungen ist i.d.R. nicht notwendig.

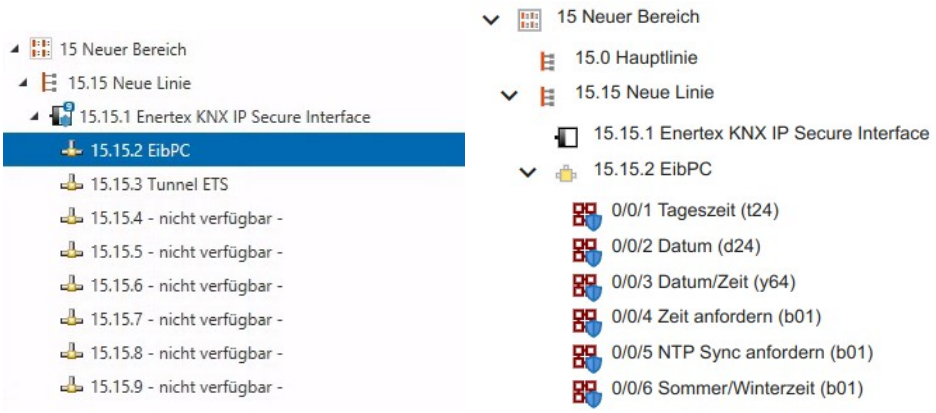


Abbildung 10: KNX Secure Tunnel für EibPC in ETS (links) und EibStudio (rechts)

Validierungsschema

Dieses Kapitel beschreibt das Validierungsschema. Bei der Kompilierung des Projekts wird ein Programm generiert, das von der EibPC-Firmware verarbeitet wird.

Objektbaum

Dies ist jedoch kein sequentielles Programm, wie man es für Mikroprozessoren kennt, sondern eine Datenabhängigkeitsbeschreibung, ein sogenannter Abhängigkeitsbaum. Die Knoten in diesem Graph werden als Programmobjekte bezeichnet. Diese sind nicht zu verwechseln mit den Objekten. Die Menge der Programmobjekte umfasst sowohl alle verwendeten Objekte, es werden aber auch alle Experte (Seite 76) auf Programmobjekte abgebildet.

Invalidierung

Statt einen Befehl nach dem anderen auszuführen wird die Ausführung in Zeitschritte (Zyklen) aufgeteilt. Innerhalb jedes Zyklus erfolgt die Abarbeitung quasi-parallel, .d.h. jede Änderung ist gleichberechtigt. Um unnötige Arbeit zu vermeiden, werden stets nur diejenigen Programmobjekte betrachtet, die sich geändert haben, deren Zustand in dem Zyklus invalidiert wurde.

Jedes Programmobjekt weiß für sich genommen,

- ob es sich geändert hat,
- ob es konstant geblieben ist,
- ob ein Ereignis aufgetreten ist,
- ob von ihm abhängige Ausdrücke sich verändern.

Wenn eine Veränderung in diesem Sinne aufgetreten ist, so wird das Programmobjekt auf den Status „ungültig“ gesetzt (d.h. es wird invalidiert) und die Auswertung des Objekts und seiner Abhängigkeiten neu initiiert. Wenn das Objekt dann verarbeitet ist, so wird es gültig (=valide). Eine „Auswertung“ einer Funktion „write“ beispielsweise ist gleichbedeutend mit dem Schreiben auf dem Bus.

Die Verarbeitung erfolgt in folgenden Schritten in jedem Zyklus, solange bis kein Objekt mehr invalid ist.

Invalidierung

Wenn ein Verarbeitungsobjekt invalid ist, heißt das zunächst nur, dass es neu berechnet werden muss. Bei Initialisierung (erster Verarbeitungszyklus) ist jedes Programmobjekt invalid. Danach muss ein Ereignis eingetreten sein. Trifft z.B. ein Telegramm vom Bus ein, wir das entsprechende Programmobjekt invalidiert. Invalidiert werden können nur Programmobjekte, die von einer Gruppenadresse, dem Systemtimer, TCP/UDP oder einer if-Anweisung abhängig sind.

Berechnung

Das Verarbeitungsobjekt wird mit den neuen Eingabewerten neu berechnet. Hat sich der Zustand geändert, wird der nächste Schritt ausgeführt.

Bedingte Invalidierung

Invalidiere alle Verarbeitungsobjekte in der Abhängigkeitsliste.

Systemstart

Die Standard-Vorbelegung von Variablen, Gruppenadressen usw. ist null (AUS, 0, 0.0 ...). Alle Programmobjekte haben die Eigenschaft „gültig“ (valid).

Dabei ist das genaue Verhalten abhängig vom Typ des Verarbeitungsobjekts.

Die folgenden Programmbeispiele können so direkt als Expertenprogramm angelegt werden.

Es wird das folgende Programm betrachtet:

```
x=2
y="SaunaDimmer-1/0/1"+3%+x
```

Durch die Kompilierung wird der in Abbildung 11 dargestellte Abhängigkeitsbaum erstellt.

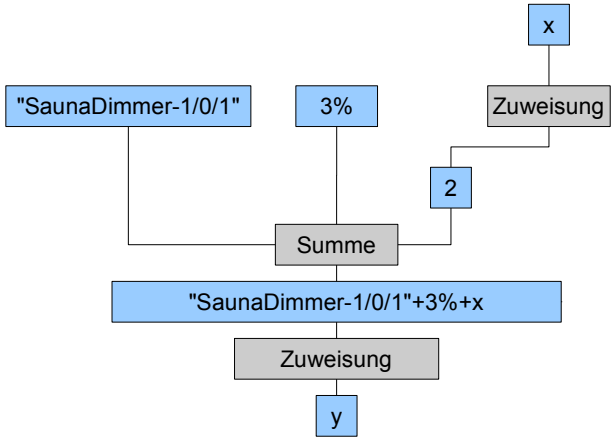


Abbildung 11: Baumstruktur für y="SaunaDimmer-1/0/1"+3%+x und x=2

Zuweisungen

Mit dem Gleichheitszeichen wird der Variablen auf der linken Seite der Wert der Konstanten, Variablen oder Funktion auf der rechten Seite zugewiesen. Nach der Zuweisung ist die Bedingung auf Gleichheit erfüllt (siehe S. 20). Eine Zuweisung ist nur dann möglich, wenn die Datentypen links und rechts identisch sind. Anderenfalls muss die Funktion convert (S. 146) auf der rechten Seite verwendet werden.

Bei Zeichenketten wird der gesamte Speicherinhalt kopiert, also nicht nur bis zum ersten 0-Byte. Dadurch kann die Zuweisung zusammen mit der Funktion stringset (S. 158) verwendet werden. Ist dabei die Zeichenkette auf der linken Seite größer als die auf der rechten Seite, wird der verbleibende Speicherinhalt auf der linken Seite mit Nullen überschrieben. Beachten Sie den Unterschied zwischen c14 und allen anderen Zeichenketten-Typen.

Variablen

Nach der Initialisierung hat x den Wert 2 und y den Wert der Gruppenadresse plus 3% plus x. Beim nächsten und allen folgenden Zyklen ändert sich in diesem Programm für x nichts mehr, da 2 konstant ist. Anders für y: Wenn vom Bus die Gruppenadresse eintrifft und sich deren Wert verändert hat, so wird y neu berechnet. Y ist also von einem Ausdruck abhängig, der sich geändert hat und wird invalidiert. Dabei wird y im nur dann ungültig, wenn sich der Wert der Gruppenadresse tatsächlich geändert hat. Gleichermaßen würde sich y neu berechnen, wenn sich x ändern würde.

Die Invalidierung vererbt sich in der Auswertereihenfolge eines Ausdrucks („von unten nach oben“ in der Abhängigkeit) bis zu der Ebene, bei der die veränderten Abhängigkeiten keine Veränderung mehr zur Folge haben.

Die Variable z ist indirekt von KNX-Telegrammen abhängig: Wenn zunächst 1/2/3 auf EIN geht, ist der ODER Ausdruck auf EIN und dieser teilt das z mit, wenn er zuvor auf AUS stand. Wenn nun 1/2/4 auf EIN geht, wird wieder ODER invalidiert und berechnet sich neu. Da aber ODER bereits auf EIN steht, wird z nicht invalidiert und muss daher nicht neu berechnet werden.

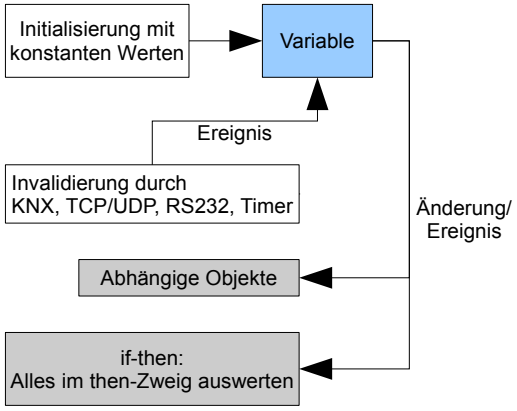


Abbildung 12: Verarbeitung von Variablen

Funktionen

Funktionen werden durch ihre Argumente invalidiert. Ändert sich also ein Argument, wird der Wert des Funktionsobjektes neu berechnet. Ändert sich dadurch das Ergebis, werden alle abhängigen Programmobjekte invalidiert.

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

Ausgabefunktionen

Eine Besonderheit stellen Ausgabefunktionen dar. Diese haben bei ihrer Ausführung Seiteneffekte, neben der Änderung ihres Zustandes. Um zu gewährleisten, dass diese nicht ungewollt angestoßen werden, werden Ausgabefunktionen nie durch ihre Argumente invalidiert, sondern nur, wenn sie innerhalb von if-then-Anweisungen vorkommen und somit durch die Bedingung invalidiert werden.

```
write("Temperatur-1/2/1",22.3)
write("Schalter-1/2/10",!"Schalter-1/2/10")
read("Temperatur-1/2/1")
```

Das Programm führt nie etwas aus. Die zweite Anweisung würde sonst im Zyklustakt des EibPC den Bus beschreiben.

Timerfunktionen

Ähnliches gilt für Timerfunktionen. Diese werden nicht von ihren Argumenten invalidiert, sondern durch die interne Uhr des EibPCs.

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

O wird nur minütig bei 19 Sekunden nach der letzten vollen Minute für einen Verarbeitungszyklus auf EIN gehen.

if-Anweisungen

Die If-Anweisung verhält sich wie eine Funktion, deren Argument die Abfragebedingung darstellt. Wenn die Abfragebedingung ungültig wird, so wird if neu ausgewertet. Diese Auswertung ist unabhängig davon, ob sich die Abfragebedingung vom Wert her ändert. Da die Abfragebedingung aber nur invalid werden kann, wenn sich deren Wert ändert, ist de facto eine ungültige Abfragebedingung auch eine veränderte Abfragebedingung für solche If-Anweisungen, die nicht in Verschachtelungen stehen.

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

Nachdem ein Gruppentelegramm für die Adresse '1/2/3' mit dem Wert 1b01 empfangen wurde, bleibt a auf 3, da sich 1 (von der Zuweisung a=1) nie ändert und dadurch a nicht mehr invalidiert wird.

Verschachteltung von if-Anweisungen

Innere if-Anweisungen werden nicht durch ihre Bedingung sondern durch die Bedingung der äußeren if-Anweisung invalidiert. So wird gewährleistet, dass die äußere Bedingung beachtet wird. Dadurch ist die Auswertung des inneren then-Zweigs allerdings nicht mehr vom Ändern der inneren Bedingung abhängig.

```
a=1
b='1/2/4'b01
z=0
if  '1/2/3'b01 then {
     if b==EIN then  a=3 endif;
     z=cos(1);
     write('1/3/4'b01,AUS)
} endif
```

Nich ausprobieren!

Das Beispiel verdeutlicht die abweichende Invalidierung bei geschachtelten if-Anweisungen:

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

Wäre die innere Anweisung nicht im then-Zweig einer anderen, so würde nie auf den Bus geschrieben werden, da die Bedingung sich nicht ändert. Durch die Verschachtelung wird die innere if-Anweisung mit jeder Änderung der Gruppenadresse invalidiert. Da die innere Bedingung erfüllt ist, wird ein Schreibtelegramm gesendet, das den Wert der Gruppenadresse invertiert. Dadurch wird die Bedingung der äußeren if-Anweisung neu ausgewertet, die wiederrum die Auswertung der inneren bedingt. Es kommt also zu einer endlosen Telegrammauslösung.

Zeitsteuerung im then-Zweig

Auch Zeitfunktionen werden in einer verschachtelten If-Anweisung eines then-Zweigs nur dann ausgewertet, wenn die if-Bedingung ein „invalid“ an ihren then-Zweig sendet.

```
Taste='1/2/3'b01
a=AUS
if Taste then {
   if htime(12,00,00) then a=EIN endif
} endif
```

a steht hier nur auf EIN, wenn Taste exakt um 12:00:00 auf EIN geht (htime generiert nur zu diesem Zeitpunkt einen EIN-Impuls). Besser wäre in diesem Fall daher die Verwendung von chtime. In diesem Fall wird a auf EIN gesetzt, wenn Taste nach 12:00 und vor Mitternacht auf EIN geht.

else-Zweig

Der else-Zweig einer if-Funktion ist wie eine zweite eigenständige if-Funktion mit negierter Abfragebedingung zu verstehen.

```
Taste='1/2/3'b01
if Taste then write('4/5/6'b01, AUS) else write('4/5/6'b01, EIN) endif
```

Dies ist gleichwertig zu

```
Taste='1/2/3'b01
if Taste then write('4/5/6'b01, AUS) endif
if !Taste then write('4/5/6'b01, EIN) endif
```

Schreiben auf Warteschlangen

Wenn die Verarbeitung eines „Programmdurchlaufs“ abgeschlossen sind, werden die internen Warteschlangen für die Ausgabe beschrieben. Dabei werden die zu diesem Zeitpunkt aktuellen Zustände der Argumente ausgewertet, d.h. ein Objekt kann bereits durch eine nachfolgende Operation verändert worden sein. Dies betrifft die folgenden Funktionen:

- sendudp
- sendudparray
- resolve
- sendmail
- sendhtmlmail
- sendcp
- sendtcparray
- connecttcp
- closetcp
- startvpn
- stopvpn
- openvpnuser
- closevpnuser
- ping

Beispiele:

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

Die Variable uIP wird mit der Adresse 192.168.1.1 initialisiert. Eine Sekunde nach dem Start wird die If-Bedingung invalidiert, und damit die Anweisungen im then-Zweig. Die ping-Anweisung wird in die Warteschlange eingereiht, die Zuweisung uIp=192.168.1.100 hingegen während des Zyklus ausgeführt. Am Ende des Zyklus wird nun die ping-Anweisung ausgeführt, jedoch mit der bereits aktualisierten Adresse.

```
b=1
s=$Hallo$
if systemstart() then {
   if b==1 then {
            sendudp(4809u16,192.168.22.1,s);
            s=$Welt$;
            b=2
   }  else {
            sendudp(4809u16,192.168.22.1,s)
    } endif
} endif
```

*Es wird (vgl. Beispiel oben) zweimal ein string **Welt** geschickt. Auch hier wird erst auf die IP-Warteschlange geschrieben, nachdem alle Objekte gültig sind.*

Asynchroner Rückgabewert

Bei einigen Funktionsaufrufen (wie etwa das Anlegen einer TCP-Verbindung) kann nicht gewährleistet werden, dass diese Ihren Rückgabewert innerhalb einer Verarbeitungsschleife des EibPC aktualisieren. Diese Funktionen aktualisieren ihren Rückgabewert entsprechend ihrer eigenen Verarbeitung „asynchron“ zur Hauptverarbeitungsschleife.

Dazu folgendes Beispiel:

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

2 Sekunden nach Systemstart wird die Connectcp-Funktion aufgerufen. Da die Verarbeitung nun den entsprechenden Thread anstößt, ist der Rückgabewert zunächst 0 (Verbindung wird aufgebaut) und die if-Anweisung verlassen. Wenn nach einiger Zeit die Verbindung besteht, aktualisiert nun connecttcp selbstständig die Variable TCP - unabhängig von der if-Abfrage - auf den Wert 1 (Verbindung besteht). Alle vom Rückgabewert abhängigen Programmobjekte werden daraufhin im nächsten Zyklus neu ausgewertet.

Makros

Makros sind wie eine Art Textersetzung zu verstehen.

Man betrachte das folgende Makro:

Dieses Makro macht nie etwas

```
: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 AUS
:end
```

Der EibParser verarbeitet diese Makros so, dass nur diejenigen Teile nach der :return Anweisung eine Verknüpfung mit dem Validierungskonzept aufweisen. Daher wird folgender Code

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

bei Sonnenaufgang nichts ausgeben. Er wird nämlich von Eibparser „expandiert“ in die Form:

```
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 AUS endif
```

Alle write's sind global!

Die write-Funktionen sind also nicht verknüpft mit der Funktion sun(). Verändern wir das Makro, indem wir nun die drei Ausaben in die :return Anweisung schreiben, so wird die Valdierung auf diese Objekte „weitergeleitet“.

```
: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
```

Somit wird nun

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

„Umleiten der Valdierung“

auf die GA die drei Nachrichten „Display “, „Message:“ und „Licht“ ausgegeben.

Mit der :return Anweisung werden die Abhängigkeiten von einer if-Abfrage „weitergeleitet“. Mit Hilfe dieser Anweisung lässt sich nun das Validierungskonzept komplett auch mit eigenen Funktionen vererben. Mit Hilfe der :return Anweisung kann aber auch ein ganzer Codeblock oder nur gezielte Teile des Makros vom aufrufenden Code abhängig gemacht werden.

Weiteres Beispiel:

```
:begin Akt_3(Aktor,Now)
   Variable=3
    if Now then write(Aktor,Variable) endif
:return AUS
:endif
```

Wenn das Makro wie folgt genutzt wird

```
if sun() then Akt_3('1/2/3'u08,chtime(5,00,00)) endif
```

gilt zu beachten, dass der Eibparser nur die Objekte in die Abhängigkeit der aufrufenden if-Funktion bringt, die bei der :return Anweisung steht. In diesem Fall wird daher nur das Objekt (Konstante) AUS in die Abhängigkeit der if sun() ... Abfrage gebracht.

:return liegt also nicht nur den Rückgabewert fest, sondern auch welche Teile des Makros mit der Validierung des aufrufenden Kontext verknüpft werden.

Nur globale Definitionen

In expandierter Form ist der Code demnach:

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

Mit der Definition

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

wird nun

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

wenn bei Sonnenaufgang bereits 5:00 Uhr verstrichen ist, *Variable* auf 3 gesetzt und dies auf den Aktor geschrieben. Expandiert wird dies zu:

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

Wichtig ist hier, dass die *Variable* durch das „Einbinden“ in den then-Zweig gar nicht mehr im globalen Kontext definiert wird, und daher die Definition *Variable*=0 in jedem Fall notwendig wird.

Rekursion

Das Programm

```
a=AUS
if a==EIN then a=!a else a=!a endif
```

verursacht eine Rekursion, wie man anhand Abbildung 13 sieht:

Beim Initialisieren ist der else-Zweig aktiv und die Objekte in dessen Abhängigkeitsliste werden verarbeitet. Diese invertieren a und weisen a neu zu. Dabei wird a invalid und nach Neuberechnung ist es verändert. Dies bedeutet, dass a nun auf EIN steht und seinerseits seine Abhängigkeiten invalidiert. Daher wird nun die if-Abfrage neu ausgewertet und der then Zweig aktiv. Die Objekte in dessen Abhängigkeitsliste werden verarbeitet. Dabei wird a invertiert. Nun wird a invalid und nach Neuberechnung ist es verändert usw. Es bildet sich also innerhalb einer Verarbeitung eine Endlosschleife aus.

Diese wird von der Firmware mit einem Event (PROC_REPITIONS) abgefangen. Wenn dieses Ereignis auftritt, wird die aktuelle Verarbeitung des Objekts und damit die Endlosschleife unterbrochen.

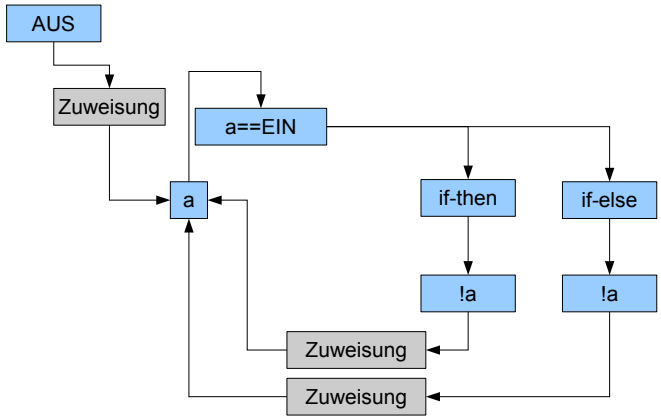


Abbildung 13: Baumstruktur für a=AUS; if a==EIN then a=!a else a=!a endif

Insgesamt gewährleistet das Validierungskonzept, dass

- der EibPC auch bei umfangreichen Programmen extrem effizient ist
- Standardaufgaben (Wenn Taste, dann Licht) einfach programmierbar sind
- alle Anweisungen parallel in einem Zyklus abgearbeitet werden.

Visualisierung

Der EibPC² bietet eine webseitenbasierte Visualisierung, die betriebssystemunabhängig auf allen modernen Browsern angezeigt werden kann. Zustandsänderungen werden dabei sofort auf der Webseite angezeigt. Die Visualisierung kann in der **Visu** im EibStudio und/oder im **Experten** erstellt werden.

Die Visualisierung gliedert sich in Gruppen von Seiten, auf denen verschiedene Elemente platziert werden. Gruppen dienen nur der Übersichtlichkeit, besitzen aber keine weiteren Eigenschaften.

Bei Elementen wird zwischen globalen und seitenbezogenen Elementen unterschieden. Globale Elemente können öfter verwendet werden, d.h. sie können auf einer oder verschiedenen Seiten mehrfach eingefügt werden. Über das Anwenderprogramm werden alle diese Elemente mit einer Funktion adressiert. Daneben gibt es seitenbezogene Elemente, die nur auf einer Seite verwendet werden können. Zur Adressierung über das Anwenderprogramm muss jeweils auch die Seite mit angegeben werden. Diese Adressierung erfolgt in Form von eindeutigen Nummern, die IDs. Diese werden beim Anlegen der Elemente vergeben und zum Zugriff durch das Anwenderprogramm verwendet.

Beim Erstellen von eigenen Visualisierungsseiten muss darauf geachtet werden, dass sich die IDs zwischen **Visu** und **Experte** nicht überschneiden (siehe IDs, S.24). Alle Seiten und Elemente müssen eindeutige IDs besitzen. Seiten und globale Elemente haben jeweils einen eigenen Nummernbereich, bei seitenbezogenen Elementen teilen sich alle Elemente auf einer Seite den gleichen Nummernbereich. Globale Elemente besitzen pro Elementtyp jeweils einen eigenen Bereich.

Elemente einer Seite werden in einem rechteckigen Gitter angeordnet (vgl. Abbildung 14). Für jede Seite kann die Anzahl der Zeilen und Spalten dieses Gitters festgelegt werden. In einer Zelle dieses Gitters kann sich nur ein Element befinden. Die meisten Elemente besitzen eine feste Größe, also eine feste Anzahl an Zeilen und Spalten, die sie zur Anzeige benötigen. Ein Überlappen von Elementen ist nicht möglich.

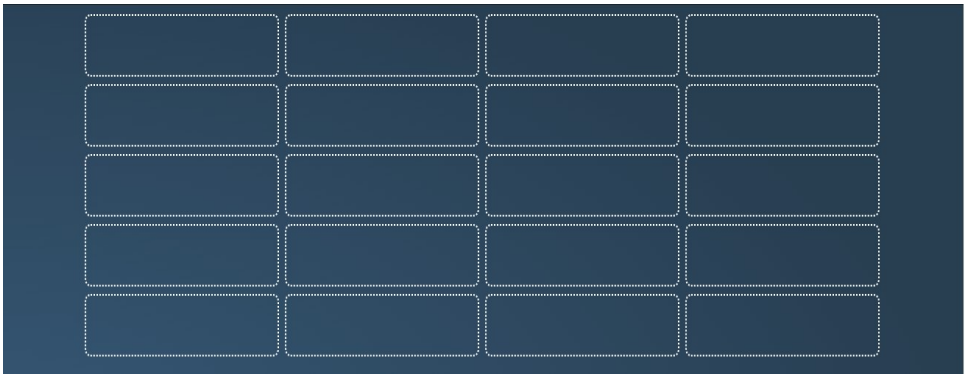


Abbildung 14: Gitter einer Seite

Zur besseren Lesbarkeit auf kleineren Anzeigen wird die Anzahl der Spalten automatisch angepasst (Responsive Design). Beispielsweise auf Smartphones wird die Visualisierung einspaltig angezeigt, unabhängig davon, wie viele Spalten für die Seite konfiguriert wurden (vgl. Abbildung 15, Abbildung 16). Die Anordnung erfolgt dabei zeilenbasiert, bezogen auf die linke obere Ecke eines Elements.

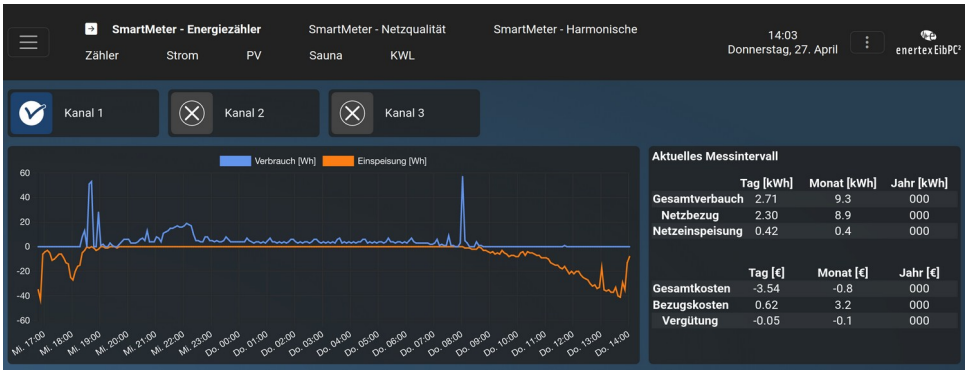


Abbildung 15: Darstellung auf dem Desktop-PC



Abbildung 16: Darstellung auf dem Smartphone

Die Seitennavigation wird automatisch generiert (vgl. Abbildung 17)



Abbildung 17: Seitennavigation

Für Seiten gibt es neben der dunklen Darstellungsvariante eine blaue (vgl. Abbildung 18). Die Auswahl erfolgt in den Seiteneigenschaften im EibStudio, bzw. mit dem entsprechenden Befehl im Expertenprogramm (S. 81).



Abbildung 18: Blaues Design

Passwortschutz

Seiten können in deren Eigenschaftsdialog individuell mit Benutzernamen/Passwort versehen werden. Dabei muss die Kombination aus Benutzername und Passwort über alle Seiten hinweg identisch sein. Diese Seiten werden in der Navigation ausgeblendet, bis sich der Benutzer im Browser bei der Seite anmeldet. Danach kann die Seite normal aufgerufen werden. Die Zugangsdaten können im Browser gespeichert werden, so dass kein neuer Login bei einem erneuten Aufruf notwendig ist.

Elemente

In Abbildung 19 sehen Sie eine Übersicht über die vorhandenen Elemente.

Buttons unterschiedlicher Breite und Icon-Anzahl sowie Mehrfachauswahl werden verwendet, um z.B. Leuchten oder Jalousien zu schalten.

Zum Dimmen können Slider und die Farbeingabe verwendet werden. Für Schaltuhren gibt es Datums- und Uhrzeitauswahl. Allgemeine Grafiken (beliebiger Webadressen) werden mit dem Picture-Element angezeigt. Mittels Plink lässt sich, zusätzlich zur Hauptnavigation, auf Visualisierungsseiten springen.

Messwerte werden entweder im Chart, ohne weitere Speicherung und mit beliebigem x- und y-Wert angezeigt, oder als Zeitreihe in einem TimeBuffer gespeichert und vom TimeChart angezeigt.

Externe Webseiten, z.B. Kamerabilder, können durch das Frame-Element direkt in der Visualisierung eingeblendet werden.

Durch Trennlinien kann eine Seite in Abschnitte unterteilt werden.

Im Visualisierungseditor werden, falls vorhanden, die seitenbezogenen Varianten der Elemente verwendet. Für den Zugriff im Expertenprogramm müssen deshalb auch die seitenbezogenen Funktionen (z.B. pdisplay) verwendet werden (vgl. S. 19).



Abbildung 19: Elemente

Funktionen

Neben den selbst konfigurierbaren Grund-Elementen stehen Elemente mit bereits hinterlegter Funktion bereit, die meist mehrere Elemente umfassen.

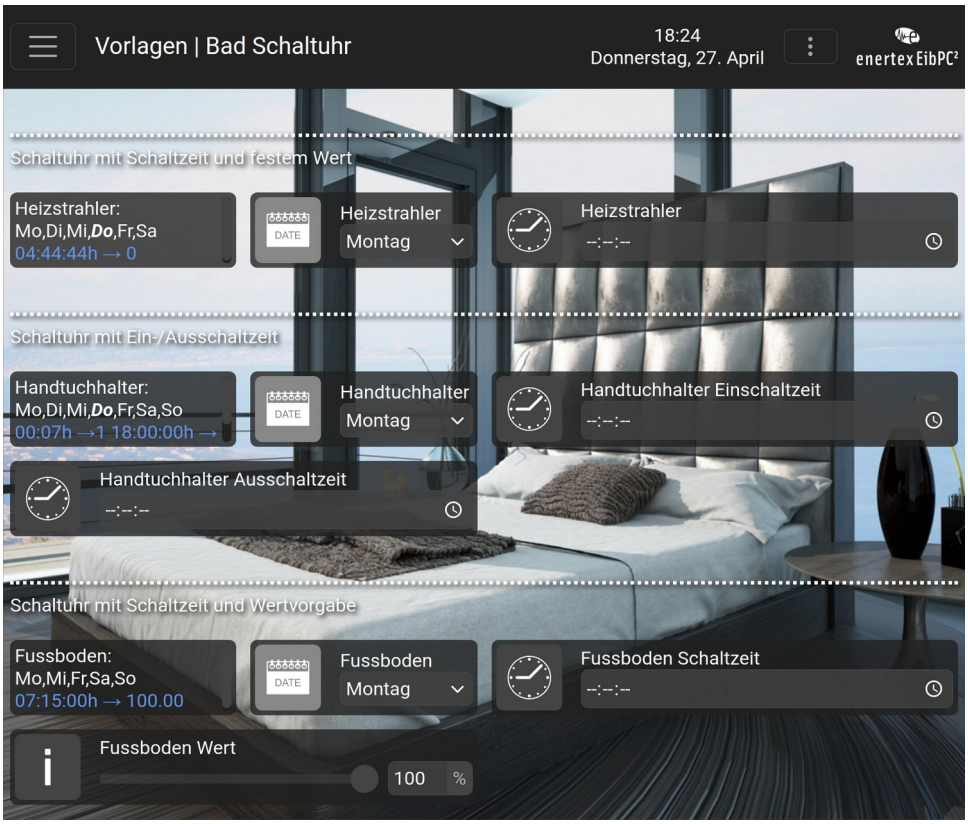


Abbildung 20: Vorgefertigte Funktionen in der Visu

Vorlagen

Unter Vorlagen finden Sie komplette Seiten, die Elemente und Funktionen bereits fertig arrangiert beinhalten. Sie können auch eigene Seitenvorlagen anlegen, um beispielsweise in verschiedenen Projekten schnell ähnliche Visualisierungsseiten zu erzeugen.

Icons

Zustände

Der EibPC verfügt über einen eingebauten Satz von Grafiken.

Es existieren die in Tabelle 3 gelisteten Symbole. Im Visualisierungseditor werden sie direkt bei der Konfiguration des Elements ausgewählt. In der Sektion [WebServer] als auch im Anwenderprogramm werden sie über den Namen oder den numerischen Index ausgewählt. Jedes Symbol kann in unterschiedlichen Ausprägungen angezeigt werden. Hierzu existieren die in Tabelle 2 gelisteten Zustände.

Hinweis: Nicht jede Symbolgruppe implementiert alle möglichen Zustände.

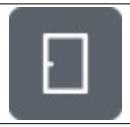
Zustand	Index
DARKRED	0u08
INACTIVE	1u08
ACTIVE	2u08
DISPLAY	3u08
STATE4	4u08
STATE5	5u08
STATE6	6u08
STATE7	7u08
STATE7	8u08
BRIGHTRED	9u08

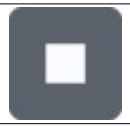
Tabelle 2: Übersicht über Zustände.

Symbole

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

Symbol	Index	DARKRED 0u08	INACTIVE 1u08	ACTIVE 2u08	DISPLAY 3u08	STATE4 4u08	STATE5 5u08	STATE6 6u08	STATE7 7u08	STATE8 8u08	BRIGHTRED 9u08
STOP	9u08										
MAIL	10u08										
SCENES	11u08										
MONITOR	12u08										
WEATHER	13u08										
ICE	14u08										
NIGHT	15u08										
CLOCK	16u08										
WIND	17u08										

Symbol	Index	DARKRED 0u08	INACTIVE 1u08	ACTIVE 2u08	DISPLAY 3u08	STATE4 4u08	STATE5 5u08	STATE6 6u08	STATE7 7u08	STATE8 8u08	BRIGHTRED 9u08
WINDOW	18u08										
DATE	19u08										
PRESENT	20u08										
ABSENT	21u08										
REWIND	22u08										
PLAY	23u08										
PAUSE	24u08										
FORWARD	25u08										
RECORD	26u08										

Symbol	Index	DARKRED 0u08	INACTIVE 1u08	ACTIVE 2u08	DISPLAY 3u08	STATE4 4u08	STATE5 5u08	STATE6 6u08	STATE7 7u08	STATE8 8u08	BRIGHTRED 9u08
HALT	27u08										
EJECT	28u08										
NEXT	29u08										
PREVIOUS	30u08										
LEFT	31u08										
RIGHT	32u08										
CROSSCIRCLE	33u08										
OKCIRCLE	34u08										
STATESWITCH	35u08										

Symbol	Index	DARKRED 0u08	INACTIVE 1u08	ACTIVE 2u08	DISPLAY 3u08	STATE4 4u08	STATE5 5u08	STATE6 6u08	STATE7 7u08	STATE8 8u08	BRIGHTRED 9u08
PLUG	36u08										
METER	37u08										
PVSOLAR	38u08										
THERMSOLAR	39u08										
PUMP	40u08										
HEATINGUNIT	41u08										
HEATPUMP	42u08										
FLOORHEATING	43u08										
WALLHEATING	44u08										

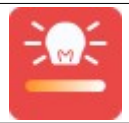
Symbol	Index	DARKRED 0u08	INACTIVE 1u08	ACTIVE 2u08	DISPLAY 3u08	STATE4 4u08	STATE5 5u08	STATE6 6u08	STATE7 7u08	STATE8 8u08	BRIGHTRED 9u08
COOLER	45u08										
MICRO	46u08										
SPEAKER	47u08										
RGB	48u08										
LUX	49u08										
RAIN	50u08										
KEY	51u08										
WASTE	52u08										
ASK	53u08										

Symbol	Index	DARKRED 0u08	INACTIVE 1u08	ACTIVE 2u08	DISPLAY 3u08	STATE4 4u08	STATE5 5u08	STATE6 6u08	STATE7 7u08	STATE8 8u08	BRIGHTRED 9u08
WARN	54u08										
NEAR	55u08										
CAMERA	56u08										
SIGNAL	57u08										
DOOR	58u08										
GARAGE	59u08										
CURTAIN	60u08										
ANGLE	61u08										
ROLLER	62u08										

Symbol	Index	DARKRED 0u08	INACTIVE 1u08	ACTIVE 2u08	DISPLAY 3u08	STATE4 4u08	STATE5 5u08	STATE6 6u08	STATE7 7u08	STATE8 8u08	BRIGHTRED 9u08
EMAIL	63u08										
PETS	64u08										
PHONE	65u08										
PERSON	66u08										
TV	67u08										
BEAMER	68u08										
RADIO	69u08										
RECIEVER	70u08										
MEDIA	71u08										

Symbol	Index	DARKRED 0u08	INACTIVE 1u08	ACTIVE 2u08	DISPLAY 3u08	STATE4 4u08	STATE5 5u08	STATE6 6u08	STATE7 7u08	STATE8 8u08	BRIGHTRED 9u08
STOVE	72u08										
FRIDGE	73u08										
WASHER	74u08										
DISHWASHER	75u08										
HOLIDAY	76u08										
SLEEP	77u08										
REFRESH	78u08										
EV	79u08										
TIMER	80u08										

Symbol	Index	DARKRED 0u08	INACTIVE 1u08	ACTIVE 2u08	DISPLAY 3u08	STATE4 4u08	STATE5 5u08	STATE6 6u08	STATE7 7u08	STATE8 8u08	BRIGHTRED 9u08
DELAY	81u08										
SCHEDULE	82u08										
ALARMCLOCK	83u08										
RESET	84u08										
MAN	85u08										
WOMAN	86u08										
CLEANING	87u08										
BEER	88u08										
BATHING	89u08										

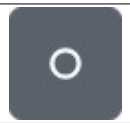
Symbol	Index	DARKRED 0u08	INACTIVE 1u08	ACTIVE 2u08	DISPLAY 3u08	STATE4 4u08	STATE5 5u08	STATE6 6u08	STATE7 7u08	STATE8 8u08	BRIGHTRED 9u08
WATCHINGTV	90u08										
LOCK	91u08										
SETTINGS	92u08										
GEARS	93u08										
COLORTEMPERATURE	94u08										
CHARTS	95u08										
CARBATTERY	96u08										
BATTERYSTORAGE	97u08										
HEATPUMPVENTILATION	98u08										

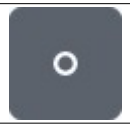
Symbol	Index	DARKRED 0u08	INACTIVE 1u08	ACTIVE 2u08	DISPLAY 3u08	STATE4 4u08	STATE5 5u08	STATE6 6u08	STATE7 7u08	STATE8 8u08	BRIGHTRED 9u08
FLUIDMETER	99u08										
WATERMETER	100u08										
HEATMETER	101u08										
ENERGYMANAGEMENT	102u08										
HEATINGROD	103u08										
HOMEVENTILATION	104u08										
WATERING	105u08										
AIRCONDITION	106u08										
AIRCONDITIONHEATING	107u08										

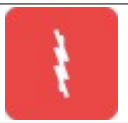
Symbol	Index	DARKRED 0u08	INACTIVE 1u08	ACTIVE 2u08	DISPLAY 3u08	STATE4 4u08	STATE5 5u08	STATE6 6u08	STATE7 7u08	STATE8 8u08	BRIGHTRED 9u08
CHRISTMAS	108u08										
STAIRSLIGHT	109u08										
SPOTLIGHT	110u08										
PENDANTLIGHT	111u08										
EXTERIORLIGHT	112u08										
HALLLIGHT	113u08										
LEDSTRIPESCEILING	114u08										
LEDSTRIPESFLOOR	115u08										
MIRRORLIGHT	116u08										

Symbol	Index	DARKRED 0u08	INACTIVE 1u08	ACTIVE 2u08	DISPLAY 3u08	STATE4 4u08	STATE5 5u08	STATE6 6u08	STATE7 7u08	STATE8 8u08	BRIGHTRED 9u08
FLOORLIGHT	117u08										
DESKLIGHT	118u08										
CEILINGLIGHT	119u08										
BATHROOM	120u08										
TOILET	121u08										
DININGROOM	122u08										
LIVINGROOM	123u08										
DRESSINGROOM	124u08										
KIDSROOM	125u08										

Symbol	Index	DARKRED 0u08	INACTIVE 1u08	ACTIVE 2u08	DISPLAY 3u08	STATE4 4u08	STATE5 5u08	STATE6 6u08	STATE7 7u08	STATE8 8u08	BRIGHTRED 9u08
KITCHEN	126u08										
GARAGEFILLED	127u08										
BASEMENT	128u08										
OFFICE	129u08										
POOL	130u08										
SAUNA	131u08										
MAGNIFIERMINUS	132u08										
MAGNIFIERPLUS	133u08										
SMALLMINUS	134u08										

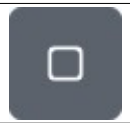
Symbol	Index	DARKRED 0u08	INACTIVE 1u08	ACTIVE 2u08	DISPLAY 3u08	STATE4 4u08	STATE5 5u08	STATE6 6u08	STATE7 7u08	STATE8 8u08	BRIGHTRED 9u08
SMALLPLUS	135u08										
POWERGRID	136u08										
TOGGLE	137u08										
FILLEDDOT	138u08										
VOLTAGE	139u08										
RGBSLIDER	140u08										
WINDSOCK	141u08										
ENERTEX	142u08										
SMALLARROWLEFT	143u08										

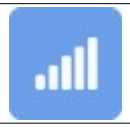
Symbol	Index	DARKRED 0u08	INACTIVE 1u08	ACTIVE 2u08	DISPLAY 3u08	STATE4 4u08	STATE5 5u08	STATE6 6u08	STATE7 7u08	STATE8 8u08	BRIGHTRED 9u08
SMALLARROWRIGHT	144u08										
SMALLFILLEDDOT	145u08										
BUILDINGPROTECTION	146u08										
COOLINGSCALE	147u08										
FLOORHEATINGSCALE	148u08										
WALLHEATINGSCALE	149u08										
COOLERSCALE	150u08										
HEATINGRODSCALE	151u08										
RGBRSCALE	152u08										

Symbol	Index	DARKRED 0u08	INACTIVE 1u08	ACTIVE 2u08	DISPLAY 3u08	STATE4 4u08	STATE5 5u08	STATE6 6u08	STATE7 7u08	STATE8 8u08	BRIGHTRED 9u08
RGBRGSCALE	153u08										
RGBBSCALE	154u08										
SPEAKERSCALE	155u08										
SLATANGLE	156u08										
AUTOMATIC	157u08										
LED	158u08										
LED3D	159u08										
WEATHERCOLORED	160u08										
AWNING	161u08										

Symbol	Index	DARKRED 0u08	INACTIVE 1u08	ACTIVE 2u08	DISPLAY 3u08	STATE4 4u08	STATE5 5u08	STATE6 6u08	STATE7 7u08	STATE8 8u08	BRIGHTRED 9u08
SAUNASTOVE	162u08										
ECO	163u08										
BAROMETER	164u08										
BALCONY	165u08										
TERRACE	166u08										
WINTERGARDEN	167u08										
FIRE	168u08										
EAT	169u08										
GAME	170u08										

Symbol	Index	DARKRED 0u08	INACTIVE 1u08	ACTIVE 2u08	DISPLAY 3u08	STATE4 4u08	STATE5 5u08	STATE6 6u08	STATE7 7u08	STATE8 8u08	BRIGHTRED 9u08
MAINTENANCE	171u08										
MOONPHASE	172u08										
EXCLAMATIONMARK	173u08										
METAPREMIUM	174u08										
METAROCKER	175u08										
DOG	176u08										
DENIED	177u08										
UPLIGHT	178u08										
DOWNLIGHT	179u08										

Symbol	Index	DARKRED 0u08	INACTIVE 1u08	ACTIVE 2u08	DISPLAY 3u08	STATE4 4u08	STATE5 5u08	STATE6 6u08	STATE7 7u08	STATE8 8u08	BRIGHTRED 9u08
WALLLIGHT	180u08										
UPDOWNLIGHT	181u08										
SHOWER	182u08										
COOKING	183u08										
SQUARELIGHT	184u08										
TUMBLEDRYER	185u08										
BARRIER	186u08										
SLIDINGGATE	187u08										
GARDENGATE	188u08										

Symbol	Index	DARKRED 0u08	INACTIVE 1u08	ACTIVE 2u08	DISPLAY 3u08	STATE4 4u08	STATE5 5u08	STATE6 6u08	STATE7 7u08	STATE8 8u08	BRIGHTRED 9u08
ENTRANCEGATE	189u08										
SCALE	190u08										
BELL	191u08										
ZERO	192u08	0	0	0	0						0
ONE	193u08	1	1	1	1						1
TWO	194u08	2	2	2	2						2
THREE	195u08	3	3	3	3						3
FOUR	196u08	4	4	4	4						4
FIVE	197u08	5	5	5	5						5

Symbol	**Index**	**DARKRED** 0u08	**INACTIVE** 1u08	**ACTIVE** 2u08	**DISPLAY** 3u08	**STATE4** 4u08	**STATE5** 5u08	**STATE6** 6u08	**STATE7** 7u08	**STATE8** 8u08	**BRIGHTRED** 9u08
SIX	198u08										
SEVEN	199u08										
EIGHT	200u08										
NINE	201u08										

Tabelle 3: Übersicht Symbole

Logik

Beispiele

Treppenhausautomat

Weitere Beispiele für Logiken finden Sie auch in unseren Lehrvideos.

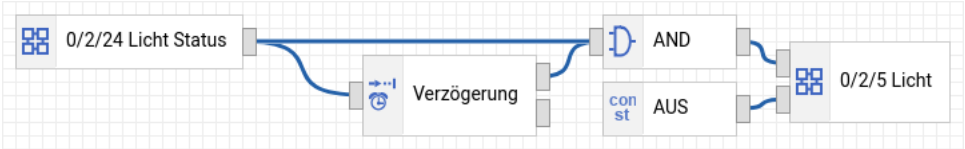


Abbildung 21: Einfache Lichtautomatik

Beispiel 1: Einfacher Treppenhausautomat: 10 Minuten nach dem Einschalten soll das Licht automatisch wieder ausgeschaltet werden.

- Starten Sie mit einem leeren Projekt, importieren Sie Gruppenadressen und kompilieren Sie das Projekt. Dadurch werden die vordefinierten Konstanten angelegt.
- Erstellen Sie eine neue Logik
- Fügen Sie die Knoten hinzu:
  Objekte/Gruppenadresse
  Objekte/Gruppenadresse
  Objekte/Konstante
  Logik/AND
  Zeit/Verzögerung
- Konfigurieren Sie den ersten Gruppenadressknoten so, dass er den aktuellen Wert der Status-Gruppenadresse ausgibt
- Der zweite Gruppenadressknoten soll bei einem Externen Auslöser Schreiben
- Es soll die Konstante AUS, die dem Wert 0b01 entspricht, geschrieben werden. Wählen Sie diesen in Konstante.
- Konfigurieren Sie die Verzögerung wie gewünscht.
- Verknüpfen Sie die Knoten wie in Abbildung 21.
- Kompilieren und starten Sie das Projekt.

Die Logik wird bei jeder Zustandsänderung der Objekte und Knoten (teilweise) neu ausgewertet (Details siehe Validierungsschema). Ändert sich der Status von 0b01 auf 1b01, wird der Verzögerungstimer neu gestartet. Sobald die Verzögerungszeit abgelaufen ist, geht ihr Auslöser-Ausgang von 0b01 auf 1b01. Ist zu diesem Zeitpunkt der Status immernoch 1b01, so wird der Wert AUS (oder 0b01) als Bustelegramm an die Gruppenadresse gesendet.

Die folgenden Sequenzen können als einzelne Programme im **Experte** angelegt werden.

IoT-Geräte via MQTT

Beispiel 2: Einbindung von Shelly Plus Plug S WLAN-Steckdosen

MQTT ist ein leichtgewichtiges Kommunikationsprotokoll, das entwickelt wurde, um Geräte wie Sensoren oder Maschinen effizient über das Internet miteinander zu verbinden und Nachrichten in Echtzeit auszutauschen. Auch für lokale Kommunikation im Netzwerk hat MQTT Vorteile, z.B. gegenüber HTTP-Requests. Verbindungen bleiben bestehen und können so vom Gerät jederzeit genutzt werden, um Ereignisse zu senden, ohne dass das Gerät zyklisch abgefragt werden muss.

Für einige IoT-Geräte gibt es vorgefertigte Funktionen in der **Visu**. Sie können aber auch auf beliebige andere Geräte mittels MQTT zugreifen. Die Einrichtung wird am Beispiel der über WLAN schaltbaren Steckdose Shelly Plus Plug S gezeigt.

Der MQTT Broker übernimmt die Rolle des Servers, der alle Verbindungen zu Clients verwaltet, Nachrichten von ihnen entgegennimmt und diese bei Bedarf unter den anderen Clients verbreitet. Der EibPC kann sowohl Broker als auch Client sein. EibStudio (Version > 5.100) bietet Logikknoten zur Konfiguration des MQTT-Brokers (S. 205) sowie zur Erzeugung von Clients (S. 206).

Broker anlegen

Legen Sie im EibStudio eine neue **Logik** an und fügen Sie den Knoten MQTT-Broker gemäß Abbildung Abbildung 22 ein. Natürlich können Sie wie üblich Starten (und Stoppen) des Brokers auch an andere Bedingungen knüpfen. Legen Sie in den Knoteneigenschaften Benutzername und Passwort fest. Übertragen Sie das Programm auf den EibPC. Sie können nun bereits andere Geräte als Client mit dem EibPC-MQTT-Broker verbinden.

Shelly Plus Plug S

Um von anderen Geräten gesendete Werte im EibPC auswerten zu können, muss noch der EibPC selbst als Client mit dem Broker verbunden werden. Fügen Sie den Knoten MQTT-Client ein, konfigurieren Sie ebenfalls Benutzername und Passwort wie oben und verwenden Sie als Server localhost. Weisen Sie den Ausgang des Clients einer neuen Variable zu (Abbildung 23). Über diese könne Sie nun in der **Visu**, **Logik** und **Experte** auf die MQTT-Verbindung zugreifen.

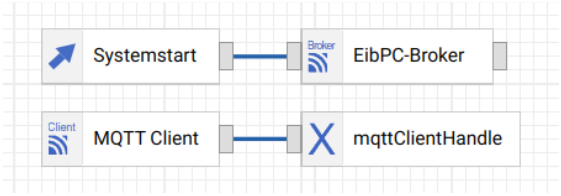


Abbildung 22: MQTT Broker starten

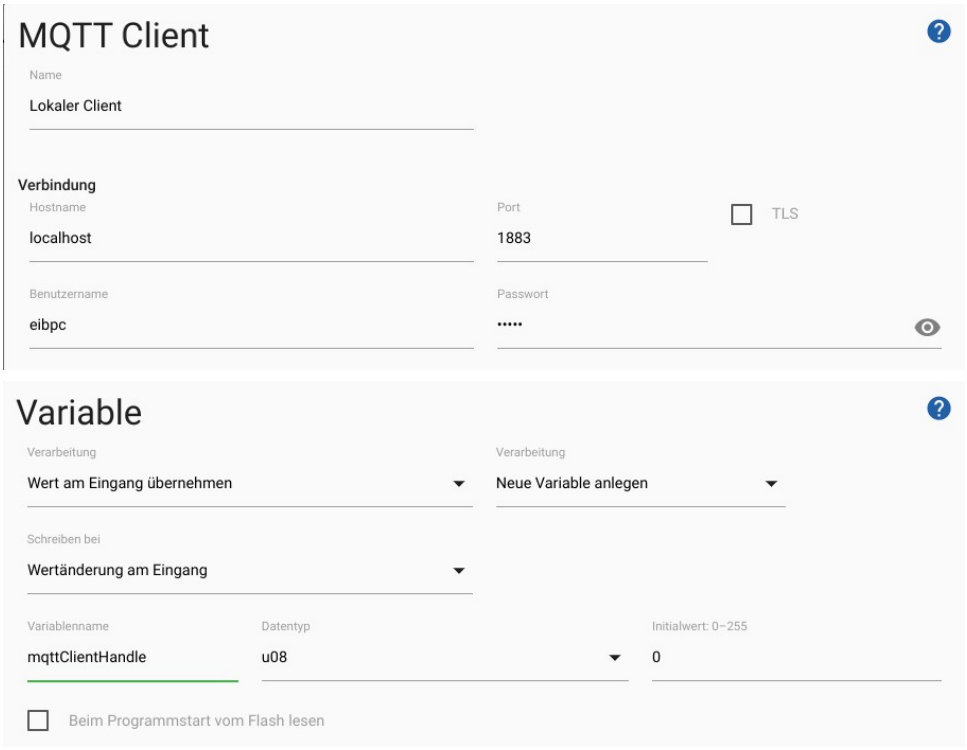


Abbildung 23: Variable für Handle anlegen

Binden Sie das Gerät entsprechend der Anleitung des Herstellers in Ihr Netzwerk ein, so dass es den EibPC erreichen kann. Aktivieren Sie anschließend MQTT wie in Abbildung 24. Setzen Sie die Haken wie gezeigt. Das MQTT-Prefix dient der Identifikation der Nachrichten des entsprechenden Geräts und wird z.B. für die VISU-Funktion benötigt. Achten Sie darauf, dass das Gerät eine Verbindung zum Broker aufbauen konnte (Status oben rechts).

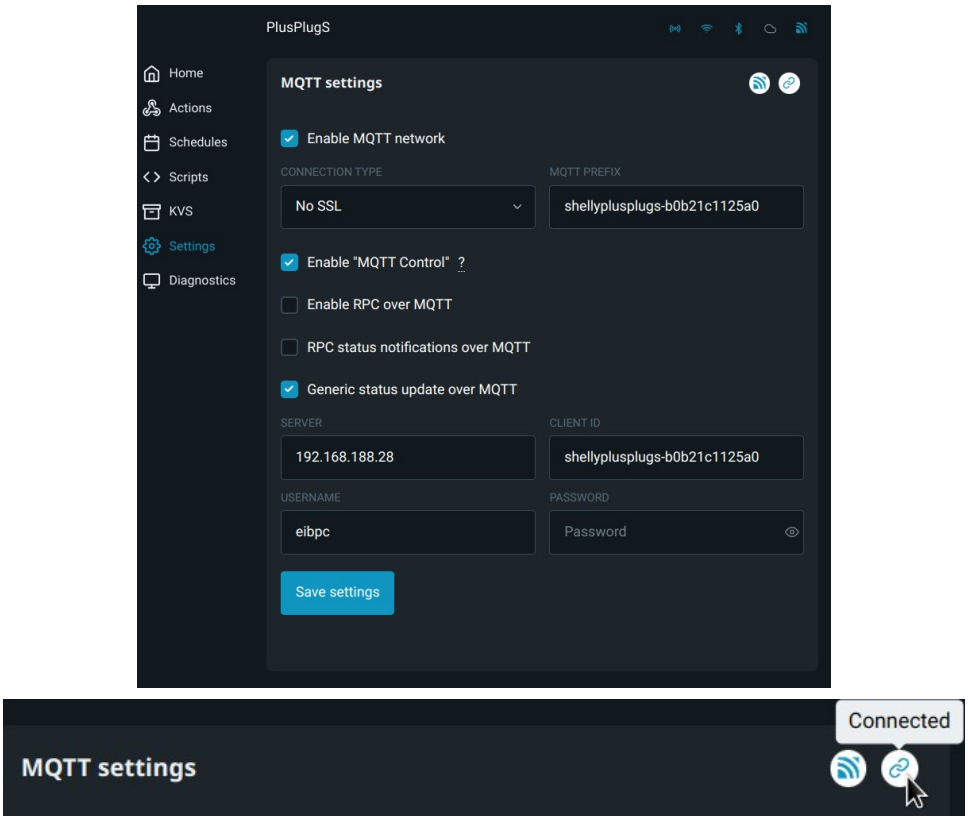


Abbildung 24:Konfiguration Shelly Plus Plug S, Verbindungsstatus oben rechts

Sie können die Kommunikation zwischen MQTT-Gerät und EibPC-Broker mit einem beliebigen MQTT-Client (z.B. MQTT Explorer, MQTTX) prüfen. Hier sehen Sie das Topic, an das die Nachrichten gesendet werden, und ob der Status wie erwartet gesendet wird, sobald es eine Zustandsänderung gibt. Außerdem sehen Sie das Format, in dem die Daten gesendet werden. Dies ist insbesondere dann wichtig, wenn Sie Geräte einbinden möchten, für die keine vorgefertigte Funktion existiert.

Nachdem Sie verifiziert haben, dass sowohl Broker als auch die Client-Verbindung des Geräts zum Broker funktionieren, fügen Sie nun die entsprechende VISU-Funktion *Shelly Plus Plug S* in der VISU ein (Abbildung 26). In den Eigenschaften verknüpfen Sie die zuvor angelegte Handle-Variable, tragen Sie das MQTT-Prefix entsprechend der MQTT-Konfiguration des Gerätes (Abbildung 24) ein und verknüpfen Sie die Gruppenadressen. In der Hilfe der Funktion finden Sie ein Beispielprojekt für die ETS mit allen nötigen Gruppenadressen.

Sie können das mittels MQTT eingebundene Gerät nun wie gewohnt in Ihre Automatisierung einbinden und über die Visualisierung und KNX steuern. Sie erhalten unmittelbar Rückmeldung, unabhängig davon, ob die Änderung vom EibPC oder anderweitig ausgelöst wurde (Abbildung 27).

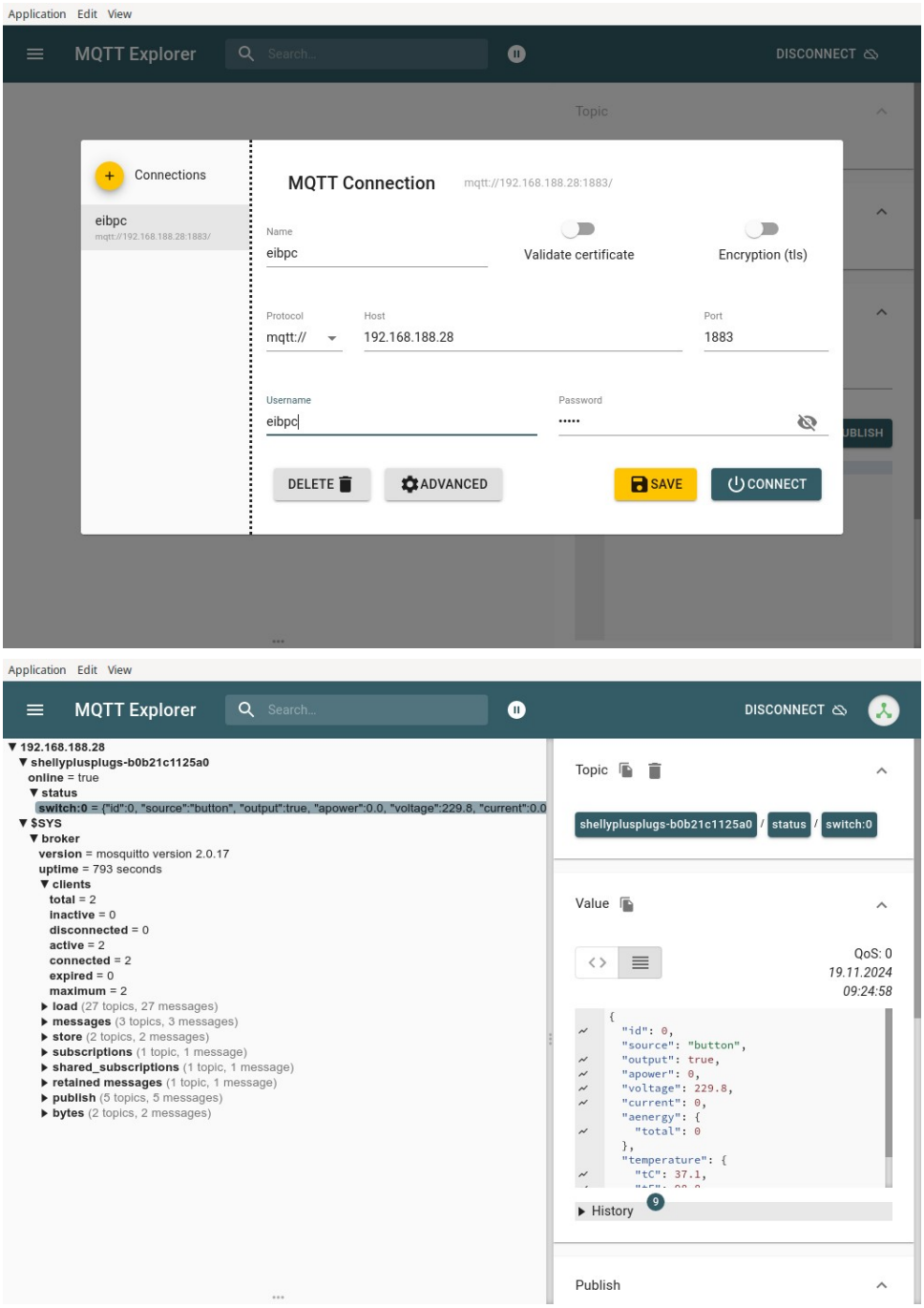


Abbildung 25: Broker prüfen und Daten anzeigen (Beispiel: MQTT-Explorer)

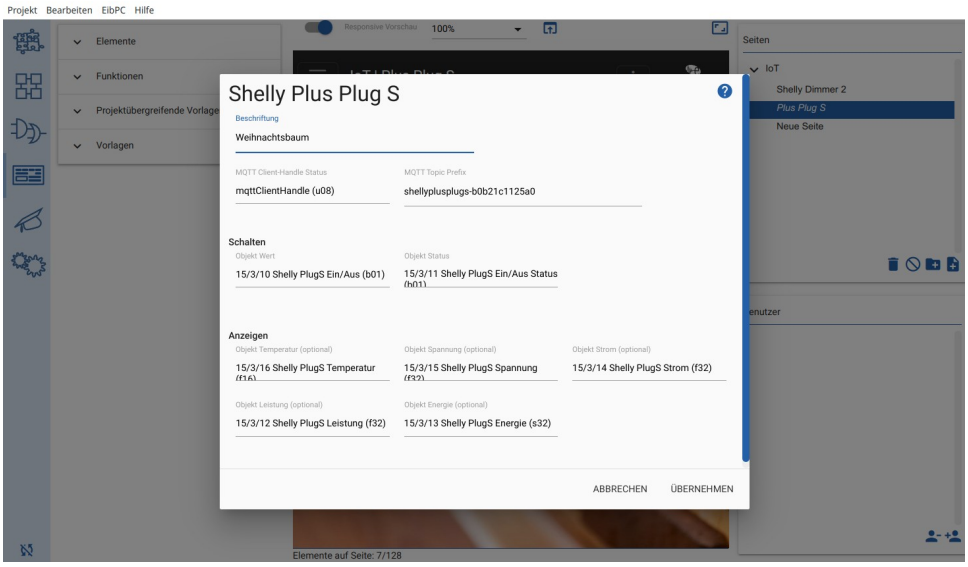


Abbildung 26: Konfiguration der Visu-Funktion Shelly Plug Plug S

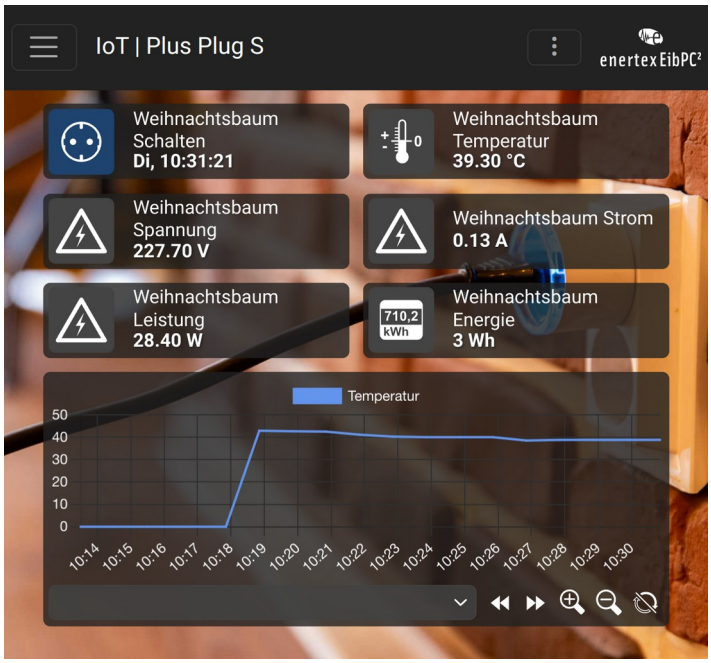


Abbildung 27: Shelly Plus Plug S in der Web-Visualisierung

Manuelle Einbindung von MQTT-Geräten

Sie können auch Geräte einbinden, für die keine fertige Funktion bereitsteht. Gehen Sie wie oben beschrieben vor. Falls noch nicht geschehen, starten Sie den MQTT-Broker auf dem EibPC und verbinden Sie das Gerät mit dem Broker. Verwenden Sie einen MQTT-Client, um das vom Gerät verwendete Prefix und das Datenformat zu sehen.

In Abbildung 28 sehen sie beispielsweise Daten von Tasmota, einer freien Firmware für IoT-Geräte. Unter dem Topic *tele/steckdose-tv/SENSOR* wird ein JSON-Objekt als Zeichenkette übertragen. Dies ist häufig der Fall. Um die einzelnen Werte zu erhalten, muss das JSON-Objekt als Ganzes empfangen und anschließend mit dem JSON-Parser zerlegt werden. Legen Sie eine neue Variable vom Typ c1400 oder c65535 an, je nach erwarteter Länge. Öffnen Sie die Eigenschaften des MQTT-Client-Logikknotens, fügen Sie ein neues Abonnement des o.g. Topics hinzu (subscribe) und wählen Sie die neue Variable als Objekt. Übertragen Sie das Programm und warten Sie, bis das Gerät einen neuen Wert an das Topic gesendet hat. Die Variable hat nun beispielsweise folgenden Inhalt:

```
{
  "Time": "2024-11-19T14:06:56",
  "ENERGY": {
    "TotalStartTime": "2020-08-22T00:25:57",
    "Total": 143.709,
    "Yesterday": 0.15,
    "Today": 0.005,
    "Period": 0,
    "Power": 72,
    "ApparentPower": 81,
    "ReactivePower": 39,
    "Factor": 0.88,
    "Voltage": 237,
    "Current": 0.344
  }
}
```

Um aus dem JSON-Objekt die Spannung zu erhalten, verbinden Sie den Ausgang der Variable mit den Logikknoten JSON. Geben Sie als JSON-Pointer */ENERGY/Voltage* ein (siehe auch parsejson). Die max. Länge der Ausgabe sollte eingeschränkt werden, um Ressourcen zu sparen, hier z.B. auf 3 Stellen. Um die Spannung nun als f32 zu erhalten, fügen Sie den Typkonvertierung-Knoten ein und wählen Sie den Datentyp in dessen Eigenschaften oder verknüpfen Sie direkt ein passendes Objekt wie in Abbildung 29.

Werden statt kompletter JSON-Objekte einzelne Werte mit einem eigenen Topic übertragen, benötigen Sie entsprechend mehrere Variablen, die Sie direkt in der Liste der Abonnierten Topics eintragen.

Beispiel aus Abbildung 28: stat*/steckdose-tv/POWER*

Wichtig ist, dass das Datenformat beachtet wird. Werden die Daten binär übertragen, dann verknüpfen Sie das Topic mit einer Variable im entsprechenden Datentyp und deaktivieren Sie „als String“. Meist wird aber wie hier eine Zeichenkette über MQTT gesendet, so dass eine Variable vom Typ cXXXXX verknüpft werden muss. Die Umwandlung in den Zieltyp erfolgt dann mit dem Logikknoten Typkonvertierung. Zur Vereinfachung kann in dem Fall die Option „als String“ aktiviert werden, bei der automatisch eine Zeichenkette vom Typ $$ (begrenzte Länge beachten!) angelegt wird, die dann in den Objekttyp konvertiert wird. Der aktuelle Wert muss mit der neuen Konstante ON vom Typ c1400 verglichen werden, um einen binären Zustand zu erhalten.

Um vom EibPC aus andere Geräte über MQTT zu steuern, muss der EibPC einen Wert an ein Topic veröffentlichen (publish). Dies kann beispielsweise durch ein Schreibtelegramm ausgelöst werden. Falls der zu sendende Wert erst erzeugt werden muss, legen Sie wieder eine neue Variable an, die Sie entsprechend vor dem Senden beschreiben. Für das Beispiel muss bei Eintreffen eines Aus-Telegramms der Wert OFF und bei einem Ein-Telegramm ON gesendet werden. Die Auswahl erfolgt durch einen Multiplexer. Fügen Sie dann im vorhandenen MQTT-Client für das Topic *cmnd/steckdose-tv/Power* eine Veröffentlichung mit der neuen Variable als Objekt ein. Damit wird bei jeder Änderung der Variablen der neue Wert an den Broker und damit alle Clients übertragen, die dieses Topic abonniert haben. Das komplette Beispiel wird in Abbildung 29 gezeigt.

Die entsprechenden Expertenfunktionen sind subscribemqtt, unsubscribemqtt und publishmqtt.

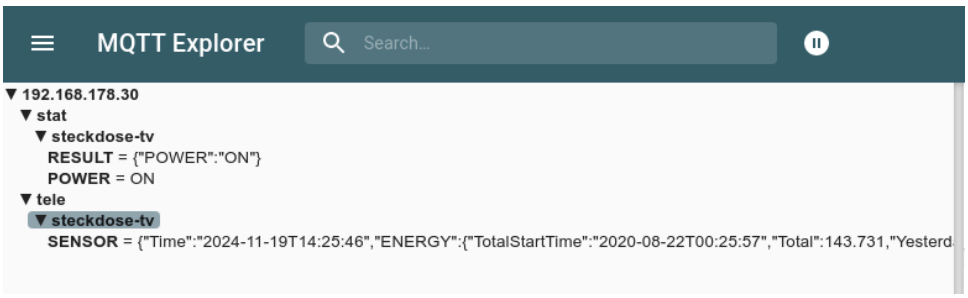


Abbildung 28: MQTT-Explorer zur Anzeige von Werten in verschiedenen Datenformaten

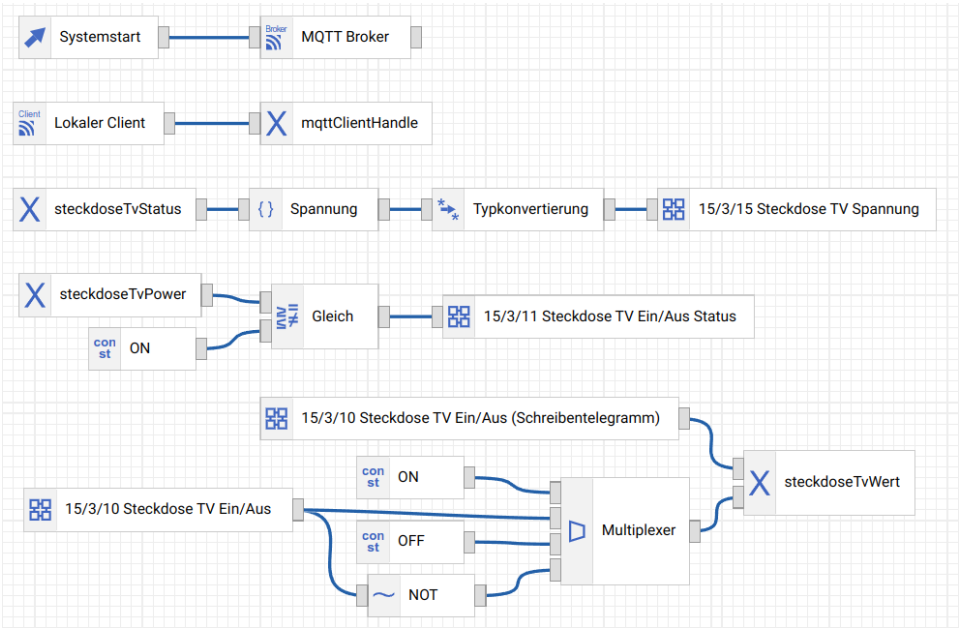


Abbildung 29: Status aus JSON-Objekt an Gruppenadresse 15/3/15 senden und beim Schreiben an 15/3/10 per MQTT schalten

Experte

Visualisierung

Dieses Kapitel ist nur relevant, falls Sie eigene Seiten innerhalb eines Expertenprogramms definieren möchten. Alternativ zur Erstellung ganzer Visualisierungsseiten im Experten können Sie auf einzelne Visualisierungselemente innerhalb von Expertenprogrammen zugreifen, indem Sie ihnen eine ID-Variable zuweisen (siehe S. 19).

Der Zugriff auf die Elemente aus dem Anwendungsprogramm erfolgt mit Hilfe der Visualisierungsfunktionen (ab S. 209).

Um im Experten Seiten zur Visualisierung hinzuzufügen, verwenden Sie im Expertenprogramm zunächst

#addto [WebServer]

Danach können die unten stehenden Befehle verwendet werden, um Seiten anzulegen, sowie Elemente hinzuzufügen. Ob Seiten, die im **Experten** definiert wurden, vor oder nach den Seiten aus der **Visu** erscheinen sollen, kann in den Projekteinstellungen geändert werden.

Zum Beenden der Web-Definitionen müssen Sie anschließend

#addto [EibPC]

verwenden. Danach kann mit dem normalen EibPC-Programm fortgefahren werden.

Seiten

Seiten können bei der Definition in Gruppen zusammengefasst werden. Es sind maximal 100 Seiten möglich, mit maximal 128 Elementen pro Seite (jew. IDs 0-127). Alle Elemente in einer Zeile werden durch ein oder mehrere Leerzeichen oder Tabulatoren voneinander getrennt. Der Compiler erkennt die Anzahl der Elemente pro Zeile und konfiguriert automatisch das Gitter (Abbildung 14). Jedes Element muss mit einer ID versehen werden, so dass vom Anwenderprogramm über die entsprechenden Funktionen darauf zugegriffen werden kann.

Definition

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

Argumente

- *ID*: Wert zwischen 1 bis 100 als Seitenindex für die Programmierung und den Zugriff auf lokale Seitenelemente (Anfangsbuchstabe „p“). Sie können auch auf u08 Variablendefinitionen in der Sektion [EibPC] zugreifen. Die Schnellauswahl (Vor- bzw. Zurückknopf in Abbildung 30) ist durch die Reihenfolge der Definition gegeben. Zwischen zwei Definitionen von page-Elementen sind die Elemente auf dieser Seite zu verwenden.
- *Gruppe*: Eine Zuordnung der Seite zu einer Gruppe. Wenn eine Seite einer Gruppe zugeordnet wird, dann bestimmt die Reihenfolge der Definitionen der Seiten die Reihenfolge der Seiten in der Auswahlbox. Auf diese Weise können Gruppen wie „Keller“, „Erdgeschoß“ etc. generiert werden.
- *Name*: Ein statischer Beschriftungstext

Zugriff im Anwenderprogramm

- keiner

Enertex® Bayern GmbH – Ebermannstädter Straße 8 - 91301 Forchheim - mail@enertex.de

Positionierung

Der Webserver ist in Einheitsgrößen aufgebaut. Alle Elemente passen in dieses Raster bzw. sind ganzzahlige Vielfache hiervon, wie bereits in Abbildung 14 erläutert. Wenn daher ein Element vierfacher Höhe (z.B. mpchart) neben ein Element einfacher Höhe konfiguriert wird,

```
[WebServer]
page(1) [$Demo$,$Compact$]
// die nächste Anweisung ist der default
compact(off)
// Zwei Elemente
mpchart(1) [DOUBLE, SXY]($Beschriftung1$,LINE) mpshifter(2) [$Keller$,$OG$][WEATHER, ICE, NIGHT, CLOCK] $Multi$
```

so entsteht in der Darstellung ein Freiraum wie in Abbildung 30.

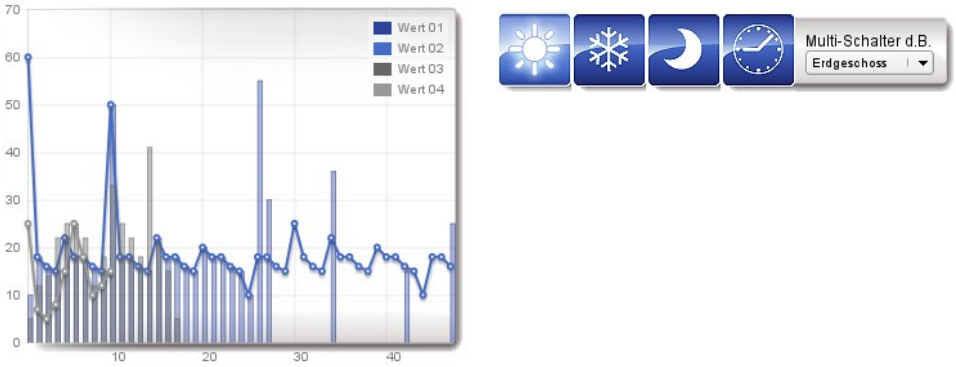


Abbildung 30: Freiraum

Bei der Konfiguration des Webservers steht jede Zeile der Textkonfiguration für eine Darstellungszeile des Webservers. Im „ausgeschalteten“ Kompaktmodus (compact(OFF)) sind die Elemente unterschiedlicher Höhe immer in einer Zeile angeordnet, d.h. die eigentliche Zeilenhöhe der Darstellung wird durch die max. Höhe aller Elemente in der jeweiligen Zeile vorgegeben. Dadurch entsteht die freie Stelle im Webserver. Anders ausgedrückt: In der Darstellung werden zusätzliche nicht sichtbare Elemente unter die Elemente gesetzt. Abbildung 31 zeigt diese „Belegung“ der Einheitsgrößen (in Blau dargestellt) der obigen Webkonfiguration.

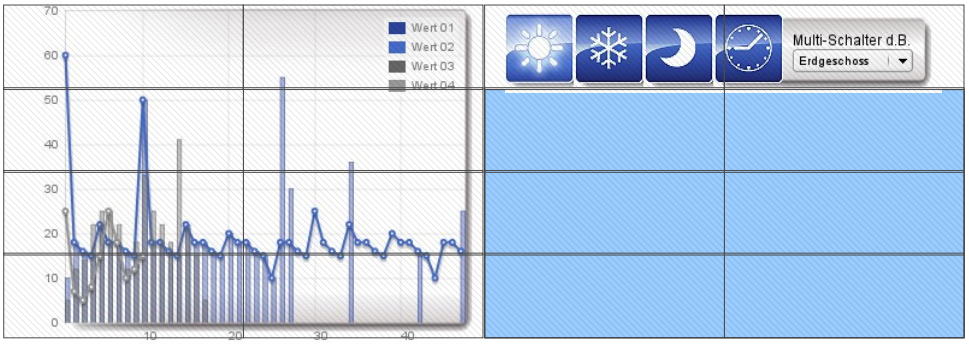


Abbildung 31: Veranschaulichung der Einheitsgrößen.

Der eibparser gibt bereits vorab im Fenster Meldungen die Konfiguration aus:

```
====== Seite: 01/Demo  ======
 mchart (1)   -  mpshifter (2)    -
      |       |         o         o
      |       |         o         o
      |       |         o         o
```

Dabei bedeutet ein Querstrich („-“), dass das Element rechts davon diesen „Platz“, d.h. diese Einheitsgröße belegt, ein senkrechter Strich „|“, dass das Element oberhalb diesen Platz belegt. Ein runder Kreis ist ein automatisch oder per Uservorgabe generiertes leeres Element (none). In Abbildung 31 sind die automatischen generierten Freiräume in blau dargestellt. Diese Ausgabe verdeutlicht somit dem Anwender beim Erstellen der Visu schematisch den Aufbau, wie dieser vom Webserver dargestellt wird. Man vergleiche hierzu die obige Ausgabe des eibparsers mit Abbildung 31.

Wenn man nun den Freiraum rechts neben den Diagramm nutzen will, so muss die Konfiguration geändert wird. Z.b: möchte man neben der Grafik weitere Multibuttons setzen (Abbildung 32).

```
page(1) [$Demo$,$Compact$]
// die nächste Anweisung ist der default
compact(on)
mpchart(1) [DOUBLE, SXY]($Beschriftung1$,LINE) mpshifter(2) [$Keller$,$OG$][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] $Multi$
```

Die erste Zeile ist wie gehabt. Nun können aber die Freiräume von Abbildung 31 genutzt werden, wenn im Kompaktmodus gearbeitet wird. Im Kompaktmodus werden die Elemente nicht zeilenweise in unterschiedlicher Höhe angeordnet. Da die Zeile

```
mpchart(1) [DOUBLE, SXY]($Beschriftung1$,LINE) mpshifter(2) [$Keller$,$OG$][WEATHER, ICE, NIGHT, CLOCK] $Multi$
```

ein mpchart doppelter Breite und vierfacher Höhe konfiguriert, ragt dessen Darstellung in drei weitere Zeilen nach unten hinein.

In den Zeilen

```
mpshifter(3) [$Keller$,$OG$][PLUS, TEMPERATURE, Minus] $Multi$
mpshifter(4) [$Keller$,$OG$][PLUS, TEMPERATURE, Minus] $Multi$
mpshifter(5) [$Keller$,$OG$][PLUS, TEMPERATURE, Minus] $Multi$
```

sind Elemente mit doppelter Breite und einfacher Höhe eingebaut. Durch das erste Element sind jeweils zusätzlich 2 Einheitselemente in der Zeile bereits „unsichtbar vorhanden“. Diesen Zeilenüberlauf gibt bereits der eibparser bereits vorab durch die Ausgabe der „-“ bzw. „|“ Zeichen aus:

```
====== Seite: 01/Demo  ======
mchart (1)    -  mpshifter (2)    -
      |       |  mpshifter (3)    -
      |       |  mpshifter (4)    -
      |       |  mpshifter (5)    -
```

an.

Man vergleiche hierzu die Abbildung 32, welche nun der Webserver ausgibt.

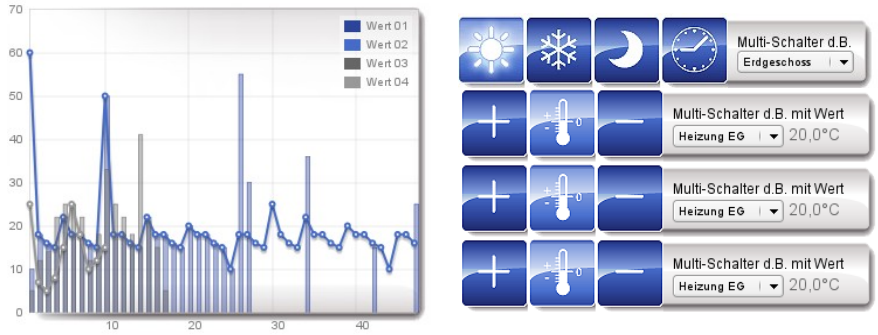


Abbildung 32: Compactmodus

Mit der Anweisung compact(EIN) kann das Platzieren von Elementen unterschiedlicher Höhe nebeneinander aktiviert werden. Dabei berechnet der Webserver selbst die Höhenüberläufe in die nächste Zeile. Der Anwender darf hier keine none-Elemente platzieren, wenn die Breite nicht vergrößert werden soll. Abbildung 33 zeigt hierzu nochmals schematisch die Anordnung der Elemente, wie das auch schon im eibparser ausgegeben wird.

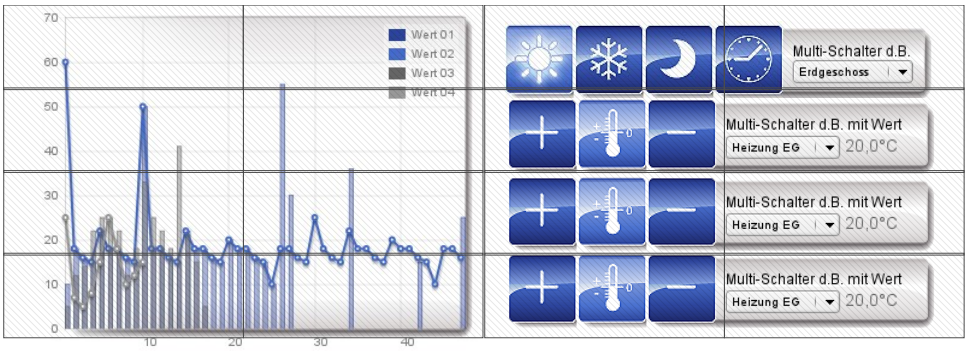


Abbildung 33: „compact“ mit Raster (zur Veranschaulichung)

Im Modus mit compact(on) des Webservers muss der Anwender also die Größe des Webelements in die nächste Zeile der Konfiguration mit berücksichtigen, um so die Anordnung der Webelemente zu steuern. Will man eine freie Zeile unter Berücksichtigung der Zeilenüberläufe generieren, so muss mit dem empty-Element gearbeitet werden.

Hierzu folgendes Beispiel

```
page(1) [$Demo$,$Compact$]
// die nächste Anweisung ist der default
compact(on)
mpchart(1) [DOUBLE, SXY]($Beschriftung1$,LINE) mpchart(2) [DOUBLE, SXY]($Beschriftung1$,LINE)
mpshifter(3) [$Keller$,$OG$][WEATHER, ICE, NIGHT, CLOCK] $Multi$
```

Die ersten beiden Elemente belegen je 2 Einheitsbreiten und 4 Einheitshöhen. Nach dem Zeilenumbruch in der Konfiguration der beiden mpcharts beginnt eine neue Zeile in der Darstellung. Diese hat einen „Übertrag“ von zwei mal zwei belegten Einheitselementen. Dann wird in der nächsten Zeile ein mpshifter konfiguriert. Daher muss die Seite mindestens 6 Einheitselemente breit sein. Diese wird auch vom eibparser so ausgegeben:

```
====== Seite: 01/Demo  ======

mchart (1)   -  mchart (2)   -         o         o
     |       |       |       |  mpshifter (3)    -
     |       |       |       |         o         o
     |       |       |       |         o         o
```

Letztlich wird der Webserver eine Darstellung wie in Abbildung 34 ausgeben.

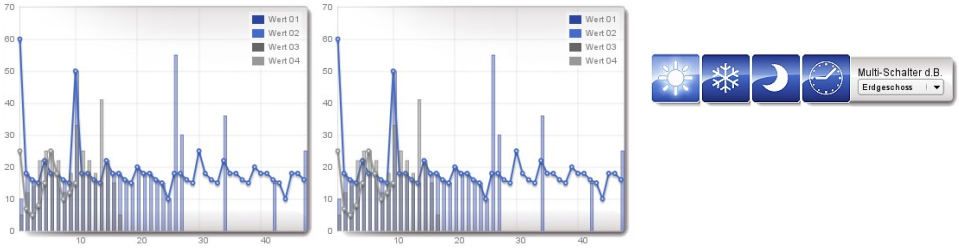


Abbildung 34: Darstellungsbeispiel für den Zeilenvorschub

Will man nun erreichen, dass der 4-Fach Button unterhalb der beiden Grafen angezeigt wird, so müssen empty-Elemente wie folgt konfiguriert werden:

```
page(1) [$Demo$,$Compact$]
// die nächste Anweisung ist der default
compact(on)
mpchart(1) [DOUBLE, SXY]($Beschriftung1$,LINE) mpchart(2) [DOUBLE, SXY]($Beschriftung1$,LINE)
empty
empty
empty
mpshifter(3) [$Keller$,$OG$][WEATHER, ICE, NIGHT, CLOCK] $Multi$
```

Die drei Empty-Elemente fügen nun Leerzeilen ein bzw. überspringen jeweils eine Zeile in der Darstellung. Auch hier kann dies schon schon vorab anhand der vom eibparser angegebenen Ausgabe im Meldungsfenster erkannt werden:

```
====== Seite: 01/Demo  ======

mchart (1)       -  mchart (2)   -
       |         |       |       |
       |         |       |       |
       |         |       |       |
mpshifter (3)    -       o       o
```

Der Seiten-Index der lokalen Elemente - , also Elemente, die nur einer Seite zugeordnet werden, - ist derjenige, der im vorangegangenen page-Kommando angegeben ist.

Kompaktmodus

Definition

- compact (*Zustand*)

Argumente

- *Zustand* Wert von 0 / 1 bzw. EIN/AUS

Passwortschutz

Definition

- *user* $*Name*$ [*Passwort*]

Argumente

- *Name*: Benutzername. Dieser Benutzer hat Zugriff auf die entsprechende Seite. Nur Zeichen A-Za-z0-9. Maximale Länge 32 Zeichen.
- *Passwort*: Der angegeben Benutzer benötigt dieses Passwort um Zugriff auf die entsprechende Seite zu bekommen. Nur Zeichen A-Za-z0-9. Maximale Länge 32 Zeichen.
- Der Befehl kann pro Seite mehrfach verwendet werden, um mehrere Benutzer zu konfigurieren.
- Die Benutzer:Passwort-Kombination muss auf jeder Seite gleich sein, ein Benutzer kann also nur ein Passwort verwenden.
- Es können maximal 20 Benutzernamen/Passwort-Kombinationen über alle Seiten hinweg angegeben werden.

Zugriff im Anwenderprogramm

- keinen

Das Benutzerpasswort wird nicht im Klartext übertragen, auch wenn die Seite per http statt https aufgerufen wird. Dennoch ist es empfehlenswert, auch lokal die Visualisierung immer per https zu öffnen.

Beispiel

```
[WebServer]
page(1) [$Benutzerverwaltung$,$Seite 1$]
user $Michael$ [PasswortM]
user $Florian$ [PasswortF]
button(1) [INFO] $Seite 1$

page(2) [$Benutzerverwaltung$,$Seite 2$]
// Passwörter werden übernommen
user $Michael$
user $Florian$
button(1) [INFO] $Seite 2$

page(3) [$Benutzerverwaltung$,$Seite 3$]
// Diese Seite ist nur für Michael
// Passwort wird übernommen
user $Michael$
button(1) [INFO] $Seite 3$

page(4) [$Benutzerverwaltung$,$Seite 4$]
// Diese Seite ist nur für Stefanie
// Passwort muss mit angegeben werden, da dieser User auf dern vorherigen Seiten nicht genannt war
user $Stefanie$ [SGut]
button(1) [INFO] $Seite 4$

page(5) [$Benutzerverwaltung$,$Seite 5$]
// Alles User
button(1) [INFO] $Seite 5$
```

Farbschema

Definition

- design $*DESIGNSTRING*$ [$*Link/Pfad zu Grafik*$] [$*CSS-Style*$]

Argumente

- $*DESIGNSTRING*$ kann mit $black$ angegeben werden, wenn das „black design“ gewählt werden soll.
- $*DESIGNSTRING*$ kann mit $blue$ das Standard-Blau Design angebenden werden.
- Das design-Kommando kann für jede Seite unterschiedlich angegeben werden.
- $*Link/Pfad zu Grafik*$ bezeichnet einen Pfad auf dem internen Flash oder einen externen Server. Die Hintergrundgrafik wird nicht skaliert. Die Anordnung der Webelemente bleibt davon unberücksichtigt, none-Elemente werden transparent.
- $*CSS-Style*$ gibt optional das CSS „style“-Attribut für den Hintergrund an. Dadurch kann die Darstellung der Hintergrundgrafik angepasst werden.
  Beispiel:
  design $*black*$ [$*/upload/livingroom.jpg*$] [$*background-position:center;filter:blur(4px)*$]
  (ab EibStudio 4.113, Firmware 4.114).
- Der Pfad auf Grafiken, die auf den EibPC geladen wurden, ist „/upload/“

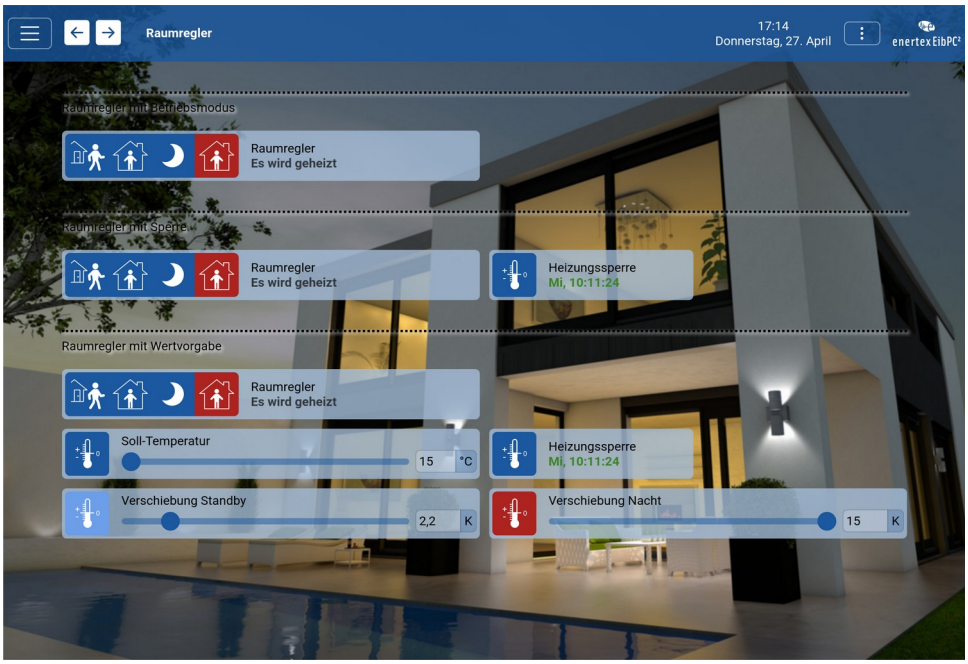


Abbildung 35: Hintergrundgrafik

Platzhalter (Kompaktmodus)

Definition

- empty

Argumente

- keine

Fügt eine Leerzeile bei der Zählung der Webelemente im Kompakt-Modus ein.

Platzhalter

Definition

- none

Argumente

- Keine. Ein leeres Element einfacher Breite wird im Webserver eingefügt.

Zugriff im Anwenderprogramm

- keinen

Trennlinie

Definition

- line(*Aussehen*) $*Text*$

Argumente

- *Aussehen* legt die Art der Zeile fest und ist optional. Folgende Werte sind möglich:
  - 0: Standard, gepunktete Linie über dem Text
  - 1: durchgehende Linie über dem Text
  - 2: durchgehende Linie über und unter dem Text
  - 3: durchgehender Rahmen um den Text
  - 4: kein Rahmen
- Der *Text* wird an die Trennlinie gebunden und ist optional. Für Zeilenumbrüche kann das HTML-Tag <br> verwendet werden.

Zugriff im Anwenderprogramm

- keinen

Responsive Design

Definition

- responsive(*Zustand*)

Argumente

- *Zustand* Steuert die automatische Anpassung der Darstellung , je nach Endgerätegröße:
  - 0: keine Anpassung, Spalten haben eine feste Breite. Zeigt Scrollbars, falls die Seite zu groß ist.
  - 1 (Standard): automatische Anpassung der Spaltenanzahl für die aktuelle Seite, füllt Zeilenweise auf
  - 2: automatische Anpassung der Spaltenanzahl für die aktuelle Seite, füllt Spaltenweise auf
  - 3: absolute Positionierung ohne Anpassung

Zugriff im Anwenderprogramm

- keinen

Enertex® Bayern GmbH – Ebermannstädter Straße 8 - 91301 Forchheim - mail@enertex.de

Eigene Logos

Definition

- navlogo(*Zustand*) $*URL*$

Argumente

- *Zustand* 0 um das Logo in der seitlichen Navigationsleiste zu deaktivieren, 1 für das Enertex-Logo (Standard), 2 für ein eigenes Logo.
- *URL* Adresse des Bildes (png, gif, jpg, svg), das oben in der Seitennavigation angezeigt werden soll. Muss bei Zustand 2 angegeben werden.

Zugriff im Anwenderprogramm

- keinen

Definition

- titlebarlogo(*Zustand*) $*URL*$

Argumente

- *Zustand* 0 um das Logo in der Leiste am oberen Bildschirmrand zu deaktivieren, 1 für das Enertex-Logo (Standard), 2 für ein eigenes Logo.
- *URL* (optional) Adresse des Bildes (png, gif, jpg, svg), das oben in der Titelleiste angezeigt werden soll. Muss bei Zustand 2 angegeben werden.

Zugriff im Anwenderprogramm

- keinen

Die folgenden Konfigurationsoptionen haben keine Auswirkung auf die Responsive Visu ab Firmware 5.000 und dienen nur zur Dokumentation.

Kopfzeile

Definition

- header(*Nummer*) $*URL*$

Argumente

- Wenn *Nummer* den Wert 0 annimmt, wird Kopfzeile ausgeblendet. Sie können auch auf u08 Variablendefinitionen in der Sektion [EibPC] zugreifen.
- Die *URL* (inkl. Pfad und führenden http://) ist optional. Dieser kann auf eine externe Quelle zugreifen. In diesem Fall muss für Nummer der Wert 2 gesetzt werden.
- Die Fußzeile ist seitenbasiert.

Zugriff im Anwenderprogramm

- keinen

Fußzeile

Definition

- footer(*Nummer*) $*URL*$

Argumente

- Wenn *Nummer* den Wert 0 annimmt, wird Kopfzeile ausgeblendet. Sie können auch auf u08 Variablendefinitionen in der Sektion [EibPC] zugreifen.
- Die *URL* (inkl. Pfad und führenden http://) ist optional. Dieser kann auf eine externe Quelle zugreifen. In diesem Fall muss für Nummer der Wert 2 gesetzt werden.
- Die Fußzeile ist seitenbasiert.

Zugriff im Anwenderprogramm

keinen

Zoom

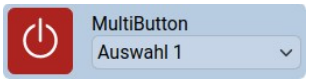
Definition

- mobilezoom*(Faktor)*

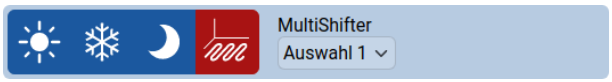
Argumente

- *Faktor*: ganzzahliger Wert von 0 bis 255 als Zoomfaktor in Prozent für den Zoom der Visualisierung auf mobilen Geräten oder Android-baisierten Panels. Der Zoomfaktor Wirkt nur auf die Seite, die mit einer vorangegangen page-Konfiguration einleitend definiert wurde.

Elemente

Gruppe	Element	Erläuterung
button		
	button, pbutton	 Die Grafik, die das eigentliche Bedienfeld darstellt, kann durch das Anwenderprogramm verändert werden. Die erste Textzeile ist statisch (nur bei der Konfiguration veränderbar). Die zweite Zeile kann durch das Anwenderprogramm verändert werden, z. B. zum Anzeigen von Variablen.
	shifter, pshifter	 Die Grafik kann durch das Anwenderprogramm verändert werden. Die erste Textzeile ist statisch (nur bei der Konfiguration veränderbar). Die zweite Zeile kann durch das Anwenderprogramm verändert werden.
	shifter, pshifter	 Die **rechte** Grafik kann durch das Anwenderprogramm verändert werden. Die linke Grafik kann nur bei der Konfiguration verändert werden. Die erste Textzeile ist statisch (nur bei der Konfiguration veränderbar). Die zweite Zeile kann durch das Anwenderprogramm verändert werden.
	shifter, pshifter	 Die **mittlere** Grafik kann durch das Anwenderprogramm verändert werden. Die beiden äußeren Grafiken können nur bei der Konfiguration verändert werden. Die erste Textzeile ist statisch (nur bei der Konfiguration veränderbar). Die zweite Zeile kann durch das Anwenderprogramm verändert werden.
	shifter	 Die **rechte** Grafik kann durch das Anwenderprogramm verändert werden. Die anderen Grafiken können nur bei der Konfiguration verändert werden. Die erste Textzeile ist statisch (nur bei der Konfiguration veränderbar). Die zweite Zeile kann durch das Anwenderprogramm verändert werden.
mbutton		
	mbutton, mpbutton	 Die Grafik, die das eigentliche Bedienfeld darstellt, kann durch das Anwenderprogramm verändert werden. Die erste Textzeile ist statisch (nur bei der Konfiguration veränderbar). Die aktive Auswahl kann durch das Anwenderprogramm verändert werden. Dabei muss dieses dann den Zustand der Grafik anpassen. In der zweiten Zeile kann kein Text angezeigt werden. Die Listbox kann maximal 254 Einträge verwalten. Bei Betätigung der Listbox wird ein Signal an das Anwendungsprogramm geschickt, das mit der Funktion mbutton (S. 210) bzw. mbbutton (S. 210) abgefragt werden kann.

Mit mshifter können Sie bis zu 254 verschiedene Aktionen in einen Knopf packen.

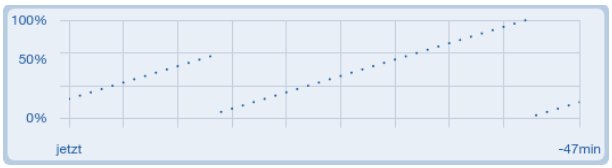
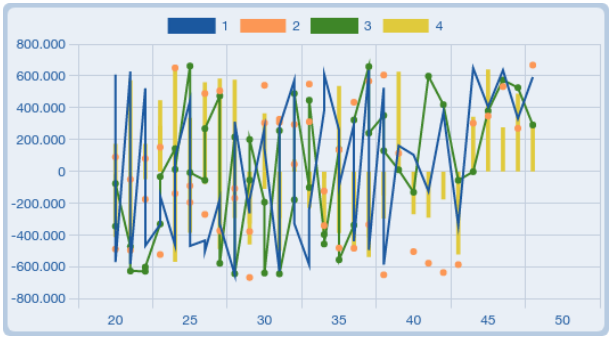
Gruppe	Element	Erläuterung
	mshifter, mpshifter	 Die Grafik kann durch das Anwenderprogramm verändert werden. Die erste Textzeile ist statisch (nur bei der Konfiguration veränderbar). Die zweite Zeile kann durch das Anwenderprogramm verändert werden. Die Listbox kann maximal 254 Einträge verwalten. Bei Betätigung der Listbox wird ein Signal an das Anwendungsprogramm geschickt, welches mit der Funktion mbutton (S. 210) bzw. mpbutton (S. 210) abgefragt werden kann.
	mshifter, mpshifter	 Die **rechte** Grafik kann durch das Anwenderprogramm verändert werden. Die linke Grafik kann nur bei der Konfiguration verändert werden. Die erste Textzeile ist statisch (nur bei der Konfiguration veränderbar). Die zweite Zeile kann durch das Anwenderprogramm verändert werden. Die Listbox kann maximal 254 Einträge verwalten. Bei Betätigung der Listbox wird ein Signal an das Anwendungsprogramm geschickt, welches mit der Funktion mbutton (S. 210) bzw. mpbutton (S. 210) abgefragt werden kann.
	mshifter, mpshifter	 Die **mittlere** Grafik kann durch das Anwenderprogramm verändert werden. Die beiden äußeren Grafiken können nur bei der Konfiguration verändert werden. Die erste Textzeile ist statisch (nur bei der Konfiguration veränderbar). Die zweite Zeile kann durch das Anwenderprogramm verändert werden. Die Listbox kann maximal 254 Einträge verwalten. Bei Betätigung der Listbox wird ein Signal an das Anwendungsprogramm geschickt, welches mit der Funktion mbutton (S. 210) bzw. mpbutton (S. 210) abgefragt werden kann.
	mshifter, mpshifter	 Die rechte Grafik kann durch das Anwenderprogramm verändert werden. Die anderen Grafiken können nur bei der Konfiguration verändert werden. Die erste Textzeile ist statisch (nur bei der Konfiguration veränderbar). Die Listbox kann maximal 254 Einträge verwalten. Bei Betätigung der Listbox wird ein Signal an das Anwendungsprogramm geschickt, welches mit der Funktion mbutton (S. 210) bzw. mpbutton (S. 210) abgefragt werden kann.
	Slider pslider	 Die Grafik und die Stellung des Sliders kann durch das Anwenderprogramm mit den Funktionen setslider bzw. setpslider verändert werden. Der Knopfdruck kann mit der Funktion mbutton (S. 210) bzw. mpbutton (S. 210) abgefragt werden.
	eslider peslider	Die Grafik und die Stellung des Sliders kann durch das Anwenderprogramm mit den Funktionen setslider bzw. setpslider verändert werden. Der Knopfdruck kann mit der Funktiion mbutton (S. 210) bzw. mpbutton (S. 210) abgefragt werden. Minimale und Maximale Stellung, sowie Inkrement des Sliders kann vorgegeben werden.

Mit mchart können Sie bis zu vier verschiedene Kurvenverläufe darstellen...

... entweder in „einfacher“ ...

.. oder „doppelter“ Höhe ...

... oder externe Bildquellen einbinden...

Gruppe	Element	Erläuterung
chart		
	chart, pchart	 Dieses Element dient zum Anzeigen von Verläufen. Die Beschriftung der Y-Achse wird bei der Konfiguration vorgeben. Die X-Achsen Beschriftung kann durch das Anwenderprogramm verändert werden. Bei Aufruf der Funktion webdisplay wird die XY-Darstellung aktiviert. Es können Werte aus dem Bereich 1..30 dargestellt werden. 0 bedeutet keine Darstellung. Die Werte werden von links beginnend dargestellt. Wenn nach 47 Aufrufen das Ende erreicht ist, werden die Werte nach links geschoben.
	mchart mpchart	 Die XY-Werte werden im Anwenderprogramm über die Funktion mchart (Seite 220) angesprochen. Ein mchart-Element verwaltet bis zu 4 XY-Diagramme, die über die gleich lautende Funktion mchart im Anwendungsprogramm mit Daten gespeist wird. Es können max. 4 Diagramme vorgegeben werden. Jedes der 4 Diagramme hat eine eigene Legende (rechts oben eingeblendet). Es werden 47 Fließkommawerte dargestellt. Die Skala wird automatisch generiert.
	picture	 Vorhersage Es wird ein externer Link auf eine Grafik eingebunden. Die Grafik kann linksbündig, zentriert oder gestreckt eingebunden werden.

...komplette Webpages einbinden...

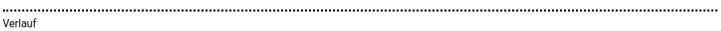
Gruppe	Element	Erläuterung
Link		
	frame dframe	 Einbetten einer einer externen Seite
	pLink	 Link auf eine interne Seite als einfacher Knopf
	link	Link auf eine externe Seite als einfacher Knopf
Dekos		
	line	 Erzwingt eine freie Zeile mit Trennlinie im Webserveraufbau. Die Überschrift ist optional.
	none	Ein Freifeld einfacher Breite

Tabelle 4: Übersicht über Webelemente.

IDs

Seitenbezogen

Im Anwenderprogramm erfolgt der Zugriff auf Visualisierungselemente anhand einer eindeutigen Nummer (ID), die bei der Definition in der [WebServer]-Sektion festgelegt wird. Grundsätzlich wird zwischen seitenbezogenen und globalen Elementen unterschieden. Bei seitenbezogenen Elementen bezieht sich die ID des Elements immer auf die ebenfalls anzugebende Seite, d.h. Elemente mit der gleichen ID auf unterschiedlichen Seiten können getrennt voneinander adressiert und geändert werden. Dabei wird nicht zwischen verschiedenen seitenbezogenen Elementtypen auf einer Seite unterschieden. Es kann pro Seite eine ID also nur einmal über alle seitenbezogenen Elemente hinweg verwendet werden.

Fehlerhaftes Beispiel:

```
#addto [WebServer]
page(6)[$Seitenbezogene Elemente$, $Buttons$]
pbutton(0)[LIGHT]$Test$
picture(0)[2x1,CENTERGRAF]($Beschriftung$,$URL$)

#addto [EibPC]
if change(CLOCKDATE_STRING) then {
   picture(0, CLOCKDATE_STRING, 6, $URL$);
} endif
```

Hier wird versucht, mit der Funktion picture ein Element vom Typ pbutton zu ändern.

Korrigiert:

```
#addto [WebServer]
page(6)[$Seitenbezogene Elemente$, $Buttons$]
pbutton(0)[LIGHT]$Test$
picture(1)[2x1,CENTERGRAF]($Beschriftung$,$URL$)

#addto [EibPC]
if change(CLOCKDATE_STRING) then {
   pdisplay(0, CLOCKDATE_STRING, LIGHT, ACTIVE, BLINKRED, 6);
   picture(1, CLOCKDATE_STRING, 6, $URL$);
} endif
```

pbutton und picture haben (seitenbezogen) eindeutige IDs und werden mit den passenden Funktionen adressiert.

Global

Demgegenüber stehen globale Elemente. Diese besitzen ebenfalls eine ID, jedoch ohne Seitenbezug und nur bezogen auf den Elementtyp. IDs können gezielt mehrfach verwendet werden, um Elemente mit identischen Eigenschaften mehrfach zu verwenden. Werden durch Verwendung der entsprechenden Funktion die Eigenschaften eines Elements mit einer bestimmten ID geändert, so betrifft das alle globalen Elemente des gleichen Typs mit der gleichen ID, über alle Seiten hinweg. Globale Elemente mit der gleichen ID können auch mehrfach auf der gleichen Seite vorkommen.

Beispiel:

```
#addto [WebServer]
page(4)[$Globale Elemente$, $Buttons$]
button(0)[LIGHT]$Test 1$
button(0)[LIGHT]$Test 2$
page(5)[$Globale Elemente$, $Buttons$]
button(0)[LIGHT]$Test 3$

#addto [EibPC]
if change(CLOCKDATE_STRING) then {
   display(0, CLOCKDATE_STRING, LIGHT, ACTIVE, BLINKRED);
} endif
```

Die sekündliche Änderung der Beschriftung erfolgt gleichzeitig für alle drei Buttons mit einem einzigen Befehl.

Elementdefinitionen

Schaltfläche einfacher Breite (global)

Definition

- button(*ID*){*X*,*Y*}[*Grafik*][*State*] $*Text*$

Argumente

- *ID*: Wert von 0 bis 127) als Index für die Programmierung und den Zugriff auf dieses Element. Sie können auch auf u08 Variablendefinitionen in der Sektion [EibPC] zugreifen.
- *X,Y:* Position im Raster oder absolut (u16), abhängig von der Konfiguration von responsive. Der gesamte {}-Block ist optional. Ohne Angabe erfolgt die Positionierung gemäß Positionierung.
- *Grafik*: Wert zwischen 0 und 127. Um die Anwendung übersichtlicher zu gestalten, sind vordefinierte Konstanten definiert (Seite 45).
- *State* (optional): Startzustand der Grafik entsprechend Tabelle 2.
- *Text*: Ein statischer Beschriftungstext (erste Zeile).

Zugriff im Anwenderprogramm

- Die Grafik und der Text werden über die Funktion display (Seite 211) angesprochen.
- Es handelt sich um einen globalen Knopf. Wenn die gleiche Definition auf mehreren Seiten verwendet wird, so werden auf allen Seiten die globalen Elemente mit ID angesprochen.
- Die Betätigung der Knöpfe ist über die Funktion button (Seite 209) auszuwerten.

Schaltfläche einfacher Breite (seitenbezogen)

Definition

- pbutton(*ID*){*X*,*Y*}[*Grafik*][*State*] $*Text*$

Argumente

- *ID*: Wert von 0 bis 127) als Index für die Programmierung und den Zugriff auf dieses Element. Sie können auch auf u08 Variablendefinitionen in der Sektion [EibPC] zugreifen.
- *X,Y:* Position im Raster oder absolut (u16), abhängig von der Konfiguration von responsive. Der gesamte {}-Block ist optional. Ohne Angabe erfolgt die Positionierung gemäß Positionierung.
- *Grafik*: Wert zwischen 0 und 127. Um die Anwendung übersichtlicher zu gestalten, sind vordefinierte Konstanten definiert (Seite 45).
- *State* (optional): Startzustand der Grafik entsprechend Tabelle 2.
- *Text*: Ein statischer Beschriftungstext (erste Zeile).

Zugriff im Anwenderprogramm

- Die Grafik und der Text werden über die Funktion pdisplay (Seite 212) angesprochen.
- Es handelt sich um ein Element, das immer nur einer Seite zugeordnet ist.
- Die Betätigung der Knöpfe ist über die Funktion pbutton (Seite 209) auszuwerten.

Enertex® Bayern GmbH – Ebermannstädter Straße 8 - 91301 Forchheim - mail@enertex.de

Schaltfläche mit Auswahl (global)

Definition

- mbutton(*ID*){*X,Y*}[$*Text1*$,$*Text2*$,... $*Text254*$][*Grafik*][*State*] $*Beschriftung*$

Argumente

- *ID*: Wert von 0 bis 127) als Index für die Programmierung und den Zugriff auf dieses Element. Sie können auch auf u08 Variablendefinitionen in der Sektion [EibPC] zugreifen.
- *X,Y:* Position im Raster oder absolut (u16), abhängig von der Konfiguration von responsive. Der gesamte {}-Block ist optional. Ohne Angabe erfolgt die Positionierung gemäß Positionierung.
- *Text1, Text2, .. Text254:* Beschriftungsfelder für den mbutton. Ab dem 2. Element sind die Elemente optional.
- *Grafik*: Wert zwischen 0 und 127. Um die Anwendung übersichtlicher zu gestalten, sind vordefinierte Konstanten definiert (Seite 45).
- *State* (optional): Startzustand der Grafik entsprechend Tabelle 2.
- *Beschriftung*: Ein statischer Beschriftungstext (erste Zeile).

Zugriff im Anwenderprogramm

- Die Grafik und der Text werden über die Funktion display (Seite 211) angesprochen.
- Es handelt sich um einen globalen Knopf. Wenn die gleiche Defintion auf mehreren Seiten verwendet wird, so werden auf allen Seiten die globalen Elemente mit ID angesprochen.
- Die Betätigung der Knöpfe ist über die Funktion mbutton (Seite mbutton) auszuwerten.
- Die Umschaltung der Listbox (Vorgabe des aktiven Listboxelements) wird über die Funktion display (Seite display) angesprochen.

Schaltfläche einfacher Breite mit Auswahl (seitenbezogen)

Definition

- mpbutton(*ID*){*X,Y*}[$*Text1*$,$*Text2*$,...$*Text254*$][*Grafik*][*State*] $*Beschriftung*$

Argumente

- *ID*: Wert von 0 bis 127) als Index für die Programmierung und den Zugriff auf dieses Element. Sie können auch auf u08 Variablendefinitionen in der Sektion [EibPC] zugreifen.
- *X,Y:* Position im Raster oder absolut (u16), abhängig von der Konfiguration von responsive. Der gesamte {}-Block ist optional. Ohne Angabe erfolgt die Positionierung gemäß Positionierung.
- *Text1, Text2, .. Text254: Beschriftungsfelder für den mpbutton. Ab dem 2. Element sind die Elemente optional.*
- *Grafik*: Wert zwischen 0 und 127. Um die Anwendung übersichtlicher zu gestalten, sind vordefinierte Konstanten definiert (Seite 45).
- *State* (optional) Startzustand der Grafik entsprechend Tabelle 2.
- *Beschriftung*: Ein statischer Beschriftungstext (erste Zeile).

Zugriff im Anwenderprogramm

- Die Grafik und der Text werden über die Funktion pdisplay (Seite 212) angesprochen. Die Umschaltung der Listbox (Vorgabe des aktiven Listboxelements) wird auch über diese Funktion angesprochen.
- Die Betätigung der Knöpfe ist über die Funktion mpbutton (Seite 210) auszuwerten.

Schaltfläche doppelter Breite (global)

Definition

- shifter(*ID*){*X*,*Y*}[*Grafik1*,*Grafik2*, *Grafik3*, *Grafik4*][*State1*, *State2*, *State3*, *State4*]$*Text*$

Argumente

- *ID*: Wert zwischen 0 bis 127) als Index für die Programmierung und den Zugriff auf dieses Element. Sie können auch auf u08 Variablendefinitionen in der Sektion [EibPC] zugreifen.
- *X,Y:* Position im Raster oder absolut (u16), abhängig von der Konfiguration von responsive. Der gesamte {}-Block ist optional. Ohne Angabe erfolgt die Positionierung gemäß Positionierung.
- *Grafik1* bis *Grafik4*: Wert zwischen 0 und 127. Um die Anwendung übersichtlicher zu gestalten, sind vordefinierte Konstanten definiert (Seite 45).
- *Grafik2* bis *Grafik4* sind optional.
- Wenn nur drei Grafiken angegeben werden, so hat das Element nur drei Knöpfe usw..
- *State1* bis *State4* (optional): Startzustand der Grafik entsprechend Tabelle 2. *State1* gibt den Zustand für *Grafik1* an, *State2* für *Grafik2* usw.
- Es können nur Zustände für tatsächlich angegebene Grafiken angegeben werden.
- *Text:* Ein statischer Beschriftungstext (erste Zeile).

Schaltfläche doppelter Breite (seitenbezogen)

Zugriff im Anwenderprogramm

- Die Grafik und der Text werden über die Funktion pdisplay (S. 212) angesprochen.
- Die Betätigung der Knöpfe ist über die Funktion button (Seite 209) auszuwerten.

Definition

- pshifter(*ID*)[*Grafik1*,*Grafik2*, *Grafik3*, *Grafik4*][*State1*, *State2*, *State3*, *State4*]$*Text*$

Argumente

- *ID*: Wert zwischen 0 bis 127) als Index für die Programmierung und den Zugriff auf dieses Element. Sie können auch auf u08 Variablendefinitionen in der Sektion [EibPC] zugreifen.
- *X,Y:* Position im Raster oder absolut (u16), abhängig von der Konfiguration von responsive. Der gesamte {}-Block ist optional. Ohne Angabe erfolgt die Positionierung gemäß Positionierung.
- *Grafik1* bis *Grafik4*: Wert zwischen 0 und 127. Um die Anwendung übersichtlicher zu gestalten, sind vordefinierte Konstanten definiert (Seite 45).
- *Grafik2* bis *Grafik4* sind optional.
- Wenn nur drei Grafiken angegeben werden, so hat das Element nur drei Knöpfe usw..
- *State1* bis *State4* (optional): Startzustand der Grafik entsprechend Tabelle 2. *State1* gibt den Zustand für *Grafik1* an, *State2* für *Grafik2* usw.
- Es können nur Zustände für tatsächlich angegebene Grafiken angegeben werden.
- *Text:* Ein statischer Beschriftungstext (erste Zeile).

Zugriff im Anwenderprogramm

- Die Grafik und der Text werden über die Funktion pdisplay (S. 212) angesprochen.
- Die Betätigung der Knöpfe ist über die Funktion pbutton (Seite 209) auszuwerten.

Enertex® Bayern GmbH – Ebermannstädter Straße 8 - 91301 Forchheim - mail@enertex.de

Schaltfläche doppelter Breite mit Auswahl (global)

Definition

- mshifter(*ID*){*X*,*Y*}[$*Text1*$,$*Text2*$,...,$*Text254*$][*Grafik1*,*Grafik2*, *Grafik3*, *Grafik4*] [*State1*, *State2*, *State3*, *State4*] $*Beschriftung*$

Argumente

- *ID*: Wert zwischen 0 bis 127) als Index für die Programmierung und den Zugriff auf dieses Element. Sie können auch auf u08 Variablendefinitionen in der Sektion [EibPC] zugreifen.
- *X,Y:* Position im Raster oder absolut (u16), abhängig von der Konfiguration von responsive. Der gesamte {}-Block ist optional. Ohne Angabe erfolgt die Positionierung gemäß Positionierung.
- *Text1, Text2, .. Text254:* Beschriftungsfelder für den *mshifter.* Ab dem 2. Element sind die Elemente optional.
- *Grafik1* bis *Grafik4*: Wert zwischen 0 und 127. Um die Anwendung übersichtlicher zu gestalten, sind vordefinierte Konstanten definiert (Seite 45).
- *Grafik2* bis *Grafik4* sind optional.
- Wenn nur drei Grafiken angegeben werden, so hat das Element nur drei Knöpfe usw..
- *State1* bis *State4* (optional): Startzustand der Grafik entsprechend Tabelle 2. *State1* gibt den Zustand für *Grafik1* an, *State2* für *Grafik2* usw.
- Es können nur Zustände für tatsächlich angegebene Grafiken angegeben werden.
- *Beschriftung*: Ein statischer Beschriftungstext (erste Zeile).

Zugriff im Anwenderprogramm

- Die Grafik und der Text werden erläutert über die Funktion pdisplay (S. 212) angesprochen. Die Umschaltung der Listbox (Vorgabe des aktiven Listboxelements) wird auch über diese Funktion angesprochen.
- Die Betätigung der Knöpfe ist über die Funktion mbutton (Seite 209) auszuwerten.

Schaltfläche doppelter Breite mit Auswahl (seitenbezogen)

Definition

- mpshifter(*ID*){*X*,*Y*}[$*Text1*$,$*Text2*$,...,$*Text254*$][*Grafik1*,*Grafik2*, *Grafik3*, *Grafik4*][*State1*, *State2*, *State3*, *State4*] $*Beschriftung*$

Argumente

- *ID*: Wert zwischen 0 bis 127) als Index für die Programmierung und den Zugriff auf dieses Element. Sie können auch auf u08 Variablendefinitionen in der Sektion [EibPC] zugreifen.
- *X,Y:* Position im Raster oder absolut (u16), abhängig von der Konfiguration von responsive. Der gesamte {}-Block ist optional. Ohne Angabe erfolgt die Positionierung gemäß Positionierung.
- *Text1, Text2, ..., Text254: Beschriftungsfelder für den mpshifter. Das 2. und 3. Element ist dabei optional.*
- *Grafik1* bis *Grafik4*: Wert zwischen 0 und 127. Um die Anwendung übersichtlicher zu gestalten, sind vordefinierte Konstanten definiert (Seite 45).
- *Grafik2* bis *Grafik4* sind optional.
- Wenn nur drei Grafiken angegeben werden, so hat das Element nur drei Knöpfe usw..
- *State1* (optional) Startzustand der Grafik entsprechend Tabelle 2. *State1* gibt den Zustand für *Grafik1* an, *State2* für *Grafik2* usw.
- Es können nur Zustände für tatsächlich angegebene Grafiken angegeben werden.
- *Beschriftung*: Ein statischer Beschriftungstext (erste Zeile).

Zugriff im Anwenderprogramm

- Die Grafik und der Text werden wie in Tabelle 1 erläutert über die Funktion pdisplay (S. 212) angesprochen. Die Umschaltung der Listbox (Vorgabe des aktiven Listboxelements) wird auch über diese Funktion angesprochen.
- Die Betätigung der Knöpfe ist über die Funktion mbutton (Seite 209) auszuwerten.

Erweitertes Wertediagramm (global)

Definition

- chart(*ID*){*X*,*Y*}[$*Y0*$,$*Y1*$,$*Y2*$]

Argumente

- ID: Wert zwischen 0 bis 127) als Index für die Programmierung und den Zugriff auf dieses Element. Sie können auch auf u08 Variablendefinitionen in der Sektion [EibPC] zugreifen.
- *X,Y:* Position im Raster oder absolut (u16), abhängig von der Konfiguration von responsive. Der gesamte {}-Block ist optional. Ohne Angabe erfolgt die Positionierung gemäß Positionierung.
- $*Y0*$, $*Y1*$,$*Y2*$: Achsenbeschriftung der Y-Achse.

Zugriff im Anwenderprogramm

- Die Y-Werte werden wie in Tabelle 1 erläutert im Anwenderprogramm über die Funktion chart (Seite 219) angesprochen.
- Es können Werte aus dem Bereich 1..30 dargestellt werden. Bei jedem Aufruf dieser Funktion werden zunächst die Werte von links beginnend dargestellt. Wenn nach 47 Aufrufen das Ende erreicht ist, werden die Werte nach links geschoben.

Erweitertes Wertediagramm (seitenbezogen)

Definition

- pchart(*ID*){*X*,*Y*}[$*Y0*$,$*Y1*$,$*Y2*$]

Argumente

- *ID*: Wert zwischen 0 bis 127) als Index für die Programmierung und den Zugriff auf dieses Element. Sie können auch auf u08 Variablendefinitionen in der Sektion [EibPC] zugreifen.
- *X,Y:* Position im Raster oder absolut (u16), abhängig von der Konfiguration von responsive. Der gesamte {}-Block ist optional. Ohne Angabe erfolgt die Positionierung gemäß Positionierung.
- *$Y0$, $Y1$,$Y2$*: Achsenbeschriftung der Y-Achse.

Zugriff im Anwenderprogramm

- Die Y-Werte werden wie in Tabelle 1 erläutert im Anwenderprogramm über die Funktion pchart (Seite 219) angesprochen.
- Es können Werte aus dem Bereich 1..30 dargestellt werden. Bei jedem Aufruf dieser Funktion werden zunächst die Werte von links beginnend dargestellt. Wenn nach 47 Aufrufen das Ende erreicht ist, werden die Werte nach links geschoben.

Wertediagramm mit mehreren Graphen (global)

Definition

- mchart(*ID*){*X*,*Y*}[*Format*,*Typ*]($*Beschriftung1*$,*Style1*, $*Beschriftung2*$,*Style2*, $*Beschriftung3*$,*Style3*, $*Beschriftung4*$,*Style4*)

Argumente

- *ID:* Wert zwischen 0 bis 127) als Index für die Programmierung und den Zugriff auf dieses Element.
- *Format:* SINGLE (2x2), DOUBLE (4x2), HALF (2x1), LONG (4x4)
- *Typ*: Wert 8 (oder Konstante XY): Es wird eine Ortskurve generiert.
- *Typ*: Wert 9 (oder Konstante SXY): Es wird eine X-Y Darstellung gewählt, welche die X-Y Wertepaare vorsortiert (für die Ausgabe von Zeitreihen, z.B: mit Zeitsprung bei 24:00).
- *$Beschriftung1$ .. $Beschriftung4$* Legende des entsprechenden Graphen
- *Style1, Style2, Style3, ,Style4*: Wert 0,1,2 oder 3 (oder Konstante LINE, DOTS, LINEDOTS, COLUMN).

Zugriff im Anwenderprogramm

- Die XY-Werte werden wie in Tabelle 1 erläutert im Anwenderprogramm über die gleichlautende Funktion mchart (Seite 220) angesprochen. Ein mchart verwaltet bis zu 4 XY-Diagramme. Die Anzahl der Diagramme wird durch die Anzahl der Argumente vorgegeben.
- Jedes XY-Diagramm hat eine Legende. Bei Darstellung von 4 XY-Diagramme werden im Diagramm 4 Legenden eingeblendet.
- Es werden 47 Fließkommawerte dargestellt. Die Skala wird automatisch generiert. Beachten Sie hier Hinweise bei der Beschreibung der Anwenderfunktion mchart() auf Seite 220.

Wertediagramm mit mehreren Graphen (seitenbezogen)

Definition

- mpchart(*ID*){*X*,*Y*}[*Höhe, Typ*]($*Beschriftung1*$,*Style1*, [$*Beschriftung2*$,*Style2*, [$*Beschriftung3*$,*Style3*, [$*Beschriftung4*$,*Style4*]]])

Argumente

- *ID* : Wert zwischen 0 bis 127) als Index für die Programmierung und den Zugriff auf dieses Element.
- *X,Y:* Position im Raster oder absolut (u16), abhängig von der Konfiguration von responsive. Der gesamte {}-Block ist optional. Ohne Angabe erfolgt die Positionierung gemäß Positionierung.
- *Höhe*: Wert 0, 1, 2 oder 3 (bzw. Konstante SINGLE, DOUBLE, HALF und LONG)
- *Typ*: Wert 8 (oder Konstante XY): Es wird eine Ortskurve generiert
- *Typ*: Wert 9 (oder Konstante SXY): Es wird eine X-Y Darstellung gewählt, welche die X-Y Wertepaare vorsortiert (für die Ausgabe von Zeitreihen, z.B: mit Zeitsprung bei 24:00)
- *$Beschriftung1$ .. $Beschriftung4$* Legende des entsprechenden Graphen
- *Style1, Style2, Style3, ,Style4*: Wert 0,1,2 oder 3 (oder Konstante LINE, DOTS, LINEDOTS, COLUMN).

Zugriff im Anwenderprogramm

- Die XY-Werte werden wie in Tabelle 1 erläutert im Anwenderprogramm über die gleichlautende Funktion mpchart (Seite 220) angesprochen. Ein mpchart verwaltet bis zu 4 XY-Diagramme. Die Anzahl der Diagramme wird durch die Anzahl der Argumente vorgegeben.
- Jedes XY-Diagramm hat eine Legende. Bei Darstellung von 4 XY-Diagramme werden im Diagramm 4 Legenden eingeblendet.
- Es werden 47 Fließkommawerte dargestellt. Die Skala wird automatisch generiert. Beachten Sie hier Hinweise bei der Beschreibung der Anwenderfunktion mpchart() auf Seite 220.

Zeitdiagramm (global)

Definition

- mtimechart(*ID*){*X*,*Y*}[*Format*,*Typ*,*Länge*,*YLMIN*,*YLMAX*,YRMIN,*YRMAX*] ($*Beschriftung1*$,*ChartPos1*,*Buffer1*,$*Beschriftung2*$,*ChartPos2*,*Buffer2*, $*Beschriftung3*$,*ChartPos3*,*Buffer3*,$*Beschriftung4*$,*ChartPos4*,*Buffer4*)

Argumente

- *ID:* Wert zwischen 0 bis 127) als Index für die Programmierung und den Zugriff auf dieses Element.
- *X,Y:* Position im Raster oder absolut (u16), abhängig von der Konfiguration von responsive. Der gesamte {}-Block ist optional. Ohne Angabe erfolgt die Positionierung gemäß Positionierung.
- *Format:* DOUBLE (2x2), TRIPLE (3x3), QUAD (4x2), LONG (4x4), EXTDOUBLE (3x2), EXTTRIPLE (4x2), EXTLONG (4x4)
- *Typ*: 0 für autoscale der linken Achse, in diesem Fall wird YLMAX etc. ignoriert (0=AUTOSCALELEFT)

  1 für autoscale der rechten Achse, in diesem Fall wird YRMAX etc. ignoriert (1=AUTOSCALERIGHT)

  2 für autoscale der beiden Achsen (2=AUTOSCALE)

  3 für kein autoscale (3=NOAUTOSCALE)
- *Länge:* Maximale Anzahl an Wertepaaren, die pro Graph angezeigt werden können (Mögliche Werte: von 32 bis 256)
- *YLMIN*: Minimalwert linke y-Achse, Fließkommazahlen
- *YLMAX*: Maximalwert linke y-Achse, Fließkommazahlen
- *YRMIN:* Minimalwert rechte y-Achse, Fließkommazahlen
- *YRMAX*: Maximalwert rechte y-Achse, Fließkommazahlen
- *$Beschriftung1$ .. $Beschriftung4$* Legende des entsprechenden Graphen
- *ChartPos1...ChartPos4*: Wert 0 (LEFTGRAF) oder 1 (RIGHTGRAF) (0 für Beschriftung an der linken y-Achse, 1 für Beschriftung an der rechten y-Achse) oder 2 (STACK) für grafisches Addieren von zwei Graphen: Die äußerste Einhüllende ist als die Gesamtsumme der Einzelgrafen zu verstehen:



- *Buffer:* ID des mit dem jeweiligen Graphen verknüpften TimeBuffers. Werte zwischen 0 bis 255 als Index für die Programmierung und den Zugriff.
- Um einen ordnungsgemäßen Betrieb zu garantieren, müssen die Buffer und mtimecharts so dimensioniert werden, dass der Speicher des EibPC nicht überlastet wird. Siehe hier unter timebufferconfig (S. 221) weitere Details.
- Die Formate EXTDOUBLE, EXTTRIPLE, EXTLONG sind Graphen mit intergrierter Zoom-, Verschiebungsfunktion und Zeitversatzeinstellung.

Zugriff im Anwenderprogramm

- Die XY-Werte werden im Anwenderprogramm über die Funktion timebufferadd S. 221 und timebufferconfig angesprochen. Ein mtimechart verwaltet bis zu 4 XY-Diagramme. Die Anzahl der Diagramme wird durch die Anzahl der Argumente vorgegeben.
- Jedes XY-Diagramm hat eine Legende. Bei Darstellung von 4 XY-Diagramme werden im Diagramm 4 Legenden eingeblendet.
- Es können bis zu 65535 Fließkommawerte dargestellt werden. Für die Skalierung beachten Sie hier Hinweise bei der Beschreibung der Anwenderfunktionen timebufferadd und timebufferconfig.
- mtimecharts sind immer global.

Farben der Graphen (seitenbezogen)

Definition

- timechartcolor *ID #HtmlFarbCode*

Argumente

- *ID*: Nummer des Graphen der timecharts , dessen Farbe geändert werden soll (1,2,3,4)
- *HtmlFarbCode*: HTML-Farbcodedefintion in Hex-Darstellung mit führendem # (vgl. https://wiki.selfhtml.org/wiki/Grafik/Farbpaletten)
- Diese Einstellung ist für alle Graphen auf einer Seite gültig und wird hinter ein page-Kommando gestellt.

Beispiel

```
[WebServer]
page (wsMeter) [$Smartmeter$, $Messung$
timechartcolor 1 #337755
timechartcolor 2 #e5a000
timechartcolor 3 #0066ff
timechartcolor 4 #ffff00
```

Langzeitaufzeichnung (global)

Definition

- historychart*(ID)*{*X*,*Y*}[*Größe, Beschriftung, StandardAnzeige*,*AchsenTyp*,*YLMIN*,*YLMAX*,*YLBeschriftung*,*YRMIN*,*YRMAX*,*YRBeschriftung*] ($*Beschriftung1*$,$*Einheit1*$,*ChartPos1*,*Buffer1*,$*Beschriftung2*$,$*Einheit2*$,*ChartPos2*,*Buffer2*, $*Beschriftung3*$,$*Einheit3*$,*ChartPos3*,*Buffer3*,$*Beschriftung4*$,$*Einheit4*$,*ChartPos4*,*Buffer4*)

Argumente

- *ID:* Wert zwischen 0 bis 127) als Index für die Programmierung und den Zugriff auf dieses Element.
- *X,Y:* Position im Raster oder absolut (u16), abhängig von der Konfiguration von responsive. Der gesamte {}-Block ist optional. Ohne Angabe erfolgt die Positionierung gemäß Positionierung.
- *Größe:* Breite x Höhe: beliebige Zahl für Höhe und Breite als Faktor der Einheitsgröße der Elemente des Webservers.
- *BeschriftungN* Überschrift des Charts, Längeu < 128 als $$ String
- *EinheitN* Einheit der Werte
- *StandardAnzeige:* Wählt die Anzeige auf der Seite. Wird das HistoryChart maximiert, kann dynamisch zwischen den Anzeigetypen gewechselt werden. Die gewählte *StandardAnzeige* wird dadurch nicht geändert.
  - 0: Aktueller Tag / Leistung
  - 1: Aktueller Tag
  - 2: Aktuelle Woche
  - 3: Aktueller Monat
  - 4: Aktuelles Jahr
  - 5: Gesamtübersicht

  Bei „Aktueller Tag“ sowie „Aktueller Tag / Leistung“ werden die Werte in der Seitenansicht stundenweise zusammengefasst. Dabei wird, abhängig vom Typ des HistoryBuffers, die Summe (Delta) oder der Mittelwert (Absolut) verwendet.
- *AchsenTyp*:
  - 0 für autoscale der linken Achse, in diesem Fall wird YLMAX etc. ignoriert (0=AUTOSCALELEFT)
  - 1 für autoscale der rechten Achse, in diesem Fall wird YRMAX etc. ignoriert (1=AUTOSCALERIGHT)
  - 2 für autoscale der beiden Achsen (2=AUTOSCALE)
  - 3 für kein autoscale (3=NOAUTOSCALE)
- *YLMIN*: Minimalwert linke y-Achse, Fließkommazahlen
- *YLMAX*: Maximalwert linke y-Achse, Fließkommazahlen
- *YLBeschriftung:* Achsenbeschriftung der linken y-Achse
- *YRMIN:* Minimalwert rechte y-Achse, Fließkommazahlen
- *YRMAX*: Maximalwert rechte y-Achse, Fließkommazahlen
- *YRBeschriftung:* Achsenbeschriftung der rechten y-Achse
- *$Beschriftung1$ .. $Beschriftung4$* Legende des entsprechenden Graphen
- *ChartPos*: Wert 0 (LEFTGRAF) oder 1 (RIGHTGRAF) (0 für Beschriftung an der linken y-Achse, 1 für Beschriftung an der rechten y-Achse) oder 2 (STACK) für grafisches Addieren von zwei Graphen, bezogen auf die linke y-Achse: Die äußerste Einhüllende ist als die Gesamtsumme der Einzelgrafen zu verstehen.
- *Buffer:* ID des mit dem jeweiligen Graphen verknüpften HistoryBuffers (siehe historybufferconfig).

Zugriff im Anwenderprogramm

- kein Zugriff

Enertex® Bayern GmbH – Ebermannstädter Straße 8 - 91301 Forchheim - mail@enertex.de

Bild (seitenbezogen)

Definition

- picture(*ID*){*X,Y*}[*Größe, Typ*]($Beschriftung$,URL)

Argumente

- *ID*: Wert zwischen 0 bis 127) als Index für die Programmierung und den Zugriff auf dieses Element.
- *X,Y:* Position im Raster oder absolut (u16), abhängig von der Konfiguration von responsive. Der gesamte {}-Block ist optional. Ohne Angabe erfolgt die Positionierung gemäß Positionierung.
- *Größe:* Wert 0, 1, 2 oder 3 (bzw. Konstante SINGLE, DOUBLE, HALF und LONG) bzw. Breite x Höhe: beliebige Zahl für Höhe und Breite als Faktor der Einheitsgröße der Elemente des Webservers.
- *Typ*: Wert 0,1,2 (oder LEFTGRAF, CENTERGRAF, ZOOMGRAF): Linksbündiges, mittiges oder gestrecktes Einbinden der Grafik.
- URL: Gültig WWW-Adresse (inkl. Pfad und führenden http://) zur externen Grafik

Zugriff im Anwenderprogramm

- Mit Hilfe der seitenbezogenen Funktion picture (S. 216) und dem dort spezifizierten Argumenten Beschriftung und URL kann die Beschriftung und der Link während der Laufzeit dynamisch verändert werden.

Einfacher Slider (global)

Definition

- slider(*ID*){*X,Y*}[*Grafik*][*State*]$*Beschriftung*$

Argumente

- *ID*: Wert zwischen 0 bis 127) als Index für die Programmierung und den Zugriff auf dieses Element. Sie können auch auf u08 Variablendefinitionen in der Sektion [EibPC] zugreifen.
- *X,Y:* Position im Raster oder absolut (u16), abhängig von der Konfiguration von responsive. Der gesamte {}-Block ist optional. Ohne Angabe erfolgt die Positionierung gemäß Positionierung.
- *Grafik*: Wert zwischen 0 und 127. Um die Anwendung übersichtlicher zu gestalten, sind vordefinierte Konstanten definiert (Seite 45).
- *State* (optional): Startzustand der Grafik entsprechend Tabelle 2.
- *Beschriftung*: Ein statischer Beschriftungstext (erste Zeile)

Zugriff im Anwenderprogramm

- Die Grafik und der Text werden wie in Tabelle 1 erläutert über die Funktion display (Seite 211) angesprochen.
- Die Betätigung des Sliders ist über die Funktion getslider (Seite 213) auszuwerten. 0 und 59
- Die Verstellung des Sliders ist über die Funktion setslider (Seite 214) anzustoßen.
- Die Betätigung des Knopfes ist über die Funktion button (Seite 209) auszuwerten.
- Der Wert kann auch direkt im numerischen Fenster des Sliders geändert werden.

Einfacher Slider (seitenbezogen)

Definition

- pslider(*ID*){*X,Y*}[*Grafik*][*State*]$*Beschriftung*$

Argumente

- *ID*: Wert zwischen 0 bis 127) als Index für die Programmierung und den Zugriff auf dieses Element. Sie können auch auf u08 Variablendefinitionen in der Sektion [EibPC] zugreifen.
- *X,Y:* Position im Raster oder absolut (u16), abhängig von der Konfiguration von responsive. Der gesamte {}-Block ist optional. Ohne Angabe erfolgt die Positionierung gemäß Positionierung.
- *Grafik*: Wert zwischen 0 und 127. Um die Anwendung übersichtlicher zu gestalten, sind vordefinierte Konstanten definiert (Seite 45).
- *State* (optional): Startzustand der Grafik entsprechend Tabelle 2.
- *Beschriftung*: Ein statischer Beschriftungstext (erste Zeile).

Zugriff im Anwenderprogramm

- Die Grafik und der Text werden wie in Tabelle 1 erläutert über die Funktion pdisplay (Seite 211) angesprochen.
- Die Betätigung des psliders ist über die Funktion getpslider (Seite 213) auszuwerten.
- Die Verstellung des pSliders ist über die Funktion setpslider (Seite 214) anzustoßen.
- Die Betätigung des Knopfes ist über die Funktion pbutton (Seite 209) auszuwerten.

Enertex® Bayern GmbH – Ebermannstädter Straße 8 - 91301 Forchheim - mail@enertex.de

Erweiterter Slider (global)

Definition

- eslider(*ID*){*X*,*Y*}[*Grafik*][*State*] (*Min*,*Inkrement*, *Max*) $*Beschriftung*$ $*Label*$

Argumente

- *ID*: Wert zwischen 0 bis 127) als Index für die Programmierung und den Zugriff auf dieses Element. Sie können auch auf u08 Variablendefinitionen in der Sektion [EibPC] zugreifen.
- *X,Y:* Position im Raster oder absolut (u16), abhängig von der Konfiguration von responsive. Der gesamte {}-Block ist optional. Ohne Angabe erfolgt die Positionierung gemäß Positionierung.
- *Grafik*: Wert zwischen 0 und 127. Um die Anwendung übersichtlicher zu gestalten, sind vordefinierte Konstanten definiert (Seite 45).
- *State* (optional): Startzustand der Grafik entsprechend Tabelle 2.
- *Beschriftung*: Ein statischer Beschriftungstext (erste Zeile)
- *Min*: Minimalwert für den eslider
- *Inkrement*: Inkrement für den eslider
- *Max*: Maximalwert für den eslider
- *Label*: Label für die Werteanzeige, max. 3 Stellen

Zugriff im Anwenderprogramm

- Die Grafik und der Text werden, wie in Tabelle 1 erläutert, über die Funktion seteslider angesprochen.
- Die Betätigung des esliders ist über die Funktion geteslider auszuwerten.
- Die Verstellung des esliders ist über die Funktion seteslider anzustoßen.
- Die Betätigung des Knopfes ist über die Funktion webbutton auszuwerten.
- Der Wert kann auch direkt im numerischen Fenster des Sliders geändert werden.

Erweiterter Slider (seitenbezogen)

Definition

- peslider(*ID*){*X*,*Y*}[*Grafik*][*State*] (*Min*,*Inkrement*, *Max*) $*Beschriftung*$ $*Label*$

Argumente

- *ID*: Wert zwischen 0 bis 127) als Index für die Programmierung und den Zugriff auf dieses Element. Sie können auch auf u08 Variablendefinitionen in der Sektion [EibPC] zugreifen.
- *X,Y:* Position im Raster oder absolut (u16), abhängig von der Konfiguration von responsive. Der gesamte {}-Block ist optional. Ohne Angabe erfolgt die Positionierung gemäß Positionierung.
- *Grafik*: Wert zwischen 0 und 127. Um die Anwendung übersichtlicher zu gestalten, sind vordefinierte Konstanten definiert (Seite 45).
- *State* (optional): Startzustand der Grafik entsprechend Tabelle 2.
- *Beschriftung*: Ein statischer Beschriftungstext (erste Zeile).
- *Min*: Minimalwert für den peslider
- *Inkrement*: Inkrement für den peslider
- *Max*: Maximalwert für den peslider
- *Label*: Label für die Werteanzeige, max. 3 Stellen

Zugriff im Anwenderprogramm

- Die Grafik und der Text werden wie in Tabelle 1 erläutert über die Funktion setpeslider angesprochen.
- Die Betätigung des Sliders ist über die Funktion getpeslider auszuwerten.
- Die Verstellung des Sliders ist über die Funktion setpeslider anzustoßen.
- Die Betätigung des Knopfes ist über die Funktion pbutton auszuwerten.
- Der Wert kann auch direkt im numerischen Fenster des Sliders geändert werden.

Eingabe für Text, Datum/Zeit, Farbe (global)

Definition

- webinput(*ID*){*X*,*Y*}[*Grafik, Stil*][*State*] $*Beschriftung*$

Argumente

- *ID*: Wert zwischen 0 bis 127) als Index für die Programmierung und den Zugriff auf dieses Element. Sie können auch auf u08 Variablendefinitionen in der Sektion [EibPC] zugreifen.
- *X,Y:* Position im Raster oder absolut (u16), abhängig von der Konfiguration von responsive. Der gesamte {}-Block ist optional. Ohne Angabe erfolgt die Positionierung gemäß Positionierung.
- *Grafik*: Wert zwischen 0 und 127. Um die Anwendung übersichtlicher zu gestalten, sind vordefinierte Konstanten definiert (Seite 45).
- *Beschriftung*: Ein statischer Beschriftungstext
- Stil ist optional. Mögliche Optionen:
  - *keine*: Einfache Texteingabe. Die Ausgabe mit webinput erfolgt als String.
  - PASSWORD (1): In diesem Fall wird die Eingabe mit Sternchen oder vom Webbrowser vorgegebene Zeichen versteckt.
  - COLORPICK (2): Auswahl einer RGB Farbe. Die Ausgabe mit webinput erfolgt als String als 24-Bit Hexadezimal-Zahl in der Darstellung 0xRRGGBB mit RR, GG, BB jew. zwischen 00-ff.
  - DATEPICK (3): Die Eingabe eines Datums mit Hilfe eines Standarddialogs (Darstellung abhängig vom Webbrowser). Die Ausgabe mit webinput erfolgt als String in der Darstellung $YYYY-MM-DD$
  - TIMEPICK (4): Die Eingabe einer Uhrzeit mit Hilfe eines Standarddialogs (Darstellung abhängig vom Webbrowser). Die Ausgabe mit webinput erfolgt als String in der Darstellung $HH-MM-SS$
  - TWPICK (5): Auswahl eines Tunable White-Wertes als Prozentwert zwischen Warmweiß (0%) und Kaltweiß (100%) und einer prozentualen Helligkeit. Die Ausgabe mit webinput erfolgt als String als 16-Bit Hexadezimal-Zahl in der Darstellung 0xBBCC mit BB als Helligkeit und CC als Farbtemperatur, jew. zwischen 00-ff.
  - INTEGERPICK (6): Auswahl einer Ganzzahl.
  - DECIMALPICK (7): Auswahl einer Zahl mit Dezimalpunkt.
- *State* (optional): Startzustand der Grafik entsprechend Tabelle 2.

Zugriff im Anwenderprogramm

- Der eingegebene Wert wird über die Funktion webinput (S. 225) abgefragt.
- Die Grafik und der Text werden über die Funktion setwebinput (Seite 225) angesprochen.

Freier Ausgabebereich (global)

Definition

- weboutput(*ID*){*X*,*Y*}[*Größe*,*Stil*]

Argumente

- *ID*: Wert zwischen 0 bis 127) als Index für die Programmierung und den Zugriff auf dieses Element. Sie können auch auf u08 Variablendefinitionen in der Sektion [EibPC] zugreifen.
- *X,Y:* Position im Raster oder absolut (u16), abhängig von der Konfiguration von responsive. Der gesamte {}-Block ist optional. Ohne Angabe erfolgt die Positionierung gemäß Positionierung.
- *Größe*: Wert 0, 1, 2...5 (bzw. Konstante SINGLE, DOUBLE und QUAD, bzw.
  Breite x Höhe: beliebige Zahl für Höhe und Breite als Faktor der Einheitsgröße der Elemente des Webservers.
- *Stil*:
  - 0u08, bzw. ICON: Rahmen, Hintergrund, Symbol
  - 1u08, bzw. NOICON: Rahmen, Hintergrund
  - 2u08, bzw. NOCOLOR: Rahmen
  - 3u08: keine optische Hervorhebung. Geeignet um beliebige Texte/Zustände anzuzeigen.

Zugriff im Anwenderprogramm

- Das Element wird über die Funktion weboutput (S. 226) angesprochen.
- Weboutputelemente sind immer global.

Interner Link (seitenbezogen)

Definition

- plink(*ID*){*X*,*Y*}[*Grafik*][*State*][*PageID*] $*Text*$

Argumente

- *ID*: Wert zwischen 0 bis 127) als Index für die Programmierung und den Zugriff auf dieses Element. Sie können auch auf u08 Variablendefinitionen in der Sektion [EibPC] zugreifen. (Grafisch ist das Element zum button identisch)
- *X,Y:* Position im Raster oder absolut (u16), abhängig von der Konfiguration von responsive. Der gesamte {}-Block ist optional. Ohne Angabe erfolgt die Positionierung gemäß Positionierung.
- *Grafik*: Wert zwischen 0 und 127. Um die Anwendung übersichtlicher zu gestalten, sind vordefinierte Konstanten definiert (Seite 45).
- *State* (optional): Startzustand der Grafik entsprechend Tabelle 2.
- *PageID*: Wert von 1 bis 100 als Sprung-Index der Seite, auf welche bei Knopfdruck gesprungen werden soll. Sie können auch auf u08 Variablendefinitionen in der Sektion [EibPC] zugreifen.
- Text: Ein dynamischer Beschriftungstext (erste Zeile).

Zugriff im Anwenderprogramm

- Grafik, Seitenverweis und Text können über die Funktion plink (S. 217) verändert werden.

Externer Link (seitenbezogen)

Definition

- link(*ID*){*X*,*Y*}[*Grafik*][*State*][$*Website*$] $*Text*$

Argumente

- *ID*: Wert zwischen 0 bis 127) als Index für die Programmierung und den Zugriff auf dieses Element. Sie können auch auf u08 Variablendefinitionen in der Sektion [EibPC] zugreifen. (Grafisch ist das Element zum button identisch)
- *X,Y:* Position im Raster oder absolut (u16), abhängig von der Konfiguration von responsive. Der gesamte {}-Block ist optional. Ohne Angabe erfolgt die Positionierung gemäß Positionierung.
- *Grafik*: Wert zwischen 0 und 127. Um die Anwendung übersichtlicher zu gestalten, sind vordefinierte Konstanten definiert (Seite 45).
- *State* (optional): Startzustand der Grafik entsprechend Tabelle 2.
- $*Website*$ http-Adresse (inkl. Pfad und führenden http://) des Ziels
- *Text*: Ein dynamischer Beschriftungstext (erste Zeile).

Zugriff im Anwenderprogramm

- Mit Hilfe der Seitenbezogenen Funktion link (S. 217) und den dort spezifizierten Argumenten *Text, Icon, Website* können diese Argumente während der Laufzeit dynamisch verändert werden.

Externe Seite einbetten

Definition

- frame {*X*,*Y*}[$*URL*$]

Argumente

- *X,Y:* Position im Raster oder absolut (u16), abhängig von der Konfiguration von responsive. Der gesamte {}-Block ist optional. Ohne Angabe erfolgt die Positionierung gemäß Positionierung.
- *URL*: Eine WWW-Adresse (inkl. Pfad und führenden http://) auf eine externe Seite, die in den Webserver integriert wird.

Zugriff im Anwenderprogramm

- keinen

Definition

- dframe {*X*,*Y*}[$*URL*$]

Argumente

- *X,Y:* Position im Raster oder absolut (u16), abhängig von der Konfiguration von responsive. Der gesamte {}-Block ist optional. Ohne Angabe erfolgt die Positionierung gemäß Positionierung.
- *URL*: Eine WWW-Adresse (inkl. Pfad und führenden http://) auf eine externe Seite, die in den Webserver integriert wird. Das eingebettte Fenster ist doppelt so hoch, wie das von frame.

Zugriff im Anwenderprogramm

- keinen

Definition

- fframe {*X*,*Y*}[*Größe*,$*URL*$]

Argumente

- *X,Y:* Position im Raster oder absolut (u16), abhängig von der Konfiguration von responsive. Der gesamte {}-Block ist optional. Ohne Angabe erfolgt die Positionierung gemäß Positionierung.
- *Größe:* Breite x Höhe: beliebige Zahl für Höhe und Breite als Faktor der Einheitsgröße der Elemente des Webservers.
- *URL*: Eine WWW-Adresse (inkl. Pfad und führenden http://) auf eine externe Seite, die in den Webserver integriert wird.

Zugriff im Anwenderprogramm

- keinen

Schaltuhr

Definition

- timeswitch{*X*,*Y*}[*Größe*,*Stil*][*Grafik*][*State*] ($*Text*$, *TimeSwitchID*)

Argumente

- *X,Y:* Position im Raster oder absolut (u16), abhängig von der Konfiguration von responsive. Der gesamte {}-Block ist optional. Ohne Angabe erfolgt die Positionierung gemäß Positionierung.
- *Größe*: Wert 0, 1, 2...5 (bzw. Konstante SINGLE, DOUBLE und QUAD, bzw.
  Breite x Höhe: beliebige Zahl für Höhe und Breite als Faktor der Einheitsgröße der Elemente des Webservers.
- *Stil*: Art der Anzeige:
  - 0u08: bedienbar, Anzeige des nächsten Schaltzeitpunktes
  - 1u08: bedienbar, Anzeige aller Schaltzeitpunkte
  - 2u08: nicht bedienbar, Anzeige des nächsten Schaltzeitpunktes
  - 3u08: nicht bedienbar, Anzeige aller Schaltzeitpunkte
- *Grafik*: Wert zwischen 0 und 127. Um die Anwendung übersichtlicher zu gestalten, sind vordefinierte Konstanten definiert (Seite 45).
- *State:* Startzustand der Grafik entsprechend Tabelle 2.
- *Text*: Ein dynamischer Beschriftungstext (erste Zeile).
- *TimeSwitchID*: ID der Schaltuhr, in timeswitch vergeben.

Zugriff im Anwenderprogramm

- keinen

Funktionen

Logische Verknüpfungen

Dieses Kapitel ist relevant, wenn Sie eigene Expertenprogramme oder Makros erstellen.

Für alle Argumente von Funktionen können anstelle von Variablen auch direkt Gruppenadressen verwendet werden.

Bitweise Und

Definition

- *A* and *B* [and *C* ... usw.]

Argumente

- Alle Argumente (*A, B, C* ...) vom gleichen Datentyp. Datentypen aber sonst beliebig.
- Beliebig viele Verknüpfungen

Wirkung

- Die Variable *A* wird bitweise mit der Variablen *B* (und der Variablen *C* usw.) „verundet“: Das Ergebnis der Operation and ist null (alle Bits), wenn eine der Variablen null (alle Bits) ist. Im anderen Fall ist das Ergebnis eine bitweise Verundung, d.h. das n-te Bit des Ergebnisses ist 0, alsbald eines der Bits der Eingangsvariablen Null ist. Ansonsten ist das n-te Bit des Ergebnisses 1, d.h. die jeweils n-ten Bits der zwei (oder mehr) Eingangsvariablen sind 1.

Rückgabewert

- Datentyp wie Argumente

Beispiel: Und-Verknüpfung

LichtAktorEin ist das Ergebnis der Und-Verknüpfung von Variable TasterEin und Variable LichtFreigabe

Die Umsetzung im Anwenderprogramm lautet dann:

```
LichtAktorEin = TasterEin and LichtFreigabe
```

Wenn *TasterEin* 1b01 ist und *LichtFreigabe* 1b01 ist, dann ist *LichtAktorEin* auf 1b01, sonst 0b01.

Beispiel: Und-Verknüpfung bei unterschiedlichen Variablen

Wenn die Variable TasterEin auf '1' und die Variable Windgeschwindigkeit genau 2.9 m/s ist, soll die Variable LichtAktorEin auf '1' gesetzt werden.

Für die Umsetzung benötigen wir die „if“ - Anweisung und den Vergleich „==“.

```
if TasterEin==1u08 and Windgeschwindigkeit==2.9f16 then LichtAktorEin=1u08 endif
```

Bitweise Oder

Definition

- *A* or *B* [or *C* ... usw.]

Argumente

- Alle Argumente (*A, B, C* ...) vom gleichen Datentyp. Datentypen aber sonst beliebig.
- Beliebig viele Verknüpfungen

Wirkung

- Die Variable *A* wird bitweise der Variablen *B* (und der Variablen *C* usw.) „verodert“, was heißt: Das Ergebnis der Operation or ist null, wenn beide der Variablen null sind. Im anderen Fall ist das Ergebnis eine bitweise Veroderung , d.h. das n-te Bit des Ergebnisses ist eins, alsbald eines der Bits der Eingangsvariablen eins ist.

Rückgabewert

- Datentyp wie Argumente

Beispiel: Oder-Verknüpfung

LichtAktorEin ist das Ergebnis der Oder-Verknüpfung von Variable TasterEin und Variable LichtFreigabe

Die Umsetzung lautet dann:

```
LichtAktorEin = TasterEin or LichtFreigabe
```

Wenn *TasterEin* 1b01 ist oder *LichtFreigabe* 1b01 oder beide 1b01 sind, dann ist *LichtAktorEin* auf 1b01, sonst 0b01.

Beispiel: Oder-Verknüpfung bei unterschiedlichen Variablentypen

Wenn die Variable TasterEin auf '1' oder die Variable Windgeschwindigkeit genau 2.9 m/s ist, soll die Variable LichtAktorEin auf '1' gesetzt werden.

Für die Umsetzung benötigen wir die „if“ - Anweisung und den Vergleich „==“. Dabei ist die gesamte if-Abfrage in Klammern zu setzen. Die Umsetzung in der EibParserdatei lautet dann:

```
if ((TasterEin==1u08) or (Windgeschwindigkeit==2.9f16)) then LichtAktorEin=1u08 endif
```

Bitweise Exklusiv-Oder

Definition

- *A* xor *B* [xor *C* ... usw.]

Argumente

- Alle Argumente (*A, B, C* ...) vom gleichen Datentyp. Datentypen aber sonst beliebig.
- Beliebig viele Verknüpfungen.

Wirkung

- Die Variable *A* wird bitweise der Variablen *B* (und der Variablen *C* usw.) „ver-x-odert“, was heißt: Das Ergebnis der Operation xor ist null (bitweise), wenn beide Variablen null oder eins sind. Im anderen Fall ist das n-te Bit des Ergebnisses eins, alsbald **nur** eines der Bits der Eingangsvariablen eins ist.

Rückgabewert

- Datentyp wie Argumente

Beispiel:

Wenn entweder Taste1 (Typ b01) oder Taste2 (Typ b01) gedrückt ist, soll LichtAktorEin auf 1b01 gehen. Wenn beide auf 0b01 oder 1b01 stehen, soll LichtAktorEin auf 0b01 gehen

Die Umsetzung lautet dann:

```
LichtAktorEin = Taste1 xor Taste2
```

Vergleichsoperatoren

Zum Erstellen von Vergleichs-Verknüpfungen sind folgende Anweisungen vorgesehen.

Definition

- *A* > *B* größer
- *A* < *B* *kleiner*
- *A* == *B* *gleich*
- *A* >= *B* *größer gleich*
- *A* <= *B* *kleiner gleich*
- *A* != *B* *nicht gleich*

Argumente

- *A, B* (u, s, f) vom gleichen Datentyp

Wirkung

- Die Variable *A* wird mit der Variablen *B* – je nach Operator – verglichen:
- Das Ergebnis der Operation > ist 1b01, wenn die Variable A größer als die Variable B ist.
- Das Ergebnis der Operation < ist 1b01, wenn die Variable A kleiner als die Variable B ist.
- Das Ergebnis der Operation == ist 1b01, wenn die Variable A den selben Wert hat wie die Variable B.
- Das Ergebnis der Operation >= ist 1b01, wenn die Variable A größer gleich der Variable B ist.
- Das Ergebnis der Operation <= ist 1b01, wenn die Variable A kleiner gleich der Variable B ist.
- Das Ergebnis der Operation != ist 1b01, wenn die Variable A nicht den selben Wert wie die Variabel B hat.
- In allen anderen Fällen ist das Ergebnis 0b01.

Rückgabewert

- Datentyp b01

Hysterese

Definition

- hysteresis(*Var*,*UnteresLimit*,*OberesLimit*)

Argumente

- Argumente (*Var*,*UnteresLimit*,*OberesLimit*) vom gleichen Datentyp.
- Datentypen: uXX, sXX, fXX mit XX beliebigen, auf Seite 27 definierten Bitlängen.

Wirkung

- Das Argument *Var* wird mit den Variablen *UnteresLimit* und *OberesLimit* einer Hysterese-funktion verglichen.
- Hat der letzte Vergleich zu einem Ergebnis 0b01 geführt und gilt (*Var*≥*OberesLimit*), so nimmt die Funktion den Wert 1b01 an.
- Hat der letzte Vergleich zu einem Ergebnis 1b01 geführt und gilt (*Var*≥*UnteresLimit*), so nimmt die Funktion den Wert 0b01 an.

Rückgabewert

- Datentyp b01

Beispiel: Temperaturgeregelte Beschattung

Wenn ein Temperaturaktor wärmer als 25°C (Gruppenadresse 1/3/4, Datentyp f16) meldet soll die Beschattung auf Gruppenadresse 4/5/77 auf EIN fahren.
Erst wenn die Temperatur wieder unter 23°C fällt, soll die Beschattung wieder hochfahren.

Umsetzung im Anwenderprogramm:

```
if hysteresis('1/3/4'f16,23f16,25f16) then write('4/5/77'b01,EIN) \\
                                      else write('4/5/77'b01,AUS) endif
```

Invertierung

Definition

- *!A*

Argumente

- Argument *A* vom Datentyp b01

Wirkung

- Die Variable *A* wird invertiert.
- Das Ergebnis der Operation ist 1b01, wenn die Variable A 0b01 ist
- Das Ergebnis der Operation ist 0b01, wenn die Variable A 1b01 ist

Rückgabewert

- Datentyp b01

Beispiel: Invertierter Taster

LichtAktorEin (b01) soll sich invers zu Taste1 (b01) verhalten.

Die Umsetzung lautet dann:

```
LichtAktorEin = !Taste1
```

Wenn *Taste1* auf 1b01 steht, so ist *LichtAktorEin* auf 0b01. Wenn *Taste1* auf 0b01 steht, so ist *LichtAktorEin* auf 1b01.

Bitweise Schieben

Definition

- shift*(Operand, Zahl)*

Argumente

- Argument *Operand* von einem beliebigen numerischen Datentyp
- Argument *Zahl* vom Datentyp s08

Wirkung

- arithmetische Verschiebung des Operanden um *Zahl*. Bei positiver Zahl Shift nach links, bei negativer Zahl nach rechts. Geshiftet werden die Anzahl Bits der Zahl des Eingangs.

Rückgabewert

- wie *Operand*

Enertex® Bayern GmbH – Ebermannstädter Straße 8 - 91301 Forchheim - mail@enertex.de

Zeitfunktionen

EibPC-Zeit setzen

Definition

- Funktion gettime(*Adresse*) mit:

Argumente

- Ein Argument vom Datentyp t24

Wirkung

- Die Systemuhr des EibPC wird mit der in *Adresse* gespeicherten Zeit überschrieben und somit neu gesetzt.

Rückgabewert

- keine

Hinweise:

1. Es ist keine Zuweisung der Form *a*=gettime(*b*) möglich (Fehlermeldung).
2. Die Funktion wird nur ausgeführt, wenn sie im then oder else Zweig einer if-Anweisung steht.

Beispiel: gettime

Wöchentlich am Sonntag um 00:00 Uhr soll die Systemuhr mit einer im KNX-Bus vorhandenen Funkuhr abgeglichen und neu gesetzt werden.

Umsetzung im Anwenderprogramm:

```
if(cwtime(0,0,0,0)) then read(“Funkuhr-1/2/1“) endif
if event(“Funkuhr-1/2/1“) then gettime(“Funkuhr-1/2/1“) endif
```

Durch die read-Funktion wird eine Leseanforderung an die Gruppenadresse generiert. Die Information, die daraufhin auf den KNX Bus gesendet wird, wird durch die gettime-Funktion in die Systemuhr des EibPC geschrieben.

EibPC-Zeit senden

Definition

- Funktion settime()

Argumente

- keines

Wirkung

- Die Systemzeit wird aus dem EibPC gelesen und einer Variablen als Wert zugewiesen. Rückgabewert ist die aktuelle Uhrzeit im DPT-Format.

Rückgabewert

- Datentyp t24

Beispiel: settime

Am 1. jeden Monats soll die Gruppenadresse "Wanduhr-4/3/5" und die Variable Uhrzeit mit der Systemuhr abgeglichen (und damit neu gesetzt) werden.

Umsetzung im Anwenderprogramm:

```
if (day(1) and !day(2)) then write(„Wanduhr-4/3/5“t24,settime()) endif
if (day(1) and !day(2)) then Uhrzeit=settime() endif
```

EibPC-Datum setzen

Definition

- Funktion getdate(*Adresse*) mit:

Argumente

- Ein Argument vom Datentyp d24.

Wirkung

- Die Systemuhr des EibPC wird mit der in *Adresse* gespeicherten Zeit überschrieben und somit neu gesetzt.

Rückgabewert

- keiner

Hinweise:

1. Es ist keine Zuweisung der Form *a*=gettime(*b*) möglich (Fehlermeldung).
2. Die Funktion wird nur ausgeführt, wenn sie im then oder else Zweig einer if-Anweisung steht.

Beispiel: GetDate

Alle sechs Monate soll das Systemdatum mit einer im KNX Bus vorhandenen Funkuhr abgeglichen und neu gesetzt werden.

Umsetzung im Anwenderprogramm:

```
if (month(1,1) or month(1,7)) then read(“Funkuhr-1/2/2“) endif
if event(“Funkuhr-1/2/2“) then getdate(“Funkuhr-1/2/2“) endif
```

EibPC-Datum senden

Definition

- Funktion setdate()

Argumente

- keines

Wirkung

- Das Systemdatum wird aus dem EibPC gelesen. Der Rückgabewert ist die Zeit im Format des Typ d24.

Rückgabewert

- Datentyp d24

Beispiel: SetDate

Am 1.1. jeden Jahres soll die Adresse “Datum-3/5/3“ mit dem Datum des EibPC abgeglichen und neu gesetzt werden.

Umsetzung im Anwenderprogramm:

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


Enertex® Bayern GmbH – Ebermannstädter Straße 8 - 91301 Forchheim - mail@enertex.de

EibPC-Zeit und -Datum setzen

Definition

- Funktion gettimedate(*Adresse*) mit:

Argumente

- Ein Argument vom Datentyp y64

Wirkung

- Die Systemuhr und das Systemdatum des EibPC werden mit der in *Adresse* gespeicherten Zeit und dem Datum überschrieben und somit neu gesetzt.

Rückgabewert

- keine

Hinweise:

1. Es ist keine Zuweisung der Form *a*=gettimedate(*b*) möglich (Fehlermeldung)
2. Die Funktion wird nur ausgeführt, wenn sie im then oder else Zweig einer if-Anweisung steht.

Beispiel: GetTimeDate

Alle sechs Monate sollen die Systemzeit und das Systemdatum mit einer im KNX Bus vorhandenen Funkuhr abgeglichen und neu gesetzt werden.

Umsetzung im Anwenderprogramm:

```
if (month(1,1) or month(1,7)) then read(“Funkuhr-1/2/3“) endif
if event(“Funkuhr-1/2/3“) then gettimedate(“Funkuhr-1/2/3“) endif
```

EibPC-Zeit und -Datum senden

Definition

- Funktion settimedate()

Argumente

- keines

Wirkung

- Die Systemzeit und das Systemdatum werden aus dem EibPC gelesen und einer Variable als Wert zugewiesen.

Rückgabewert

- Datentyp y64

Beispiel: SetDate

Am 1.1. jeden Jahres soll die Adresse “Funkuhr-1/2/1“ mit der Systemzeit und dem Systemdatum des EibPC abgeglichen und neu gesetzt werden.

Umsetzung im Anwenderprogramm:

```
if (month(1,1)) then write(“Funkuhr-1/2/1“y64, settimedate()) endif
```

Aktuelle Stunde

Definition

- Funktion hour()

Argumente

- keines

Wirkung

- Die Systemzeit (Stunde) wird in eine Variable gespeichert

Rückgabewert

- Datentyp u08

Beispiel:

Stoppuhr siehe S. 115

Aktuelle Minute

Definition
- Funktion minute()

Argumente
- keines

Wirkung
- Die Systemzeit (Minute) wird in eine Variable gespeichert

Rückgabewert
- Datentyp u08

Beispiel:

Stoppuhr siehe S. 115

Aktuelle Sekunde

Definition
- Funktion second()

Argumente
- keines

Wirkung
- Die Systemzeit (Sekunde) wird in eine Variable gespeichert

Rückgabewert
- Datentyp u08

Beispiel:Stoppuhr

Stoppen der Zeit in Sekunden, an welcher die Variable Stopper_Go auf ein steht. Es soll ein c1400 String angegeben werden, der die Zeit Formatiert in 000d:000h:000m:000s (Tage, Stunden, Minuten, Sekunden) ausgibt.

Hier die Umsetzung, wobei die Sekunden in der Variablen *Stopper_time* und die formatierte Ausgabe in *Stopper* zu finden sind. Vgl. Sie hierzu auch das Beispiel Stoppuhr V2 auf S. 161.

Stringformat für die formatierte Ausgabe/Umwandlung

```
[EibPC]
Stopper=$$
Stopper_start=0s32
Stopper_time=1s32
Stopper_Go=AUS

// Starte Stoppuhr (Offset berechnen)
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

// Ende Stoppzeit
if !Stopper_Go then {

Stopper_time=convert(hour(),0s32)*3600s32+convert(minute(),0s32)*60s32+convert(second(),0s32)+Stopp
er_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
```

Stunde ändern

Definition

- Funktion changehour(*Arg*)

Argumente

- *Arg*, Datentyp u08

Wirkung

- Die Systemzeit (Stunde) wird auf den Wert von *Arg* gesetzt.
- Beachten Sie, dass durch das Setzen bzw. Verändern der Systemzeit die Timerfunktionen in ihrem Ablauf gestört werden können.
- Sollte ihr EibPC eine ntp-Verbindung aufbauen können, so wird diese die Zeit wieder zurücksetzen (vgl. Hinweis auf S. 23).

Rückgabewert

- keine

Minute ändern

Definition

- Funktion changeminute(*Arg*)

Argumente

- *Arg*, Datentyp u08

Wirkung

- Die Systemzeit (Minute) wird auf den Wert von *Arg* gesetzt.
- Beachten Sie, dass durch das Setzen bzw. Verändern der Systemzeit die Timerfunktionen in ihrem Ablauf gestört werden können.
- Sollte ihr EibPC eine ntp-Verbindung aufbauen können, so wird diese die Zeit wieder zurücksetzen (vgl. Hinweis auf S. 23).

Rückgabewert

- keine

Sekunde ändern

Definition

- Funktion changesecond(*Arg*)

Argumente

- *Arg*, Datentyp u08

Wirkung

- Die Systemzeit (Sekunde) wird auf den Wert von *Arg* gesetzt.
- Beachten Sie, dass durch das Setzen bzw. Verändern der Systemzeit die Timerfunktionen in ihrem Ablauf gestört werden können.
- Sollte ihr EibPC eine ntp-Verbindung aufbauen können, so wird diese die Zeit wieder zurücksetzen (vgl. Hinweis auf S. 23).

Rückgabewert

- keine

String in Unixzeit (UTC)

Definition

- Funktion utc(*Zeit*)

Argumente

- *Zeit* (c) im Format YYYY-MM-DD HH:MM:SS

Wirkung

- Vergangene Zeit seit dem 1. Januar 1970, 00:00:00 Uhr ohne Schaltsekunden (Unixzeit) bis *Zeit* in Millisekunden (UTC).

Rückgabewert (u64)

Aktuelle Zeit in Unixzeit (UTC)

Definition

- Funktion utctime()

Argumente

- keine

Wirkung

- Zeit seit dem 1. Januar 1970, 00:00:00 Uhr ohne Schaltsekunden (Unixzeit) in Millisekunden (UTC).

Rückgabewert (u64)

Unixzeit in String (UTC)

Definition

- Funktion utcconvert(*Unixzeit*)

Argumente

- *Unixzeit* (u64)

Wirkung

- Konvertiert *Unixzeit* (Zeit seit dem 1. Januar 1970, 00:00:00 Uhr ohne Schaltsekunden) in Millisekunden in einen String (UTC).

Rückgabewert (c1400)

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

Beispiel:

```
// Aktuelle Zeit als Unixzeit (UTC)
unixtime=utctime()

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

// Wandelt 2012-09-03 20:00:17 in Unixzeit (UTC) um. Ergebnis: 1346702417000
utcZ=utc($2012-09-03 20:00:17$)

// Tage im Februar - Schaltjahr?
uTageFeb2020=(utc($2020-03-01 00:00:00$) - utc($2020-02-01 00:00:00$))/(24u64*3600u64*1000u64)
uTageFeb2019=(utc($2019-03-01 00:00:00$) - utc($2019-02-01 00:00:00$))/(24u64*3600u64*1000u64)
```

String in Unixzeit (Lokalzeit)

Definition

- Funktion localtime(*Zeit*)

Argumente

- *Zeit* (c) im Format YYYY-MM-DD HH:MM:SS

Wirkung

- Vergangene Zeit seit dem 1. Januar 1970, 00:00:00 Uhr ohne Schaltsekunden (Unixzeit) bis *Zeit* in Millisekunden (Lokalzeit).

Rückgabewert (u64)

Beispiel:

```
// Anzahl Tage seit einem bestimmten Datum
uStart=utc($2023-07-31 23:59:00$)
uNow=localtime($2023-09-11 00:00:00$)
uDelta = uNow-uStart
uDeltaDays=uDelta/86400000u64

uStartLocal = localtime($2023-07-31 23:59:00$)
uDeltaLocal = uNow-uStartLocal
uDeltaDaysLocal=uDeltaLocal/86400000u64

uDayOffsetLocal=uDeltaLocal-(uDeltaDaysLocal*86400000u64)
uHoursOffset=mod(uDayOffsetLocal, 3600000u64)/60000u64
uMinOffset=mod(uDayOffsetLocal, 60000u64)

// uDeltaDays: 40
// uDeltaDaysLocal: 41
```

Unixzeit in String (Lokalzeit)

Definition

- Funktion localtimeconvert(*Unixzeit*)

Argumente

- *Unixzeit* (u64)

Wirkung

- Konvertiert *Unixzeit* (Zeit seit dem 1. Januar 1970, 00:00:00 Uhr ohne Schaltsekunden) in Millisekunden in einen String (Lokalzeit).

Rückgabewert (c1400)

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

Beispiel:

```
// Gestern zur gleichen Zeit
now=utctime()
gesternLokal=localtimeconvert(now-(24u64*3600000u64))
```

Zeitdifferenz zwischen Lokalzeit und UTC

Definition

- Funktion difftime()

Argumente

- keine

Wirkung

- Abweichung der Lokalzeit von der UTC-Zeit in Millisekunden. Entspricht der Zeitverschiebung durch die gewählte Zeitzone (und ggf. Sommerzeit) in Bezug auf UTC. Für Zentraleuropa bspw. UTC+1, also +3600000s64 (MEZ) bzw. +7200000s64 (MESZ).

Rückgabewert (s64)

Enertex® Bayern GmbH – Ebermannstädter Straße 8 - 91301 Forchheim - mail@enertex.de

Datumssteuerung

Datumsvergleich

Definition

- Funktion date(*dd,mm,yyy*) mit
  - *dd*: Tag (1..31)
  - *mm*: Monat (1=Januar, 12=Dezember)
  - *yyy*: Jahresdifferenz (0..255) vom Jahr 2000 an

Argumente

- Alle vom Datentyp u08

Wirkung

- Der Ausgang ist 1b01, wenn das Datum erreicht oder bereits verstrichen ist. Liegt das Datum vor dem eingestellten Wert, geht der Ausgang auf 0.

Rückgabewert

- Datentyp b01

Beispiel: Datumsvergleichszeitschaltuhr

Am 01. Oktober 2009 soll die Variable EinzugEin auf 1u08 gesetzt werden.

Umsetzung im Anwenderprogramm:

```
if date(10,1,09) then EinzugEin=1 endif
```

Monatsvergleich

Definition

- Funktion month(*dd,mm*) mit:
  - *dd*: Tag (1..31)
  - *mm*: Monat (1=Januar, 12=Dezember)

Argumente

- Zwei Argumente vom Datentyp u08

Wirkung

- Der Ausgang ist 1b01, wenn das Datum erreicht oder bereits verstrichen ist. Liegt das Datum vor dem eingestellten Wert, geht der Ausgang auf 0b01. Mit dem Beginn eines neuen Jahres (Januar, 1) geht der Ausgang auf 0b01, bis der Monat und Tag den eingestellten Wert erreichen.

Rückgabewert

- Datentyp b01

Beispiel: Monatsvergleichszeitschaltuhr

Jedes Jahr am 01. Dezember soll die Variable WeihnachtenBeleuchtungEin auf 1 gesetzt werden.

Umsetzung im Anwenderprogramm:

```
if month(1,12) then WeihnachtenBeleuchtungEin=1 endif
```

Beispiel: Variablendefinition „Sommer“

Es solle eine Variable Sommer definiert werden, die vom 1.5. bis 30.9. eines jeden Jahres auf 1b01 (EIN) steht.

Umsetzung im Anwenderprogramm:

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

Tagesvergleich

Definition

- Funktion day(*dd*) mit:
  - *dd*: Tag (1..31)

Argumente

- Argument Datentyp u08

Wirkung

- Der Ausgang ist 1b01, wenn der Tag erreicht oder bereits verstrichen ist. Liegt der Tag vor dem eingestellten Wert, geht der Ausgang auf 0b01. Mit dem Beginn eines neuen Monats geht der Ausgang auf 0b01, bis der Tag den eingestellten Wert erreicht.

Rückgabewert

- Datentyp b01

Beispiel: Tagesvergleichszeitschaltuhr

Jeden 6. im Monat soll die Variable RasensprengerEin auf 1 gesetzt werden.

Die Umsetzung in der EibParserdatei lautet dann:

```
if day(6) then RasensprengerEin=1 endif
```

Wochentag

Definition

- Funktion dayofweek() mit:

Argumente

- keine

Wirkung

- Der Ausgang gibt den aktuellen Wochentag [0{Sonntag}..6{Samstag} zurück.

Rückgabewert

- Datentyp u08

Beispiel: Tagesvergleichszeitschaltuhr

Abfrage, ob heute Sonntag ist und wenn ja, dann die Variable RasensprengerEin auf EIN setzten

Die Umsetzung in der EibParserdatei lautet dann:

```
if dayofweek()==SONNTAG then RasensprengerEin=1 endif
```

Tag (relativ zu) Ostersonntag

Definition

- Funktion easterday(*Offset*)

Argumente

- Argument *Offset* Datentyp s16

Wirkung

- Tag des Ostersonntags berechnen. Dabei wird ein *Offset* für das Berechnungsergebnis angegeben, z.B. Ostersonntag +40 Tage, Ostersonntag – 30 Tage.

Rückgabewert

- Datentyp u08

Monat (relativ zu) Ostersonntag

Definition

- Funktion eastermonth(*Offset*)

Argumente

- Argument *Offset* Datentyp s16

Wirkung

- Monat des Ostersonntags berechnen. Dabei wird ein *Offset* für das Berechnungsergebnis angegeben, z.B. Ostersonntag +40 Tage, Ostersonntag – 30 Tage.

Rückgabewert

- Datentyp u08

Beispiel: Berechnung des Aschermittwochs; (Aschermittwoch ist 46 Tage vor Ostersonntag:)

```
uAschermittwochTag=easterday(-46s16)
uAschermittwochMonat=eastermonth(-46s16)
```

Beschattung und Sonnenstand

Tag oder Nacht

Definition

- Funktion sun()

Argumente

- keine

Wirkung

- Die Sonnenstand-Funktion gibt aus, ob es Tag oder Nacht ist. Das Programm muss dazu die geographische Länge und Breite des betreffenden Ortes kennen. Diese können wie auf S. 23 beschrieben, im EibStudio eingegeben werden.
- Rückgabewert: Der Rückgabewert ist 1 binär, wenn es Tag ist und 0 binär, wenn es Nacht ist.

Rückgabewert

- Datentyp b01

Beispiel 1: Sonnenstand

Wenn es Tag ist soll die Variable JalousieEin auf 0 gesetzt werden.

Die Umsetzung im Anwenderprogramm lautet dann:

```
if (sun()==1b01) then JalousieEin=0 endif
if (sun()==HELL) then JalousieEin=0 endif
```

“HELL“ ist eine vordefinierte Konstante mit dem Binärwert 1b01 und kann deshalb als Vergleichsoperator an Stelle von 1b01 angegeben werden.

Azimut

Definition

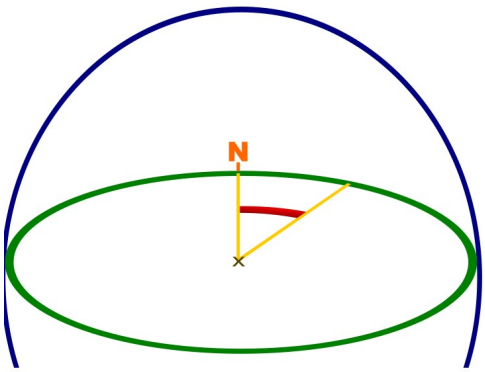
- Funktion azimuth()

Argumente

- keine. Der EibPC muss die geographische Länge und Breite des betreffenden Ortes kennen. Diese können im EibStudio eingegeben werden (s. Seite 23).

Wirkung

- Diese Funktion berechnet zyklisch (Zeitraster: 5 Minuten) den Azimutwinkel der Sonne in Grad, Nord über Ost.



(Quelle: Wikipedia)

Rückgabewert

- Datentyp f32

Beispiel 2: Azimut berechnen

Berechnen Sie alle 5 Minuten Azimutwinkel der Sonne am Aufstellungsort des EibPC

Die Umsetzung in der EibParserdatei lautet dann:

```
AWinkel=azimuth()
```

Hinweis:

Diese Funktion wird bei Hausbeschattungen benötigt. In der Bibliothek EnertexBeschattung finden Sie ausführliche Beispiele.

Elevation

Definition

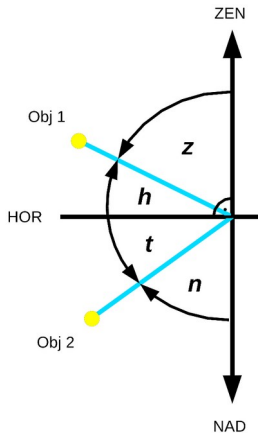
- Funktion elevation()

Argumente

- keine. Der EibPC muss jedoch die geographische Länge und Breite des betreffenden Ortes kennen. Diese können im EibStudio eingegeben werden (S. 23).

Wirkung

- Diese Funktion berechnet zyklisch (Zeitraster: 5 Minuten) den Höhenwinkel der Sonne in Grad.



- (Quelle: Wikipedia)

Rückgabewert

- Datentyp f32

Beispiel:

Berechnen Sie um 5:00 den Höhenwinkel der Sonne am Aufstellungsort des EibPC

Die Umsetzung in lautet:

```
HWinkel=0f32
if htime(5,00,00) then HWinkel=elevation() endif
```

Hinweis:

Diese Funktion wird bei Hausbeschattungen benötigt. In der Bibliothek EnertexBeschattung finden Sie ausführliche Beispiele.

Relativ zu Sonnenaufgang/Sonnen-untergang

Definition

- Funktion presun(*hh,mm*)
  hh: Stunden (0... 23)
  mm: Minuten (0... 59)

Argumente

- zwei Argument vom Datentyp u08

Wirkung

- Gibt in der angegeben Zeit vor dem Wechsel auf Tag eine 1 aus, bzw. vor dem Wechsel auf Nacht eine 0 aus. Das Programm muss dazu die geographische Länge und Breite des betreffenden Ortes kennen.

Rückgabewert

- Sonnenstand, 1= Tag, 0 = Nacht vom Datentyp b01

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

Stunde des Sonnenaufgangs

Definition
- Funktion sunrisehour()

Argumente
- keine

Wirkung
- Der Rückgabewert ist die Stunde (0 bis 23), zu der die Sonne aufgeht.

Rückgabewert
- Datentyp u08

Minute des Sonnenaufgangs

Definition
- Funktion sunriseminute()

Argumente
- keine

Wirkung
- Rückgabewert: Der Rückgabewert ist die Minute (0 bis 59), zu der die Sonne aufgeht.

Rückgabewert
- Datentyp u08

Beispiel: Sonnenaufgang visualisieren

Schreiben Sie auf die Gruppenadresse 1/4/4 (Typ c14) die Zeit für den Sonnenaufgang.

Dies setzen Sie wie folgt um:

```
if htime(sunrisehour(),sunriseminute(),0) then \\
  write('1/4/4'c14, convert(sunrisehour(),$$c14)+$:$c14+convert(sunriseminute(),$$c14))  \\
endif
```

Stunde des Sonnenuntergang

Definition
- Funktion sunsethour()

Argumente
- keine

Wirkung
- Der Rückgabewert ist die Stunde (0 bis 23), zu der die Sonne am aktuellen Tag untergeht.

Rückgabewert
- Datentyp u08

Minute des Sonnenuntergangs

Definition
- Funktion sunsetminute()

Argumente
- keine

Wirkung
- Der Rückgabewert ist die Minute (0 bis 59), zu welcher die Sonne am aktuellem Tag untergeht.

Rückgabewert
- Datentyp u08

Beispiel: siehe obiges Beispiel „Sonnenaufgang visualisieren“

```
if htime(sunsethour(),sunsetminute(),0) then \\
  write('1/4/4'c14, convert(sunsethour(),$$c14)+$:$c14+convert(sunsetminute(),$$c14))  endif
```

Zeitschaltuhren

Zeitschaltuhren sind Funktionen, die beim Eintreten der angegebenen Tageszeit für einen Verarbeitungszyklus des EibPC ihren Rückgabewert von AUS auf EIN und dann wieder auf AUS wechseln. Zeitschaltuhren sind Objekte, die regelmäßige Aktionen anstoßen, z.B. jede Nacht um 1:00 Uhr wird die Garagenbeleuchtung ausgeschaltet etc..

Um die Anwendung einfach zu gestalten, unterscheiden wir vier Typen von Zeitschaltuhren:

- Die Wochenzeitschaltuhr, die eine Aktion pro Woche anstößt,
- die Tageszeitschaltuhr, die pro Tag eine Aktion ausführt,
- die Stundenzeitschaltuhr, die pro Stunde aktiv wird und schließlich
- die Minutenzeitschaltuhr, die pro Minute eine Aktion anstößt.

Um die Aktion auszuführen, muss die Zeitschaltuhr exakt den Zeitpunkt durchlaufen. Dies ist bei der Programmierung zu berücksichtigen. Als Referenzzeit für alle Zeitschaltuhren wird die Systemzeit des EibPC verwendet.

Wochenzeitschaltuhr

Definition

- wtime(*hh,mm,ss,dd*) mit
  - *hh*: Stunden (0..23)
  - *mm*: Minuten (0..59)
  - *ss*: Sekunden (0..59)
  - *dd*: Tag (0=Sonntag, 6=Samstag,7=Werktags, 8=Wochenende)

Argumente

- 4 Argumente vom Datentyp u08

Wirkung

- Der Rückgabewert ist 0b01, wenn die aktuelle Zeit und der Tag der Systemuhr nicht gleich *hh*:*mm*:*ss* und Tag *dd* sind. Wenn der Zeitpunkt eintrifft (und exakt übereinstimmt), geht der Ausgabewert auf 1b01. Wenn der Zeitpunkt überschritten wird, wieder auf 0b01.

Rückgabewert

- Datentyp b01

Beispiel Wochenzeitschaltuhr

Jeden Dienstag, 01:00 Uhr, 30 Sekunden, soll die Variable LichtAktorEin auf 0b01 gesetzt werden.

Umsetzung im Anwenderprogramm:

```
if wtime(01,00,30,DIENSTAG) then LichtAktorEin=0b01 endif
```

Hinweis:

Für die Tage und Wochenende und Werktags sind Konstanten (geschrieben in Großbuchstaben) definiert (MONTAG, DIENSTAG, WERKTAGS, WOCHENENDE etc.)

Tageszeitschaltuhr

Definition

- htime(*hh,mm,ss*) mit
  - *hh*: Stunden (0..23)
  - *mm*: Minuten (0..59)
  - *ss*: Sekunden (0..59)

Argumente

- 3 Argumente vom Datentyp u08

Wirkung

- Der Rückgabewert ist 0b01, wenn die aktuelle Zeit der Systemuhr nicht gleich *hh*:*mm*:*ss* ist. Wenn der Zeitpunkt eintrifft (und exakt übereinstimmt), geht der Ausgabewert auf 1b01. (Wenn der Zeitpunkt überschritten wird, wieder auf 0b01)

Rückgabewert

- Datentyp b01

Beispiel Tageszeitschaltuhr

Jeden Tag, 22:04 Uhr, 7 Sekunden, soll die Variable LichtAktorEin auf '0' gesetzt werden.

Umsetzung im Anwenderprogramm:

```
if htime(22,04,07) then LichtAktorEin=0b01 endif
```


Enertex® Bayern GmbH – Ebermannstädter Straße 8 - 91301 Forchheim - mail@enertex.de

Stundenzeitschaltuhr

Definition

- mtime(*mm,ss*) mit
  - *mm*: Minuten (0..59)
  - *ss*: Sekunden (0..59)

Argumente

- Zwei Argumente vom Datentyp u08

Wirkung

- Der Rückgabewert ist 0b01, wenn die aktuelle Minuten-Sekunden-Zeit der Systemuhr nicht gleich *mm*:*ss* ist (Stunde beliebig). Wenn der Zeitpunkt eintrifft (und exakt übereinstimmt), geht der Ausgabewert auf 1b01. (Wenn der Zeitpunkt überschritten wird, wieder auf 0b01).

Rückgabewert

- Datentyp b01

Beispiel Stundenzeitschaltuhr

Stündlich, immer nach 22 Minuten, 7 Sekunden nach einer vollen Stunde, soll die Variable LichtAktorEin auf '0' gesetzt werden.

Umsetzung im Anwenderprogramm:

```
if mtime(22,07) then LichtAktorEin=0b01 endif
```

Minutenzeitschaltuhr

Definition

- stime(*ss*) mit
  - *ss*: Sekunden (0..59)

Argumente

- Ein Argumente vom Datentyp u08

Wirkung

- Der Rückgabewert ist 0b01, wenn die aktuelle Sekunden-Zeit der Systemuhr nicht gleich *ss* ist (Stunde, Minute beliebig). Wenn der Zeitpunkt eintrifft (und exakt übereinstimmt), geht der Ausgabewert auf 1b01. (Wenn der Zeitpunkt überschritten wird, wieder auf 0b01).

Rückgabewert

- Datentyp b01

Beispiel Minutenzeitschaltuhr

Immer nach 34 Sekunden nach einer vollen Minute soll die Variable Fensterkontakte auf '0' gesetzt werden.

Immer nach 5 Sekunden nach einer vollen Minute, soll die Variable auf '1' gesetzt werden.

Umsetzung im Anwenderprogramm:

```
if stime(34) then Fensterkontakte=0 endif
if stime(5) then Fensterkontakte=1 endif
```


Enertex® Bayern GmbH – Ebermannstädter Straße 8 - 91301 Forchheim - mail@enertex.de

Vergleichszeitschaltuhren

Vergleichszeitschaltuhren sind Objekte, die einen Zeitvergleich ermöglichen. Abhängig von dem Ergebnis des Vergleichs lässt sich dann ein Bustelegramm anstoßen, z.B. jede Nacht von 1:00 Uhr bis 6:00 Uhr wird die Garagenbeleuchtung ausgeschaltet. Wenn der eingestellte Zeitpunkt erreicht ist, sind sie bis zum nächsten Tag auf 1b01, im Gegensatz zur den Schaltzeituhren, die nur zum Zeitpunkt selbst auf 1b01 springen und danach sofort wieder auf 0b01. Damit sind Vergleichszeitschaltuhren sehr ähnlich zu den üblicheren Zeitschaltuhren, haben aber den Vorteil, dass der Zeitpunkt nicht exakt eingetroffen sein muss (z.B. Stromausfall, Neustart).

Als Referenzzeit für alle Vergleichszeitschaltuhren wird die Systemzeit des EibPC verwendet.

Um die Anwendung einfach zu gestalten, unterscheiden wir vier Typen von Vergleichszeitschaltuhren:

- Die Wochenvergleichszeitschaltuhr, die eine Aktion pro Woche anstößt,
- die Tagesvergleichszeitschaltuhr, die pro Tag eine Aktion ausführt,
- die Stundenvergleichszeitschaltuhr, welche pro Stunde aktiv wird und schließlich
- die Minutenvergleichszeitschaltuhr, welche pro Minute eine Aktion anstößt.

Wochenvergleichszeitschaltuhr

Definition

- cwtime(*hh,mm,ss,dd*) mit
  - *ss*: Sekunden (0..59)
  - *mm*: Minuten (0..59)
  - *hh*: Stunden (0..23)
  - *dd*: Tag (0=Sonntag, 6=Samstag,7=Werktags, 8=Wochenende)

Argumente

- 4 Argumente vom Datentyp u08

Wirkung

- Der Rückgabewert ist 0b00, wenn die aktuelle Zeit und der Tag vor *hh*:*mm*:*ss* und Tag *dd* liegen. Nach diesem Zeitpunkt geht der Ausgabewert auf 1b01 und bleibt auf diesem Wert bis zum nächsten Sonntag, 00:00:00 Uhr.

Rückgabewert

- Datentyp b01

Beispiel: Wochenvergleichszeitschaltuhr

Jede Woche ab Dienstag, 01:00 Uhr, 30 Sekunden, soll die Variable LichtAktorEin auf '0' gesetzt werden. Mit Beginn einer neuen Woche soll die Variable wieder auf '1' gesetzt werden.

Umsetzung im Anwenderprogramm:

```
if cwtime(01,00,30,DIENSTAG) then LichtAktorEin=0 else LichtAktorEin=1 endif
```

Hinweis:

1. Für alle Tage einzeln, das Wochenende und Werktags sind Konstanten (geschrieben in Großbuchstaben) definiert, z.B.

   ```
   if cwtime(01,00,30,WERKTAGS) then LichtAktorEin=0 else LichtAktorEin=1 endif
   ```

2. cwtime geht am Schalttag auf 1b01 und bleibt dies bis Sonntag 24:00. Wenn daher MONTAG oder WERKTAGS angegeben wird, ist cwtime immer auf 1b01.

Tagesvergleichszeitschaltuhr

Definition

- chtime(*hh,mm,ss*) mit
  - *ss*: Sekunden (0..59)
  - *mm*: Minuten (0..59)
  - *hh*: Stunden (0..23)

Argumente

- 3 Argumente vom Datentyp u08

Wirkung

- Der Rückgabewert ist 0b01, wenn die aktuelle Zeit der Systemuhr nicht gleich *hh*:*mm*:*ss* ist. Wenn der Zeitpunkt eintrifft, geht der Ausgabewert auf 1b01 und bleibt auf diesem Wert bis zum nächsten Tag (d.h. 00:00:00 Uhr).

Rückgabewert

- Datentyp b01

Beispiel Tagesvergleichszeitschaltuhr

Jeden Tag ab 22:04 Uhr, 7 Sekunden, soll die Variable LichtAktorEin auf '0' gesetzt werden. Mit dem Beginn eines neuen Tages wird die Variable wieder auf '1' gesetzt.

Umsetzung im Anwenderprogramm:

```
if chtime(22,04,07) then LichtAktorEin=0 else LichtAktorEin=1 endif
```

Stundenvergleichszeitschaltuhr

Definition

- cmtime(*mm,ss*) mit
  - *ss*: Sekunden (0..59)
  - *mm*: Minuten (0..59)

Argumente

- Zwei Argumente vom Datentyp u08

Wirkung

- Der Rückgabewert ist 0b01, wenn die aktuelle Minuten-Sekunden-Zeit nicht gleich *mm*:*ss* ist.
- Wenn der Zeitpunkt eintrifft, geht der Ausgabewert auf 1b01 und bleibt auf diesem Wert bis zur nächsten Stunde.

Rückgabewert

- Datentyp b01

Beispiel Stundenvergleichszeitschaltuhr

Stündlich, immer nach 22 Minuten, 7 Sekunden, soll die Variable LichtAktorEin auf '0' gesetzt werden. Zur vollen Stunde soll die Variable wieder auf '1' gesetzt werden.

Umsetzung im Anwenderprogramm:

```
if cmtime(22,07) then LichtAktorEin=0 else LichtAktorEin=1 endif
```


Enertex® Bayern GmbH – Ebermannstädter Straße 8 - 91301 Forchheim - mail@enertex.de

Minutenvergleichszeitschaltuhr

Definition

- cstime(*ss*) mit
  - *ss*: Sekunden (0..59)

Argumente

- Ein Argument vom Datentyp u08

Wirkung

- Der Rückgabewert ist 0b01, wenn die aktuelle Sekunden-Zeit der Systemuhr nicht gleich *ss* ist.
- Wenn der Zeitpunkt eintrifft, geht der Ausgabewert auf 1b01 und bleibt auf diesem Wert, bis zur nächsten Minute.

Rückgabewert

- b01

Beispiel Sekundenvergleichszeitschaltuhr

Immer nach 34 Sekunden nach einer vollen Minute soll die Variable Fensterkontakte auf '0' gesetzt werden. Zu Beginn einer neuen Minute bis zum Erreichen der eingestellten Zeit soll die Variable auf '1' gesetzt werden.

Umsetzung im Anwenderprogramm:

```
if cstime(34) then Fensterkontakte=0 else Fensterkontakte=1 endif
```

Konfigurierbare Zeitschaltuhren

Definition

timeswitch(*ID, Sperre, Objekt1, Name1, Objekt2, Name2, … ObjektN, NameN*)

Argumente

- *ID* (u08) Eindeutige ID der Zeitschaltuhr
- *Sperre* (b01): Nur wenn *Sperre* den Wert 0b01 hat werden die hinterlegten Aktionen beim Erreichen eines Schaltzeitpuntkes ausgeführt.
- *Objekt1* bis *ObjektN* vom Datentyp b01, (u, s)(08, 16, 32, 64, s64), f16, f32. Maximal 100.
- *Name1* bis *NameN* vom Datentyp c65534, max. 100 Zeichen.

Wirkung

- Legt eine Schaltuhr an, die in der Visualisierung durch das Web-Element timeswitch angezeigt oder konfiguriert werden kann.
- Es können mehrere unabhängige Schaltzeiten angelegt werden (max. 25 pro Schaltuhr).
- Für jede Schaltzeit kann gewählt werden, welches Objekt mit welchem Wert geschrieben werden soll. Es müssen nicht immer alle Objekte beteiligt sein.
- Für die Anzeige der Objekte in der Visualisierung wird der Name verwendet.
- Für jedes Objekt muss ein Name angegeben werden.

Rückgabewert

- 0u08 bei Erfolg
- 1u08 bei Fehler

Enertex® Bayern GmbH – Ebermannstädter Straße 8 - 91301 Forchheim - mail@enertex.de

Spezielle Zeitfunktionen

Mit Hilfe von delay und after können sehr kurze Zeitkonstanten generiert werden, wie sie z.B. bei der Steuerung von Bewegungsmeldern (Leuchtdauer, Entprellen gegen Wiedereinschalten) oder bestimmten Regelalgorithmen notwendig werden.

Die minimale Verzögerungszeit beträgt 1 ms, die maximal einstellbare Verzögerungszeit beträgt ca. 30 Jahre.

Delay

Definition

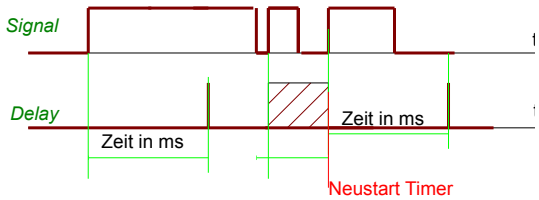
- delay(*Signal*, *Zeit*)

Argumente

- Argument *Signal* vom Datentyp b01
- Argument *Zeit* vom Datentyp u64

Wirkung

- Die Funktion startet beim Übergang der Variablen *Signal* von AUS auf EIN einen Timer. Nach Ablauf der *Zeit* in ms erzeugt die Delay-Funktion einen EIN-Impuls.



- Wenn ein neuer AUS-EIN Impuls auftritt, während der interne Timer läuft, wird der Timer neu gestartet.

Rückgabewert

- b01

Hinweis:

- delay darf nicht im then oder else Zweig einer if-Anweisung stehen.
- Wenn der EibPC mit delay (mit Hilfe einer if-Anweisung und write) ein Telegramm auf den Bus schreibt, kann je nach Buslast und Busgeschwindigkeit eine zusätzliche (variable) Verzögerungszeit von wenigen ms entstehen, bis dieses dort erscheint. Verzögerungszeiten von wenigen ms entstehen, bis dieses dort erscheint.

Beispiel: verzögerte Variablenzuweisung

Wenn die Variable LichtAktor (Datetyp f16) kleiner als 1000f16 ist, soll nach 10s die Variable Licht (Datentyp b01) auf EIN gehen

Umsetzung im Anwenderprogramm:

```
Licht=AUS
if delay(LichtAktor<1000f16,10000u64) then Licht=EIN endif
```

Beispiel: Einschaltverzögerung

Wenn LichtTaster (Typ b01) EIN ist, soll nach 1300 ms die Variable LichtAktor (Typ b01) auf EIN gehen.

Umsetzung im Anwenderprogramm:

```
if delay(LichtTaster,1300u64) then LichtAktor=1b01 endif
```

Alternative 1

```
if delay(LichtTaster==1b01,1300u64) then LichtAktor=1b01 endif
```

Alternative 2

```
if (delay(LichtTaster,1300u64)==1b01) then LichtAktor=1b01 endif
```

Man beachte, dass “LichtAktor“ nur gesetzt, nicht aber gelöscht wird. Siehe dazu auch das folgende Beispiel.

Beispiel: Ausschaltverzögerung

Wenn LichtTaster (Datentyp b01) *AUS* ist, soll nach 4000 ms die Variable LichtAktor auf *AUS* gehen.

Die Umsetzung in der EibParserdatei lautet dann:

```
if (delay(LichtTaster==AUS,4000u64)) then LichtAktor=0b01 endif
```

Beispiel: Verschiedene Ein- und Ausschaltverzögerung

Wenn LichtTaster (Datentyp b01) *EIN* ist soll nach 1300 ms die Variable LichtAktor (Datentyp b01) auf *EIN* ist, wenn LichtTaster (Datentyp b01) *AUS* ist, soll nach 4000 ms die Variable LichtAktor auf *AUS* gehen.

Umsetzung im Anwenderprogramm:

```
if (delay(LichtTaster==EIN,1300u64)) then LichtAktor=EIN endif
if (after(LichtTaster==AUS,4000u64)) then LichtAktor=AUS endif
```

Delayc

Definition

- delayc(*Signal*, *Zeit*,*xT*)

Argumente

- *Signal* vom Datentyp b01
- *Zeit* vom Datentyp u64
- *xT* vom Datentyp u64

Wirkung

- Wie delay.
- Über die Variable *xT* kann die verbleibende Zeit ausgefragt werden.
  ACHTUNG: Wenn Sie dieselbe Variable *xT* für verschiedene Funktionen delayc im Programmcode nutzen, führt dies zu einem undefiniertem Verhalten.
- Die Initialisierung der Variable *xT* wird bei Aktivierung des Timers überschrieben.

Rückgabewert

- b01

Hinweis:

- delayc darf nicht im then oder else Zweig einer if-Anweisung stehen.
- Wenn der EibPC mit delayc (mit Hilfe einer if-Anweisung und write) ein Telegramm auf den Bus schreibt, kann je nach Buslast und Busgeschwindigkeit eine zusätzliche (variable) Verzögerungszeit von wenigen ms entstehen, bis dieses dort erscheint.

Beispiel Verbleibende Zeit ausfragen:

Wenn LichtTaster (Typ b01) EIN ist, soll nach 4000 ms die Variable LichtAktor (Typ b01) auf EIN gehen. Die noch verbleibende Zeit - gemessen ab LichtTaster == EIN bis Ende der Verzögerung 4000 ms - soll alle 200ms auf die Adresse '2/2/2' geschrieben werden.

```
xT=0u64
debug='2/2/2'u64
// Der Timer kann nun über xT ausgefragt werden
if (delayc(LichtTaster==EIN,4000u64),xT) then LichtAktor=1b01 endif
if (change(xT/200u64)) then write('2/2/2'u64, xT) endif
```

After

Definition

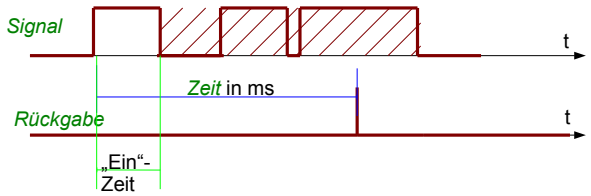
- after(*Signal*, *Zeit*)

Argumente

- *Signal* vom Datentyp b01
- *Zeit* vom Datentyp u64

Wirkung

- Die Funktion startet beim Übergang der Variablen *Signal* von AUS auf EIN einen Timer. Nach Ablauf der *Zeit* in ms erzeugt die After-Funktion einen EIN-Impuls.



- Während des Zeitintervalls Totzeit ist die Funktion gesperrt, d.h. neu ankommende Impulse werden ignoriert

Rückgabewert

- b01

Hinweis:

- Wenn der EibPC mit after (mit Hilfe einer if-Anweisung und write) ein Telegramm auf den Bus schreibt, kann je nach Buslast und Busgeschwindigkeit eine zusätzliche (variable) Verzögerungszeit von wenigen ms entstehen, bis dieses dort erscheint.

Beispiel: Ein- und Auschaltverzögerung

Die Variable LichtAktor (Datetyp b01) soll nach 1000 ms der Variablen LichtTaster (Datentyp b01) folgen.

Umsetzung im Anwenderprogramm:

```
LichtAktor =AUS
if after(LichtTaster,1000u64) then LichtAktor =EIN endif
```

Beispiel: Einschaltverzögerung

Wenn LichtTaster (Typ b01) EIN ist, soll nach 1300 ms die Variable LichtAktor (Typ b01) auf EIN gehen.

Umsetzung im Anwenderprogramm:

```
if (after(LichtTaster,1300u64)==1b01) then LichtAktor=1b01 endif
```

Alternative 1

```
if after(LichtTaster==1b01,1300u64) then LichtAktor=1b01 endif
```

Alternative 2

```
if after(LichtTaster,1300u64) then LichtAktor=1b01 endif
```

Man beachte, dass “LichtAktor“ nur gesetzt, nicht aber gelöscht wird. Siehe dazu auch das folgende Beispiel.

Beispiel: Ausschaltverzögerung

Wenn LichtTaster (Datentyp b01) *AUS* ist, soll nach 4000 ms die Variable LichtAktor auf *AUS* gehen.

Umsetzung im Anwenderprogramm:

```
if (after(LichtTaster==AUS,4000u64)) then LichtAktor=0b01 endif
```

Beispiel: Verschiedene Ein- und Ausschaltverzögerung

Wenn LichtTaster (Datentyp b01) *EIN* ist soll nach 1300 ms die Variable LichtAktor (Datentyp b01) *EIN* ist gehen, wenn LichtTaster (Datentyp b01) *AUS* ist, soll nach 4000 ms die Variable LichtAktor auf *AUS* gehen.

Umsetzung im Anwenderprogramm:

```
if (after(LichtTaster==EIN,1300u64)) then LichtAktor=EIN endif
if (after(LichtTaster==AUS,4000u64)) then LichtAktor=AUS endif
```

Afterc

Definition

- afterc(*Signal*, *Zeit, xT*)

Argumente

- *Signal* vom Datentyp b01
- *Zeit* vom Datentyp u64
- *xT* in ms vom Datentyp u64

Wirkung

- Wie after.
- Über die Variable *xT* kann die verbleibende Zeit ausgefragt werden.
  ACHTUNG: Wenn Sie dieselbe Variable *xT* für verschiedene Funktionen afterc im Programmcode nutzen, führt dies zu einem undefiniertem Verhalten.

Rückgabewert

- b01

Hinweis:

- Wenn der EibPC mit afterc (mit Hilfe einer if-Anweisung und write) ein Telegramm auf den Bus schreibt, kann je nach Buslast und Busgeschwindigkeit eine zusätzliche (variable) Verzögerungszeit von wenigen ms entstehen, bis dieses dort erscheint.

Beispiel: Verbleibende Zeit anzeigen – Zeit Messen

Die Variable LichtAktor (Datetyp b01) soll nach 1000 ms der Variablen LichtTaster (Datentyp b01) folgen. Die noch verbleibende Zeit - gemessen ab LichtTaster == EIN bis Ende der Verzögerung 4000 ms - soll alle 300ms auf die Adresse '2/2/2' geschrieben werden.

Umsetzung im Anwenderprogramm:

```
LichtAktor =AUS
xT=0u64
if afterc(LichtTaster,1000u64,xT) then LichtAktor =EIN endif
if (change(xt/300u64)) then write('2/2/2'u64, xC) endif
```

Zyklustimer – cycle

Definition

- cycle(*mm,ss*)

Argumente

- *mm:* Minuten (0...255) vom Datentyp u08
- *ss:* Sekunden (0..59) vom Datentyp u08

Wirkung



- Der Rückgabewert geht für einen Verarbeitungzyklus wiederkehrend auf 1b01, sonst ist er 0b01. Die Wiederholungszeit ist in mm:ss (Minuten:Sekunden) definiert.

Rückgabewert

- b01

Beispiel: Cycle

Immer nach 1 Minute und 5 Sekunden soll eine Leseanforderung an die Adresse “Licht1-0/0/1“ gesendet werden.

Umsetzung im Anwenderprogramm:

```
if cycle(01,05) then read("Licht1-0/0/1") endif
```

Remanentspeicher

Sie können den Flash-Speicher des EibPCs nutzen, um Variablen im Speicher abzulegen. Dieser Speicher bleibt bei Neustart, Übertragen eines neuen Programms oder Firmwareupdate erhalten, und wird nur beim Zurücksetzen auf Werkseinstellungen gelöscht.

Einlesen

Definition

- readflash(*Variable*, *Speicherplatz)*

Argumente

- *Variable* beliebiger Datentyp
- *Speicherplatz* vom Datentyp u16. Gültige Werte sind 0u16 bis 999u16

Wirkung

- Es werden vom *Speicherplatz* (Nummer 0u16 bis 999u16) im eingebauten Flash soviele Binärdaten auf den Speicher der *Variable* zurückgeschrieben, wie diese aufnehmen kann (vgl. Bitlänge Seite 27). Der Rückgabewert ist 0b01 wenn, das Lesen erfolgreich war, sonst wird 1b01 zurückgegeben.
- Es wird empfohlen, bei neuen Programmen die Funktionen readflashvar und writeflashvar zu nutzen (vgl. S. 137).

Rückgabewert

- b01

Speichern

Definition

- writeflash(*Variable*, *Speicherplatz)*

Argumente

- *Variable* beliebiger Datentyp
- *Speicherplatz* vom Datentyp u16. Gültige Werte sind 0u16 bis 999u16

Wirkung

- Es werden auf den *Speicherplatz* (Nummer 0u16 bis 999u16) im eingebauten Flash die Binärdaten des Speicherinhalts (vgl. Bitlänge Seite 27) der *Variable* abgelegt. Der Rückgabewert ist 0b01 wenn, das Schreiben erfolgreich war, sonst wird 1b01 zurückgegeben.
- Es wird empfohlen, bei neuen Programmen die Funktionen readflashvar und writeflashvar zu nutzen (vgl. S. 137).

Rückgabewert

- b01

Beispiel:

Beim Systemstart sollen zehn 1400-Bytes Strings (c1400) in das Flash auf die ersten 10 Speicherplätze geschrieben werden und anschließend wieder ausgelesen werden. Sollte Schreiben oder Auslesen Probleme bereiten, so soll auf eine Gruppenadresse '8/5/2'c14 eine Textmeldung ausgeben werden.
Das Ergebnis des Auslesens soll ebenso auf diese Gruppenadresse geschrieben werden.

```
[EibPC]
a=$: Zahl$
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
```

Beispiel 2:

Der letzte Wert, der auf den Bus gesendet wird, soll im Flash gespeichert werden und nach einem Neustart automatisch auf den Bus gesendet werden.

```
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
```

Variable einlesen

Definition

- readflashvar(*Variable)*

Argumente

- *Variable* beliebiger Datentyp

Wirkung

- Im eingebauten Flash werden die Binärdaten auf den Speicher der *Variable* zurückgeschrieben, wie diese aufnehmen kann (vgl. Bitlänge Seite 27). Der Rückgabewert ist 0b01 wenn, das Lesen erfolgreich war, sonst wird 1b01 zurückgegeben.
- Das Lesen bzw. die De-Referenzierung erfolgt über den Variablennamen.

Rückgabewert

- Datentyp b01

Variable speichern

Definition

- writeflashvar(*Variable)*

Argumente

- *Variable* beliebiger Datentyp

Wirkung

- Es werden im eingebauten Flash die Binärdaten des Speicherinhalts (vgl. Bitlänge Seite 27) der *Variable* abgelegt. Der Rückgabewert ist 0b01 wenn, das Schreiben erfolgreich war, sonst wird 1b01 zurückgegeben.
- Das Schreiben bzw. die Referenzierung erfolgt ausschließlich über den Variablennamen..

Rückgabewert

- Datentyp b01

Beispiel :

Der letzte Wert einer Variable soll um Mitternacht oder vor dem Einspielen einer neuen Benutzerprogrammierung im Flash gespeichert werden und nach einem Neustart automatisch in die Variable geladen werden.
Hinweis: Die vordefinierte Variable SHUTDOWN wird vom EibStudio vor Einspielen einer neuen Anwenderprogrammierung automatisch auf EIN gesetzt, sodass die Anwendung genügend Zeit bekommt, z.B. Werte auf das Flash zu speichern (vgl.S 23)

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

Rechnen

Mit dem EibPC können nicht nur (logische und zeitliche) Abläufe programmiert, sondern auch mathematische Ausdrücke ausgewertet und daraus entsprechende Reaktionen auf die KNX–Vernetzung, z.B. durch Senden von entsprechenden Adressen, hervorgerufen werden.

Absolutwert

Definition

- abs(*Variable*)

Argumente

- Datentypen: uXX, sXX und fXX, mit XX beliebige auf Seite 27 definierte Bitlänge

Wirkung

- Rückgabewert: Absolutwert der *Variable*

Rückgabewert

- Datentyp wie Argument

Beispiel Absolutwert:

Man berechne von a (=2.5f32) den Absolutwert und speichere diesen in b.

Die Umsetzung in der EibParserdatei lautet dann:

```
a=-2.5f32
b=abs(a)
```

Addition

Definition

- *Variable1* + *Variable2 [...]*

Argumente

- alle Argumente vom gleichen Datentyp
- Datentypen: uXX, sXX und fXX, mit XX beliebige auf Seite 27 definierte Bitlänge

Wirkung

- Die Werte der Variablen werden addiert. Es können nur Werte vom gleichen Datentyp addiert werden. Wollen Sie dennoch beispielsweise einen vorzeichenlosen 8-Bit Wert mit einem vorzeichenbehafteten 16-Bit Wert addieren, benutzen Sie die convert-Funktion (siehe Seite 146)

Rückgabewert

- Datentyp wie Argumente

Hinweis:

- Mit gleicher Syntax können Sie Zeichenketten verketten.

Arkuscosinus

Definition

- acos(*Variable*)

Argumente

- Ein Argument *Variable* von Datentyp f32

Wirkung

- Berechnung des Arkuscosinus der *Variable,* Ergebnis angegeben in RAD
- Wenn das Argument größer 1f32 oder kleiner -1.0f32 ist erfolgt keine Berechnung

Rückgabewert

- Datentyp f32

Beispiel Arkuscosinus:

In Variable b steht das Ergebnis des Arkuscosinus von Variable a.

Die Umsetzung in der EibParserdatei lautet dann:

```
a=5f32
b=acos(a)
```

Arkussinus

Definition

- asin(*Variable*)

Argumente

- Ein Argument *Variable* von Datentyp f32

Wirkung

- Berechnung des Arkussinus der *Variable,* Ergebnis angegeben in RAD
- Wenn das Argument größer 1f32 oder kleiner -1.0f32 ist erfolgt keine Berechnung

Rückgabewert

- f32

Beispiel Arkussinus:

```
In Variable b steht das Ergebnis des Arkussinus von Variable a.
```

Umsetzung im Anwenderprogramm

```
a=5f32
b=asin(a)
```

Arkustangens

Definition

- atan(*Variable1*)

Argumente

- Ein Argument *Variable* von Datentyp f32

Wirkung

- Berechnung des Arkustangens der *Variable,* Ergebnis angegeben in RAD

Rückgabewert

- f32

Beispiel Arkustangens:

```
In Variable b steht das Ergebnis des Argustangens von Variable a.
```

Umsetzung im Anwenderprogramm:

```
a=5f32
b=atan(a)
```

Cosinus

Definition

- cos(*Variable*)

Argumente

- Ein Argument *Variable* von Datentyp f32

Wirkung

- Berechnung des cosinus der *Variable,* Ergebnis angegeben in RAD

Rückgabewert

- f32

Beispiel Cosinus:

```
In Variable b steht der Cosinus von Variable a.
```

Umsetzung im Anwenderprogramm

```
a=5f32
b=cos(a)
```

Division

Definition

- *Variable1 / Variable2 [...]*

Argumente

- alle Argumente vom gleichen Datentyp.
- Datentypen: uXX, sXX und fXX, mit XX beliebige auf Seite 27 definierte Bitlänge

Wirkung

- Variable1 wird durch Variable2 dividiert.

Rückgabewert

- Datentyp wie Argumente

Beispiel

Der Heizungsvorlauf soll abhängig von der Außentemparatur eingestellt werden. Ist die Außentemperatur unter 0°C geht der Vorlauf auf 55°C. Bei 30°C Außentemparatur ist der Vorlauf auf 30°C eingestellt.

Außentemperatur = 15°C

V = 30 + 25/30 * (30 - Außentemperatur)

Umsetzung im Anwenderprogramm

```
V = 30f16 + 25f16 / 30f16 * (30f16 – "Außentemperatur-3/5/0"f16)
```

Durchschnitt

Definition

- Funktion average(*Objekt1*, *Objekt2, [...])*

Argumente

- alle Argumente vom gleichen Datentyp.
- Datentypen: uXX, sXX und fXX, mit XX beliebige auf Seite 27 definierte Bitlänge

Wirkung

- Rückgabewert: Der Durchschnittswert der angegebenen Variablen. Die Genauigkeit der Berechnung ergibt sich durch den verwendeten Datentypen.

Rückgabewert

- Datentyp wie Argumente

Beispiel: Durchschnitt berechnen

Es soll der durchschnittliche Wert der Heizaktoren ermittelt werden.

Umsetzung im Anwenderprogramm:

```
c=average("HeizungKeller1-1/0/2","HeizungKeller2-1/0/3","HeizungKeller3-1/0/4" /
          "HeizungKeller4-1/0/5","HeizungKeller5-1/0/6")
```

Exponentialfunktion

Definition
- exp(*Variable*)

Argumente
- Ein Argument *Variable* von Datentyp f32

Wirkung
- Berechnung des Exponentialfunktion der *Variable*

Rückgabewert
- Datentyp f32

Beispiel E-Funktion:

```
In Variable b steht das Ergebnis der E-Funktion von Variable a.
```

Die Umsetzung in der EibParserdatei lautet dann:

```
a=5f32
b=exp(a)
```

Aufrunden

Definition
- ceil(*Variable*)

Argumente
- Ein Argument *Variable* von Datentyp f32 oder f16

Wirkung
- Kleinste Ganzzahl ≥ *Variable* berechnen

Rückgabewert
- f32

Abrunden

Definition
- floor(*Variable)*

Argumente
- Ein Argument *Variable* von Datentyp f32 oder f16

Wirkung
- Rückgabwert: Größte Ganzzahl ≤ *Variable* ausgeben.

Rückgabewert
- f32

Logarithmus

Definition
- log(*Variable1, Variable2*)

Argumente
- Variable1: Basis vom Datentyp f32
- *Variable2*: Numerus vom Datentyp f32

Wirkung
- Rückgabwert: Das Ergebnis der Logarithmusrechnung.
- Falls ein nicht-positives Argument oder eine nicht-positive Basis vorgegeben wird, erfolgt keine Berechnung.

Rückgabewert
- f32

Enertex® Bayern GmbH – Ebermannstädter Straße 8 - 91301 Forchheim - mail@enertex.de

Maximum

Definition

- max(*Variable1*, *Variable2, [...]*)

Argumente

- alle Argumente vom gleichen Datentyp.
- Datentypen: uXX, sXX und fXX, mit XX beliebige auf Seite 27 definierte Bitlänge.

Wirkung

- Rückgabewert: Das Maximum der angegebenen Variablen, die alle vom selben Datentyp sein müssen.

Rückgabewert

- Datentyp wie Argumente

Beispiel: Maximum von 5 Prozentwerten

Es soll der maximale Wert der Heizaktoren ermittelt werden.

Umsetzung im Anwenderprogramm:

```
c=max("HeizungKeller1-1/0/2","HeizungKeller2-1/0/3","HeizungKeller3-1/0/4" /
        "HeizungKeller4-1/0/5","HeizungKeller5-1/0/6")
```

Minimum

Definition

- min(*Variable1*, *Variable2, [...]*)

Argumente

- alle Argumente vom gleichen Datentyp.
- Datentypen: uXX, sXX und fXX, mit XX beliebige auf Seite 27 definierte Bitlänge

Wirkung

- Rückgabewert: Das Minimum der angegebenen Argumente, die alle vom selben Datentyp sein müssen.

Rückgabewert

- Datentyp wie Argumente

Beispiel: Minimum von 5 Prozentwerten

Es soll der minimale Wert der Heizaktoren ermittelt werden.

Umsetzung im Anwenderprogramm:

```
c=min("HeizungKeller1-1/0/2","HeizungKeller2-1/0/3","HeizungKeller3-1/0/4" /
        "HeizungKeller4-1/0/5","HeizungKeller5-1/0/6")
```

Modulo

Definition

- mod*(Variable1, Variable2)*

Argumente

- alle Argumente vom gleichen Datentyp.
- Datentypen: uXX und sXX mit XX beliebige auf Seite 27 definierte Bitlänge

Wirkung

- Funktion führt Modulo-Berechnung Rückgaberwert = *Variable1* modulo *Variable2*

Rückgabewert

- Datentyp wie Argumente

Multiplikation

Definition

- *Variable1 * Variable2 [...]*

Argumente

- Alle Argumente vom gleichen Datentyp.
- Datentypen: uXX, sXX und fXX, mit XX beliebige auf Seite 27 definierte Bitlänge

Wirkung

- Die Werte der Variablen werden multipliziert.

Rückgabewert

- Datentyp wie Argumente

Potenz

Definition

- pow(*Variable1, Variable2*)

Argumente

- *Variable1*: Basiszahl vom Datentyp f32
- *Variable2*: Exponent vom Datentyp f32

Wirkung

- Rückgabewert: Das Ergebnis der Potenzrechnung.
- Aktualisierung der Abhängigkeiten bei Wertänderung. Falls eine negative Basis vorgegeben wird, erfolgt keine Berechnung.

Rückgabewert

- f32

Quadratwurzel

Definition

- sqrt(*Variable*)

Argumente

- Ein Argument vom Datentyp f32

Wirkung

- Quadratwurzel der *Variable* berechnen. Die Variable muss als Fließkomma 32Bit Wert angegeben werden. Siehe hierzu Tabelle 1 und die dort gemachten Erläuterungen.
- Falls *Variable* kleiner 0 ist, erfolgt keine Berechnung.

Rückgabewert

- f32

Beispiel Quadratwurzel:

```
In Variable b steht das Ergebnis der Quadratwurzel-Funktion von Variable a.
```

Die Umsetzung in der EibParserdatei lautet dann:

```
a=5f32
b=sqrt(a)
```

Sinus

Definition

- Funktion sin(*Variable*)

Argumente

- Ein Argument vom Datentyp f32

Wirkung

- Rückgabewert: Sinus der *Variable*, Ergebnis angegeben in RAD.

Rückgabewert

- f32

Beispiel Sinus:

```
In Variable b steht der Sinus von Variable a.
```

Die Umsetzung in der EibParserdatei lautet dann:

```
a=4f32
b=sin(a)
```

Subtraktion

Definition

- *Variable1 - Variable2 [...]*

Argumente

- Alle Argumente vom gleichen Datentyp.
- Datentypen: uXX, sXX und fXX, mit XX beliebige auf Seite 27 definierte Bitlänge.

Wirkung

- Die Werte der Variablen werden subtrahiert.

Rückgabewert

- Datentyp wie Argumente

Tangens

Definition

- tan(*Variable*)

Argumente

- Ein Argument vom Datentyp f32

Wirkung

- Tangens der *Variable*, Ergebnis angegeben in RAD.

Rückgabewert

- f32

Beispiel Tangens:

```
In Variable b steht das Ergebnis des Tangens von Variable a.
```

Die Umsetzung in der EibParserdatei lautet dann:

```
a=5f32
b=tan(a)
```

Sonderfunktionen

Objekt-Änderung

Definition

- change(*Variable*)

Argumente

- Ein Argument von beliebigem Datentyp

Wirkung

- Rückgabewert: EIN bei Änderung der Überwachten Adresse bzw. Variable im vorhergehenden Verarbeitungszyklus.

Rückgabewert

- b01

Eine Besonderheit der change-Funktionen ist, dass diese nicht bei if-Anweisungen mit else-Zweig stehen dürfen. Ähnlich zur event-Funktion (siehe Seite 171) geht die change-Funktion immer nur für einen Verarbeitungszyklus auf EIN und wird dann bei der if-Anweisung den then-Zweig ausführen. Im nächsten Zyklus geht change wieder auf AUS und nun würde der else-Zweig ausgeführt. Um den Anwender die Programmierung zu vereinfachen, wurde die Verwendung der change-Funktion an dieser Stelle durch den Compiler beschränkt.

Eine Änderung des Arguments aktiviert change in der **darauffolgenden** Verarbeitungsschleife.

Beispiel: Change

Wenn sich die maximale Heizleistung ändert, dann soll der Heizungsvorlauf nach geregelt werden.

Umsetzung im Anwenderprogramm

```
if change(HeizungMax) then write("Heizvorlauf-0/0/1",HeizungBedarf) endif
```

Typkonvertierung

Definition

- convert(*Variable1*, *Variable2*)

Argumente

- Zwei Argumente von beliebigem Datentyp

Wirkung

- Konvertiert den Datentyp von *Variable1* in den von *Variable2*.
- Es sind alle Umwandlungen ineinander erlaubt mit der Ausnahme von b01.
- Wird vom Datentyp f16 in c14 bzw. c1400 umgewandelt, so wird für die Darstellung der Zeichenkette eine Festkommadarstellung mit zwei Nachkommastellen gewählt.
- Wird vom Datentyp f32 in c14 bzw. c1400 umgewandelt, so wird für die Darstellung der Zeichenkette Exponentialdarstellung gewählt wird.
- Wird von einer Zeichenkette in einen numerischen Datentyp konvertiert, wird der Zahlenwert interpretiert. Beginnt die Zeichenkette mit 0x oder 0X, wird die Zahl hexadezimal interpretiert.
- Der Wert von *Variable2* wird immer ignoriert. Dieses Argument dient nur zu Angabe des Datentyps.

Rückgabewert

- Das Ergebnis der Konvertierung der *Variable1* in den Datentyp von *Variable2*.

Hinweis

- Durch die Umwandlung von Datentypen können Informationen z.B. durch „Abschneiden“ von Bits verloren gehen.

Beispiel: Convert-Funktion

Ein vorzeichenloser 8-Bit Wert soll mit einem vorzeichenbehafteten 16-Bit Wert addiert werden.

Die Umsetzung in der EibParserdatei lautet dann:

```
Var1=10u08
Var2=300s16
Var3=convert(Var1,Var2)+Var2
```

Seriennummer

Definition
- devicenr()

Argumente
- keine

Wirkung
- Seriennummer des EibPC abfragen

Rückgabewert
- Datentyp u32

Beispiel: Devicenr

Die Seriennummer soll der Variable SNR zugewiesen werden.

Die Umsetzung in der EibParserdatei lautet dann:

```
SNR= devicenr()
```

Fehlerspeicher auslesen

Definition
- elog()

Argumente
- keine

Wirkung
- Auslesen des ältesten Eventspeicher Eintrags.
- Nach dem Auslesen des Logeintrags wird dieser gelöscht.
- Die Funktion gibt im Klartext die aufgetretenen Ereignisse zurück. Dabei wird folgendes Format in den String geschrieben: Objekttext.:EventID@Zeitstempel. Der Objekttext stellt den Programmcode dar, welcher das Ereignisse auslöst. Der Programmcode ist auf 32 Zeichen begrenzt.

Rückgabewert
- Datentyp c1400 String

Beispiel: siehe elognum S. 147

Fehleranzahl

Definition
- elognum()

Argumente
- keine

Wirkung
- Gibt die Anzahl der Einträge im Fehlerspeicher zurück.

Rückgabewert
- Datentyp u16

Beispiel: elog

Die letzte EventNummer auslesen und den Speicher um eins zurücksetzen.

Die Umsetzung in der EibParserdatei lautet dann:

```
EventInfo=$$
EventNr=elognum()
if change(EventNr) then EventInfo=elog() endif
```

Auswertung erzwingen

Definition

- Funktion eval(*Arg*)

Argumente

- Ein Argument, Datentyp beliebig

Wirkung

- Unabhängig vom Validierungsschema wird die Auswertung des Ausdrucks vorgenommen. Dies ist insbesondere bei der if-Abfrage von Bedeutung, wenn Verschachtelungen in der üblichen Syntax von C-Programmen realisiert werden sollen.

Rückgabewert

- gleicher Datentyp wie Argument

Beispiel: Zähler

Sie möchten einen Zähler programmieren, der mit dem Verarbeitungszyklus des EibPCs bis 100 hochzählt.

Umsetzung:

```
Counter=0
if eval(Counter<100) then Counter=Counter+1 endif
```

Hinweise:

Die Programmierung mit Hilfe des Validierungsschemas ist ein Garant für eine stabile und optimierte, ereignisorientierte Verarbeitung der Telegramme: Nur bei Änderung wird ein Ausdruck/Variable/Funktion ungültig, so dass der EibPC **nur** die davon abhängigen Ausdrücke neu verarbeitet. Die eval-Funktion unterbricht das Valdierungsschema bei der Verarbeitung und erzeugt somit zum Einen eine höhere Systemlast bei der Verarbeitung.

Wenn Sie anstelle von

```
if '1/0/0'b01 then write('1/0/1'b01,AUS) endif
```

versehentlich mit *if eval('1/0/0'b01)* arbeiten würden, können Sie ihre KNX Installation zum Absturz bringen. Wir empfehlen die Verwendung der eval-Funktion nur erfahrenen Programmieren, da das Validierungsschema optimal auf die Verwendung im EibPC und desses Programmierung abgestimmt ist.

Eine Anweisung

```
if Counter<100 then Counter=Counter+1 endif
```

würde im Normalfall beim Systemstart oder Neusetzen der Variablen *Counter* von z.B. 102 auf 10 nur ein einziges mal ausgeführt, da ja dann *Counter*<100 gültig ist und eine weitere Auswertung nicht mehr vorgesehen ist.

Wir empfehlen, bei Verschachtelungen anstelle der Eval-Funktion soweit möglich mit and zu arbeiten.

Verarbeitungszeit

Definition

- Funktion processingtime()

Argumente

- keines.

Wirkung

- Der EibPC benötigt für die Verarbeitung seines Programms pro Zyklus eine gewisse Zeit. Diese Verarbeitungszeit wird mit dieser Funktion in ms zurückgegeben.

Rückgabewert

- Verarbeitungszeit in ms als Datentyp u16.

Beispiel:

Es soll die max. Dauer der Verarbeitung pro Sekunde in einem Diagramm visualisiert werden.

Die Maximalwert über alle Zyklen soll zudem angegeben werden.

```
[WebServer]
page(1) [$Test$,$Processingtime$]
mtimechart(1)[EXTLONG,AUTOSCALE,256,0,10,0,1]($Time in ms $,LEFTGRAF, Buffer0)
[EibPC]
Buffer0=0
timebufferconfig(Buffer0, 0, 3600u16, t)

// per Second
t=0u16
if t < processingtime() then t=processingtime() endif
// Maximum
m=0u16
if m < processingtime() then m=processingtime() endif

// write to chart
if cycle(0,1) then {
        timebufferadd(Buffer0,t);
        t=0u16;
} endif

// Generate some load
y=0f32
if cycle(0,10) then
y=cos(34f32)+sqrt(234f32)+tan(34f32)*7f32+cos(34f32)+sqrt(234f32)+tan(34f32)*7f32+cos(34f32)+sqrt(234f32)+tan(34f32)*7f32+cos(34f32)+sqrt(234f32)+tan(34f32)*7f32+cos(34f32)+sqrt(234f32)+tan(34f32)*7f32+cos(34f32)+sqrt(234f32)+tan(34f32)*7f32+cos(34f32)+sqrt(234f32)+tan(34f32)*7f32 endif
```

Startereignis

Definition

- Funktion systemstart()

Argumente

- keine

Wirkung

- Nach dem Einspielen eines neuen Anwenderprogramm oder einem Neustart des EibPC wechselt diese Funktion während des ersten Verarbeitungszyklus von AUS auf EIN.

Rückgabewert

- Datentyp b01

Beispiel: systemstart

Beim Systemstart sollen die Variablen LichterAus und RolloRauf einmalig auf 0b01 gesetzt werden.

Umsetzung

```
if systemstart() then LichterAus=AUS; RolloRauf=RUNTER endif
```

Start abgeschlossen

Definition

- Funktion ready()

Argumente

- keine

Wirkung

- Ändert seinen Zustand von 0b01 zu 1b01 nach Abschluss der Initialisierung, wenn die normale Verarbeitung beginnt.

Rückgabewert

- Datentyp b01

Unerwarteter Neustart

Definition

- Funktion blackout()

Argumente

- keine

Wirkung

- Ist während der gesamten Programmlaufzeit 1b01 falls das Programm nach nach einem unerwarteten Neustart gestartet wurde.

Rückgabewert

- Datentyp b01

Programmende

Ein eigentliches Programmende gibt es beim EibPC nicht. Ein Programm wird dadurch beendet, dass entweder die Stromversorgung getrennt wird, oder der Anwender ein neues Programm einspielt. Im letzteren Fall setzt EibStudio die eingebaute Variable SHUTDOWN auf EIN, sodass im Anwenderprogramm entsprechende Aktionen davon abhängig ausgeführt werden können. EibStudio wartet dann eine einstellbare Zeit (Standard: 5 Sekunden), bevor das Anwendungsprogramm gestoppt wird. Laufende Schreibaktionen auf das Flash werden noch ordentlich ausgeführt.

Beispiel siehe S. 137.

Zufallszahl

Definition

- Funktion random(Max)

Argumente

- Ein Argument Max vom Datentyp u32

Wirkung

- Gibt eine Zufallszahl im Bereich von 0 bis Max zurück.

Rückgabewert

- Datentyp u32

Beispiell Schaltimpuls mit Zufallszeit

Jeden Abend um 22:00 Uhr plus einer Zufallszeit von 3 Min soll die Variable RolloRunter auf EIN gesetzt werden.

Umsetzung

```
// Zufallszahl von 0 bis 180 (32 Bit vorzeichnlos)
Zufall=convert(random(180u32),0u08)
// Umwanden in Minuten und Sekunden
Min=Zufall/60
Sek=Zufall-Min*60
if htime(22, Min, Sek) then RolloRunter=AUS endif
```

Letzte Änderung

Definition

- comobject(*Variable1*, *Variable2, [...]*)

Argumente

- alle Argumente vom gleichen Datentyp.
- Datentypen: uXX, sXX und fXX, mit XX beliebige auf Seite 27 definierte Bitlänge.

Wirkung

- Rückgabewert: Der Wert der Variablen, der sich als letzter verändert hat.

Rückgabewert

- Datentyp wie Argumente

Beispiel: Ein Aktor mit mehreren Variablen – Status ermittlen

Sie wollen den Status eines Aktors (1 Bit) ermitteln. Der Aktor wird über drei Gruppenadressen "GA_a-1/2/3","GA_b-1/2/4" und "GA_c-1/2/5" angesprochen.
Wenn der Aktor 3 Minuten eingeschaltet war, und nicht bereits manuell wieder ausgeschaltet wurde, soll er auf AUS geschaltet werden.

Umsetzung

```
StatusAktor=comobject("GA_a-1/2/3","GA_b-1/2/4","GA_c-1/2/5")
if delay(StatusAktor==EIN,180000u64) and StatusAktor==EIN then write("GA_a-1/2/3", AUS) endif
```

Passivmodus

Definition

- sleep(Status)

Argumente

- Ein Argument Status vom Datentyp b01.

Wirkung

- Falls der Eingang auf AUS geht, übergibt der EibPC ausgehende EIB-Telegramme und UDP-Pakete an die entsprechende Sendewarteschlange. Liegt am Eingang der Wert EIN an, so werden ausgehende EIB-Telegramme und UDP-Pakete verworfen, d. h. nicht mehr an die Sendewarteschlangen übergeben. Bereits in den Sendewarteschlangen befindliche Daten werden nicht gelöscht und bei Verfügbarkeit der jeweiligen Schnittstelle auf den Bus bzw. das Ethernet geschrieben.

Rückgabewert

- keine

Beispiel: EibPC in den Passivmodus setzen.

Sie möchten einen EibPC in über die Gruppenadresse 2/5/6 (b01) in den Passivmodus setzten

Umsetzung

```
if '2/5/6'b01 then sleep(EIN) else sleep(AUS) endif
```

Hinweis:

Diese Funktion ist hilfreich, wenn in einem bestehenden System ein Programm getestet werden soll ohne tatsächlich auf den Bus zu schreiben. Ohne die Anwender oder etwa ein Programm eines anderen EibPC zu stören, können Sie neue Progamme austesten (der Webserver lässt sich dabei normal bedienen). Wenn der EibPC im Passivmodus läuft, arbeitet er intern normal weiter, d.h. es werden Variablen berechnet, Zustände geändert und der Webserver angepasst etc.

Warteschlangen

Zu Debugzwecken kann es hilfreich sein, die aktuelle Länge der internen Warteschlangen abzufragen. Da die Länge der Warteschlangen begrenzt ist, sollte die Anzahl der Einträge möglichst kurz und in etwa konstant sein. Wird eine Warteschlange immer länger, können neue Befehle oder Ereignisse nicht mehr gespeichert werden. Die maximale Länge einer Warteschlange hängt von ihrem Typ ab.

Diese Funktion kann verwendet werden, um bei Erreichen eines bestimmten Grenzwertes z.B. eine Alarmmeldung zu senden und weitere Abfragen auszusetzen.

Definition

- queuelength(WarteschlangenNummer)

Argumente

- WarteschlangenNummer (u08)

Wirkung

- Liefert die aktuelle Länge der Warteschlange WarteschlangenNummer:
  KnxIn = 1
  KnxOut = 2
  UdpIn = 3
  UdpOut = 4
  VisuResponse = 5
  VisuDisplay = 6
  VisuWeboutput = 7
  VisuDialog = 8
  VisuCharts = 9
  VisuMultiCharts = 10
  VisuTimeCharts = 11
  TcpIn = 12
  Tcp0 = 13
  Tcp1 = 14
  Tcp2 = 15
  Tcp3 = 16
  Ftp = 17
  Vpn = 18
  Resolve = 19
  Ping = 20
  Error = 21
  HttpRequest = 22
  Mail = 23
  ModbusMaster = 24
- Wird nicht durch ihr Argument invalidiert, sondern durch if.

Rückgabewert (u32)

- Aktuelle Länge der Warteschlange

Beispiel: Überwachung der Modbus-TCP Warteschlange

Falls die Modbus-TCP-Warteschlange zu viele Befehle enthält, soll eine E-Mail zur Warnung ausgegeben werden

```
uModbusRequests=0u32
if cycle(1,0) then uModbusRequests=queuelength(24) endif
if (uModbusRequests > 500u32) then {
        sendmail($mail@enertex.de$, $EibPC Warnung$, \
                $Modbus-Master-Warteschlange sehr voll: $+convert(uModbusRequests , $$));
} endif
```

Szenen

Es können pro Szenenfunktion („Szenenaktor“) bis zu 64 Szenen gespeichert und aufgerufen werden. Die Anzahl der Szenenfunktionen (Szenenaktoren) ist dabei nicht begrenzt.

Gespeicherte Szenen bleiben bei einem Neustart erhalten. Bei Änderung der Gruppenadressen, die für die Szenen von Bedeutung sind, sollten die Szenen über das Menü **Projekteinstellungen → Dateien** zurückgesetzt werden.

Szenenaktor

Definition

- Funktion scene(*GruppenadresseSzenebaustein*, *Akt1, Akt2, ..., AktorN*)

Argumente

- *GruppenadresseSzenebaustein* vom Datentyp u08, sonst weitere beliebige (Datentypen und Anzahl) Gruppenadressen
- *AktorXX* (XX 0 bis max. 65000)
  Im Gegensatz zu herkömmlichen KNX-Szenenaktoren, können hier neben Gruppenadressen auch Variablen genutzt werden (s. Beispiel presetscene)

Wirkung

- Es wird ein KNX Szenenaktor mit der Gruppenadresse *GruppenadresseSzenebaustein* erstellt. Dieser kann entweder mit Hilfe von KNX Schaltern und passender ETS Parametrierung oder über unten aufgeführten Funktionen storescene oder callscene angesprochen werden.
- Sie können beliebig viele Szenebausteine definieren.
- Vorgabewerte können mit presetscene vorgegeben werden.

Rückgabewert

- keine

Anmerkung

- Ein inaktiv Schalten von Eingängen ist mit der Funktion presetscene möglich.
- Sie können (wie für alle anderen Funktionen auch) beliebig viele „Szenenbausteine“ (also scene-Funktionen) definieren.
- Jeder so definierte Szenenbaustein verwaltet 64 Szenen (Nummer 1 bis 64)

Beispiel: Lichtszene

Sie möchten einen Szenenaktor für einen Dimmer und eine Lampe erstellen.

Umsetzung im Anwenderprogramm

```
scene(“SceneBaustein-1/4/3“u08, “Dimmer-1/1/2“, “DimmerWert-1/1/3“, “Lampe-1/1/1“)
```

Vorbelegung für Szenenaktor

Definition

- Funktion presetscene(*GruppenadresseSzenebaustein*, *SzenenNummer, OptionOverwrite, WertVar1,KonfVar1,[WertVar2,KonfVar2,...WertVarN, KonfVarN*)

Argumente

- *GruppenadresseSzenebaustein* und *Nummer* vom Datentyp u08
- *OptionOverwrite* vom Datentyp b01
- *WertVarXX* und *WertVarXX;* XX das XX-te Argument passend zur entsprechenden *Variable bzw. GruppenadresseAktor* die mit Funktion scene definiert wurde (XX 0 bis max. 65000)
- *KonfVarXX* vom Datentyp b01

Wirkung

- Vorbelegung für den Szenenaktor mit der Gruppenadresse *GruppenadresseSzenebaustein* und entsprechender *SzenenNummer* erstellen.
- Ist die *OptionOverwrite* mit 1b01 vorbelegt, wird ein bereits bestehender Datensatz bei Neustart überschrieben. Die Funktion wird dann in jedem Fall die Daten auf dem Flash aktualisieren.

  Durch eine Vorbelegung mit 0b01 wird eine bereits gespeicherte Szene nicht überschrieben. Die Funktion wird keinesfalls die Daten auf dem Flash aktualisieren, falls diese bereits angelegt sind. Ist die Szene mit der angegeben *SzenenNummer* noch nicht angelegt, wird diese dennoch auf dem Flash wie parametriert geschrieben.
- *KonfVarXX* dient zu Vorgabe, ob das entsprechende Eingangsobjekt in dieser Szenennummer aktiv ist. Inaktiv bei 0b01, aktiv bei 1b01. Wenn aktiv, ist *WertVarXX* der entsprechende Wert der Vorbelegung.

Rückgabewert

- keine

Beispiel: Lichtszene mit presetscene

Sie möchten einen Szenenaktor für einen Dimmer und eine Lampe erstellen. Zusätzlichen sollen die Variablen Var1 und Var2 verändert werden.

Szeneaktor “SceneBaustein-1/4/3“u08, Nummer 13 soll wie folgt vorbelegt werden:
- Bereits gespeicherte Szenen sollen überschrieben werden
- Der Dimmer sollen inaktiv sein (müssen aber angegeben werden, z.B. 0b01 und 49%)
- Die Lampe sowie die beiden Variablen Var1 und Var2 sollen aktiv sein (wobei auf die Gruppenadresse “Lampe-1/1/1“ ein EIN Signal gesendet, die Var1 auf -20 und Var2 auf „scene on“ gesetzt werden sollen)

Umsetzung im Anwenderprogramm

```
Var1=123s32
Var2=$scene off$c14

scene(“SceneBaustein-1/4/3“u08, “Dimmer-1/1/2“, “DimmerWert-1/1/3“, “Lampe-1/1/1“, Var1, Var2)

presetscene(“SceneBaustein-1/4/3“u08, 13, EIN, 0b01, AUS, 49%, AUS, “Lampe-1/1/1“, EIN,  -20s32, EIN, $scene on$, EIN)
```

Hinweis:

Die Szenenfunktionen scene und presetscene stehen „toplevel“, also nicht in Abhängigkeit einer if-Anweisung. Für die einfache Verwendung von Szenen steht eine Makrobibliothek EnertexSzene zur Verfügung.

Szene speichern

Definition

- storescene(*GruppenadresseSzenebaustein*, *Nummer*)

Argumente

- Zwei Argumente: *GruppenadresseSzenebaustein* und *Nummer* vom Datentyp u08

Wirkung

- Diese Funktion setzt voraus, dass ein Szenenaktor auf diese Gruppenadresse parametriert wurde (entweder über KNX Szenenaktoren oder scene-Funktionen).
- Die Funktion löst ein Telegramm auf *GruppenadresseSzenebaustein* und damit ein Speichern der Szene mit *Nummer* aus.

Rückgabewert

- keine

Beispiel:

Sie möchten eine definierte Szene des obigen Beispiels auf der Nummer 1 speichern.

Umsetzung im Anwenderprogramm

```
if TasterSceneSpeichern==EIN then storescene(“SceneBaustein-1/4/3“u08,1) endif
```

Szene aufrufen

Definition

- callscene(*GruppenadresseSzenebaustein*, *Nummer*)

Argumente

- Zwei Argumente: *GruppenadresseSzenebaustein* und *Nummer* vom Datentyp u08

Wirkung

- Diese Funktion setzt voraus, dass ein Szenenaktor auf die *GruppenadresseSzenebaustein* parametriert wurde (entweder über KNX Szenenaktoren oder scene-Funktionen).
- Die Funktion löst ein Telegramm auf *GruppenadresseSzenebaustein* und damit beim Szenenaktor ein Abrufen der Szene mit *Nummer* aus.

Rückgabewert

- keine

Beispiel:

Sie möchten, die in Beispiel zur scene-Funktion definierte Szene, mit der Nummer 1 abrufen.

Umsetzung im Anwenderprogramm

```
if TasterSceneAbrufen==EIN then callscene(“SceneBaustein-1/4/3“u08,1) endif
```

Stringfunktionen

Strings können variabel von 1 bis 65534 Byte definiert werden. Dazu ist hinter der Zeichenkette der entsprechende Endpunkt anzugeben. z.B. ein String mit einer Länge von 55 Byte wird wie folgt definiert: String= $$c55.

Der Datentyp c14 wird vom Compiler separat behandelt, da er mit dem KNX Datentyp EIS15 kompatibel ist und im Gegensatz zu allen anderen Strings keine Nullterminierung am Ende hat, sowie keine Sonderzeichen erlaubt.

Verketten

Definition

- *String1* + *String2* [+ *String3* ... *StringN*]

Argumente

- Beliebig viele Argumente
- Entweder alle (c14) oder alle vom Datentyp (c)

Wirkung

- Die Strings werden aneinander gefügt. Von links beginnend werden immer zwei Argumente konkateniert und jeweils der größere Datentyp für das Zwischenergebnis verwendet.. Übersteigt die resultierende Länge die maximale Länge, so werden diese Zeichen „abgeschnitten“

Rückgabewert

- Datentyp wie Argumente

Beispiel: Addition von Strings

Die Zeichenketten string1 und string2 sollen „addiert“ bzw. aneinander gehängt werden.

Die Zeichenketten string3 und 4 sind selbst definiert und sollen ebenfalls addiert werden.

Umsetzung im Anwenderprogramm

```
string1=$Zeichen$
string2=$kette$
string3=$55 Byte$c55
string4=$65534 Byte$c65534
// Ergebnis: „Zeichenkette“
result=string1+string2  // Länge: $$c1400
result2=string3+string4 // Länge: $c65534
```

Im String suchen

Definition

- find(*String1, String2, Pos1*)

Argumente

- *String1*(c)
- *String2* (c)
- *Pos1* (u16)

Wirkung

- *String1*: Zeichenkette, in der eine (Teil-)Zeichenkette gesucht werden soll
- *String2*: zu findende Zeichenkette
- *Pos1*: Die ersten *Pos1* - Vorkommen der zu findenden Zeichenkette ignorieren

Rückgabewert (u16)

- Position des ersten Zeichens
- Sie gibt 65535u16 (Konstante EOS) zurück, falls die Zeichenkette nicht gefunden wurde.

Beispiel: Suche Zeichenfolge

In der Variable string=$Zeichenkette$ soll die Zeichenkette „kette“ gesucht werden. Es sollen keine (0) Vorkommen ignoriert werden.

Wenn „kette“ nicht gefunden wird, soll die Variable Error auf 1 gesetzt werden.

Umsetzung im Anwenderprogramm

```
Error
String=$Zeichenkette$
Suche=$kette$
Result=find(String,Suche,0u16)
if Result==65535u16 then Error=EIN endif
```

Daten aus Zeichenkette extrahieren

Definition

- stringcast(*String, Data, Pos*)

Argumente

- 3 Argumente, *String* vom Datentyp c1400,*Data* vom beliebigen Datentyp, *Pos* vom Datentyp u16

Wirkung

- *String* Zeichenkette (1400 Bytes bzw. einer selbst definierten Stringlänge), von der eine bestimmte Anzahl von Bytes in einen anderen Datentyp kopiert werden sollen. Die Anzahl der Bytes wird durch den Datentyp von *Data* festgelegt. Dabei werden nur die Rohdaten kopiert (cast) und keine Umwandlung der Datentypen vorgenommen.
- *Pos*: Die Position des ersten Zeichens im String, das in den Zieltyp kopiert werden sollen.

Rückgabewert

- n Bits (n = Bytelänge des Datentyps *Data*) aus dem *String,* d.h. die Rohdaten werden zurückgegeben

Beispiel: Konvertierung eines Strings in eine Fließkommazahl

In der Variable a=98 sollen die ersten zwei Byte in eine Fließkommazahl geschrieben werden

Umsetzung im Anwenderprogramm

```
a=$98$
z=stringcast(a,0.0,0u16)
// z interpretiert 0x39 0x38 (ASCII „98“) als „72.9600000“
```

Hinweis:

In Verbindung mit stringset und stringcast können Strings zur Verwaltung von Datenarrays genutzt werden. Siehe hierzu Beispiel zu stringset auf S. 158

Daten in Zeichenkette speichern

Definition

- stringset(*Stringg, Data, Pos*)

Argumente

- 3 Argumente, *String* vom Datentyp c1400 bzw. einer selbst definierten Stringlänge, *Data* vom beliebigen Datentyp, *Pos* vom Datentyp u16

Wirkung

- *String* Zeichenkette, in der ein oder mehrere Bytes verändert werden sollen.
- *Data*: Diese Bytes (=Zeichen) ersetzen einen Teil von *String*. Es werden also soviele Bytes in den String kopiert, wie *Data* selbst enthält. Falls *String* vom Typ c ist, wird das abschließende Null-Byte von *String* übersprungen
- *Pos*: Die Startposition der Bytes im String, die ausgetauscht werden sollen. Die Anzahl der Bytes ergibt sich aus dem Datentyp von *Data*.

Rückgabewert

- keine

Beispiel: Ersetze Zeichenfolge

In der Variable a=$ nnette$ soll das 1.te Zeichen auf den Wert 65 =('A') gesetzt werden

Umsetzung im Anwenderprogramm

```
a=$ nnette$
if systemstart() then stringset(a,65,0u16) endif
```

Beispiel: Datenarray - Erstellen und Auslesen

In einem Array sollen alle 15 Min-Werte der Temperatur von der Gruppenadresse '1/1/1'f16 abgelegt werden. Gleichzeitig soll die Temperaturdifferenz der letzten Änderung aus diesem Array entnommen werden.

Die Umsetzung ist wie folgt. Dabei gilt es zu berücksichtigen, dass bei Stringset der Anwender die Bytegröße angeben muss.

Mit dem Debugger (S. 25) können Sie auch die „Rohdaten“ im Array anschauen. Allerdings ist dies wohl nur für Ganzzahltypen sinnvoll.

Es können bis zu 65534 Bytes in Strings genutzt werden.

```
[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
```

Wert als String formatieren

Definition

- Funktion stringformat(*Data, Umwandlungtyp, Format, Feldbreite, Präzision*)

Argumente

- Argument *Data* vom Datentyp uXX, sXX, fXX, mit XX beliebigen, auf Seite 27 definierten Bitlängen.
- Argumente *Format, Feldbreite, Präzision, Umwandlungtyp* vom Datentyp u08

Wirkung

- *Umwandlungstyp*
  - 0: uXX / sXX → Dezimaldarstellung
  - 1: uXX / sXX → Oktaldarstellung
  - 2: uXX / sXX → Hexadezimaldarstellung ('x')
  - 3: uXX / sXX → Hexadezimaldarstellung ('X')
  - 4: fXX → Fließkommadarstellung
  - 5: fXX → Exponentialdarstellung ('e')
  - 6: fXX → Exponentialdarstellung ('E')
- Mit Angabe von *Format* wird wie folgt formatiert:
  - 0: (entfällt)
  - 1: Leerzeichen bei positiver Zahl (nur zulässig, falls *Data* vom Datentyp sXX oder fXX und keine Umwandlung ins Oktal- oder Hexadezimalformat)
  - 2: Vorzeichen bei positiver Zahl (nur zulässig, falls *Data* vom Datentyp sXX oder fXX und keine Umwandlung ins Oktal- oder Hexadezimalformat)
  - 3: Auffüllen mit Nullen (wird ignoriert, falls *Data* vom Datentyp uXX oder sXX bzw. falls bei den Fließkommazahlen die *Präzision* anderweitige Ausdrücke generiert).
  - 4: Auffüllen mit Nullen und Leerzeichen bei positiver Zahl (nur zulässig, falls *Data* vom Datentyp sXX oder fXX und keine Umwandlung ins Oktal- oder Hexadezimalformat; wird ignoriert, falls *Data* vom Datentyp uXX oder sXX bzw. falls bei den Fließkommazahlen die *Präzision* anderweitige Ausdrücke generiert.
  - 5: Auffüllen mit Nullen und Vorzeichen bei positiver Zahl (nur zulässig, falls *Data* vom Datentyp sXX oder fXX und keine Umwandlung ins Oktal- oder Hexadezimalformat; wird ignoriert, falls *Data* vom Datentyp uXX oder sXX bzw. falls bei den Fließkommazahlen die *Präzision* anderweitige Ausdrücke generiert.)
  - 6: Linksbündig
  - 7: Linksbündig und Leerzeichen bei positiver Zahl (nur zulässig, falls *Data* vom Datentyp sXX oder fXX und keine Umwandlung ins Oktal- oder Hexadezimalformat)
  - 8: Linksbündig und Vorzeichen bei positiver Zahl (nur zulässig, falls *Data* vom Berechnungstyp sXX oder fXX und keine Umwandlung ins Oktal- oder Hexadezimalformat)
  - 9: Alternative Formatierung (s.u) , nur zulässig, falls keine Umwandlung ins Dezimalformat:
  - 10: Alternative Formatierung (s.u) und Leerzeichen bei positiver Zahl (nur zulässig, falls *Data* vom Berechnungstyp fXX)
  - 11: Alternative Formatierung (s.u) und Vorzeichen bei positiver Zahl (nur zulässig, falls *Data* vom Berechnungstyp fXX)
  - 12: Alternative Formatierung (s.u) und Auffüllen mit Nullen (nur zulässig, falls keine Umwandlung ins Dezimalformat; wird ignoriert, falls *Data* vom Datentyp uXX oder sXX und *Präzision* angegeben wurde)
  - 13: Alternative Formatierung (s.u), Auffüllen mit Nullen und Leerzeichen bei positiver Zahl (nur zulässig, falls *Data* vom Datentyp fXX)
  - 14: Alternative Formatierung (s.u), Auffüllen mit Nullen und Vorzeichen bei positiver Zahl (nur zulässig, falls *Data* vom Datentyp fXX)
  - 15: Alternative Formatierung (s.u) und linksbündig (nur zulässig, falls keine Umwandlung ins Dezimalformat)
  - 16: Alternative Formatierung (s.u), linksbündig und Leerzeichen bei positiver Zahl (nur zulässig, falls *Data* vom Datentyp fXX)
  - 17: Alternative Formatierung (s.u), linksbündig und Vorzeichen bei positiver Zahl (nur zulässig, falls *Data* vom Datentyp fXX)
  - 18: 0x-Präfix auch bei 0 und Auffüllen mit Nullen (nur zulässig bei Umwandlung ins Hexadezimalformat 'x'. Wird ignoriert, wenn *Präzision* angegeben wurde)
  - 19: 0x-Präfix auch bei 0 und linksbündig (nur zulässig bei Umwandlung ins Hexadezimalformat 'x')
  - 20: 0X-Präfix auch bei 0 und Auffüllen mit Nullen (nur zulässig bei Umwandlung ins Hexadezimalformat 'X'. Wird ignoriert, wenn *Präzision* angegeben wurde)
  - 21: 0X-Präfix auch bei 0 und linksbündig (nur zulässig bei Umwandlung ins Hexadezimalformat 'X')

- *Feldbreite:* Angabe der minimalen Feldbreite
- *Präzision* Angabe der Genauigkeit
- Alternative Formatierung:

  Bei einer Umwandlung ins Oktalformat wird das Zeichen "0" vorangestellt, bei einer Umwandlung ins Hexadezimalformat die Zeichenkette "0x" bzw. "0X".
  Bei einer Umwandlung in Fließkomma- oder Exponentialdarstellung enthält die Ausgabe ein Dezimalzeichen, auch falls diesem keine Ziffern folgen.

  Bei Umwandlungen in andere Formate ist das Ergebnis undefiniert.

Rückgabewert

- Datentyp c1400

Beispiel: Stoppuhr V2 (Vgl. Beispiel:Stoppuhr, S. 115).

Stoppen der Zeit in Sekunden, an der die Variable Stopper_Go auf ein steht. Es soll ein c1400 String angegeben werden, der die Zeit Formatiert in 000d:000h:000m:000s (Tage, Stunden, Minuten, Sekunden) ausgibt.

Hier die Umsetzung, wobei die Sekunden in der Variablen *Stopper_time* und die formatierte Ausgabe in *Stopper* zu finden sind. Im Unterschied zu Beispiel:Stoppuhr (S. 115) wird die Zeitspanne mittels after gezählt.

```
Stopper=$$
Stopper_time=0s32
Stopper_Go=AUS
if (Stopper_Go) then {
        Stopper_time=1s32;
        write(address(85u16),$Starte$c14)
} endif
if after(change(Stopper_time),1000u64) then Stopper_time=Stopper_time+1s32 endif

// Ende Stoppzeit
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
```

Häufige Anwendungsfälle:

Wert	Funktionsaufruf	Ergebnis	Bedeutung
pi=3.1415926535f32	stringformat(pi, 4, 1, 0, 2)	*$ 3.14$*	Leerzeichen oder – als Vorzeichen, zwei Nachkommastellen
	stringformat(-pi, 4, 1, 0, 1)	*-3.1*	eine Nachkommastelle
	stringformat(pi, 4, 6, 0, 2)	*3.14*	linksbündig, zwei Nachkommastellen
	stringformat(pi, 4, 1, 0, 4)	*$ 3.1416$*	Leerzeichen oder – als Vorzeichen, vier Nachkommastellen
	stringformat(pi, 4, 1, 10, 4)	*$ 3.1416$*	insgesamt 10 Zeichen, inkl „.“, mit Leerzeichen links auffüllen
e=.00000000000000000016f32	stringformat(e, 5, 6, 0, 2)	*$1.60e-19$*	In Exponentialdarstellung
nowH=5u32	stringformat(nowH, 0, 3, 2, 2)	*05*	Mit 0 auffüllen auf zwei gültige Ziffern
	stringformat(nowH, 0, 3, 4, 2)	*$ 05$*	Mit 0 auffüllen auf zwei gültige Ziffern, links mit Leerzeichen auf insgesamt 4 Stellen auffüllen.
rgb1=0x0000ffu24 rgb2=255u24	stringformat(rgb1, 2, 18, 0, 6) stringformat(rgb2, 2, 18, 0, 6)	*$0x0000ff$* *$0x0000ff$*	Als Hex-Wert Kleinbuchstaben, mit führendem 0x, auf 6 Stellen mit 0 auffüllen

Teilstring

Definition

- split(*String*, *Pos1, Pos2*)

Argumente

- *String* vom Datentyp c1400 bzw. einer selbst definierten Stringlänge
- *Pos1* vom Datentyp u16
- *Pos2* vom Datentyp u16

Wirkung

- *String*: Zeichenkette, aus der eine Zeichenkette entnommen werden soll
- *Pos1*: erstes Zeichen der abzutrennenden Zeichenkette
- *Pos2*: letztes Zeichen der abzutrennenden Zeichenkette. Wenn *Pos2* gleich 65534u16 (vordefinierte Konstante END) ist, so wird die Zeichenkette bis zu deren Ende abgetrennt.
- Die Variable *String* muss vom Datentyp c1400 sein.
- Rückgabewert: die von Variable String abgetrennte Zeichenkette

Rückgabewert

- Eine Zeichenkette vom Datentyp c1400

Beispiel: split

Aus der Variable string=$Zeichenkette$ soll die Zeichenkette „kette“ entnommen werden. Das erste abzutrennende Zeichen ist an Position 7 (Zählvorgang beginnt bei 0), das letzte Zeichen ist an Position 11.

Die Umsetzung in der EibParserdatei lautet dann:

```
string=$Zeichenkette$
result=split(string, 7u16,11u16)
```

Beispiel: Suche Zeichenfolge (2)

In der Variable string=$Zeichenkette: Hallo$ soll die Zeichenkette „Hallo“ abgetrennt werden.

Umsetzung:

```
String=$Zeichenkette: Hallo$
TeilString=split(String,find(String,$:$,0u16),1399u16)
```

Stringlänge

Definition

- size(*String, Codierung*)

Argumente

- *String* (c)
- *Codierung* (c14) optional

Wirkung

- Von der Zeichenkette *String* soll die Länge bestimmt werden. Die Länge ist durch das Terminierungszeichen „\0“ am Ende von Zeichenketten gegeben.
- Wird keine *Codierung* angegeben, wird ASCII verwendet.
- *Codierung* wie bei encode (S. 164).

Rückgabewert

- Datentyp u16

Beispiel: size

Die Länge des string=$Zeichenkette$ soll bestimmt werden.

Umsetzung im Anwenderprogramm

```
string=$Zeichenkette$
result=size(string)
```

Maximale Länge

Definition

- capacity(*String*)

Argumente

- Ein Argument, *String* vom Datentyp c1400 bzw. mit einer selbst definierten Stringlänge

Wirkung

- Von der Zeichenkette *String* soll die maximal verfügbare Länge bestimmt werden.

Rückgabewert

- u16

Beispiel:

Die maximal verfügbare Länge des strings=$Zeichenkette$ soll bestimmt werden.

Umsetzung im Anwenderprogramm

```
string=$Zeichenkette$
// result ist 1400, da der String c1400 max. 1400 Zeichen aufnehmen kann.
result=capacity(string)
```

Text ersetzen

Definition

- replace(*String, What, Replacement, Count*)

Argumente

- *String* (cXXXXX) Zeichenkette, in der ersetzt werden soll
- *What* (cXXXXX) Zeichenkette, die ersetzt wird.
- *Replacement* (cXXXXX) Zeichenkette, durch die ersetzt wird.
- *Count (*u16) Anzahl der Ersetzungen. 0u16 um alle Vorkommen von *What* zu ersetzen.

Wirkung

- Ersetzt *Num* Vorkommen von *What* in *String* durch *Replacement*. Die Länge von *String* ändert sich dadurch nicht. Überzählige Zeichen werden abgeschnitten.

Rückgabewert

- Eine Zeichenkette gleichen Typs wie *String*.

Beispiel:

Im String $Der EibPC ist der neueste Logikserver von Enertex Bayern GmbH. Kaufen Sie noch heute einen EibPC!$ sollen alle Vorkommnisse von EibPC durch EibPC2 ersetzt werden.

Umsetzung im Anwenderprogramm

```
string=$Der EibPC ist der neueste Logikserver von Enertex Bayern GmbH. Kaufen Sie noch heute einen EibPC!$
result=replace(string, $EibPC$, $EibPC2$, 0u16)
```

Sonderzeichen eingeben

Definition

- tostring(*char1[,char2, ... charN]*)

Argumente

- Mindestens ein Argument, *char1* vom Datentyp u08 als Charactercode der UTF-8 Codierung (vgl. http://de.wikipedia.org/wiki/UTF-8)

Wirkung

- Es wird Zeichenkette aus den einzelnen Bytes gebildet, die terminierende Null wird automatisch angehängt.

Rückgabewert

- Datentyp c1400

Beispiel:

Es soll das Eurozeichen '€' in UTF-8-Codierung im String gespeichert werden.

Umsetzung im Anwenderprogramm

```
Eurozeichen=tostring(0xE2,0x82,0xAC)
```

Zeichencodierung umwandeln

Definition

- encode(*String,Quellcodierung, Zielcodierung*)

Argumente

- *String* vom Datentyp c1400 bzw. mit einer selbst definierten Stringlänge
- *Quellcodierung* mit den üblichen Bezeichnungen, z.B. „UTF-8“ als c14 String.
- *Zielcodierung* mit den üblichen Bezeichnungen, z.B. „UTF-8“ als c14 String.

Wirkung

- Eine Zeichenkette *String*, die in der Quellcodierung vorliegt, wird in die Zielcodierung übertragen.

Rückgabewert

- Datentyp String-Format

Beispiel: encode

Einen String umkodieren von UTF-8 nach ISO-8859

Umsetzung im Anwenderprogramm

```
// String
s1=$Hallöchen$c4000

// String von UTF nach Windows (Deutsch) kodieren;
sDE=encode(s1,$UTF-8$c14,$ISO-8859-15$c14)
```

Einen String umkodieren von ISO-8859 nach UTF-8 codieren

```
// String von UTF nach Windows (Europa) kodieren:
sEU=encode(s1,$UTF-8$c14,$ISO-8859-1$c14)
sUTF=encode(sDE,$ISO-8859-1$c14,$UTF-8$c14)
```

URL Decodieren

Definition

- urldecode(*String,Quellcodierung, Zielcodierung*)

Argumente

- *String* vom Datentyp c1400 bzw. mit einer selbst definierten Stringlänge
- *Quellcodierung* mit den üblichen Bezeichnungen, z.B. „UTF-8“ als c14 String.
- *Zielcodierung* mit den üblichen Bezeichnungen, z.B. „UTF-8“ als c14 String.

Wirkung

- Eine Zeichenkette *String*, die in der Quellcodierung vorliegt, wird in die Zielcodierung übertragen, wobei die URL-Codierung verwendet wird.

Rückgabewert

- Datentyp String-Format

Beispiel:

```
Einen String $ÜberMich.de$,umkodieren
```

Umsetzung im Anwenderprogramm

```
// String:org: $Hallöchen auf http:\\enertex.de$
org=urldecode($Hall%c3%b6chen%20auf%20http%3a%5c%5cenertex.de$,$utf-8$c14,$utf-8$c14)
```

URL-Codierung

Definition

- urlencode(*String,Quellcodierung, Zielcodierung*)

Argumente

- *String* (c)
- *Quellcodierung* (c14) mit den üblichen Bezeichnungen, z.B. „UTF-8“
- *Zielcodierung* (c14) mit den üblichen Bezeichnungen, z.B. „UTF-8“

Wirkung

- Eine Zeichenkette *String*, die in der Quellcodierung vorliegt, wird in die Zielcodierung übertragen, wobei die URL-Codierung verwendet wird.

Rückgabewert

- Datentyp String-Format

Beispiel: encode

```
Einen String $ÜberMich.de$,umkodieren
```

Umsetzung im Anwenderprogramm

```
// 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 Prüfsumme

Definition

- md5sum(*String*)

Argumente

- *String* (c)

Wirkung

- Es wird die MD5-Summe des Strings berechnet. Das Ergebnis wird als String zurückgegeben.

Rückgabewert

- Datentyp cXXXXX mit der gleichen Stringlänge, wie *String*.

Hinweis

- Durch die Funktion hash ersetzt.

Beispiel

```
Der Wert der MD5-Summe des String $fdzehkdkhfckdhk%%$ soll ermittelt werden
```

```
string=$fdzehkdkhfckdhk%%$
md5=md5sum(string)
```

Universelle Hash-Funktion

Definition

- hash(*Algorithmus, String, Länge*)

Argumente

- *Algorithmus* (u08)
- *String* (c)
- *Länge* (u16) optional

Wirkung

- Es wird die Prüfsumme von *String* mit dem gewählten *Algorithmus* berechnet. Das Ergebnis wird als String zurückgegeben.
- *Algorithmus* muss einen der folgenden Werte besitzen:
  HASH_MD5=0u08,
  HASH_SHA1=1u08,
  HASH_SHA256=2u08,
  HASH_SHA512=3u08
- *Länge* gibt die Anzahl Byte an, die gehasht werden (Standard: size(*String*)).

Rückgabewert (c)

- Hex-String des Hashwertes in ASCII-Codierung(c1400).

Beispiel

Die SHA1-Hash des String $Enertex$ soll ermittelt werden

In Kleinbuchstaben umwandeln

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

Definition

- tolower(*String*)

Argumente

- *String* (c)

Wirkung

- Alle Zeichen in *String* werden in ASCII-Kleinbuchstaben konvertiert.

Rückgabewert (c)

- Zeichenkette der gleichen Länge wie *String*

Beispiel

Der String $Enertex$ soll in Kleinbuchstaben konvertiert werden

In Großbuchstaben umwandeln

```
input1=$AlLeSgRosS$
lower_ascii=tolower(input1)
// Ergebnis ist $allesgross$
```

Definition

- toupper(*String*)

Argumente

- *String* (c)

Wirkung

- Alle Zeichen in *String* werden in ASCII-Kleinbuchstaben konvertiert.

Rückgabewert (c)

- Zeichenkette der gleichen Länge wie *String*

Beispiel

Der String $Enertex$ soll in Großbuchstaben konvertiert werden

```
input1=$AlLeSgRosS$
upper_ascii=toupper(input1)
// Ergebnis ist $ALLESGROSS$
```

Als Base64 codieren

Definition

- base64encode(*String, Länge*)

Argumente

- *String* (c)
- *Länge* (u16) gibt die Länge des zu konvertierenden String an. Optional. Standard: size(*String*).

Wirkung

- Alle Bytes in *String* (max. *Länge*) werden Base64-codiert. Wird keine Länge angegeben, wird bei Zeichenketten beim ersten 0-Byte abgebrochen. Das 0-Byte wird nicht mit codiert.
- Achtung: Base64-Codierung benötigt mehr Speicher als die Eingabe. Der Datentyp von *String* muss groß genug für das Ergebnis sein.

Rückgabewert (c)

- Zeichenkette der gleichen Größe wie *String*

Beispiel

```
Der String $Enertex$ soll in base64 konvertiert werden
```

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

Base64 decodieren

Definition

- base64decode(*String*)

Argumente

- *String* (c)

Wirkung

- Alle Zeichen in *String* werden decodiert. Steuerzeichen werden ebenfalls decodiert.

Rückgabewert (c)

- Zeichenkette der gleichen Länge wie *String*

Beispiel

```
Der base64-String $RW5lcnRleA==$ soll zurückkonvertiert werden
```

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

TLS-Zertifikate, Private Keys, Wurzelzertifikate/CA-Zertifikate

Definition

- pem(*String*)

Argumente

- *String* (c)

Wirkung

- *String* wird im PEM-Format korrekt formatiert, um ihn für Funktionen verwenden zu können, die Zertifikate benötigen.
- Dies ist nötig, da Zeichenketten nicht mit Zeilenumbrüchen eingegeben werden können.
- Um eine Zertifikatskette zu erstellen müssen mehrere mit pem() erstellte Zeichenketten mit CR als Trennzeichen verkettet werden.
- Beachten Sie, dass Zertifikate ggf. länger sind als die Standard-Stringlänge von 1400 Zeichen.

Rückgabewert (c)

- Zeichenkette der gleichen Länge wie *String*

Beispiel

Das selbstsignierte Zertifikat eines lokalen Webservers akzeptieren

```
cert=pem($-----BEGIN CERTIFICATE-----
MIIDUDCCAjigAwIBAgIJALvECSjcmOhXMA0GCSqGSIb3DQEBCwUAMB8xHTAbBgNVBAMMFEVuZXJ0Z
XggRU5BIFNOMTExIENBMB4XDTIyMDgzMTEwNDgxOVoXDTM4MDExNzEwNDgxOVowHzEdMBsGA1U
EAwwURW5lcnRleCBFTkEgU04xMTEgQ0EwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDI
yL1tsDMp8d98yDHQPvWRUYZD5nyrHTmkdyiz4nckHvm9H8wx1bO8EjXn+m7AXdgIIRulf6Ni48alvnb77Ld9
XgjlLeHJUeuiX651OIDwR8BBYsQfLp5qzp/L5gwSDKo2Or1Hs+GISqedaLNN3+h/tit2d/
g04j9vjK5qE97HIKoRfJv0wVuuGtyy6azHwXGjbKYIFjbIDH+FXHpL5WTZScxyOyISVFCjXcYvuyWVGhQKS
W+vpOUA3S3IAWj7YB+yvINeEXYAZgZ5kcawa9dvVM/zdgoPe42cL8wuVRsBzng9XQjAcCqibv/
ComRCm4l6jhbJL2dWZCYcAtkZwQQ1AgMBAAGjgY4wgYswHQYDVR0OBBYEFMpsNzzdS9s7/
JfA2LIKn2z2m7m3ME8GA1UdIwRIMEaAFMpsNzzdS9s7/
JfA2LIKn2z2m7m3oSOkITAfMR0wGwYDVQQDDBRFbmVydGV4IEVOQSBTTjExMSBDQYIJALvECSjcmOh
XMAwGA1UdEwQFMAMBAf8wCwYDVR0PBAQDAgEGMA0GCSqGSIb3DQEBCwUAA4IBAQAJyPComoQ
FZrLG8rdd0yXEP3OuNsVjYxU4ZswZ56qWyrMk6aEHH2FghbEzERxjkdJGgNm7ZWpAhhlb0ZMfh0qUc9toQ
cNvT7fRV7YXSRQ/dhkQFBeVVd0Dx75GFhqpDBf3GSwVZGM799nPPj3rPmxiXy9S6OQXyyKVrhoJyQ/
vTm3HX/URZ/
+05m8hdgcK6TZ6SNVCWPs07pUZgsMyZzf1Vzya3uOwaBHQ0C7alU+2PGPGUE3ld3uDzfyLnmt9NPvYFD
BHoqGiV3p82N1HUQfoJOh/
PkBLG9UqdTNVbraW+SE8ZHpeHyDcOLa3HKjgsmW4GoKryz6MUzuOxud8PvgC-----END
CERTIFICATE-----$c1400)

// cert ist $-----BEGIN CERTIFICATE-----
MIIDUDCCAjigAwIBAgIJALvECSjcmOhXMA0GCSqGSIb3DQEBCwUAMB8xHTAbBgNV
BAMMFEVuZXJ0ZXggRU5BIFNOMTExIENBMB4XDTIyMDgzMTEwNDgxOVoXDTM4MDEx
NzEwNDgxOVowHzEdMBsGA1UEAwwURW5lcnRleCBFTkEgU04xMTEgQ0EwggEiMA0G
CSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDIyL1tsDMp8d98yDHQPvWRUYZD5nyr
HTmkdyiz4nckHvm9H8wx1bO8EjXn+m7AXdgIIRulf6Ni48alvnb77Ld9XgjlLeHJ
UeuiX651OIDwR8BBYsQfLp5qzp/L5gwSDKo2Or1Hs+GISqedaLNN3+h/tit2d/g0
4j9vjK5qE97HIKoRfJv0wVuuGtyy6azHwXGjbKYIFjbIDH+FXHpL5WTZScxyOyIS
VFCjXcYvuyWVGhQKSW+vpOUA3S3IAWj7YB+yvINeEXYAZgZ5kcawa9dvVM/zdgoP
e42cL8wuVRsBzng9XQjAcCqibv/ComRCm4l6jhbJL2dWZCYcAtkZwQQ1AgMBAAGj
gY4wgYswHQYDVR0OBBYEFMpsNzzdS9s7/JfA2LIKn2z2m7m3ME8GA1UdIwRIMEaA
FMpsNzzdS9s7/JfA2LIKn2z2m7m3oSOkITAfMR0wGwYDVQQDDBRFbmVydGV4IEVO
QSBTTjExMSBDQYIJALvECSjcmOhXMAwGA1UdEwQFMAMBAf8wCwYDVR0PBAQDAgEG
MA0GCSqGSIb3DQEBCwUAA4IBAQAJyPComoQFZrLG8rdd0yXEP3OuNsVjYxU4ZswZ
56qWyrMk6aEHH2FghbEzERxjkdJGgNm7ZWpAhhlb0ZMfh0qUc9toQcNvT7fRV7YX
SRQ/dhkQFBeVVd0Dx75GFhqpDBf3GSwVZGM799nPPj3rPmxiXy9S6OQXyyKVrhoJ
yQ/vTm3HX/URZ/+05m8hdgcK6TZ6SNVCWPs07pUZgsMyZzf1Vzya3uOwaBHQ0C7a
lU+2PGPGUE3ld3uDzfyLnmt9NPvYFDBHoqGiV3p82N1HUQfoJOh/PkBLG9UqdTNV
braW+SE8ZHpeHyDcOLa3HKjgsmW4GoKryz6MUzuOxud8PvgC
-----END CERTIFICATE-----$c1400
```

Parser

Die folgenden Funktionen sind insbesondere zur Verarbeitung von HTTP-Request hilfreich.

XML

Definition

- parsexml(*String, XPath, Rückgabelänge*)

Argumente

- *String* (c)
- *XPath* (c)
- *Rückgabelänge* (c)

Wirkung

- Der XML-String *String* wird eingelesen und der von *XPath* referenzierte XML-Knoten wird als String zurückgegeben. Eine Erklärung möglicher XPath-Ausdrücke finden Sie z.B. unter https://www.w3schools.com/xml/xml_xpath.asp
- Es können einzelne Attribute, Knotenwerte und Teilbäume ausgewählt werden. Werden mehrere Attribute adressiert, wird nur das letzte davon ausgegeben.
- Wird eine Menge von Knoten ausgewählt, werden diese unter dem neuen Knoten <root/> als String zurückgegeben, der erneut von der Funktion verarbeitet werden kann.
- Wird kein passender Knoten gefunden, ist das Ergebnis leer.
- Die maximale Länge des Ergebnisses wird vom Parameter *Rückgabelänge* bestimmt.
- Ist *String* oder *XPath* leer, so ist das Ergebnis ebenfalls leer.

Rückgabewert (c)

- Zeichenkette der gleichen Länge wie *Rückgabelänge*

Hinweis

- Beim indizierten Zugriff auf Knotenlisten beginnt die Zählung bei 1

Beispiel

Aus einem XML-String soll ein Attribut eines nicht-leeren Knotens ausgelesen werden

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

JSON

Definition

- parsejson(*String, JSONPointer, Rückgabelänge*)

Argumente

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

Wirkung

- Der JSON-String *String* wird eingelesen und das von *Pointer* Objekt als String zurückgegeben. Eine Erklärung möglicher JSONPointer-Ausdrücke finden Sie z.B. unter https://tools.ietf.org/html/rfc6901
- Es können einzelne Werte, und ganze Objekte ausgewählt werden, die Auswahl muss jedoch eindeutig sein. Objekte werden als neues Objekt zurückgegeben, das erneut von der Funktion verarbeitet werden kann.
- Wird kein passender Wert gefunden, ist das Ergebnis leer.
- Die maximale Länge des Ergebnisses wird vom Parameter *Rückgabelänge* bestimmt.

Rückgabewert (c)

- Zeichenkette der gleichen Länge wie *Rückgabelänge*

Hinweis

- Beim indizierten Zugriff auf Arrays beginnt die Zählung bei 0, siehe Beispiel.

Beispiel

Aus einem JSON-String soll eine Eigenschaft ausgelesen werden

```
json=${"number": 5, "array": ["x","y"]}$
number=parsejson(json, $/number$, $$c1)  // number=$5$c1
arrayElement=parsejson(json, $/array/1$,$$c1) // arrayElement=$y$, erstes Element bei Index 0!
```

KNX-Telegramme

Gruppentelegramm senden

Definition

- write(*Gruppenadresse, Wert*)

Argumente

- Zwei Argumente vom gleichen Datentyp. Datentypen aber sonst beliebig.
- *Gruppenadresse*: Importierte oder manuelle KNX™ Gruppenadresse
- *Wert*: Der Wert, welcher auf die KNX™ Gruppenadresse (auf den KNX™-Bus) geschrieben werden soll.

Wirkung

- Auf dem Bus wird ein gültiges KNX™ Telegramm generiert, welches den *Wert* auf die *Gruppenadresse* schreibt.

Rückgabewert

- keine

Leseanforderung senden

Definition

- read(*Gruppenadresse*)

Argumente

- *Gruppenadresse*: Importierte oder manuelle KNX™ Gruppenadresse

Wirkung

- Auf dem Bus wird ein gültiges KNX™ Telegramm generiert, das über das „Lese-Flag“ alle auf die *Gruppenadresse* parametrierte Aktoren abfragt.
- *Gruppenadresse* kann optional mit dem !-Zeichen negiert werden (ohne dass diese ein Auswirkung auf die Wirkung der Funktion hat).

Rückgabewert

- keine

Hinweis

- Damit der Aktor im KNX™ Netzwerk auch antwortet, muss das Lese-Flag bei dessen ETS Programmierung auch gesetzt werden.

Beispiel: Aktuellen Temperaturwert vom Bus „erfragen“

Auf der Adresse 2/3/4 kann von einem Temperatursensor eine Temperatur auf den Bus im Fließkommaformat f16 (16 Bit) gesendet werden. Das Bit “Leseanforderung“ ist in der ETS gesetzt, d.h. die Temperatur kann per Leseanforderung abgefragt werden.
Jeden Tag, 18:30 Uhr, 20 Sekunden, soll die Variable Temperatur von Sensor erfragt werden.

Umsetzung:

```
Temperatur='2/3/4'f16
if chtime(18,30,20) then read('2/3/4'f16) endif
```

Mit der Anweisung *Variable = Gruppenadresse* wird die Information, die, angestoßen durch die read-Funktion, an die *Gruppenadresse* gesendet wird, einer Variablen zugewiesen.
Insgesamt kann der Ablauf des Beispiels mit Abbildung 36 dargestellt werden.

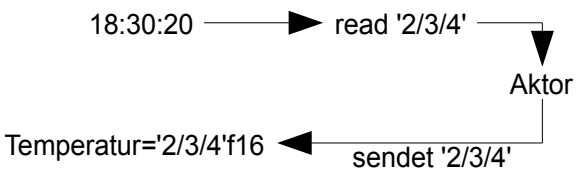


Abbildung 36: Funktionsweise von read

Sobald die Uhrzeit 18:30:20 erreicht wurde, geht die chtime auf EIN, die Bedingung der if-Anweisung ist erfüllt und die read sendet die Leseanforderung. Der Aktor antwortet nun und schickt den Wert auf die Gruppenadresse '2/3/4'f16. Der EibPC schreibt das Ergebnis in die Variable Temperatur.

Busereignis

Definition

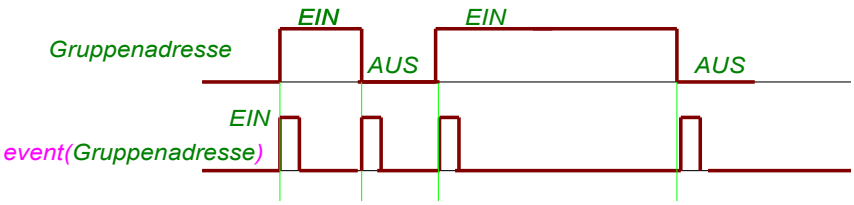
- event(*Gruppenadresse*)

Argumente

- *Gruppenadresse*: Importierte oder manuelle KNX Gruppenadresse
- *Gruppenadresse* kann optional mit dem !-Zeichen negiert werden (ohne dass diese eine Auswirkung auf die Funktion hat).

Wirkung

- Bei UDP, TPC/IP oder RS232 Telegrammen kann die Funktion ebenfalls angewendet werden.
- Diese Funktion reagiert immer, wenn ein Telegramm für die überwachte Adresse auf den Bus geschrieben wird. Sie reagiert nicht auf Variablen.
-
- Im Zusammenhang mit UDP, TPC/IP oder RS232 Telegrammen reagiert sie auf das Eintreffen von Paketen.
- Rückgabewert: 1b01 (EIN-Impuls), wenn am KNX Bus ein Telegramm mit der Gruppenadresse geschickt wird, unabhängig von dessen Inhalt.



Rückgabewert

- Datentyp b01

Hinweis

- Eine Besonderheit der event-Funktionen ist, dass diese nicht bei if-Anweisungen mit else-Zweig stehen dürfen.
- Die event-Funktion geht immer nur für einen Verarbeitungszyklus auf EIN gehen und wird bei Eintreffen eines Telegramms auf die Gruppenadresse bei der if-Anweisung den then-Zweig ausführen. Im nächsten Zyklus geht event wieder auf AUS und nun wird der else-Zweig ausgeführt. Um den Anwender die Programmierung zu vereinfachen, wurde die Verwendung der event-Funktion an dieser Stelle durch den Compiler beschränkt.

Ein Beispiel für die Verwendung der event-Funktion.

Immer wenn die Adresse “BewegungsMelder-3/2/3“ oder !“BewegungsMelder-3/2/4“ ein Ereignis bekommt, soll die Variable Licht auf EIN gesetzt werden.
Nach 3 Minuten soll die Variable Licht wieder auf AUS gesetzt werden.

Die Umsetzung lautet dann:

```
if (event(“BewegungsMelder-3/2/3“)) or (event(“BewegungsMelder-3/2/4“)) then Licht=EIN endif
if(after(Licht,180000u64)==EIN) then Licht=AUS endif
```

Das Überwachen der Busaktivität auf eine Gruppenadresse wird mit der Hilfe der event-Funktion realisiert.
Mit den nun folgenden Event-Funktionen kann analysiert werden, ob das am Bus geschriebene Telegramm

1. ein „normales“ Schreibtelegramm,
2. eine Leseanforderung oder
3. eine Antwort auf eine Leseanforderung

darstellt.

Ereignis: Leseanforderung

Definition
- eventread(*Gruppenadresse*)

Argumente
- *Gruppenadresse*: Importierte oder manuelle KNX Gruppenadresse
- *Gruppenadresse* kann optional mit dem !-Zeichen negiert werden (ohne, dass diese eine Auswirkung auf die Funktion hat).

Wirkung
- Rückgabewert: 1b01 (EIN-Impuls), wenn am KNX Bus eine Leseanforderung mit der Gruppenadresse geschickt wird, unabhängig von dessen Inhalt.

Rückgabewert
- Datentyp b01

Ereignis: Antworttelegramm

Definition
- eventresponse(*Gruppenadresse*)

Argumente
- *Gruppenadresse*: Importierte oder manuelle KNX Gruppenadresse
- *Gruppenadresse* kann optional mit dem !-Zeichen negiert werden (ohne dass diese eine Auswirkung auf die Funktion hat).

Wirkung
- Rückgabewert: 1b01 (EIN-Impuls), wenn am KNX Bus eine Antwort auf eine Leseanforderung mit der Gruppenadresse geschickt wird. Der Inhalt spielt dabei keine Rolle.

Rückgabewert
- Datentyp b01

Ereignis: Gruppentelegramm

Definition
- eventwrite(*Gruppenadresse*)

Argumente
- *Gruppenadresse*: Importierte oder manuelle KNX Gruppenadresse
- *Gruppenadresse* kann optional mit dem !-Zeichen negiert werden (ohne dass diese eine Auswirkung auf die Funktion hat).

Wirkung
- Rückgabewert: 1b01 (EIN-Impuls), wenn am KNX Bus auf die Gruppenadresse geschrieben wird, unabhängig von deren Inhalt.

Rückgabewert
- Datentyp b01

Antworttelegramm senden

Definition
- writeresponse(*Gruppenadresse,Wert*)

Argumente
- *Gruppenadresse*: Importierte oder manuelle KNX Gruppenadresse
- *Wert*: Der Wert, der auf die KNX Gruppenadresse (auf den KNX-Bus) geschrieben werden soll

Wirkung
- Antwortet auf eine Leseanforderung, indem ein gültiges KNX Telegramm generiert wird, das den *Wert* auf die *Gruppenadresse* schreibt. Das Antwortflag wird im Telegramm gesetzt.

Rückgabewert
- keine

Beispiel writeresponse

Auf eine Leseanforderung soll geantwortet werden.

```
if eventread('GA'b01) then writeresponse('GA'b01,a) endif
```

Leseanforderung beim Gerätestart

Definition

- initga(*Gruppenadresse*)

Argumente

- *Gruppenadresse*: Importierte oder manuelle KNX Gruppenadresse
- Die Gruppenadresse kann optional invertiert werden (Verwendung des !-Vorzeichens).

Wirkung

- Die Funktion bewirkt das selbe Verhalten wie die Aktivierung der Initialisierung in **Objekte → Gruppenadressen**.
- Die Funktion kann nur „toplevel“ genutzt werden. Sie kann nicht im then- oder else Zweig einer if-Abfrage stehen.
- Die Funktion kann auch in Verbindung mit der Funktion comobject (S. 152) genutzt werden

Rückgabewert

- keine

Beispiel

```
// Temperatur als Manuelle Gruppenadresse
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
```

Beispiel 2 - comobject

Folgendes Beispiel zeigt die Verwendung in Kombination mit der Funktion comobject.

```
initga(!"Licht KG Treppe-0/0/2")
initga(comobject("Licht EG -Decke Flur-0/0/14","Licht EG Speis-0/0/18"))
```

Sowohl die Verwendung von Negationen als auch die Funktion comobject sind in Verbindung mit der Funktion initga möglich. Dies bringt ins Besondere bei der Programmierung von Makros erhebliche Vorteile mit sich.

Telegramm erstellen

Mit dieser Funktion können KNX-Telegramme auf der untersten Ebene erstellen werden. In diesem Fall können Geräte z. B. auch über ihre physikalische Adresse angesprochen werden, wie es bei der Programmierung von Anwendungsdaten der Fall ist. Der EibPC arbeitet intern im Gruppentelegrammmodus und daher auch nur Gruppentelegramme mitloggt und aufzeichnet, die an eine Gruppenadresse gesendet werden.

Sollen auch andere Telegramme (z.B. an eine physikalische Adresse gesendete) beobachtet werden, so muss die ETS im Busmonitor und mit einer Busmonitor fähigen Schnittstelle arbeiten. Geeignete Schnittstellen für den ETS Busmonitor zur Beobachtung von vom EibPC gesendeten Telegrammen mit physikalischer Adressierung sind:

- EIBMarkt IF-RS232 (Sie müssen umschalten auf FT 1.2 und Busmonitormodus)
- EIB-IP-Router N146 (Sie müssen in der ETS mit dem Busmonitor arbeiten und die Schnittstelle im Routing-Modus betreiben)
- EIB KNX IP Schnittstelle PoE (Sie müssen in der ETS mit dem Busmonitor arbeiten)

Definition

- eibtelegramm(Kontrollfeld, Zieladresse, Telegramminformation, Nutzinformation1 .. Nutzinformation18)

Argumente

- Kontrollfeld vom Datentyp u08

  Das Kontrollfeld ist das erste Zeichen eines jeden Telegramms.

	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							

Abbildung 37: Kontrollfeld eines KNX-Telegramms

  Bit W: Wiederholung; hat den Wert 1

  P1 und P0 geben die Prioritätsstufe an. Diese ist im Normalfall niedrig; d.h. P1=P0=1

  Somit ergibt sich für das Kontrollfeld die Struktur: 10111100b = 188u08

- Zieladresse (physikalische Adresse oder Gruppenadresse) vom Datentyp u16

Bit:	16 .. 12	11 .. 8	7 .. 0
Physikalische Adresse	Bereichs-adresse	Linien-adresse	Teilnehmeradresse
z.B.	1	3	5
Binär:	0001	0011	0101
	1*4096 +	1*512+1*256	+ 0*8+1*4+0*2+1*1
u16-Datentyp	4869		

  Abbildung 38: Umwandlung der physikalischen Adresse in einen u16-Datentyp

- Telegramminformation vom Datentyp u08 enthält:

  a) die Kennzeichnung des Typs der Adresse des Empfängers in Bit 7 (MSB)

  Wert = 0 → physikalische Adresse

  Wert = 1 → Gruppenadresse

  b) den Routing-Zähler in den Bits 4..6

  Zählerstand 7: Ein Telegramm mit diesem Zählerstand wird unverändert und beliebig oft über alle Koppler weiter vermittelt

  Zählerstand 6..1: Ein Telegramm mit einem solchen Zählerstand wird weitervermittelt, sein Routing-Zähler wird beim Durchlaufen eines Kopplers jeweils um eins vermindert

  Zählerstand 0: Ein Telegramm mit diesem Zählerstand wird nicht weitervermittelt

  c) die Länge der Nutzdaten in den Bits 0..3, welche aber aus der Anzahl der Eingänge berechnet wird und deshalb nicht angegeben werden muss

	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 Datentyp	112							

Abbildung 39: Telegramminformationen zum Senden mit physikalischer Adresse

- Nutzinformation1... Nutzinformation18 vom Datentyp u08

  Die ersten beiden Bytes des Nutzdatenbefehls enthalten den auszuführenden Befehl und in den meisten Fällen finden auch die zu übertragenden Informationen darin Platz.

  Eine genau Codierung entnehmen Sie bitte entsprechender Fachliteratur.

Wirkung

- Die Zustandsspeicher der Eingangsobjekte werden an die entsprechenden Stellen eines allgemeinen KNX Telegramm Objekts kopiert.
- Die individuelle Adresse des Senders kann nicht angegeben werden, sie wird durch die Adresse der Schnittstelle ersetzt.

Rückgabewert

- keine

Beispiel: physikalische Adresse eines Teilnehmers ansprechen

Alle 10 Minuten soll an den Aktor mit der physikalischen Adresse 1/3/5 eine Leseanforderung gesendet werden.

Umsetzung

```
if cycle(10,0) then eibtelegramm(188u08,4869u16,112u08,0u08) endif
//Äquivalent dazu können die Daten auch als Hex-Werte angegeben werden:
if cycle(10,0) then eibtelegramm(0xbc,0x1105u16,0x70,0x00) endif
```

Mit den beiden Funktionen address und readknx kann der EibPC wie ein frei programmierbarer Router für KNX Telegramme eingesetzt werden: Wenn Sie beispielsweise per TCP/IP Client die Gruppenadresse (als Zahl) an den EibPC übermitteln, können mit address direkt diese Gruppenadressen schreiben, ohne eine weitere Programmierung vornehmen zu müssen. Gleichermaßen kann ein eintreffendes Telegramm mit readknx an den Client zurückgemeldet werden. address kann in den Funktionen für den Buszugriff, wie z.B. event, write, scene etc., anstelle einer Gruppenadresse verwendet werden.

Gruppenadresse „erstellen“

Definition

- address(*Variable*)

Argumente

- Ein Argument vom Datentyp u16

Wirkung

- Rückgabewert: eine Gruppenadresse, wie sie von write, read etc. verwendet werden kann.

Rückgabewert

- Datentyp Gruppenadresse

Eine Besonderheit der Funktionen für den Buszugriff ist, dass diese als Argumente Gruppenadressen erwarten. So muss z.B. das 1. Argument in write('5/3/11'b01, EIN) eine Gruppenadresse sein. Die Funktion address konvertiert eine 16-Bit Zahl in eine Gruppenadresse. Diese Zahl berechnet sich aus Adresse=HG x 2048+MG x 256 + UG, wobei im Beispiel '5/3/11' HG=5, MG=3 und UG=11 ist. Sie müssen diese Zahl selbst berechnen oder die Funktion getaddress benutzen.

Beispiel:

Sie möchten auf die Gruppenadresse *'5/3/11'b01* den EIN schreiben, wenn das System gestartet wurde.

Umsetzung im Anwenderprogramm

```
if systemstart() then write(address(11019u16),EIN) endif
```

Gruppenadresse auswerten

Definition

- getaddress(*Gruppenadresse*)

Argumente

- *Gruppenadresse* Manuelle oder importierte Gruppenadresse

Wirkung

- Die Funktion gibt den 16 Bit Wert (vorzeichenlos) zurück, welche die Gruppenadresse repräsentiert.

Rückgabewert

- u16

Um 12:00 soll die Gruppenadresse 1/1/27 gelesen und um 12:30 soll der Wert 10% auf diese geschrieben werden.

```
[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
```

Im Normalfall benötigen Sie diese Funktion nicht, da Sie direkt read("Dimmer-1/1/27") etc. verwenden können.

Rohdaten

Definition

- gaimage(*Nummer)*

Argumente

- *Nummer* vom Datentyp u16.

Wirkung

- Die Funktion gibt den Inhalt des internen Abbildspeichers im EibPC der betreffenden Gruppenadresse zurück. Dabei werden die gespeicherten Binärdaten des Telegramms in einen String kopiert (vgl. convert) und das Ergebnis als Rückgabewert weitergegeben.
- Die *Nummer* stellt die Gruppenadresse eines ETS-Exports dar.
- Die Funktion dient zur Übermittlung von im EibPC gespeicherten Zuständen von Gruppenadressen.

Rückgabewert

- c1400

Eine Besonderheit der Funktion ist, dass sie als Argument die Gruppenadressen als 16 Bit Adresse erwartet. Die Nummer stellt eine 16-Bit Zahl dar. Sie berechnet sich aus Adresse=HG x 2048+MG x 256 + UG, wobei zum Beispiel '5/3/11' HG=5, MG=3 und UG=11 ist. Sie müssen diese Zahl selbst berechnen oder die Funktion getaddress benutzen.

Der aktuelle Wert einer Gruppenadresse '1/2/1'f16 soll um 12:00 Uhr als Text in eine Variable gespeichert werden.

```
MyVar=$$
if htime(12,0,0) then {
        MyVar=convert(gaimage(getaddress('1/2/1'f16)),$$)
}endif
```

Name der Gruppenadresse

Definition

- getganame(*Gruppenadresse, Codierung)*

Argumente

- Gruppenadresse Eine importierte Gruppenadresse
- Codierung mit der üblichen Bezeichnung, z.B. $UTF-8$c14 als c14 String, dient der direkten Umwandlung der GA in eine beliebige Systemcodierung.

Wirkung

- Die Funktion gibt den Namen der Gruppenadresse im EibPC-Format zurück, wenn diese Gruppenadresse im Anwendungsprogramm importiert wurde (ESF Import)

Rückgabewert

- c1400

Der Name einer Gruppenadresse soll als Text in der Standard-Windowscodierung (iso8859-15) in einer Variable gespeichert werden.

```
// MyVar=$"LüftenArbeiten-0/0/2"$
MyVar=getganame("LüftenArbeiten-0/0/2",$utf-8$c14)
```

Beliebiges Telegramm

Definition

- readknx(*Nummer*, *Ausgabe*)

Argumente

- *Nummer* vom Datentyp u16.
- *Ausgabe* vom Datentyp c1400.

Wirkung

- Bei Eintreffen eines Telegramms schreibt die Funktion auf das Argument *Nummer* die Gruppenadresse des Telegramms. Die Binärdaten des Telegramms werden in den String *Ausgabe* kopiert. Dabei nimmt nun *Ausgabe* den Typ des zuletzt gesendeten Datentyps an. Um diesen in Text zu wandeln, kann convert die Daten in lesbare Form bringen.

Rückgabewert

- Das Ergebnis der Konvertierung

Hinweis

- Auf die Funktion readknx kann die Event-Funktion angewendet werden.

Beispiel: Verschicken der aktuell eintreffenden Telegramme per UDP

Sie möchten die jeweils, aktuelle am KNX Bus geschriebene, Telegramminformation per UDP an den Empfänger 192.168.22.199 schicken. Dabei soll die Gruppenadresse im Format u16 und die Information im Format GA:XXXXX INF:YYYYYYY übermittelt werden.

```
adr=0u16
info=$$
if event(readknx(adr,info)) then {
        sendudp (5000u16, 192.168.22.199,$GA:$+convert(adr,$$)+$INF:$+info)
}endif
```

Beliebiges Roh-Telegramm

Definition

- readrawknx(*Kontrollfeld, PhyAdresse, Zieladresse, IsGroubAdress, RoutingZähler, Bits, Nutzdaten*)

Argumente

- *Kontrollfeld* vom Datentyp u08.
- *PhyAdresse* des Absenders vom Datentyp u16 (in der üblichen Schreibweise z.B. 2.1.14).
- *Zieladresse* vom Datentyp u16.
- *IsGroubAdress* vom Datentyp b01.
- *RoutingZähler* vom Datentyp u08.
- *BitLänge* vom Datentyp u08.
- *Nutzdaten* vom Datentyp $$.

weiter Informationen zum Telegrammaufbau finden Sie auf S. 174

Wirkung

- Wenn ein beliebiges KNX-Telegramm am Bus beobachtet wird, aktualisiert die Funktion readrawknx ihre Argumente.
- Wenn dies der Fall ist, so werden die Argumente der Funktion soweit mit Daten „gefüllt“, bis die empfangene Datenmenge (eingetroffenes KNX Telegramm) zur Datenlänge der Argumente der readrawknx Funktion passt. In jedem Fall werden die Variablen *PhysAdresse* und *Gruppenadresse* von der Funktion readrawknx bei jedem neu eintreffenden Telegramm mit den aktuellen Daten des Versenders überschrieben.
- Die *PhyAdresse* wird in der üblichen Schreibweise (2.1.4) angegeben.
- Wenn *IsGroubAdress* auf EIN steht, wurde ein Telegramm an eine Gruppenadresse geschickt, im anderen Fall an einen Aktor.
- *BitLänge* ist die Bitlänge des Telegramms (Datentyp) in Byte+1.
- Um zu prüfen, ob ein Telegramm eingetroffen ist, kann die Funktion event auf readrawknx angewendet werden. Dies wird notwendig, wenn Telegramme mit identischen Inhalt ausgewertet werden müssen (s. Funktion readupd S. 182).

Rückgabewert

- keine

Beispiel: KNX Telegramme auswerten

Es soll gezählt werden, wie viele Telegramme von der physikalischen Adresse 1.3.14 auf die Gruppenadresse 2/4/44 gesendet wurden. Dabei muss unterschieden werden, ob das Telegramm an eine Gruppenadresse oder an eine physikalische Adresse geschrieben wurde.

Umsetzung im Anwenderprogramm:

```
Raw_Kontroll=0
Raw_Sender=10.2.1
Raw_GA=0u16
Raw_IsGa=AUS
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
```

Beispiel: Aktorüberwachung

Es wird überprüft, ob vom einem KNX Gerät mindestens alle 120 Minuten ein Telegramm eintrifft. Außerdem ein paar Statistiken zum Bustransfer.

Umsetzung im Anwenderprogramm:

```
// ----------------------------------------
// Physikalische Geräteadresse
// ----------------------------------------
Raw_Dev=1.1.60

// Auswertung
// ----------------------------------------
// Maximale Zeit zwischen zwei Telegrammen vom selben Gerät seit Beginn der Aufzeichnung
Raw_MaxTime=0u16
// Minimale Zeit zwischen zwei Telegrammen vom selben Gerät seit Beginn der Aufzeichnung
Raw_MinTime=65365u16
// Letzte ermittlete Zeit
Raw_CalcTime=0u16
// Durchschnittswert über alle Telegramme vom selben Gerät
Raw_AvgTime=0u64

// Fehlerzeit: Wann soll ein Fehler erkannt werden
Raw_TimeWatch=120u64*60000u64


// Argumente von readrawknx:
Raw_Kontroll=0
Raw_Sender=0.0.0
Raw_GA=0u16
Raw_IsGa=AUS
Raw_RoutingCnt=0
Raw_Len=0
Raw_Data=$$


// ----------------------------------------
// Hilfsvariablen
Raw_AvgTrigger=0u64
Raw_Error=AUS
Raw_AvgTimeSum=0u64
// In welcher Zeitskala wird gerechnet:  1000 entspricht in Sekundengenauigkeit
//                                       60000 entspricht in Minutengenauigkeit
Raw_TimeScale=1000u64

Raw_Time=Raw_TimeWatch


// Nur bei Gruppentelegrammen auf den EibPC reagieren und auch nur, wenn die SenderAdresse die
richtige ist

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 {
        // Zeit seit letzem Diagramm in Sekunden wandeln und Min und Max berechnen
        // Raw_Time wird über die Delayc-Funktion berechnet und kann damit hier
ausgewertet werden
        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;
        // Durchschnittswert=Raw_AvgTime/Raw_Trigger
        Raw_AvgTimeSum=Raw_AvgTimeSum+convert(Raw_CalcTime,0u64);
        Raw_AvgTrigger=Raw_AvgTrigger+1u64;
        Raw_AvgTime=Raw_AvgTimeSum/Raw_AvgTrigger;
} endif
```

```
// Alle Raw_TimeWatch wird ein Telgramm erwartet: Dann wird delay immer wieder neu getriggert
// sonst Fehlerbedingung!
if delayc(change(Raw_AvgTrigger),Raw_TimeWatch,Raw_Time) then {
        Raw_Error=EIN
} endif
```

Hinweis:

Die event-Funktion bzw. in der if-Anweisung die Verknüpfung mit *Telegramm* stellt sicher, dass in jedem Fall der then-Zweig aufgerufen wird. Damit wird zudem gewährleistet, dass die Daten auf den Bus geschrieben werden, wenn identische UDP-Telegramme mehrfach auftreten (und aufgrund des Validierungsschemas, siehe auch S. 33).

Netzwerkfunktionen

Um die Netzwerkfunktionen nutzen zu können, benötigen Sie die Option NP.

Die Ports, über welche der EibPC kommuniziert, können über die **Projekteinstellungen → Verbindung** geändert werden.

UDP

Der EibPC selbst verschickt die Daten beim UDP Transfer standardmäßig über seinen Port 4807. Der Zielport des Empfängers kann beliebig gewählt werden.

Der EibPC empfängt Daten beim UDP Transfer standardmäßig über seinen Port 4806. Der Sender muss daher diesen Zielport angegeben. Der Port, über welchen der Senders Telegramme verschickt, kann vom EibPC ermittelt werden.

UDP-Paket empfangen

Definition

- readudp(*Port, IP, arg 1*[, *arg2, ... argN*])

Argumente

- *Port* vom Datentyp u16. Ausgangsport des Absenders.
- *IP* vom Datentyp u32. IP des Absenders in der üblichen Schreibweise z.B. 192.168.22.100
- *arg2* bis *argN* beliebig

Wirkung

- Übertragene „Nutzdaten“ beginnen ab dem 3 Argument. Diese sind beliebig in Anzahl und Datentyp.
- Wenn ein beliebiges UDP-Paket an den EibPC geschickt wird, aktualisiert jede Funktion readudp ihre Argumente. Wenn dies der Fall ist, so werden die Argumente der Funktion soweit mit Daten „gefüllt“, bis die empfangene Datenmenge (eingetroffenes UDP-Paket) zur Datenlänge der Argumente der readudp Funktion passt. In jedem Fall werden die Variablen *Port* und *IP* von der Funktion readudp bei jedem neu eintreffenden Telegramm mit den aktuellen Daten des Versenders überschrieben.
- Die IP Adresse (Variable *IP*) wird in der üblichen Schreibweise (xxx.xxx.xxx.xxx mit xxx: Zahl zwischen 0 und 255) abgegeben.
- Wenn Ihr LAN Gerät über DNS einen Namen ansprechbar ist, kann die Funktion resolve anstelle einer IP Adresse stehen.
- Um zu prüfen, ob ein Paket eingetroffen ist, kann die Funktion event auf readudp angewendet werden. Dies wird notwendig, wenn Pakete mit identischen Inhalt ausgewertet werden müssen (s.u.).

Rückgabewert

- keine

Beispiel: Daten von UDP-Paket auf den KNX Bus schreiben

Ein UDP-Paket wird vom Absender 122.32.22.1 an den EibPC über den Port 2243u16 des Absenders geschickt. Die Nutzdaten sind drei u08 Daten und sollen immer, wenn ein Paket eingetroffen ist, an die Gruppenadressen 3/4/0,3/4/1,3/4/2 geschickt werden.

Umsetzung im Anwenderprogramm:

```
Port=0u16
IP=0u32
Data1=0;Data2=0;Data3=0
Telegramm=event(readudp(Port, IP,Data1,Data2,Data3))
if (Port==2243u16) and (IP==122.32.22.1) and Telegramm then \\
                    write('3/4/0'u08,Data1);                \\
                    write('3/4/1'u08,Data2);                \\
                    write('3/4/2'u08,Data3)             \\
                    endif
```

Hinweis:

Die event-Funktion bzw. in der if-Anweisung die Verknüpfung mit *Telegramm* stellt sicher, dass in jedem Fall der then-Zweig aufgerufen wird und die Daten auf den Bus geschrieben werden, auch wenn identische UDP-Telegramme mehrfach geschickt werden.

Objekte als UDP-Paket senden

Definition

- sendudp(*Port, IP, arg 1*[*, arg2, ... argN*])

Argumente

- Argument *Port* vom Datentyp u16
- Argument *IP* vom Datentyp u32 (Adresse des Empfängers in der üblichen Schreibweise z.B. 192.168.22.100)
- *arg2* bis *argN* beliebig

Wirkung

- Argument *Port* ist der Port, an den der EibPC Daten verschickt.
- Der EibPC selbst verschickt die Daten über seinen Port 4807 (bzw
- Übertragene „Nutzdaten“ beginnen ab dem 3 Argument. Diese sind beliebig in Anzahl und Datentyp.
- Die IP Adresse (Variable *IP*) wird in der üblichen Schreibweise (xxx.xxx.xxx.xxx mit xxx: Zahl zwischen 0 und 255) abgegeben.
- Wenn Ihr LAN Gerät über DNS einen Namen ansprechbar ist, kann die Funktion resolve anstelle einer IP Adresse stehen.
- Wenn *arg2* bis *argN* vom Datentyp c1400 sind, werden die terminierenden Nullzeichen der Strings mit übermittelt.

Rückgabewert

- keine

Beispiel: UDP-Pakete versenden

Alle 2 Minuten soll ein UDP-Paket an den Empfänger www.enertex.de an den Port 5555u16 vom EibPC geschickt werden. Als Nutzendaten soll ein 32-Bit Zähler für die Pakete und der String „Ich lebe noch“ geschickt werden.

Umsetzung im Anwenderprogramm:

```
Count=0u32
if cycle(2,00) then sendudp(5555u16,resolve($www.enertex.de$, Count,$Ich lebe noch$); \\
                    Count=Count+1u32 endif
```

Daten als UDP-Paket senden

Definition

- sendudparray(*Port, IP, arg ,size*)

Argumente

- Argument *Port* vom Datentyp u16
- Argument *IP* vom Datentyp u32 (Adresse des Empfängers in der üblichen Schreibweise z.B. 192.168.22.100) oder über die Funktion resolve
- *arg* vom Datentyp c1400 bzw. einer selbst definierten Stringlänge
- Argument *size* vom Datentyp u16 (Anzahl der zu sendenden Bytes)

Wirkung

- Argument *Port* ist der Port, an dem der EibPC Daten verschickt.
- Der EibPC selbst verschickt die Daten über seinen Port 4807.
- Übertragene „Nutzdaten“ beginnen ab dem 3 Argument. Diese sind beliebig in Anzahl und Datentyp.
- Die IP Adresse (Variable *IP*) wird in der üblichen Schreibweise (xxx.xxx.xxx.xxx mit xxx: Zahl zwischen 0 und 255) abgegeben.
- Wenn Ihr LAN Gerät über DNS einen Namen ansprechbar ist, kann die Funktion resolve anstelle einer IP Adresse stehen.
- Keine terminierenden Nullzeichen.

Rückgabewert

- keine

Beispiel: UDP Telegramme versenden

Alle 2 Minuten soll ein UDP Telegramm soll an den Empfänger www.enertex.de an den Port 5555u16 vom EibPC geschickt werden. Als Nutzendaten sollen die ersten 5 Bytes des String „Ich lebe noch“ geschickt werden.

Umsetzung im Anwenderprogramm:

```
String =$Ich lebe noch$
if cycle(2,00) then sendudparray(5555u16,resolve($www.enertex.de$), String, size(String),5u16)  endif
```

TCP Server und Client

Der EibPC arbeitet sowohl als Server als auch als Client. Alle 100 ms wird auf eine neue Verbindungsanfrage reagiert. Ist der EibPC verbunden, so beantwortet er die Anfragen mit der Zykluszeit der Verarbeitungsschleife.

Der TCP/IP Server des EibPC nimmt Verbindungsanfragen immer über seinen Port 4809 entgegen. Alternativ kann der Port wie auf S. 22 beschrieben verändert werden.

TCP-Verbindung aufbauen

Definition

- connecttcp(*Port, IP*)

Argumente

- *Port* vom Datentyp u16
- *IP* vom Datentyp u32 (Adresse des Empfängers in der üblichen Schreibweise z.B. 192.168.22.100)

Wirkung

- Der EibPC arbeitet als Client. Er baut eine Verbindung zum angegeben Zielrechner (definiert durch dessen *IP*-Adresse und *Port*) auf.
- Die Funktion liefert den Status ihrer Verarbeitung zurück:
  - bei Erfolg = 0
  - bei andauernder Verarbeitung = 1
  - bei Fehler = 2
  - bei Fehler, wenn eine entsprechende Verbindung schon besteht = 3
  - bei Fehler, da zu viele Verbindungen in Betrieb sind = 4
  - bei automatisch getrennter Verbindung nach timeout = 6
  - nach vom Anwender mit closetcp getrennter Verbindung = 7
  - TCP Gegenstelle hat die Verbindung beendet = 8
  - Initialwert = 9
- nach 30 Sekunden ohne Aktivität auf einer bestehenden Verbindung trennt der Enertex EibPC diese automatisch

Rückgabewert

- u08 (Der Rückgabewert erfolgt asynchron zur Hauptverarbeitungsschleife - siehe S.38)

TCP-Verbindung trennen

Definition

- closetcp(*Port, IP*)

Argumente

- Argument *Port* vom Datentyp u16
- Argument *IP* vom Datentyp u32 (Adresse des Empfängers in der üblichen Schreibweise z.B. 192.168.22.100)

Wirkung

- Der EibPC schließt die Client-Verbindung zum angegeben Zielrechner (definiert durch dessen *IP*-Adresse und *Port*).
- Die Funktion liefert den Status ihrer Verarbeitung zurück:
  - bei Erfolg = 0,
  - bei andauernder Verarbeitung = 1 und
  - bei Fehler = 2
  - bei nicht existenter Verbindung = 5
  - Initialwert = 9

Rückgabewert

- u08

Objekte aus TCP-Paket lesen

Definition

- readtcp(*Port, IP, arg 1*[*, arg2, ... argN*])

Argumente

- *Port* vom Datentyp u16 (und ist der Port, über den der Absender Daten verschickt).
- *IP* vom Datentyp u32 (die Adresse des Absenders in der üblichen Schreibweise z.B. 192.168.22.100)
- *arg2* bis *argN* beliebig

Wirkung

- Übertragene „Nutzdaten“ beginnen ab dem 3 Argument. Diese sind beliebig in Anzahl und Datentyp.
- Wenn ein beliebiges TCP/IP Telegramm an den EibPC geschickt wird, aktualisiert jede Funktion readtcp ihre Argumente. Wenn dies der Fall ist, so werden die Argumente der Funktion soweit mit Daten „gefüllt“, bis die empfangene Datenmenge (eingetroffenes TCP/IP Telegramm) zur Datenlänge der Argumente der readtcp Funktion passt. In jedem Fall werden die Variablen *Port* und *IP* von der Funktion readtcp bei jedem neu eintreffenden Telegramm mit den aktuellen Daten des Versenders überschrieben.
- Die IP Adresse (Variable *IP*) wird in der üblichen Schreibweise (xxx.xxx.xxx.xxx mit xxx: Zahl zwischen 0 und 255) abgegeben.
- Wenn Ihr LAN Gerät über DNS einen Namen ansprechbar ist, kann die Funktion resolve anstelle einer IP Adresse stehen.
- Um zu prüfen, ob ein Telegramm eingetroffen ist, kann die Funktion event auf readtcp angewendet werden. Dies wird notwendig, wenn Telegramme mit identischen Inhalt ausgewertet werden müssen (s.u.).

Rückgabewert

- keine

Objekte als TCP-Paket senden

Definition

- sendtcp(*Port, IP, arg 1*[*, arg2, ... argN*])

Argumente

- *Port* vom Datentyp u16
- *IP* vom Datentyp u32 (Adresse des Empfängers in der üblichen Schreibweise z.B. 192.168.22.100)
- *arg2* bis *argN* beliebig

Wirkung

- Argument *Port* ist der Port, an dem der EibPC Daten verschickt.
- Übertragene „Nutzdaten“ beginnen ab dem 3 Argument. Diese sind beliebig in Anzahl und Datentyp.
- Die IP Adresse (Variable *IP*) wird in der üblichen Schreibweise (xxx.xxx.xxx.xxx mit xxx: Zahl zwischen 0 und 255) abgegeben.
- Wenn Ihr LAN Gerät über DNS einen Namen ansprechbar ist, kann die Funktion resolve anstelle einer IP Adresse stehen.
- Wenn *arg2* bis *argN* vom Datentyp c1400 sind, werden die terminierenden Nullzeichen der Strings mit übermittelt.

Rückgabewert

- keine

Daten als TCP-Paket senden

Definition

- sendtcparray(*Port, IP, arg* ,*size*)

Argumente

- *Port* vom Datentyp u16
- *IP* vom Datentyp u32 (Adresse des Empfängers in der üblichen Schreibweise z.B. 192.168.22.100)
- *arg* vom Datentyp c1400 bzw. einer selbst definierten Stringlänge
- *size* Datentyp u16 (Anzahl der zu sendenden Bytes)

Wirkung

- Argument *Port* ist der Port, an dem der EibPC Daten verschickt.
- Übertragene „Nutzdaten“ beginnen ab dem 3 Argument. Diese sind beliebig in Anzahl und Datentyp.
- Die IP Adresse (Variable *IP*) wird in der üblichen Schreibweise (xxx.xxx.xxx.xxx mit xxx: Zahl zwischen 0 und 255) abgegeben.
- Wenn Ihr LAN Gerät über DNS einen Namen ansprechbar ist, kann die Funktion resolve anstelle einer IP Adresse stehen.
- Keine terminierenden Nullzeichen.

Rückgabewert

- keine

TCP-Pakete versenden

Alle 2 Minuten soll ein TCP-Paket an den Empfänger www.enertex.de an den Port 5555u16 vom EibPC geschickt werden. Als Nutzendaten sollen die ersten 5 Bytes des Strings „Ich lebe noch“ geschickt werden.

(Die Socketverbindung wurde bereits mit connecttcp geöffnet und besteht.)

Umsetzung im Anwenderprogramm:

```
String =$Ich lebe noch$
if cycle(2,00) then sendtcparray(5555u16,resolve($www.enertex.de$), String, size(String),5u16)  endif
```

Ping

Definition

- ping(*IP*)

Argumente

- Argument *IP* vom Datentyp u32 (Adresse des Empfängers in der üblichen Schreibweise z.B. 192.168.22.100)

Wirkung

- Ausführung des Ping Befehls
- Die Funktion liefert den Status ihrer Verarbeitung zurück:
  - bei Erfolg = 0
  - bei andauernder Verarbeitung = 1
  - bei Fehler bzw. nicht gefunden = 2

Rückgabewert

- u08

  (Der Rückgabewert erfolgt asynchron zur Hauptverarbeitungsschleife - siehe S.38)

Beispiel

Die Adresse www.enertex.de soll kurz nach dem Systemstart angepingt werden. Bei Erfolg soll auf die GA 2/2/2 geschrieben werden.

```
A=3
If after(systemstart(),1u64) 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,$gefunden$c14) endif
```

Hinweis:

Beachten Sie, dass wohl resolve als auch ping asynchrone Funktionen sind. Dies bedeutet, dass bei etwa ein ineinander setzen von diesen Funktionen zu unerwarteten Verhalten führen kann:

```
If after(systemstart(),10u64) then a=ping(resolve($www.enertex.de$)) endif
```

Wenn die Funktion ping und resolve angestoßen werden und unterschiedlich abgearbeitet werden, ist das Verhalten der Abfragen und damit der Wert von *a* u.U. nicht reproduzierbar.

Namensauflösung

Definition

- resolve(*hostname*)

Argumente

- Ein Argument *hostname* vom Datentyp c1400 bzw. einer selbst definierten Stringlänge

Wirkung

- Die Funktion ermittelt die IP-Adresse des angegebenen Hostenamens
- Im Fehlerfall wird 0u32 zurück gegeben.

Rückgabewert

- u32

  (Der Rückgabewert erfolgt asynchron zur Hauptverarbeitungsschleife - siehe S.38)

Beispiel resolve

Der Hostname enertex.de soll aufgelöst werden.

Umsetzung im Anwenderprogramm:

```
hostname=$www.enertex.de$
IP=resolve(hostname)
```

E-Mail
Text-Email versenden

Definition

- sendmail(*Empfängeradresse(n), Betreff, Nachricht*)

Argumente

- 3 Argumente vom Datentyp c1400 bzw. einer selbst definierten Stringlänge

Wirkung

- Es wird an die *Empfängeradresse(n)* (Zeichenkette) eine E-Mail mit dem *Betreff* und der *Nachricht* verschickt. Einzelne Empfänger werden durch Komma getrennt.
- Ab Firmware 4.113 kann der Betreff bis zu 512 Bytes und die Nachricht bis zu 65534 Bytes lang sein.
- Ein Zeilenumbruch wird durch Einfügen der zwei Zeichen '\n' (bzw. Konstante LF) generiert: $Zeile1\nZeile2$ oder $Zeile1$+LF+$Zeile2$.
- Rückgabewert < Firmware 4.113:
  0 = E-Mail erfolgreich versendet
  1 = in Bearbeitung
  2 = Fehler
- Rückgabewert ab Firmware 4.113:
  0 = E-Mail erfolgreich versendet
  1 = in Bearbeitung
  2 = Zu wenig Arbeitsspeicher verfügbar
  3 = Ungültige Serveradresse
  4 = Authentifikation fehlgeschlagen
  5 = TLS Fehler
  6 = Sendefehler, z.B. PLAIN oder STARTTLS nicht unterstützt
  7 = Unerwartete Serverantwort
  8 = Timeout nach 5 Sekunden

Rückgabewert

- Datentyp u08

  (Der Rückgabewert erfolgt asynchron zur Hauptverarbeitungsschleife - siehe S.38)

Hinweis:

Um die Funktion sendmail nutzen zu können, muss die Grundkonfiguration des E-Mail vorgenommen werden (Seite 23).

Die Funktion sendmail sendet reine Textmails. Für das Verschicken von html-Mails steht sendhtmlmail zur Verfügung.

Beispiel: sendmail

Jeden Montag um 8.00 Uhr soll eine E-Mail an eibpc@enertex.de versendet werden.
Der Betreff ist “EibPC“ und der Nachrichtentext “alles prima“ in der ersten und „mit dem EibPC“ in der zweiten Zeile.

Umsetzung:

```
email=$eibpc@enertex.de$
betreff=$EibPC$
nachricht=$alles prima\nmit dem EibPC$
if wtime(08,00,00,MONTAG) then sendmail(email, betreff, nachricht) endif
```

Das geht auch mit zwei oder mehr Empfängern

```
email=$eibpc@enertex.de, knxrouter@enertex.de$
```

HTML-E-Mail versenden

Definition

- sendhtmlmail(*Empfängeradresse, Betreff, Nachricht*)

Argumente

- Drei Argumente vom Datentyp c1400 bzw. einer selbst definierten Stringlänge

Wirkung

- Es wird an die *Empfängeradresse* (Zeichenkette) eine E-Mail mit dem *Betreff* und der *Nachricht* verschickt.
- Ab Firmware 4.113 kann der Betreff bis zu 512 Bytes und die Nachricht bis zu 65534 Bytes lang sein.
- Rückgabewert < Firmware 4.113:
  0 = E-Mail erfolgreich versendet
  1 = in Bearbeitung
  2 = Fehler
- Rückgabewert ab Firmware 4.113:
  0 = E-Mail erfolgreich versendet
  1 = in Bearbeitung
  2 = Zu wenig Arbeitsspeicher verfügbar
  3 = Ungültige Serveradresse
  4 = Authentifikation fehlgeschlagen
  5 = TLS Fehler
  6 = Sendefehler, z.B. PLAIN oder STARTTLS nicht unterstützt
  7 = Unerwartete Serverantwort
  8 = Timeout nach 5 Sekunden

Rückgabewert

- u08

Hinweis:

Um die Funktion sendhtmlmail nutzen zu können, muss die Grundkonfiguration des E-Mail vorgenommen werden (Seite 23).

Die Funktion sendhtmlmail sendet HTML-Mails. Für das Verschicken von rein textbasierten Mails steht sendlmail zur Verfügung.

Beispiel: sendmail

Jeden Montag um 8.00 Uhr soll eine E-Mail an eibpc@enertex.de versendet werden.
Der Betreff ist “EibPC“ und der Nachrichtentext “alles prima“ in der ersten und „mit dem EibPC“ in der zweiten Zeile.

Umsetzung im Anwenderprogramm

```
email=$eibpc@enertex.de$
betreff=$EibPC$
nachricht=$<html><head><meta name="qrichtext" content="1" /></head><body style="font-size:11pt;font-family:Sans Serif"> <p><span style="font-weight:600">alles prima, </span></p> <p>mit dem EibPC</p> </body></html>$
if wtime(08,00,00,MONTAG) then sendhtmlmail(email, betreff, nachricht) endif
```

VPN

VPN-Dienst starten

Definition
- startvpn()

Argumente
- keine

Wirkung
- Startet den VPN Dienst im EibPC. Zuvor muss das VPN im Anwendungsprogramm mit dem EibStudio® konfiguriert werden.
- Nach einem Neustart ist das VPN standardmäßig gestoppt. Deshalb sollte der Dienst mit if systemstart() gestartet werden (siehe Beispiel).
- Für alle eingerichteten Benutzer ist VPN nach diesem Funktionsaufruf sofort geöffnet (um für einen Benutzer VPN zu öffnen vgl. openvpnuser()).
- Beim Überspielen eines neuen Anwendungsprogramms an den EibPC bleibt VPN geöffnet. Ein empfohlener, zusätzlicher Aufruf von startvpn() unterbricht eine bestehende Verbindung nicht. Nur bei einem Neustart wird der Dienst gestoppt.
- Über den Info-Button im EibStudio® kann ausgelesen werden, ob der VPN-Dienst läuft und welche Benutzer freigeschaltet sind.

Rückgabewert
- keine

VPN-Dienst stoppen

Definition
- stopvpn()

Argumente
- keine

Wirkung
- Stoppt den VPN Dienst im EibPC.
- Nach einem Neustart ist das VPN standardmäßig gestoppt.
- Für alle eingerichteten Benutzer ist VPN nach diesem Funktionsaufruf sofort geschlossen (um für einen Benutzer VPN zu öffnen vgl. openvpnuser()).
- Über den Info-Button im EibStudio® kann ausgelesen werden, ob der VPN-Dienst läuft und welche Benutzer freigeschaltet sind.

Rückgabewert
- keine

Aktive VPN-Benutzer ausgeben

Definition
- getvpnusers()

Argumente
- keine

Wirkung
- Liefert eine Liste mit derzeit aktiven VPN-Benutzern

Rückgabewert
- c1400

Hinweis: In der EnertexVPN finden sich Funktionen, die das Handling von VPN mit dem EibPC einfach gestaltet.

Benutzer freischalten

Definition

- openvpnuser(*Benutzername*)

Argumente

- Ein Argument *Benutzername* vom Datentyp c1400 bzw. einer selbst definierten Stringlänge

Wirkung

- Öffnet einen VPN Benutzerzugang. Der Zugang wird erst aktiv, wenn die Funktion startvpn() aufgerufen wurde.
- Nach einem Neustart bleibt der Benutzerzugang aktiv, allerdings muss der VPN Dienst mit startvpn() neu gestartet werden.
- Über den Info-Button im EibStudio® kann ausgelesen werden, ob der VPN-Dienst läuft und welche Benutzer freigeschaltet sind.

Rückgabewert

- keine

Benutzer sperren

Definition

- closevpnuser(*Benutzername*)

Argumente

- Ein Argument *Benutzername* vom Datentyp c1400 bzw. einer selbst definierten Stringlänge

Wirkung

- Schließt einen VPN Benutzerzugang. Der Zugang wird inaktiv unabhängig davon, ob der VPN-Dienst ausgeführt wird oder nicht.
- Nach einem Neustart bleibt der Benutzerzugang aktiv, allerdings muss der VPN Dienst mit startvpn() neu gestartet werden.
- Über den Info-Button im EibStudio® kann ausgelesen werden, ob der VPN-Dienst läuft und welche Benutzer freigeschaltet sind.

Rückgabewert

- keine

Hinweis:

Wenn ein Benutzerzugang mit closevpnuser geschlossen wird, so ist eine noch bestehende Verbindung davon nicht betroffen. Erst nach Abmelden der Verbindung – oder einen Stop und Start des VPN Service - ist ein geschlossener Zugang tatsächlich gesperrt.

Beispiel

Über eine Gruppenaddresse 1/1/1 soll bei einem EIN-Telegramm ein bereits angelegter VPN Teilnehmer „USER1“ der Zugang frei geschaltet werden. Bei einem AUS Telegramm, soll dieser Zugang wieder geschlossen werden. Über die Adresse 1/1/2 soll ein zweiter Anwender „USER2“ frei geschaltet werden. Über die Adresse 1/2/3 soll bei einem EIN Telegramm der gesamte VPN gesperrt werden.

```
[EibPC]
if after(systemstart(),500u64) then startvpn() endif
if "OpenUser1-1/1/1"==EIN then openvpnuser($User1$) else closevpnuser($User1$) endif
if "OpenUser2-1/1/2"==EIN then openvpnuser($User2$) else closevpnuser($User2$) endif
if "StopVPN-1/1/3"==EIN then stopvpn() endif
```

FTP

Der FTP Transfer schreibt Dateien auf einen externen FTP Server, wobei die maximale Dateigröße 64 kB ist.

Dazu können verschiedene Handles angelegt werden, die ihrerseits per Warteschlange gepuffert bis zu 64 kB große Dateien auf dem Server anlegen. Die Dateien werden per Timeout auch früher (und dann ggf. weniger Bytes) oder per flushftp vom User initiiert geschrieben.

Die Dateien werden von der Firmware automatisch nach Datum und Uhrzeit benannt.

Als Eingabe können Strings geschrieben werden. Die Datei ist demnach im ASCII Format und werden mit der Funktion sendftp in die Warteschlange geschrieben.

Dabei wird am Ende der Datenübermittlung von sendftp ein CR-LF (Zeilenumbruch für Windows geeignet) eingefügt. Ein Aufruf von sendftp kann mehrere Teilstrings übergeben, aber nicht mehr als 1400 Bytes insgesamt übernehmen. Es können nicht mehr als vier Handles definiert werden. Nicht zu verwechseln ist dies mit der zyklischen Auslagerung der KNX Telegramme (vgl. S 23).

Verbindung Konfigurieren

Definition

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

Argumente

- *server* vom Datentyp c1400
- *user* vom Datentyp c1400
- *password* vom Datentyp c1400
- *path* vom Datentyp c1400
- *timeout* vom Datentyp u32 in Sekunden

Wirkung

- Konfiguration eines FTP-Servers
- Aktualisierung der Abhängigkeiten bei Wertänderung oder während des möglichen Aufrufs der Systemstart-Funktion.
- Der FTP Transfer schreibt Dateien auf einen externen FTP Server, wobei die maximale Dateigröße 64 kB ist. Dazu können verschiedene Handles angelegt werden, die ihrerseits per Warteschlange gepuffert bis zu 64 kB große Dateien auf dem Server anlegen. Die Dateien werden per Timeout auch früher (und dann ggf. weniger Bytes) oder per flushftp() vom User initiiert geschrieben. Die Dateien werden von der Firmware automatisch nach Datum und Uhrzeit benannt.
- Es können nicht mehr als vier Handles definiert werden.

Rückgabewert

- Im Fehlerfall = 0
- bei Erfolg wird eine Handle-Nummer zwischen 1 und 4 zurückgegeben

Daten senden

Definition

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

Argumente

- *handle* vom Datentyp u08
- *data[x]* von beliebigem Datentyp, maximal 1400 Bytes

Wirkung

- Beliebige Daten werden in die Warteschlange des Handles geschrieben.
- Die Zuweisung erfolgt asynchron.

Rückgabewert

- bei Erfolg = 0
- Im Fehlerfall = 1

Zustand eines FTP-Handles Abfragen

Definition
- ftpstate(handle)

Argumente
- Argument *handle* vom Datentyp u08

Wirkung
- Gibt Informationen über den Status der FTP-Konfiguration zurück.

Rückgabewert
- u08
- Konfiguriert / fehlerfrei = 0
- Letzte Übertragung fehlerfrei = 1
- Server nicht erreichbar = 2
- Passwort/User nicht erlaubt = 3
- Fehler Verzeichnis nicht existent und konnte nicht angelegt werde = 4
- Warteschlange Überlauf, wenn vorher Fehler = 5
- Handle nicht definiert = 6

Zeit seit letzter Übertragung

Definition
- ftptimeout(handle)

Argumente
- Argument *handle* vom Datentyp u08

Wirkung
- Gibt die bereits verstrichene Zeit seit dem letzten Transfer in Sekunden zurück

Rückgabewert
- u32

Füllstand abfragen

Definition
- ftpbuffer(handle)

Argumente
- Argument *handle* vom Datentyp u08

Wirkung
- Gibt den Füllstand der Warteschlange des Transfers aus.

Rückgabewert
- u16

FTP-Übertragung sofort ausführen

Definition
- flushftp(handle)

Argumente
- Argument *handle* vom Datentyp u08

Wirkung
- Daten manuell auf den FTP-Server schreiben

Rückgabewert
- Erfolg = 0
- Server nicht erreichbar = 1
- Fehler beim Hochladen der Datei = 2
- Passwort/User nicht erlaubt = 3
- Fehler Verzeichnis nicht existent und konnte nicht angelegt werde = 4
- Übertragung wird eben durchgeführt (asynchrone Aktualisierung) = 5

HTTP-Request

Definition

- httprequest(*Typ*, *URL*, *Query*, *Header*, *Body*, *TLS*, *Timeout*, *Prioritaet*, *HTTP-Status*, *Reply-Header, Reply-Body*)

Argumente

- *Typ (*u08):
  GET=0u08, POST=1u08, PUT=2u08, DELETE=3u08, PATCH=4u08
- *URL* (c), maximal 256 Zeichen
  Format:
  `http[s]://[benutzer:passwort@]enertex.de[:Port]/kompletter/pfad`
- *Query* (c)
- *Header* (c)
- *Body* (c)
- *TLS (*b01):
  TLS_VERIFY_CERT=1b01, TLS_IGNORE_CERT=0b01
- *Timeout* (u08)
- *Prioritaet* (u08)
- *HTTP-Status* (u16)
- *Reply-Header* (c)
- *Reply-Body* (c)

Wirkung

- Sendet einen HTTP-Request an die übergebene Adresse.
- Verwenden Sie `https` statt `http` in der *URL* für eine verschlüsselte Übertragung.
- Hat *TLS* den Wert TLS_IGNORE_CERT, wird die Verbindung auch bei ungültigem Server-Zertifikat zugelassen
- Falls der Server eine Authentifikation benötigt, geben Sie Benutzernamen/Passwort als Teil der *URL* an (s.o.).
- Der Port kann nach dem Host angegeben werden. Anderenfalls werden die Standardports 80/443 für http/https verwendet (s.o.).
- Die Argumente *Query* werden durch & getrennt URL-codiert angegeben, z.b. `arg1=wert1&arg2=wert2`. Sie werden bei der Ausführung automatisch durch ? getrennt an die *URL* angehängt.
- Der *Body* wird ohne Veränderung übertragen. Die Codierung bzw. nötige Header (z.B. Content-Type) müssen selbst vorgenommen werden.
- *Header* ist eine durch LF getrennte Liste, z.b.
  `$Content-Type: application/json$+LF+$Accept: text/plain$`
  Standard: `User-Agent: Enertex EibPC2`
- Nach Ablauf von *Timeout* Sekunden wird die Anfrage abgebrochen und das Ergebnis entsprechend gesetzt. Bei Angabe von 0 wird der Standard-Timeout von 10 Sekunden verwendet.
- HTTP-Requests werden der Reihe nach bearbeitet. Durch Vergabe einer die *Priorität* können wichtige HTTP-Requests gegenüber weniger wichtigen HTTP-Requests priorisiert werden, z.B. das Schalten eines IoT-Gerätes gegenüber der Wetterabfrage. Die niedrigste Priorität hat den Wert 0u08, die Höchste den Wert 255u08
- Pro Sekunde werden maximal 10 HTTP-Requests abgearbeitet (max. 2 bei Firmware < 4.105).
- Es werden maximal 5 Weiterleitungen beachtet, falls der Server mit einem Status 3xx antwortet (keine bei Firmware < 4.108).
- Ab Firmware 4.110 können Weiterleitungen optional nicht beachtet werden: Addieren Sie 128 zum Parameter *Typ,* zB GET ohne Redirect: 128, POST ohne Redirect 129.
- Die Funktion überschreibt die Argumente *HTTP-Status, Reply-Header, Reply-Body* mit dem Ergebnis der Serverantwort, auch im Fehlerfall.
  Verwenden Sie deshalb immer eigene Rückgabevariablen, und nicht z.B. 0u16 oder $ $!

Rückgabewert (u08)

- 0u08 bei Erfolg
- 1u08: in Warteschlange eingereiht
- 2u08: Argumente ungültig
- 3u08: Fehler während der Ausführung
- 4u08: Ungültige URL oder keine Verbindung möglich
- 5u08: Zugriff nicht erlaubt, z.B. weil Authentifizierung nötig ist
- 6u08: Server-Zertifikat ungültig und Option TLS_IGNORE_CERT nicht gesetzt
- 7u08: Server hat nicht rechtzeitig geantwortet
- 8u08: Zu viele gleichzeitige HTTP-Requests (Grenze: 1000)
- 9u08: Zu viele HTTP-Weiterleitungen
- Der Rückgabewert erfolgt asynchron zur Hauptverarbeitungsschleife - siehe S.38.

Beispiel

Es soll täglich die die aktuelle Firmware-Version abgefragt und in einer Variablen gespeichert werden.

```
// 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
```

HTTP-Download

Diese Funktion ist sehr ähnlich zu httprequest. Statt die empfangenen Daten in einer Variable zu speichern, werden sie direkt im Flash-Speicher im Web-Verzeichnis */upload* abgelegt (siehe Dateien S. 24). Beachten Sie die maximale Dateigröße!

Definition

- httpdownload(*Typ*, *URL*, *Query*, *Header*, *Body*, *TLS*, *Timeout*, *Prioritaet*, *Dateiname*, *MaximaleDateigroesse, HTTP-Status*, *Reply-Header, Dateigröße, Ausführungsdauer*)

Argumente

- *Typ (*u08):
  GET=0u08, POST=1u08, PUT=2u08, DELETE=3u08, PATCH=4u08
- *URL* (c), maximal 256 Zeichen
  Format:
  `http[s]://[benutzer:passwort@]enertex.de[:Port]/kompletter/pfad`
- *Query* (c)
- *Header* (c)
- *Body* (c)
- *TLS (*b01):
  TLS_VERIFY_CERT=1b01, TLS_IGNORE_CERT=0b01
- *Timeout* (u08)
- *Prioritaet* (u08)
- *Dateiname* (cXXXXX): max. 32 Zeichen, darf nur die folgenden Zeichen enthalten: a-z, A-Z, 0-9, ".", "-"
- *MaximaleDateigroesse* (u32) in Byte
- *HTTP-Status* (u16)
- *Reply-Header* (c)
- *Dateigröße* (u32) in Byte
- *Ausführungsdauer* (u32) in ms

Wirkung

- Die Argumente sind analog zu httprequest.
- *Dateiname* Name der Zieldatei. Der Pfad zur Verwendung im Webserver ist */upload/Dateiname*.
- Ist die Datei größer als *Maximale-Größe*, wir die Übertragung abgebrochen.
- *Dateigröße* enthält bei Erfolg die Größe der übertragenen Datei in Byte.
- *Ausführungsdauer* enthält bei Erfolg die Dauer der Übertragung. Die Übertragungsgeschwindigkeit ergibt sich damit aus *Dateigröße*/*Ausführungsdauer.*
- Die Funktion überschreibt die Argumente *HTTP-Status, Dateigröße, Ausführungsdauer.* **Verwenden Sie deshalb immer eigene Rückgabevariablen, und nicht z.B. 0u16 oder $ $!**

Rückgabewert (u08)

- analog zu httprequest
- 10u08: Maximale Größe überschritten
- Der Rückgabewert erfolgt asynchron zur Hauptverarbeitungsschleife - siehe S.38.

Beispiel1

```
zURL= $https://enertex.de/assets/img/$
// Nicht für den Download nötig
zQuery= $$
zHeader =$$
zBody = $$
//
uTimeout = 60
uPrioritaet= 255
zDateiname= $logo-text.png$
// 10 MB
uMaximaleDateigroesse = 1024u32*1024u32*10u32
// Antworten:
uHTTPStatus=0u16
zReplyHeader=$$
uDateigroesseUebertragung=0u32
uAusfuehrungsdauer=0u32
uRueckgabe = 0


bDowloadStart='0/0/1'b01
if (bDowloadStart) then uRueckgabe=httpdownload(GET, zURL, zQuery, zHeader, zBody,
TLS_VERIFY_CERT, uTimeout, uPrioritaet, zDateiname, uMaximaleDateigroesse, uHTTPStatus,
zReplyHeader, uDateigroesseUebertragung, uAusfuehrungsdauer) endif
```

Beispiel2

Es soll alle 10 Sekunden das aktuelle Kamerabild von einer Hikvision DS-2CD2T86G2 Überwachungskamera heruntergeladen und in der Visualisierung mit dem picture-Element dargestellt werden.

Voraussetzung:

In der Kamera wurde ein Benutzer mit der Berechtigung "Remote Liveansicht" angelegt. Die WEB-Authentifizierung wurde auf "digest/basic" konfiguriert.

In **VISU** wurde ein picture-Element (S. 99) angelegt, mit dem **EXPERTEN** verknüpft und als "Eindeutige Variable" *uPicture* konfiguriert (Abbildung 40).



Abbildung 40: Konfiguration des Picture-Elements

```
uStatus=255
uHttpStatus=0u16
zHeader=$$
uSize=0u32
uDuration=0u32

if cycle(0,10) then {
   uStatus=httpdownload(GET,$http://user:password@192.168.1.10/ISAPI/Streaming/channels/1/picture$, $
$, $$, $$, TLS_IGNORE_CERT, 10, 4, $cam.jpg$, 1000000u32, uHttpStatus, zHeader, uSize, uDuration);
} endif

if uStatus==0 && uHttpStatus == 200u16 then {
   picture(uPicture, $Cam | $+CLOCKDATE_STRING+$ | $ + convert(uSize,$$) + $ b | geladen in $
+convert(uDuration, $$)+$ ms$, uPicture_P, $/upload/cam.jpg?$+convert(utctime(), $$))
} endif
```

HTTP-API

Der EibPC nimmt unter der Web-Adresse (URL)

`http(s)://<IP oder Hostname EibPC>/visu/api?key=`*Key*`&value=`*Value*

Befehle entgegen, um Objekte der EibPC Verarbeitung von anderen Geräten aus zu verändern. Die Objekte müssen vorher mit Hilfe der Funktion httpapi konfiguriert werden. Wenn ein HTTPS-Zertifikat vorhanden ist, ist die Adresse auch über HTTPS aufrufbar. Falls für die Web-Visualisierung ein HTTPS-Benutzer konfiguriert ist, betrifft dies auch die HTTP-API. Die Funktion wird automatisch beim Programmstart ausgewertet. Änderungen des *Keys* wirken sich beim nächsten Client-Aufruf aus.

Die vom Client verwendete HTTP-Request-Methode wird nicht ausgewertet. Es macht also keinen Unterschied, ob der Aufrufer GET, POST oder andere Methoden verwendet werden. Ebenso werden die übergebenen Header und der Body ignoriert.

Argumente beim API-Aufruf durch den Client werden, wie oben gezeigt, als URL Query-String übergeben und müssen korrekt URL-codiert sein, also bspw. %20 statt Leerzeichen verwenden. Passende Encoder finden sich auf diversen Webseiten.

Die API meldet an den Aufrufer immer einen HTTP-Status 200 zurück, wenn die Argumente korrekt empfangen wurden und an die Firmware weitergegeben wurden. Es wird dabei nicht geprüft, ob Zugriff auf diese Objekte besteht, der *Key* also mittels httpapi angelegt wurde. Existiert für eines dieser Objekte kein gültiger API-Zugriff, wird das Ereignis ERR_HTTPAPI_INVALID_KEY aufgezeichnet.

Definition

- httpapi(*Key, Value*)

Argumente

- *Key* (cXXXXX) max. 1400 Byte
- *Value* (cXXXXX) max. 1400 Byte

Wirkung

- Erstellt den HTTP-API-Aufruf
  `http(s)://<IP oder Hostname EibPC>/visu/api?key=`*<URL-codierter Key>*`&value=`*<URL-codierter Value>*
- Die vom Client übergebenen URL Query-Argumente werden für die Verarbeitung im EibPC automatisch URL-decodiert, d.h. *Key* und *Value* sind nicht (mehr) URL-codiert.
- Wird vom Client ein neuer Wert übergeben, wird die Variable *Value* und alle Abhängigkeiten aktualisiert.
- Um unabhängig vom gesendeten *Value* auf API-Aufrufe durch den Client zu reagieren, ändert die Funktion für einen Zyklus ihren Zustand auf 1 und aktualisiert ihre Abhängigkeiten (analog z.B. zu pbutton).

Rückgabewert (u08)

- 1u08 für einen Zyklus, wenn ein Client die URL (s.o.) mit dem passenden *Key* aufgerufen hat

Beispiel

Beim Klingeln einer Türsprechstelle soll ein KNX-Telegramm ausgelöst werden, um die Türklingel zu aktivieren. In der Konfiguration der Türsprechstelle wird der folgende HTTP-Aufruf konfiguriert:
GET `http://eibpc2/visu/api?key=`*T%C3%BCrklingel*`&value=`*1*

```
zArgument=$$c1
if httpapi($Türklingel$,zArgument) == 1 and convert(zArgument, 0b01)==EIN then write("0/0/1-Glocke", EIN)
endif
```

Modbus TCP

Der EibPC² kann als Modbus TCP Master und Slave verwendet werden, d.h. er kann auf die Ressourcen anderer Geräte zugreifen, und kann eigene Objekte anderen Geräten zur Verfügung stellen.

Modbus-Ressourcen werden unterschieden in

- MB_COIL: 1 Bit, Adressbereich 1-9999
- MB_DISCRETE_INPUT: 1 Bit, nur lesbar, Adressbereich 10001-19999
- MB_INPUT_REGISTER: 16 Bit, nur lesbar, Adressbereich 30001-39999
- MB_HOLDING_REGISTER: 16 Bit, Adressbereich 40001-49999

Für die Adressierung der Ressourcen wird von den unten stehenden Funktionen eine 0-basierte Adressierung der Register und eine explizite Angabe des Adressbereichs verwendet. Für einen Zugriff auf das erste Holding Register wird somit MB_HOLDING_REGISTER mit Index 0 verwendet.

Byte-Reihenfolge

Modbus-Ressourcen sind entweder 1 Bit oder 16 Bit breit. Beim Lesen, Schreiben und beim Modbus Slave werden diese auf Objekte des EibPC abgebildet. Die Abbildung für b01-Objekte auf MB_DISCRETE_INPUT oder MB_COIL erfolgt direkt, es wird nur auf das eine Bit zugegriffen. Bei 16-Bit-Datentypen, z.B. u16 analog beim Zugriff auf MB_INPUT_REGISTER oder MB_HOLDING_REGISTER.

Beim Zugriff auf Register ist zudem die Byte-Reihenfolge, die sog. Endianess, relevant. Diese gibt an, ob das höherwertige (Big Endian) oder das niederwertige Byte (Little Endian) zuerst im Speicher steht.

Der Registerwert 0x1234 (dezimal 4660) besteht aus zwei Bytes 0x12 und 0x34. Wird der Wert im Modbus-Gerät als 0x3412 (Little Endian) abgespeichert und soll vom EibPC richtig als 0x1234 interpretiert werden, kann durch Setzen des Arguments Byte-Reihenfolge auf LITTLE_ENDIAN die Interpretation entsprechend angepasst werden.

Wort-Reihenfolge

Ist der EibPC-Datentyp größer als die Modbus-Ressource, wird automatisch auf mehrere zusammenhängende Ressourcen zugegriffen. So können also z.B. 8 einzelne 1-Bit-Register als u08 gelesen werden. Die Reihenfolge der einzelnen Worte (hier der einzelnen Bits, alternativ: 16 Bit bei Registerzugriff) wird durch das Argument *Wort-Reihenfolge* bestimmt. Einer Ressource mit niedrigerem Index wird bei BIG_ENDIAN eine höhere Stelle im Ergebnis zugeordnet.

Die Bits 1, 0, 0, 1, 1, 0, 0, 0 ab Index 7 werden bei Verwendung von BIG_ENDIAN als Binärzahl 10011000 oder hex 0x98 oder dezimal 152 interpretiert, bei Verwendung von LITTLE_ENDIAN als Binärzahl 00011001 oder 0x19 oder dezimal 25.

Master

Das Prinzip ist ähnlich zu den FTP-Funktionen (S. 193). Es wird ein Modbus Master-Handle erzeugt, das anschließend zum Lesen und Schreiben verwendet wird. Das Handle speichert intern die Verbindungsinformationen. Falls die Verbindung unterbrochen wird, wird diese automatisch wieder aufgebaut.

Definition

- modbusmaster(*Host*, *Port*, *Timeout*, *SlaveAdresse*)

Argumente

- *Host* (c)
- *Port* (u16)
- *Timeout* (u32)
- *SlaveAdresse* (u08)

Wirkung

- Erstellt einen Modbus-TCP Handle zur Verwendung von den Funktionen readmodbus und writemodbus
- Als *Host* kann eine IP-Adresse oder ein Hostname angegeben werden. Dieser wird bei der Auswertung der Funktion aufgelöst.
- Der Modbus TCP Standard*port* ist 502u16.
- Der *Timeout* in Sekunden gibt an, wie lange auf eine Antwort beim Lesen oder Schreiben geantwortet werden soll.
- Pro Sekunde werden maximal 10 Lese- oder Schreibanforderungen abgearbeitet (maximal 2 pro Sekunde für Firmware < 4.106).
- Die meisten Geräte verwenden als *SlaveAdresse* die Adresse 1u08 oder 255u08. Der Modbusmaster kommuniziert dabei mit dieser Gerätezieladresse (*SlaveAdresse*).

Ergebnis (u08)

- 0u08 bei Fehler
- Modbus Master-Handle (u08 > 0u08), der für die Funktionen readmodbus und writemodbus verwendet werdet wird.

Master Timings anpassen

Manche Modbus-Geräte erfordern zusätzliche Wartezeiten für die Kommunikation. So kann eine Wartezeit vor dem ersten Lese- und Scheibanforderungen nach dem Verbindungsaufbau erforderlich sein, eine Wartezeit zwischen Lesen und Scheibanforderungen und nach erneutem Verbindungsaufbau/anfrage notwendig werden. Mit dieser Funktion kann dies parametriert werden.

Definition

- modbusmastertimings(*MasterHandle*, *NachVerbindungsaufbau*, *ZwischenRequests*, *Reconnect*)

Argumente

- *MasterHandle* (u08)
- *NachVerbindungsaufbau* (u16)
- *ZwischenRequests* (u16)
- *Reconnect* (u16)

Wirkung

- Verändert die übergebenen Wartezeiten in ms für alle nachfolgend ausgeführten Lese- und Schreibrequests, einschließlich Aufrufe von readmodbus und writemodbus, die bereits in der Warteschlange befinden.
- Wird die Funktion nicht verwendet, sind die Wartezeiten 0 ms, d.h. es wird nicht extra gewartet.
- Konfigurierte Verzögerungen beeinflussen auch die Verarbeitungsgeschwindigkeit anderer Handles.
- *NachVerbindungsaufbau* verändert die Zeit zwischen dem erfolgreichen Verbindungsaufbau und dem Senden des ersten Lese- oder Schreibbefehls
- *ZwischenRequests* verändert die Zeit zwischen zwei aufeinanderfolgenden Befehlen. Ist diese noch nicht abgelaufen, wird der nachfolgende Befehl entsprechend verzögert.
- *Reconnect* verändert die Zeit zwischen dem Beenden einer (fehlerhaften) Verbindung und dem Aufbau einer neuen Verbindung.

Ergebnis (u08)

- 0u08 bei Erfolg
- 1u08 bei Fehler

Beispiel:

Modbusverbindung erstellen und Timeouts anpassen

```
// Modusbusverbindung konfigurieren
zServer=$modbusserver$
uPort=502u16
uTimeout=10u32
uZielSlaveAdresse=16
FD1=modbusmaster(zServer, uPort, uTimeout,uZielSlaveAdresse)
// Detailierte Timings erstellen (all in ms)
uNachVerbindungsaufbau=20u16
uZwischenRequests=110u16
uReconnect=32u16
x=modbusmastertimings(FD1, uNachVerbindungsaufbau, uZwischenRequests,  uReconnect)
```

Ressource lesen

Definition

- readmodbus(*MasterHandle*, *Typ*, *Index*, *Ausgabeobjekt*, *ByteReihenfolge*, *WortReihenfolge, [SlaveAdresse]*)

Argumente

- *MasterHandle (*u08)
- *Typ* (u08)
- *Index* (u16)
- *Ausgabeobjekt* (Variable vom Typ b01, b02, b04, u08, s08, u16, s16, f16, u24, u32, s32, f32, u64, s64, cXXXXX)
- *ByteReihenfolge* (u08)
- *WortReihenfolge* (u08)
- *SlaveAdresse* (u08) optional

Wirkung

- Liest die Ressource vom *Typ* und *Index* und schreibt das Ergebnis in *Ausgabeobjekt.*
- Der *Typ* muss einer der vordefinierten Typen MB_DISCRETE_INPUT, MB_COIL, MB_INPUT_REGISTER, MB_HOLDING_REGISTER sein.
- Die Reihenfolge der Bits, bzw. der Bytes bei der Abbildung auf das *Ausgabeobjekt* wird durch die Argumente *Byte-Reihenfolge* und *Wort-Reihenfolge* festgelegt.
- Wird *SlaveAdresse* angegeben, wird wir diese Anfrage *SlaveAdresse* statt der im modbusmaster definierten verwendet. Die Adresse für andere (nachfolgende) Anfragen wird nicht geändert.
- Besonderheiten *Ausgabeobjekt* mit Typ cXXXXX:
  - Die Anzahl der gelesenen Register richtet sich nach der Länge der Zeichenkette, z.B. werden bei c34 insgesamt 17 Register zu je 2 Byte gelesen. Bei ungeraden Längen wird abgerundet.
  - Alle ASCII-Steuerzeichen werden in der Ausgabe übersprungen
  - Die *Byte-Reihenfolge* beeinflusst die Reihenfolge der Bytes innerhalb eines Registers, die *Wort-Reihenfolge* die Anordnung der Register in der Zeichenkette. Bei LITTLE_ENDIAN wird jeweils die Reihenfolge der Bytes vertauscht, bzw. die Reihenfolge der einzelnen 2-Byte-Worte in der Zeichenkette umgekehrt.
  - Es sind nur Zugriffe auf MB_INPUT_REGISTER und MB_HOLDING_REGISTER erlaubt.
- Die Aktualisierung der Rückgabe und vom *Ausgabeobjekt* erfolgt asynchron zur Verarbeitung.

Rückgabewert (u08)

- 0u08 bei Erfolg
- 1u08 bei laufender Verarbeitung
- 2u08 bei Fehler

Beispiel

```
Es sollen alle 10 Sekunden die Wirkleistung und der aktuelle Ladezustand eines
Batteriespeichers ausgelesen und einer Variable zugewiesen werden. Die Slave-
Adresse (unit ID) ist 255, der Port ist 502 (Standard).
```

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		✔	✔	✔	✔	✔

Abbildung 41: Modbus-Register eines Batteriespeichers (Quelle: 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
```

Ressource schreiben

Definition

- writemodbus(*MasterHandle*, *Typ*, *Index*, *Quellobjekt*, *ByteReihenfolge*, *WortReihenfolge, [SlaveAdresse]*)

Argumente

- *MasterHandle (*u08)
- *Typ* (u08)
- *Index* (u16)
- *Quellobjekt* (b01, b02, b04, u08, s08, u16, s16, f16, u24, u32, s32, f32, u64, s64, cXXXXX)
- *ByteReihenfolge* (u08)
- *WortReihenfolge* (u08)
- *SlaveAdresse* (u08) optional

Wirkung

- Schreibt den aktuellen Wert von *Quellobjekt* auf die Ressource vom *Typ* ab *Index*.
- Der *Typ* muss einer der vordefinierten Typen MB_COIL oder MB_HOLDING_REGISTER sein.
- Die Reihenfolge der Bits, bzw. der Bytes bei der Abbildung von *Quellobjekt* wird durch die Argumente *ByteReihenfolge* und *WortReihenfolge* festgelegt.
- Wird *SlaveAdresse* angegeben, wird wir diese Anfrage *SlaveAdresse* statt der im modbus-master definierten verwendet. Die Adresse für andere (nachfolgende) Anfragen wird nicht geändert.
- Besonderheiten *Quellobjekt* mit Typ cXXXXX:
  - Die Anzahl der geschriebenen Register richtet sich nach der Länge der Zeichenkette, nicht nach der Länge der gespeicherten Zeichenkette. Bei c34 werden insgesamt 17 Register zu je 2 Byte geschrieben. Bei ungeraden Längen wird abgerundet.
  - Es werden alle Bytes gesendet.
  - Die *ByteReihenfolge* beeinflusst die Reihenfolge der Bytes innerhalb eines Registers, die *WortReihenfolge* die Anordnung der Register in der Zeichenkette. Bei LITTLE_ENDIAN wird jeweils die Reihenfolge der Bytes vertauscht, bzw. die Reihenfolge der einzelnen 2-Byte-Worte in der Zeichenkette umgekehrt.
  - Es sind nur Zugriffe auf MB_HOLDING_REGISTER erlaubt.
- Die Aktualisierung der Rückgabe erfolgt asynchron zur Verarbeitung.

Rückgabewert (u08)

- 0u08 bei Erfolg
- 1u08 bei laufender Verarbeitung
- 2u08 bei Fehler

Beispiel

```
Es soll der Skalierungsfaktor der Wirkleistung des obigen Batteriespeichers
geändert werden, um den Wert in kW abzufragen.
```

2066	active power SF	WR	SINT16	1	-	exponent for active power	active power = (value in #1066) * 10 ^ (value in #2066)	✘	✘	✔	✘	✔

Abbildung 42: Modbus-Register eines Batteriespeichers (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

Bei der Verwendung als Modbus TCP Slave stellt der EibPC² ausgewählte interne Objekte anderen Modbus TCP-Mastern zur Verfügung. Ändert sich ein Objekt, wird beim In einem festgelegten Intervall von 5 Sekunden werden die aktuellen Objektinhalte zwischengespeichert und sind als Modbus Ressourcen abrufbar. Die Anzahl gleichzeitiger Verbindungen von Modbus TCP Mastern zum EibPC² ist auf 4 begrenzt.

Der TCP-Port für den Verbindungsaufbau kann in den Projekteinstellungen (S. 22) konfiguriert werden.

Allen Modbus TCP-Mastern stehen die gleichen Ressourcen zur Verfügung. Beim Schreiben auf Ressourcen erfolgt die Aktualisierung des zugehörigen Objekts asynchron zur Verarbeitung. Ist der Typ MB_DISCRETE_INPUT oder MB_INPUT_REGISTER, so darf nicht schreibend darauf zugegriffen werden

Definition

- modbusslave(*Typ*, *Index*, *Quellobjekt*, *Byte-Reihenfolge*, *Wort-Reihenfolge*)

Argumente

- *Typ* (u08)
- *Index* (u16)
- *Quellobjekt* (Variable vom Typ b01, b02, b04, u08, s08, u16, s16, f16, u24, u32, s32, f32, u64, s64)
- *Byte-Reihenfolge* (u08)
- *Wort-Reihenfolge* (u08)

Wirkung

- Stellt das *Quellobjekt* als Modbus-Ressource vom *Typ* an *Index* zum Auslesen durch Modbus TCP Master zur Verfügung
- Der *Typ* muss einer der vordefinierten Typen MB_DISCRETE_INPUT, MB_COIL, MB_INPUT_REGISTER, MB_HOLDING_REGISTER sein
- Die Reihenfolge der Bits, bzw. der Bytes bei der Abbildung vom *Quellobjekt* wird durch die Argumente *Byte-Reihenfolge* und *Wort-Reihenfolge* festgelegt.
- Die Aktualisierung der Rückgabe erfolgt asynchron zur Verarbeitung.

Rückgabewert (u08)

- 0u08 wenn die Modbus-Ressource korrekt erstellt wurde
- 1u08 bei laufender Verarbeitung
- 2u08 bei Fehler

Beispiel

```
Der EibPC soll von einem Modbus TCP-Master abgefragt werden. Dazu soll an
Register Adresse 0 ein 1-Bit-Wert und an Register Adresse 100/101 (zwei
aufeinander folgende 16-Bit-Register) ein 32-Bit Wert abgefragt werden können.
flag=1b01
val=0x12345678u32
modbusslave(MB_COIL, 0u16, flag, BIG_ENDIAN, BIG_ENDIAN);
modbusslave(MB_INPUT_REGISTER, 100u16, val, BIG_ENDIAN, BIG_ENDIAN);
```

MQTT

Der EibPC² mit Option NP unterstützt das MQTT-Protokoll zum einfachen Datenaustausch mit anderen Geräten (Firmware > 5.100). Der EibPC² bietet einen MQTT-Broker, der Nachrichten entgegennimmt und an andere Clients verteilt.

Um Nachrichten von anderen MQTT-Clients in der Verarbeitung nutzen zu können, um sie z.B. als KNX-Telegramm auf dem Bus weiterzuleiten, muss ein MQTT-Client konfiguriert werden. Dies gilt auch, wenn der EibPC² selbst der MQTT-Broker ist.

MQTT Broker

Definition

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

Argumente

- *Port* (u16) Standardport ist 1883u16 ohne TLS, 8883 mit TLS
- *TLS* (b01) Aktiviert Verschlüsselung mit TLS
- *Username* (c) Benutzername zur Authentifikation oder leere Zeichenkette
- *Password* (c) Passwort zur Authentifikation oder leere Zeichenkette

Wirkung

- Startet den integrierten MQTT-Broker.
- Falls *TLS* aktiviert (=1b01) ist, wird die Kommunikation mit TLS verschlüsselt. Als Serverzertifikat wird das Webserver-Zertifikat verwendet.
- Wenn *Username* und/oder *Password* leere Zeichenketten sind, wird die Authentifikation deaktiviert.
- Die Aktualisierung der Rückgabe erfolgt asynchron zur Verarbeitung.
- Es können sich max. 100 Clients gleichzeitig am Broker anmelden.
- Falls der Broker bereits läuft, wird er nur neu gestartet, falls sich *Port* oder *TLS* geändert haben. Anderenfalls wird nur die Benutzerkonfiguration neu gelesen.

Rückgabewert (u08)

- 0u08 wenn der MQTT-Broker korrekt gestartet wurde und läuft
- 1u08: wird gerade gestartet
- 2u08: ist gestoppt
- 3u08: Start fehlgeschlagen, z.B. kein Zertifikat aber *TLS* aktiviert
- 4u08: Konfigurationsfehler
- 5u08: Konfiguration wurde neu geladen
- Der Rückgabewert erfolgt asynchron zur Hauptverarbeitungsschleife - siehe S.38.

Beispiel

```
Der EibPC soll beim Start den MQTT-Broker starten. TLS-Verschlüsselung ist
deaktiviert, aber Clients müssen sich mit Benutzername und Passwort
(eibpc:geheim) anmelden.
uBrokerStatus=255
if systemstart() then uBrokerStatus=startmqttbroker(1883u16, 0b01, $eibpc$,
$geheim$) endif
```

Definition

- stopmqttbroker()

Argumente

- keine

Wirkung

- Stoppt den laufenden MQTT-Broker

Rückgabewert (u08)

- keiner

MQTT-Client

Definition

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

Argumente

- *Host* (c) Hostname oder IP-Adresse als Zeichenkette
- *Port* (u16) Standardport ist 1883u16 ohne TLS, 8883 mit TLS
- *TLS* (b01) Aktiviert Verschlüsselung mit *TLS*
- *Username* (c) Benutzername zur Authentifikation oder leere Zeichenkette
- *Password* (c) Passwort zur Authentifikation oder leere Zeichenkette
- *ValidateServerCert* (b01) TLS_VERIFY_CERT oder TLS_IGNORE_CERT
- *CACert* (c) Wurzelzertifikat, um Gültigkeit des Server-Zertifikats zu prüfen, PEM-Format
- *ClientCert* (c) Client Zertifikat, PEM-Format
- *ClientKey* (c) Unverschlüsselter Private Key des Client Zertifikats, PEM-Format

Wirkung

- Erstellt ein MQTT-Client-Verbindungshandle. Es sind insgesamt bis zu vier Handles möglich.
- Die Verbindung wird beim Zugriff automatisch aufgebaut, falls nötig. Falls beim Verbindungsaufbau ein Fehler auftritt, z.B. weil der Broker nicht erreichbar ist, wird bis zum nächsten Versuch 60 Sekunden gewartet.
- Ist *Username* oder *Password* leer, erfolgt keine Authentifikation
- Ist *ValidateServerCert* auf TLS_VERIFY_CERT=1b01 gesetzt, wird die Adresse des Server-Zertifikat überprüft. Nur gültig wenn TLS verwendet wird. Abgelaufene oder selbstsignierte Zertifikate werden auch mit TLS_VERIFY_CERT=0b01 nicht akzeptiert.
- Ist *CACert* leer, wird gegen die integrierte Zertifikatsliste validiert, falls TLS aktiv ist.
- Wird ein *ClientCert* und *ClientKey* angegeben, authentifiziert sich der Client zusätzlich per Zertifikat, falls TLS aktiv ist.
- Als MQTT-Client-ID wird "eibpc-<seriennummer>-<handle>" verwendet.

Rückgabewert (u08)

- 0u08 bei Fehler
- MQTT-Handle (u08 > 0u08), der für die Funktionen subscribemqtt, unsubscribemqtt, publishmqtt verwendet werdet wird.

Definition

- subscribemqtt(*Handle*, *Topic*, *QualityOfService*, *Ausgabeobjekt*, *[AusgabeTopic]*)

Argumente

- *Handle* (u08) Verbindungshandle von mqttclient
- *Topic* (c) Topic ggf. mit Wildcards (+, #)
- *QualityOfService* (u08) Gültige Werte: 0u08 (QoS 0), 1u08 (QoS 1), 2u08 (QoS 2)
- *Ausgabeobjekt* (Variable vom Typ b01, b02, b04, u08, s08, u16, s16, f16, u24, u32, s32, f32, u64, s64, cXXXXX)
- *AusgabeTopic* (c) optional

Wirkung

- Abonniert ein MQTT-Topic.
- Das Topic kann Wildcards enthalten:
  - sensors/+/temp für einzelne Level beliebigen Wertes
  - sensors/# für alle Topics beliebiger (Sub)-Levels. # muss das letzte Zeichen sein.
- Die Verbindung wird beim Zugriff automatisch aufgebaut, falls nötig.
- *QualityOfService* steuert die Zuverlässigkeit der Zustellung:
  - QoS 0: maximal einmal
  - QoS 1: mindestens einmal
  - QoS 2: genau einmal
- Das *Ausgabeobjekt* wird bei jeder neuen Nachricht aktualisiert, falls sich diese vom aktuellen Zustand des Objekts unterscheidet.
- Bei Änderung von *Ausgabeobjekt* wird, falls angegeben, *AusgabeTopic* mit dem Topic aktualisiert, mit dem die Nachricht an den Broker gesendet wurde. Dadurch kann bei Wildcard-Subscriptions unterscheiden werden, an welches Topic die Nachricht adressiert wurde.
- Falls möglich, wird der Datentyp entsprechend des Ausgabeobjekts decodiert. Viele Geräte senden Werte als Zeichenkette. In diesem Fall muss das *Ausgabeobjekt* ebenfalls eine Zeichenkette ausreichender Größe sein. Der Wert kann anschließend z.B. mit parsejson oder convert konvertiert verarbeitet werden.
- Es können maximal 1000 Subscriptions aktiv sein (Firmware 5.107: max. 100).

Rückgabewert (u08)

- 0u08 Erfolg
- 1u08 Fehler
- 2u08 Subscription existiert bereits
- 3u08 Maximale Anzahl Subscriptions erreicht

Beispiel

```
Der MQTT-Broker auf dem EibPC ist aktiviert. Änderungen an einem MQTT-Topic
sollen auf eine Gruppenadresse abgebildet werden.
uMqttHandleEibPC = mqttclient($localhost$, 1883u16, AUS, $eibpc$, $geheim$,
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, Ausgabeobjekt*)

Argumente

- *Handle* (u08) Verbindungshandle von mqttclient
- *Topic* (c) Topic aus subscribemqtt
- *Ausgabeobjekt* Objekt aus subscribemqtt

Wirkung

- Entfernt das Abonnement vom MQTT-Topic für das Ausgabeobjekt. Andere Abonnements (auch des gleichen Topics) für andere Ausgabeobjekte bleiben bestehen.
- Das *Ausgabeobjekt* wird nicht geändert und dient nur der Zuordnung der Abonnements.

Rückgabewert (u08)

- 0u08 bei Erfolg
- 1u08 bei Fehler

Definition

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

Argumente

- *Handle* (u08) Verbindungshandle von mqttclient
- *Topic* (c) Topic ohne Wildcards
- *QualityOfService* (u08) siehe 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

Wirkung

- Sendet *Objekt* an den MQTT-Broker.
- Die Nachricht enthält die in *Objekt* gespeicherten Daten, ggf. auf eine Länge von *Size* Bytes verkürzt.
- *Size* gibt die Länge der Daten in Byte an, die gesendet werden sollen. Falls *Size* == 0u16, wird bei numerischen Datentypen das gesamte Objekte gesendet, bei Strings die tatsächliche Länge der Zeichenkette (size*(Objekt)*)
- *Retain* weist den Broker an, die Nachricht zu speichern und automatisch an neue Subscriber eines passenden Topics zu senden.

Rückgabewert (u08)

- 0u08 bei Erfolg
- 1u08 bei Fehler

Beispiel

```
Der MQTT-Broker auf dem EibPC ist aktiviert. Änderungen an einer Gruppenadresse
sollen als MQTT-Topic veröffentlicht werden.
uMqttHandleEibPC = mqttclient($localhost$, 1883u16, AUS, $eibpc$, $geheim$,
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
```

Visualisierung

Um die Web-Visualisierung des EibPC nutzen zu können, müssen Sie die Option NP im EibPC freischalten. Der Freischaltcode ist immer an die Seriennummer des Gerät gebunden und ist nicht übertragbar auf andere Geräte.

Die folgenden Funktionen werden verwendet, um auf Visualisierungselemente zuzugreifen.

Visualisierungselemente werden unterschieden in globale und seitenbezogene Elemente (siehe S. 18).

Schaltelemente

Visualisierungen, die über **VISU** erstellt werden, verwenden stets seitenbezogene Elemente, falls verfügbar. Wie eine Web-Visualisierung in **EXPERTE** selbst erstellt wird, ist in Visualisierung (S. 76) beschrieben.

Schaltfläche gedrückt (global)

Definition

- button(*ID*)
- identisch zu webbutton früherer Releases

Argumente

- *ID* vom Datentyp u08. Darf sich zur Laufzeit des Programms nicht ändern.

Wirkung

- Bei „Betätigen“ bzw. Drücken der Schaltfläche eines Webbuttons (z.B. *button* oder *shifter*) mit der *ID* (Seite 85 ff.) geht die Funktion für einen Verarbeitungszyklus auf einen Wert ungleich 0. In allen anderen Fällen ist der Rückgabewert 0.
- Bei einem *button*-Element wechselt bei Betätigung die Rückgabe auf 1.
- Bei einem *shifter*-Element wechselt bei Betätigung die Rückgabe auf 1, 2, 3 oder 4 (u08), je nachdem, welches Schaltelement des Webbuttons betätigt wird. Die Nummern beziehen sich dabei auf die Anordnung der Schaltelemente von links nach rechts

Rückgabewert

- u08, Werte 0,1,2,3,4.

Schaltfläche gedrückt (seitenbezogen)

Definition

- pbutton(*ID,PageID*)

Argumente

- Argument *ID* vom Datentyp u08. Darf sich zur Laufzeit des Programms nicht ändern.
- Argument *PageID* vom Datentyp u08. Darf sich zur Laufzeit des Programms nicht ändern.

Wirkung

- Bei „Betätigen“ bzw. Drücken der Schaltfläche eines seitenbezogenen Buttons (z.B. *pbutton* oder *pshifter*) mit der *ID* (Seite 85 ff.) auf der Webseite mit der *PageID* geht die Funktion für einen Verarbeitungszyklus auf einen Wert ungleich 0. In allen anderen Fällen ist der Rückgabewert 0.
- Bei einem *pbutton*-Element wechselt bei Betätigung die Rückgabe auf 1.
- Bei einem *pshifter*-Element wechselt bei Betätigung die Rückgabe auf 1, 2, 3 oder 4 (u08), je nachdem, welches Schaltelement des Webbuttons betätigt wird. Die Nummern beziehen sich dabei auf die Anordnung der Schaltelemente von links nach rechts.

Rückgabewert

- Typ u08, Werte 0,1,2,3,4.

Enertex® Bayern GmbH – Ebermannstädter Straße 8 - 91301 Forchheim - mail@enertex.de

Mehrfachauswahl abfragen (global)

Definition

- mbutton(*ID, Auswahl*)

Argumente

- *ID* vom Datentyp u08. Darf sich zur Laufzeit des Programms nicht ändern.
- *Auswahl* vom Datentyp u08.

Wirkung

- Bei „Betätigen“ bzw. Drücken der Schaltfläche eines Multibuttons und der gegebenen Auswahl mit Index *Auswahl* (z.B. *mbutton* oder *mshifter*) mit der *ID* (Seite 85 ff.) geht die Funktion für einen Verarbeitungszyklus auf den Wert *Auswahl*. *Auswahl* ist dabei ein Wert 1, 2 ... 254 und bezeichnet die Stelle im Auswahlfeld, an welcher das Auswahlelement steht.
- Wenn das Auswahlfeld betätigt wurde, ist der Rückgabewert 255. In allen anderen Fällen ist die Rückgabe 0.
- Bei einem *mbutton*-Element wechselt bei Betätigung die Rückgabe auf 1.
- Bei einem *mshifter*-Element wechselt bei Betätigung die Rückgabe auf 1, 2, 3 oder 4 (u08), je nachdem, welches Schaltelement des Webbuttons betätigt wird. Die Nummern beziehen sich dabei auf die Anordnung der Schaltelemente von links nach rechts.

Rückgabewert

- u08.

Mehrfachauswahl (seitenbezogen)

Definition

- mpbutton(*ID, Auswahl,PageID*)

Argumente

- *ID* vom Datentyp u08. Darf sich zur Laufzeit des Programms nicht ändern.
- *PageID* vom Datentyp u08. Darf sich zur Laufzeit des Programms nicht ändern.
- *Auswahl* vom Datentyp u08.

Wirkung

- Bei „Betätigen“ bzw. Drücken der Schaltfläche eines seitenbezogenen Multibuttons und der aktiven Auswahl mit Index *Auswahl* (z.B. *mpbutton* oder *mpshifter*) mit der *ID* (Seite 85 ff.) auf der Webseite mit der *PageID* geht die Funktion für einen Verarbeitungszyklus auf den Wert 1. *Auswahl* ist dabei ein Wert 1, 2 ... 254 und bezeichnet den Index im Auswahlfeld, an welcher das Auswahlelement steht. Wenn die Auswahl auf *Auswahl* geändert wurde ist der Rückgabewert 255. In allen anderen Fällen ist die Rückgabe 0.
- Bei einem *mpbutton*-Element wechselt bei Betätigung die Rückgabe auf 1.
- Bei einem *mpshifter*-Element wechselt bei Betätigung die Rückgabe auf 1, 2, 3 oder 4 (u08), je nachdem, welches Schaltelement des Webbuttons betätigt wird. Die Nummern beziehen sich dabei auf die Anordnung der Schaltelemente von links nach rechts.

Rückgabewert

- u08

Schaltfläche ändern (global)

Definition

- display(*ID, Text, Icon, State, TextStil,[Mbutton]*)
- webdisplay(*ID, Text, Icon, State, TextStil,[Mbutton]*)

Argumente

- *ID, Icon, State, Textstil* und *Mbutton* vom Datentyp u08
- *Text* beliebiger Datentyp

Wirkung

- Die Funktion spricht den (*button, shifter*) an. Wenn *ID* mehrfach vorhanden ist, dann werden alle Elemente dieser ID angesprochen.
- Mit Hilfe des optionalen Arguments *Mbutton* kann die angezeigte Auswahl des Dropdownmenüs bei mbutton, mshifter verändert werden.
- Bei Aufruf der Funktion wird das Icon des Webelements mit der *ID* auf das Symbol mit der Nummer (Typ u08) *Icon* gesetzt. Mögliche Grafiken sind in Tabelle 3 (Seite 68) aufgelistet.
- Das Argument *Text* bezeichnet eine beliebige Variable, deren in einen String gewandelter Wert auf die variable Textzeile des Webelements ausgegeben wird.
- Jedes Icon hat mindestens die Zustände ACTIVE (==1), INACTIVE (==2), DARKRED (==0) und BRIGHTRED (==9). Einer dieser Zustände kann im Argument *State* übergeben werden. Eine Übersicht über die möglichen Zustände gibt Tabelle 2 (Seite 45).
- Der auszugebende Text kann in den Stilen GREY (==0), GREEN (==1), BLINKRED(==2) und BLINKBLUE (==3) dargestellt werden.

Rückgabewert

- keine

Beispiel Uhrzeit anzeigen

An einem *button*-Element soll die Uhrzeit angezeigt werden.

Umsetzung im Anwenderprogramm:

```
[WebServer]
button(ClockWebID)[CLOCK]$Uhrzeit$
[EibPC]
ClockWebID=0
if stime(0) then webdisplay(ClockWebID,settime(),CLOCK,INACTIVE,GREY) endif
```

Hinweis:

1. Der Rückgabetyp von *settime*() ist t24. Er wird in diesem Fall in einen String der lesbaren Form „Fr. 12:33:55“ gewandelt.
2. Sie können auf u08 Variablendefinitionen für die ID zugreifen. Diese müssen jedoch durch eine Konstante initialisiert werden und dürfen sich zur Laufzeit nicht ändern.

Schaltfläche ändern (seitenbezogen)

Definition

- pdisplay(*ID, Text, Icon, State, TextStil, PageID, [Mbutton]*)

Argumente

- *ID* vom Datentyp u08.
- *PageID* vom Datentyp u08.
- *Icon, State* und *Textstil* vom Datentyp u08
- *Text* beliebiger Datentyp

Wirkung

- Die Funktion spricht den (*pbutton* oder *pshifter*) an. Wenn *ID* mehrfach vorhanden ist, dann werden alle Elemente dieser ID auf der Webseite mit der *PageID* angesprochen.
- Mit Hilfe des optionalen Arguments *Mbutton* kann die angezeigte Auswahl des Dropdownmenüs verändert werden.
- Bei Aufruf der Funktion wird das Icon des Webelements mit der *ID* auf der Webseite mit der *PageID* auf das Symbol mit der Nummer (Typ u08) *Icon* gesetzt. Mögliche Grafiken sind auf Seite 45 dargestellt und werden über vordefinierte Zahlen (u08) ausgewählt. Zudem erleichtern vordefinierte Konstanten die Auswahl. Die Zuordnung hierfür ist in Tabelle 3 (Seite 68) aufgelistet.
- Das Argument *Text* bezeichnet eine beliebige Variable, deren in einen String gewandelter Wert auf die variable Textzeile des Webelements ausgegeben wird.
- Jedes Icon hat mindestens die Zustände ACTIVE (==1), INACTIVE (==2), DARKRED (==0) und BRIGHTRED (==9). Einer dieser Zustände kann im Argument *State* übergeben werden. Eine Übersicht über die möglichen Zustände gibt Tabelle 2 (Seite 45).
- Der auszugebende Text kann in den Stilen GREY (==0), GREEN (==1), BLINKRED(==2) und BLINKBLUE (==3) dargestellt werden.

Rückgabewert

- keine

Slider

Wert abfragen(global)

Definition

- getslider(*ID*)

Argumente

- Argument *ID* vom Datentyp u08. Darf sich zur Laufzeit des Programms nicht ändern.

Wirkung

- Die Funktion spricht den *slider* an und gibt dessen Stellung (0 bis 255) zurück. Wenn *ID* mehrfach vorhanden ist, dann werden alle Elemente dieser ID angesprochen.

Rückgabewert

- u08

Wert abfragen (seitenbezogen)

Definition

- getpslider(*ID, PageID)*

Argumente

- Argument *ID* vom Datentyp u08. Darf sich zur Laufzeit des Programms nicht ändern.
- Argument *PageID* vom Datentyp u08. Darf sich zur Laufzeit des Programms nicht ändern.

Wirkung

- Die Funktion spricht den seitenbezogenen *pslider* an und gibt dessen Stellung (0 bis 255) zurück. Wenn *ID* mehrfach vorhanden ist, dann werden alle Elemente dieser ID auf der Webseite mit der *PageID* angesprochen.

Rückgabewert

- u08

Erweiterten Slider abfragen (global)

Definition

- geteslider(*ID*)

Argumente

- Argument *ID* vom Datentyp u08. Darf sich zur Laufzeit des Programms nicht ändern.

Wirkung

- Die Funktion spricht den *eslider* an und gibt dessen Stellung (0 bis 255) zurück. Wenn *ID* mehrfach vorhanden ist, dann werden alle Elemente dieser ID angesprochen.

Rückgabewert

- f32

Erweiterten Slider abfragen (seiten-bezogen)

Definition

- getpeslider(*ID, PageID)*

Argumente

- Argument *ID* vom Datentyp u08. Darf sich zur Laufzeit des Programms nicht ändern.
- Argument *PageID* vom Datentyp u08. Darf sich zur Laufzeit des Programms nicht ändern.

Wirkung

- Die Funktion spricht den seitenbezogenen *peslider* an und gibt dessen Stellung zurück. Wenn *ID* mehrfach vorhanden ist, dann werden alle Elemente dieser ID auf der Webseite mit der *PageID* angesprochen.

Rückgabewert

- f32

Slider-Wert setzen (global)

Definition

- setslider(*ID*,*Value*,*Icon*, *State*)

Argumente

- Alle Argumente vom Datentyp u08

Wirkung

- Die Funktion spricht den *slider* an und stellt diesen auf den Wert von *Value.* Wenn *ID* mehrfach vorhanden ist, dann werden alle Elemente dieser ID angesprochen.
- Bei Aufruf der Funktion wird das Icon zudem auf das Symbol mit der Nummer (Typ u08) *Icon* gesetzt. Mögliche Grafiken sind auf Seite 45 dargestellt und werden über vordefinierte Zahlen (u08) ausgewählt. Zudem erleichtern vordefinierte Konstanten die Auswahl. Die Zuordnung hierfür ist in Tabelle 3 (Seite 68) aufgelistet.
- Jedes Icon hat mindestens die Zustände ACTIVE (==1), INACTIVE (==2), DARKRED (==0) und BRIGHTRED (==9). Einer dieser Zustände kann im Argument *State* übergeben werden. Eine Übersicht über die möglichen Zustände gibt Tabelle 2 (Seite 45).

Rückgabewert

- keine

Slider-Wert setzen (seitenbezogen)

Definition

- setpslider(*ID, Value, Icon, State, PageID*)

Argumente

- Alle Argumente vom Datentyp u08

Wirkung

- Die Funktion spricht den seitenbezogenen *slider* an der *ID* auf der Seite *PageID* und stellt diesen auf den Wert von *Value.* Wenn *ID* mehrfach vorhanden ist, dann werden alle Elemente dieser ID auf der Webseite mit der *PageID* angesprochen.
- Bei Aufruf der Funktion wird das Icon zudem auf das Symbol mit der Nummer (Typ u08) *Icon* gesetzt. Mögliche Grafiken sind auf Seite 45 dargestellt und werden über vordefinierte Zahlen (u08) ausgewählt. Zudem erleichtern vordefinierte Konstanten die Auswahl. Die Zuordnung hierfür ist in Tabelle 3 (Seite 68) aufgelistet.
- Jedes Icon hat mindestens die Zustände ACTIVE (==1), INACTIVE (==2), DARKRED (==0) und BRIGHTRED (==9). Einer dieser Zustände kann im Argument *State* übergeben werden. Eine Übersicht über die möglichen Zustände gibt Tabelle 2 (Seite 45).

Rückgabewert

- keine

Erweiterten Slider-Wert setzen (global)

Definition

- seteslider(*ID*,*Value*,*Icon*, *State*)

Argumente

- *ID, Icon, State, PageID* vom Datentyp u08
- *Value* vom Datentyp f32

Wirkung

- Die Funktion spricht den *eslider* an und stellt diesen auf den Wert von *Value.* Wenn *ID* mehrfach vorhanden ist, dann werden alle Elemente dieser ID angesprochen.
- Bei Aufruf der Funktion wird das Icon zudem auf das Symbol mit der Nummer (Typ u08) *Icon* gesetzt. Mögliche Grafiken sind auf Seite 45 dargestellt und werden über vordefinierte Zahlen (u08) ausgewählt. Zudem erleichtern vordefinierte Konstanten die Auswahl. Die Zuordnung hierfür ist in Tabelle 3 (Seite 68) aufgelistet.
- Jedes Icon hat mindestens die Zustände ACTIVE (==1), INACTIVE (==2), DARKRED (==0) und BRIGHTRED (==9). Einer dieser Zustände kann im Argument *State* übergeben werden. Eine Übersicht über die möglichen Zustände gibt Tabelle 2 (Seite 45).

Rückgabewert

- keine

Erweiterten-Slider-Wert setzen (seitenbezogen)

Definition

- setpeslider(*ID, Value, Icon, State, PageID*)

Argumente

- *ID, Icon, State, PageID* vom Datentyp u08
- *Value* vom Datentyp f32

Wirkung

- Die Funktion spricht den seitenbezogenen *peslider* an der *ID* auf der Seite *PageID* und stellt diesen auf den Wert von *Value.* Wenn *ID* mehrfach vorhanden ist, dann werden alle Elemente dieser ID auf der Webseite mit der *PageID* angesprochen.
- Bei Aufruf der Funktion wird das Icon zudem auf das Symbol mit der Nummer (Typ u08) *Icon* gesetzt. Mögliche Grafiken sind auf Seite 45 dargestellt und werden über vordefinierte Zahlen (u08) ausgewählt. Zudem erleichtern vordefinierte Konstanten die Auswahl. Die Zuordnung hierfür ist in Tabelle 3 (Seite 68) aufgelistet.
- Jedes Icon hat mindestens die Zustände ACTIVE (==1), INACTIVE (==2), DARKRED (==0) und BRIGHTRED (==9). Einer dieser Zustände kann im Argument *State* übergeben werden. Eine Übersicht über die möglichen Zustände gibt Tabelle 2 (Seite 45).

Rückgabewert

- keine

Bilder

Definition

- picture(*ID, Beschriftung, PageID, URL*)

Argumente

- *ID* vom Datentyp u08
- *Beschriftung* beliebiger Datentyp
- *PageID* vom Datentyp u08
- *URL* vom Datentyp c1400

Wirkung

- Die Funktion ändert das Element (*picture*). Wenn *ID* mehrfach vorhanden ist, dann werden alle Elemente dieser ID auf der Webseite mit der *PageID* angesprochen.
- Das Argument *Text* bezeichnet eine beliebige Variable, deren in einen String gewandelter Wert auf die variable Textzeile des Webelements ausgegeben wird.
- *URL*: Gültig WWW-Adresse (optional mit führenden http://) zur externen Grafik als neues Sprungziel.

  Aus Kompatibilitätsgründen kann der Weberver nur die ersten 383 Zeichen des Links auswerten. Längere Links werden abgeschnitten.

Rückgabewert

- keine

Beispiel

Ein Beispiel finden sie auf Seite 218.

Links

Externen Link ändern (seitenbezogen)

Definition

- link(*ID, Text, Icon, Iconzustand, PageID, Website*)

Argumente

- *ID* vom Datentyp u08.
- *Text* vom Datentyp c1400
- *Icon* vom Datentyp u08
- *Iconzustand* vom Datentyp u08
- *PageID* vom Datentyp u08
- *Website* vom Datentyp c1400

Wirkung

- Die Funktion spricht den (*link*) an. Wenn *ID* mehrfach vorhanden ist, dann werden alle Elemente dieser ID auf der Webseite mit der *PageID* angesprochen.
- Bei Aufruf der Funktion wird das Icon des Webelements mit der *ID* auf der Webseite mit der *PageID* auf das Symbol mit der Nummer (Typ u08) *Icon* gesetzt. Mögliche Grafiken sind auf Seite 45 dargestellt und werden über vordefinierte Zahlen (u08) ausgewählt. Zudem erleichtern vordefinierte Konstanten die Auswahl. Die Zuordnung hierfür ist in Tabelle 3 (Seite 68) aufgelistet.
- Das Argument *Text* wird auf die variable Textzeile des Webelements ausgegeben.
- Im Argument *Website* (http-Adresse (inkl. Pfad und führenden http://) des Ziels) wird die neue URL als Sprungziel angegeben.

  Aus Kompatibilitätsgründen kann der Weberver nur die ersten 479 Zeichen des Links auswerten. Längere Links werden abgeschnitten.

Rückgabewert

- keine

Beispiel

Ein Beispiel finden sie auf Seite 218.

Link auf Visualisierungsseite ändern (seitenbezogen)

Definition

- Funktion plink(*ID, Text, Icon, Iconzustand, PageID, PageSprungIndex*)

Argumente

- *ID* vom Datentyp u08
- *Text* vom Dtaentyp c1400
- *Icon* vom Datentyp u08
- *Iconzustand* vom Datentyp u08
- *PageID* vom Datentyp u08
- *PageSprungIndex* vom Datentyp u08

Wirkung

- Die Funktion spricht den (*plink*) an. Wenn *ID* mehrfach vorhanden ist, dann werden alle Elemente dieser ID auf der Webseite mit der *PageID* angesprochen.
- Bei Aufruf der Funktion wird das Icon des Webelements mit der *ID* auf der Webseite mit der *PageID* auf das Symbol mit der Nummer (Typ u08) *Icon* gesetzt. Mögliche Grafiken sind auf Seite 45 dargestellt und werden über vordefinierte Zahlen (u08) ausgewählt. Die Zuordnung hierfür ist in Tabelle 3 (Seite 68) aufgelistet.
- Das Argument *Text* wird auf die Textzeile des Webelements ausgegeben.
- Im Argument *PageSprungIndex* wird die Seitennummer des neuen Sprungziels angegeben.

Rückgabewert

- keine

Beispiel

Dynamisches Verändern von Web-Links im Webserver

```
[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
```

Werte-Diagramme

Diagramm mit einem Graphen

Definition

- Funktion chart(*ID, Var, X1, X2*)
- kompatibel zur Funktion webchart

Argumente

- *ID, Var* vom Datentyp u08
- *X1, X2* Datentyp c14

Wirkung

- Die Funktion spricht das XY-Diagramm *chart* an. Wenn *ID* mehrfach vorhanden ist, dann werden alle Elemente dieser ID angesprochen.
- Bei Aufruf der Funktion wird die XY-Darstellung des Wertes *Var* aktiviert. Es können Werte aus dem Bereich 1..30 dargestellt werden. 0 bedeutet keine Darstellung, Werte größer als 30 sind unzulässig und werden wie 0 interpretiert. Bei jedem Aufruf der Funktion werden zunächst die Werte von links beginnend dargestellt. Wenn nach 47 Aufrufen das Ende erreicht ist, werden die Werte nach links geschoben.
- Die X-Achsenberschriftung wird mit den Argumenten *X1, X2* (Datentyp c14) vorgegeben.

Rückgabewert

- u08 (interner Zustand des Webcharts).

Beispiel Prozentwert Anzeigen

An einer XY-Grafik im Webserver (*chart*-Element) soll ein Prozentwert angezeigt werden

Umsetzung im Anwenderprogramm:

```
[WebServer]
chart(ChartWebID)[$0%$,$50%$,$100%$]
[EibPC]
Prozentwert='1/3/5'u08
ChartWebID=0
if stime(0) then\\
webchart(ChartWebID,convert(convert(Prozentwert,0f32)/8.5f32,0), $jetzt$c14,$-47min$c14) endif
```

Diagramm (seitenbezogen)

Definition

- pchart(*ID, Var, X1, X2, PageID*)w

Argumente

- *ID, Var, PageID* vom Datentyp u08
- *X1, X2* Datentyp c14

Wirkung

- Die Funktion spricht das XY-Diagramm *chart* an. Wenn *ID* mehrfach vorhanden ist, dann werden alle Elemente dieser ID auf der Webseite mit der *PageID* angesprochen.
- Bei Aufruf der Funktion wird die XY-Darstellung des Wertes *Var* aktiviert. Es können Werte aus dem Bereich 1..30 dargestellt werden. 0 bedeutet keine Darstellung, Werte größer als 30 sind unzulässig und werden wie 0 interpretiert. Bei jedem Aufruf der Funktion werden zunächst die Werte von links beginnend dargestellt. Wenn nach 47 Aufrufen das Ende erreicht ist, werden die Werte nach links geschoben.
- Die X-Achsenberschriftung wird mit den Argumenten *X1, X2* (Datentyp c14) vorgegeben.

Rückgabewert

- u08 (interner Zustand des Webcharts).

Diagramm für bis zu 4 Graphen (global)

Definition

- mchart(*ID, VarX, VarY, Index*)

Argumente

- *ID, Index* vom Datentyp u08
- *VarX, VarY* vom Datentyp f16

Wirkung

- Die Funktion spricht das Element *mchart* mit der *ID* an. Wenn *ID* mehrfach vorhanden ist, dann werden alle Elemente dieser ID angesprochen.
- Ein *mchart* stellt vier verschiedene Graphen dar. *Index* (0,1,2,3) gibt an, welcher Graph angesprochen wird.
- Es werden 48 Werte gespeichert. Wenn mehr als 48 Werte in den selben *Index* von mchart gespeichert werden, fällt der an erste Stelle gespeicherte Wert weg.
- Die Einordnung der Werte im Graphenverlauf erfolgt durch die Vorgabe der Wertepaare.
- Die Achsenbeschriftung wird automatisch generiert.

Rückgabewert

- u08 (interner Zustand).

Diagramm für bis zu 4 Graphen (seitenbezogen)

Definition

- mpchart(*ID, VarX, VarY, Index,PageID*)

Argumente

- *ID, PageID, Index* vom Datentyp u08
- *VarX, VarY* Datentyp f16

Wirkung

- Die Funktion spricht das seitenbezogene Element *mpchart* mit der *ID* auf der Webseite mit der *PageID* an. Wenn *ID* mehrfach vorhanden ist, dann werden alle Elemente dieser ID angesprochen.
- Ein *mpchart* stellt vier verschiedene Graphen dar. *Index* (0,1,2,3) gibt an, welcher Graph angesprochen wird.
- Es werden 48 Werte gespeichert. Wenn mehr als 48 Werte in den selben *Index* von mpchart gespeichert werden, fällt der an erste Stelle gespeicherte Wert weg.
- Die Einordnung der Werte im Graphenverlauf erfolgt durch die Vorgabe der Wertepaare.
- Die Achsenbeschriftung wird automatisch generiert.

Rückgabewert

- u08 (interner Zustand)

Enertex® Bayern GmbH – Ebermannstädter Straße 8 - 91301 Forchheim - mail@enertex.de

Zeitdiagramme

Im Vergleich zu einfachen Wertediagrammen wird bei Zeitdiagrammen nur der y-Wert angegeben. Die x-Ache ist immer der aktuelle Zeitpunkt. Ein Graph in einem Zeitdiagramm stellt immer Werte aus einem TimeBuffer dar. Um Zeitdiagramme zu verwenden muss vorher der TimeBuffer konfiguriert werden. Dieser kann bei Bedarf abgespeichert werden, so dass die Werte auch nach einem Neustart noch vorhanden sind.

TimeBuffer konfigurieren

Definition

- timebufferconfig(*TimeBufferID, MemType, Laenge, DataType*)

Argumente

- *TimeBufferID* (u08) Eindeutige ID des TimeBuffers, Werte 0u08-254u08
- *MemType* (u08) Speichertyp: 0u08: Ringspeicher, 1u08: linearer Speicher
- *Laenge* (u16) Anzahl der Datensätze. Maximal 65535 Datensätze mit max. 4 Bytes Länge.
- Der Speicher ist vom Datentyp *DataType* des Eingangsobjektes

Wirkung

- Es wird ein TimeBuffer angelegt bzw. konfiguriert. Dabei kann mit dem Speichertyp festgelegt werden, ob dieser nach Befüllen mit den Werten vollläuft (linearer Speicher) oder ob der jeweils älteste Wert verworfen wird (Ringspeicher).
- ACHTUNG: Der EibPC² hat einen RAM von 512MB.

  Um einen ordnungsgemäßen Betrieb zu garantieren, müssen die TimeBuffer so dimensioniert werden, dass der Speicher des EibPC nicht überlastet wird. Mit der Funktion können bis zu 255 TimeBuffer definiert werden. Dabei gilt für den benötigten Speicherbedarf = (Anzahl der Werte) * 12. Daher belegt z.B. ein Puffer mit 65000 Werten etwa 780 kB.
- Sie können den Puffer auch jederzeit im Flash ablegen, sodass bei einem Neustart die Werte nicht verloren sind, siehe timebufferstore und timebufferread.

Rückgabewert

- 0u08 Erfolg
- 1u08 Fehler: maximale Anzahl von TimeBuffern überschritten
- 2u08 Fehler: TimeBuffer bereits definiert

Wert speichern

Definition

- timebufferadd(*TimeBufferID, Daten*)

Argumente

- *TimeBufferID* (u08)
- *Daten* (Typ wie *DataType* des TimeBuffers) Wert, der dem TimeBuffer hinzugefügt werden soll.

Wirkung

- Es wird hier ein Wertepaar in den TimeBuffer am Ende eingefügt.

Rückgabewert

- 0u08 Erfolg, 1u08 Fehler

Alle Werte löschen

Definition

- timebufferclear(*TimeBufferID*)

Argumente

- *TimeBufferID* (u08)

Wirkung

- Den Inhalt des TimeBuffers löschen (im Speicher und ggf. auf den Flash, falls vorhanden)
- Der TimeBuffer selbst wir nicht gelöscht.

Rückgabewert

- 0u08 Erfolg, 1u08 Fehler

TimeBuffer abspeichern

Definition

- timebufferstore(*TimeBufferID*)

Argumente

- *TimeBufferID* (u08)

Wirkung

- TimeBuffer dauerhaft im Flash speichern.

Rückgabewert

- 0u08 Erfolg, 1u08 Fehler

TimeBuffer von Speicher lesen

Definition
- timebufferread(*TimeBufferID*)

Argumente
- *TimeBufferID* (u08)

Wirkung
- Es wird ein TimeBuffer aus dem Flash gelesen.

Rückgabewert
- 0u08 Erfolg, 1u08 Fehler

Füllstand

Definition
- timebuffersize(*TimeBufferID*)

Argumente
- *TimeBufferID* (u08)

Wirkung
- Den aktuellen Füllstand des TimeBuffer ausgeben.

Rückgabewert
- Anzahl Werte im TimeBuffer (u16)

Wert auslesen

Definition
- timebuffervalue(*TimeBufferID, utcZeit, Data, utcZeitWert*)

Argumente
- *TimeBufferID* (u08)
- *utcZeit* vom Datentyp u64, welcher der Zeitstempel angibt, der größer gleich dem Zeitpunkt des nächsten Datenpunkts in der Zeitreihe ist.
- *Data* (Typ wie *DataType* des TimeBuffers). Die Funktion ändert beim Aufruf den Wert dieses Arguments auf den gespeicherten Wert zum gesuchten Zeitpunkt.
- *utcZeitWert* Die exakte Zeit des Aufnahmezeitpunkts des Wertes *Data*. Die Funktion ändert beim Aufruf den Wert dieses Arguments auf den Wert

Wirkung
- Es wird ein Wertepaar im TimeBuffer gesucht.

Rückgabewert
- 0u08 Erfolg, 1u08 Fehler

Beispiel: Werte auslesen

Ein TimeBuffer speichert f16-Datentypen und zeichnet seit 1.1.2016 Werte auf. Es soll der Wert im TimeBuffer zum Zeitpunkt 12:00:00 am 2.1.2016 täglich um 9:30:00 ausgelesen werden. Wenn ein Wert im Puffer vorhanden ist, der plus oder minus eine Sekunde zu diesem Zeitpunkt mit timebufferadd in den Puffer geschrieben wurde, soll dieser auf die GA '1/2/3'f16 ausgegeben werden.

```
uBf=0
timebufferconfig(uBf,0,2500u16,0f16)
// gesuchter Zeitpunkt
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
```

Bereich eines TimeBuffers im TimeChart anzeigen

Definition

- mtimechartpos(TimeChartID,Graph,TimeBufferID,StartPos,EndPos)

Argumente

- TimeChartID vom Datentyp u08
- Graph (u08) (0..3)
- TimeBufferID (u08) ID des TimeBuffers, der vom TimeChart angezeigt werden soll.
- StartPos (u16) Startposition der Anzeige, einschließlich
- EndPos (u16) Endposition der Anzeige, einschließlich

Wirkung

- Den darzustellenden Bereich eines TimeBuffers für das Webelement vorgeben.
- Der Wertebereich der Position ist von 0 bis zum aktuellen Füllstand, wobei 0 der älteste Eintrag ist. Dies gilt auch bei Ringspeichern.
- Ist EndPos größer als der aktuelle Füllstand, werden EndPos-StartPos+1 Werte bis zum aktuellen Füllstand angezeigt
- Sind StartPos und EndPos gleich 0u16, werden die neuesten Werte im TimeChart angezeigt. Die Anzahl entspricht der Anzahl der Werte des TimeCharts.

Rückgabewert

- keine

Intervall eines TimeBuffers im TimeChart anzeigen

Definition

- mtimechart(TimeChartID,Graph,TimeBufferID,StartZeit,EndZeit)

Argumente

- TimeChartID (u08)
- Graph (u08) (0..3)
- TimeBufferID (u08) ID des TimeBuffers, der vom TimeChart angezeigt werden soll.
- StartZeit (u64) Startposition der Anzeige als UTC Zeit in ms, einschließlich.
- EndZeit (u64) Endposition der Anzeige als UTC Zeit in ms, einschließlich.

Wirkung

- Den darzustellenden Bereich eines TimeBuffers für das Webelement vorgeben.
- Es wird jeweils der erste Wert gesucht, dessen Zeitstempel größer-gleich StartZeit bzw. EndZeit ist.

Rückgabewert

- keine

Langzeitaufnahmen

Während Zeitdiagramme für flexible Aufzeichnungen genutzt werden können, dienen Langzeitaufnahmen der automatischen Aufzeichnung mit einem festgelegten Intervall. Der EibPC² kümmert sich selbst um die Aufzeichnung, die Speicherung auf dem Flash und das Auslesen der passenden Daten, wenn der Anzeigebereich in der Visualisierung geändert wird.

Analog zu mtimechart und timebufferconfig sind Datenspeicher und Diagramm-Element auch hier getrennt. Das zugehörige Web-Element ist historychart. Das erlaubt es, eine HistoryBuffer in mehreren verschiedenen HistoryCharts anzuzeigen, ohne zusätzlichen Speicherplatz zu benötigen.

HistoryBuffer konfigurieren

Definition

- historybufferconfig(*HistoryBufferID , AufzeichnungsTyp, Intervall, Objekt, Valid*)

Argumente

- *HistoryBufferID* (u08) Eindeutige ID des HistoryBuffers, Werte 0u08-63u08
- *AufzeichnungsTyp* (u08):
  - 0u08: Delta
  - 1u08: Absolut
  - 2u08: Delta falls positiv, 0 anderenfalls.
- *Intervall* (u08) Minuten zwischen zwei Aufzeichnungen, gültige Werte:
  1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, 60
- *Objekt* (b01, s08, u08, s16, u16, s32, u32, s64, u64, f16, f32): Objekt, das aufgezeichnet wird. Es wird im festgelegten Intervall immer der aktuelle Wert verwendet.
- *Valid* (b01) legt fest, ob *Objekt* gültig ist.

Wirkung

- Es wird ein HistoryBuffer angelegt bzw. konfiguriert.
- HistoryBuffer werden beim Start automatisch geladen und beim Beenden des Programms gespeichert.
- Der Objekttyp entspricht dem intern verwendeten Speichertyp.
- Wird ein vorzeichenloses Objekt gespeichert, käme es bei eigentlich negativem Delta zu einem Wertebereichsüberlauf. In diesem Fall wird das Delta zu 0 geändert.
- Verwenden Sie für monotone Zählerstände (z.B. Wasserzähler, Stromzähler) den *AufzeichnungsTyp* 2u08. Dabei werden negative Sprünge nicht gespeichert sondern die Zählung wird neu gestartet.

  Beispiel: 15 Minuten-Intervall, *Valid*==1b01

  00:00: Zählerstand 1234u32, zuletzt: 1230u32 → Aufzeichnung 4u32
  00:15: Zählerstand 1235u32, zuletzt: 1234u32 → Aufzeichnung 1u32
  00:30: Zählerstand 25u32, zuletzt: 1235u32 → Aufzeichnung 0u32
  00:45: Zählerstand 27u32, zuletzt: 25u32 → Aufzeichnung 2u32
- Nur wenn *Valid* 1b01 ist, wird der aktuelle Wert von *Objekt* zum Füllen des Puffers verwendet und der letzte Wert gespeichert, um ggf. Delta zu berechnen. Andernfalls wird ein Nullwert gespeichert. Verwenden Sie dieses Objekt, wenn Werte aus externen Quellen gelesen werden, z. B. Modbus TCP oder MQTT.

Rückgabewert

- 0u08 Erfolg
- 1u08 Fehler: Ungültige ID
- 2u08 Fehler: HistoryBuffer bereits definiert
- 3u08 Fehler: Ungültiges Intervall

Gespeicherte Werte löschen

Definition

- historybufferclear(*HistoryBufferID*)

Argumente

- *HistoryBufferID* (u08)

Wirkung

- Den Inhalt des HistoryBuffers löschen.
- Der HistoryBuffer selbst wir nicht gelöscht sondern zeichnet beim nächsten Intervall neu auf.

Rückgabewert

- 0u08 Erfolg
- 1u08 Fehler

Eingaben

Definition

- webinput(ID)

Argumente

- *ID* des Webinputelements vom Datentyp u08. Darf sich zur Laufzeit des Programms nicht ändern.

Wirkung

- Liest die Daten des Webelements webinput aus. Dabei wird die Eingabe des Benutzers in den Rückgabe-String geschrieben.
- Webinputelemente sind immer global.
- Mit der Funktion setwebinput kann der angezeigte Zustand geändert werden, so dass dieser bei einem erneuten Öffnen der Seite angezeigt wird.

Rückgabewert

- String c1400 als Ergebnis der Betätigung. Abhängig von der Konfiguration des Elements hat der Rückgabewert folgendes Format:
  - kein Stil/PASSWORD: eingegebener Text
  - DATEPICK: $YYYY-MM-DD$
  - TIMEPICK: $HH-MM-SS$
  - COLORPICK: $0xRRGGBB$ mit RR, GG, BB jew. zwischen 00-ff.
  - TWPICK: $0xRRGGBB$ mit RR, GG, BB jew. zwischen 00-ff.

Beispiel: Farbe als RGB-Wert mittels Webinput vorgeben und Status anzeigen

```
[WebServer]
page(1)[$Enertex$,$Webserver$]
webinput(1)[COLORPICKER] $Farbe wählen$

[EibPC]
// Farbwahl
zColor=webinput(1) // z.B. $0xff0000$
uColor=convert(zColor,0u24) // z.B. 0xff0000
if change(uColor) then write("211-Dimmen Absolut RGB-6/0/3",uColor) endif

// Status
zColorStatus=stringformat("248-Status RBG-6/0/11", 2,3,0,6);
if change(zColorStatus) then {
   setwebinput(1, $0x$+zColorStatus , RGB, ACTIVE);
} endif
```

Definition

- setwebinput(*ID, Text, Icon, State*)

Argumente

- *ID, Icon, State* vom Datentyp u08
- *Text* beliebiger Datentyp

Wirkung

- Die Funktion spricht den *webinput* an. Wenn *ID* mehrfach vorhanden ist, dann werden alle Elemente dieser ID angesprochen.
- Bei Aufruf der Funktion wird das Icon des Webelements mit der *ID* auf das Symbol mit der Nummer (Typ u08) *Icon* gesetzt. Mögliche Grafiken sind in Tabelle 3 (Seite 68) aufgelistet.
- Das Argument *Text* gibt den auszugebenden Text an. Das Format muss zum Stil des Webinputs passen.
- Jedes Icon hat mindestens die Zustände ACTIVE (==1), INACTIVE (==2), DARKRED (==0) und BRIGHTRED (==9). Einer dieser Zustände kann im Argument *State* übergeben werden. Eine Übersicht über die möglichen Zustände gibt Tabelle 2 (Seite 45).

Rückgabewert

- keine

Ausgabe

Definition

- weboutput(ID,Data)

Argumente

- *ID* des Weboutputelements vom Datentyp u08
- *Data* Daten, die im Weboutputelement angezeigt werden

Wirkung

- Gibt Daten an das Weboutputelement aus.
- Weboutputelemente sind immer global.
- Die Ausgabe kann durch Verwendung von HTML/CSS nach Belieben gestaltet werden.
- **Es muss darauf geachtet werden, den HTML-Baum der Visualisierung z.B. durch nicht geschlossene HTML-Tags nicht zu zerstören!**

Rückgabewert

- keine

```
[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
```

Dialog anzeigen

Definition

- webdialog(*UserListe*, *AlarmIndex*, *Timeout*, *Titel*, *Inhalt*, *Buttons*)

Argumente

- *UserListe* (cXXXXX)
- *AlarmIndex* (u08)
- *Timeout* (u08) in Sekunden
- *Titel* (cXXXXX)
- *Inhalt* (cXXXXX)
- *Buttons* (cXXXXX)

Wirkung

- Öffnet ein Dialogfenster in aktiven Web-Visualisierungen.
- *UserListe* enthält eine Komma-separierte Liste von Benutzernamen, für die der Dialog geöffnet werden soll.
- Ist *UserListe* leer, wird der Dialog auf allen geöffneten Visualisierungen angezeigt.
- Beim Öffnen des Dialogs kann der *AlarmIndex* (Nummern 1-5) abgespielt werden, oder 0 für keinen Ton. Es werden die Dateien */upload/alarm[1-5].mp3* verwendet. Zum Hochladen von eigenen Alarmtönen siehe Dateien. Der Alarm kann erst nach einmaliger Interaktion abgespielt werden.we
- *Timeout* Zeit bis der Dialog automatisch geschlossen wird. Hat Timeout den Wert 0, muss er vom Benutzer geschlossen werden. Wird ein Timeout angegeben, wird die aktuelle Restzeit automatisch am Ende der Titelzeile angezeigt.
- *Titel* Titelzeile des Dialogs, kann zur Formatierung HTML enthalten
- *Inhalt* Eigentlicher Inhalt des Dialogs, kann zur Formatierung HTML enthalten
- *Buttons* Bereich für Buttons, kann zur Formatierung HTML enthalten
- *Buttons* werden mit HTML konfiguriert. Es können mehrere Buttons mit unterschiedlichen Funktionen hintereinander platziert werden. Jeder Button hat die Form
  <button class="..." type="button" data-action="..." data-action-args="...">Beschriftng</button>

  Für das Attribut class sind folgende Werte möglich:
  - visu-btn
  - visu-btn-outline
  - visu-btn-alarm

  Für das Attribut data-actions sind folgende Werte möglich:
  - navigate-page (Seitennummer, siehe plink)
  - navigate-url(URL)
  - close

  wobei die Argumente im Attribut data-action-args übergeben werden.

Rückgabewert

- keine

Beispiel

```
zUserliste=$$ // alle User
uAlarmindex = 1 // Alarmton Alarm1.mp3 muss vorhanden sein
uTimeOut= 60 // Eine Minute bis zum Schließen, 0 => User muss schließen
zTitel =$Neue Firmware$
zInhalt =$Eine neue Firmware-Version ist verfügbar$
zFusszeile =$<button class="visu-btn" type="button" data-action="close">Schließen</button>$  + \\
$<button class="visu-btn-alarm" type="button" data-action="navigate-url"
data-action-args="https://www.enertex.de">Enertex aufrufen</button>$


bNeueFirmware='0/0/1'b01
if (bNeueFirmware) then webdialog(zUserliste,uAlarmindex,uTimeOut, zTitel, zInhalt, zFusszeile) endif
```

Sprachausgabe (Text-to-speech)

Definition

- webtts(*UserListe*, *Sprache*, *Stimme*, *Text*)

Argumente

- *UserListe* (cXXXXX)
- *Sprache* (cXXXXX)
- *Stimme* (cXXXXX)
- *Text* (cXXXXX)

Wirkung

- Lässt einen Text über die im Browser integrierte Web Speech API ausgeben.
- *UserListe* enthält eine Komma-separierte Liste von Benutzernamen, für die die Ausgabe erfolgt.
- Ist *UserListe* leer, erfolgt die Ausgabe auf allen geöffneten Visualisierungen.
- *Sprache* Sprachcode im Format `de` oder `de-DE` (BCP 47). Unterstützung ist Browserabhängig. Verwenden Sie $$ für die eingestellte Standardsprache des Browsers.
- *Stimme* Zu verwendende Stimme. Unterstützte Stimmen sind browserabhängig. Verwenden Sie $$ für die Standard-Stimme der Sprache.
- *Text* Der auszugebende Text.
- Die Funktion muss durch if aktiviert werden.

Rückgabewert

- keine

Doorbird Türsprechstelle

Es können Doorbird-Türsprechstellen eingebunden werden. Die Web-Visualisierung zeigt das Kamerabild und erlaubt Gegensprechen, falls die Web-Visualisierung per https aufgerufen wurde. Beim Aufruf per http wird lediglich der Ton des Mikrofons der Sprechanlage ausgegeben, ein Zugriff auf das Mikrofon über den Web-Broser ist nicht möglich. Diese Einschränkung ist eine Sicherheitsbeschränkung moderner Web-Browser.

In der Doorbird-Konfigurationsapp muss der Benutzer mit den entsprechenden Berechtigungen angelegt werden. Dies kann selbst über die Doorbird-App oder über den Doorbird-Logikknoten im EibStudio erfolgen.

Die Expertenfunktion doorbird ist nur zur internen Verwendung. Verwenden Sie zur Einbindung die **VISU**-Funktion Doorbird.

Definition

- doorbird(*Name*, *Host*, *Benutzername*, *Passwort*)

Argumente

- *Name* (cXXXXX)
- *Host* (cXXXXX)
- *Benutzername* (cXXXXX)
- *Passwort* (cXXXXX)
-

Wirkung

- Erlaubt den Zugriff auf Kamera und Ton einer Doorbird-Türsprechstelle über die lokale LAN-API. Die Kamera selbst benötigt keinen Internetzugriff.
- *Name* ist der interne Name der Kamera, um diese in der Visualisierung anzuzeigen. Der Name muss eindeutig sein und darf nur Zeichen aus a-z, A-Z, 0-9 enthalten. Maximale Länge 64 Bytes.
- *Host* IP-Adresse oder DNS-Name der Kamera.
- *Benutzername, Passwort* In der Doorbird-App konfigurierte Zugangsdaten für den EibPC. Es wird empfohlen, für den EibPC eigene Zugangsdaten zu verwenden. Die Berechtigung für den Zugriff auf das Live-Kamerabild muss vergeben sein.

Rückgabewert (u08)

- 0u08 Erfolg
- 1u08 Fehlgeschlagen

IP-Kameras (RTSP)

In der Web-Visualisierung können Kamerastreams von IP-Kameras eingebunden werden, wenn diese das RTSP-Protokoll unterstützen. Dabei werden die Daten über den EibPC weitergeleitet, ein direkter Kamerazugriff wird nicht benötigt. Dies ist insbesondere beim Fernzugriff hilfreich.

Da die Ressourcen des EibPC beschränkt sind, ist die Anzahl gleichzeitig aktiver RTSP-Videostreams begrenzt. Es werden jedoch nur Ressourcen benötigt, wenn in der Visualisierung der entsprechende Stream auch aktuell geöffnet ist.

Der RTSP-Stream wird vom EibPC für den Web-Browser als WebRTC-Stream abrufbar gemacht.

Definition

- rtsp(*Name*, *RtspUrl*, *Transport*, *AlwaysOn*)

Argumente

- *Name* (cXXXXX)
- *RtspUrl* (cXXXXX)
- *Transport* (u08)
- *AlwaysOn* (b01)

Wirkung

- Erlaubt den Zugriff auf den RTSP-Stream einer IP-Kamera.
- *Name* ist der interne Name der Kamera, um diese in der Visualisierung anzuzeigen. Der Name muss eindeutig sein und darf nur Zeichen aus a-z, A-Z, 0-9 enthalten. Maximale Länge 64 Bytes.
- *RtspUrl* Adresse der Kamera als RTSP-URL im Format
  `rtsp://<Benutzername>:<Passwort>@<IP-Adresse oder DNS-Name>:<Port>/Pfad`
  z.B. `rtsp://eibpc:geheim@192.168.178.42:554/stream/profile0`
- *Transport* ändert das Protokoll zwischen EibPC und IP-Kamera:
  - 0u08: automatisch
  - 1u08: TCP
  - 2u08: UDP
- Falls *AlwaysOn* den Wert 1b01 hat, bleibt der Video-Stream intern immer aktiv. Dies reduziert die Zeit beim Öffnen.
- Konfigurieren Sie einen Weboutput mit dem folgenden HTML-Code, um den Stream anzuzeigen:
  `<video class="visu-rtsp" data-name="`*Name*`"></video>`

Rückgabewert

- 0u08 Erfolg
- 1u08 Fehlgeschlagen

Makros

Mit Hilfe von Makros, die wir auch (fertige) Funktionsblöcke nennen, wird

- die Programmierung des EibPC für den versierten Anwender schematisiert. Der Anwender kann komplette Codefragmente von Programmteilen, die er immer wieder verwendet, in eine eigene Bibliothek auslagern und somit die Programmierung bei unterschiedlichen Projekten jederzeit direkt wiederverwenden.
- Sie können die Makros auch wie eingebaute Funktionen im Anwendungsprogramm im EibPC nutzen.
- Sie können den Makroassistenten verwenden, der Ihnen bei der Parametrierung der Makros unterstützt. Dies bedeutet, dass Sie jeden Parameter mit Erklärungen mit Hilfe von EibStudio eingeben können. Bei ggf. später notwendiger Änderung können Sie wieder den Assistenten benutzten, um die Parameter neu einzugeben.

Definition

Ein Makro ist ein (Teil eines) Anwenderprogramms, das in eine Bibliothek ausgelagert wird. Als eigenständiger Programmteil eines anderen Anwenderprogramm können diese Makros in andere Projekte eingebunden werden. Im Makro kann man verschiedene Eingänge (Argumente) definieren, die projektspezifische Daten enthalten.

Am einfachsten kann die Makroprogrammierung anhand eines Beispiel erläutert werden. In einem Projekt haben Sie die Doppelbelegung eines KNX Tasters programmiert: Ein Tastendruck schreibt ein EIN-Telegramm auf die Adresse 0/0/1. Wenn der Taster innerhalb von 800ms zweimal gedrückt wird, so soll der EibPC ein EIN Telegramm auf die Adresse 3/4/6 senden: Es ergibt sich folgendes Anwenderprogramm:

```
DoppelKlick=0
if event('0/0/1'b01) and ('0/0/1'b01==EIN) then DoppelKlick=DoppelKlick+1 endif
if after(DoppelKlick==1, 800u64) then write('3/4/5'b01, EIN) endif
if after(DoppelKlick==1, 800u64) and DoppelKlick==2 then write('3/4/6'b01, EIN) endif
if after(DoppelKlick==1, 1000u64) then DoppelKlick=0 endif
```

Wenn diese Funktionalität auf weitere Taster und Gruppenadressen übertragen werden soll, so können Sie den Text per Copy und Paste im Texteditor des EibStudio ändern. Allerdings kann diese Methode u.U. fehleranfällig werden.

Mit einem Makro sind Sie in der Lage, in solchen Situationen Schablonen zu schaffen, die die Programmierung einfach gestalten. Hierzu legen Sie sich eine neue Makrobibliothek an und schreiben:

Ein Makro beginnt mit :begin

... endet mit :end

```
:begin Doppelklick(Name,TasteGA,TasteWert,Klick1GA,Klick1Wert,Klick2GA,Klick2Wert)
Name^DoppelKlick=0
if event(TasteGA) and (TasteGA==TasteWert) then Name^DoppelKlick=Name^DoppelKlick+1 endif
if after(Name^DoppelKlick==1, 800u64) then write(Klick1GA,Klick1Wert) endif
if after(Name^DoppelKlick==1, 800u64) and Name^DoppelKlick==2 then write(Klick2GA,Klick2Wert) endif
if after(Name^DoppelKlick==1, 1000u64) then Name^DoppelKlick=0 endif
:end
```

Ein Makro beginnt mit dem Schlüsselwort :begin und endet mit :end. Die Definition selbst ist der Name des Makros gefolgt von durch Komma getrennte Argumente, die von Klammern umgeben sind und befindet sich unmittelbar hinter dem :begin.

Die Argumente des Makros werden als Textersetzungen im Makrocode genutzt. Die Syntax ist exakt wie die des „gewöhnlichen“ Anwenderprogramms. Der aus den Makros gleichsam wie aus Textschablonen generierte Code wird vom Compiler zusammen mit den übrigen Programmen übersetzt. Sie können Ihren vom Compiler generierten Macrocode auch in der Datei „tmpMacroOut.txt“ im Arbeitsverzeichnis des EibStudio anschauen.

Wenn das obige Makro gespeichert wird, so vereinfacht sich der „Doppelklick“ auf einen KNX Taster:

```
Doppelklick(Keller,'0/0/1'b01,EIN,'3/4/5'b01,EIN,'3/4/6'b01,EIN)
```

Der Compiler schreibt bei unserem Beispiel „tmpMacroOut.txt“ (im Arbeitsverzeichnis vom EibStudio):

```
KellerDoppelKlick=0
if event('0/0/1'b01) and ('0/0/1'b01==EIN) then KellerDoppelKlick=KellerDoppelKlick+1 endif
if after(KellerDoppelKlick==1, 800u64) then write('3/4/5'b01,EIN) endif
if after(KellerDoppelKlick==1, 800u64) and KellerDoppelKlick==2 then write('3/4/6'b01,EIN) endif
if after(KellerDoppelKlick==1, 1000u64) then KellerDoppelKlick=0 endif
```

Sonderzeichen

Ein Sonderzeichen bei der Textersetzung ist das „^“-Zeichen. Mit diesem kann die Textersetzung so erweitert werden, dass aus zwei Wörtern zusammengesetzte Variablen entstehen. Das „^“-Zeichen wird dabei gelöscht. Die gleiche Wirkung erzielt man mit dem „_“-Zeichen, wobei dieses Zeichen nicht gelöscht wird. Mit Hilfe dieses Vorgehens können (indirekt) Variablen in Makros generiert werden, die aufgrund der Namensgebung gleichsam „gekapselt“ sind, z.B.

```
Name^DoppelKlick=0
```

im obigen Beispiel.

Auf diese Weise können Sie nun Variablen ähnlich zu objektorientierten Programmiersprachen „kapseln“. Im Beispiel ist die Variable „Doppelklick“ immer wieder genutzt. Wenn es nicht für jedes Makro eine „eigene“ Doppelklick-Variable gäbe, würde das Programm fehlerhaftes Verhalten generieren.

Beim Ersetzen von Makro-Argumenten in Zeichenketten ist zu beachten, dass diese nur ersetzt werden, wenn sie von Trennzeichen umschlossen werden. Ein Makro mit Argument

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

aufgerufen als

```
stringTest(Parameter)
```

wird jeweils wie folgt ersetzt

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

Laufzeit- und Syntaxfehler

Laufzeitfehler oder Syntaxfehler aufgrund falscher Verwendung z.B. Gruppenadressenzuweisungen treten erst bei der „Expansion“ des Makros auf.

Makroasisstent

Sie können Ihre Makros im Quellcode direkt für die Anwendung dokumentieren. Dazu gibt es das Schlüsselwort :info. An erster Stelle nach diesem Schlüsselwort steht hier die Funktionsbeschreibung, gefolgt von einer Beschreibung für jedes Argument. Die Beschreibungen sind von zwei „$“ Zeichen eingefasst.

Die Beschreibung können Sie mit „:info“ selbst erzeugen.

Jede Beschreibung der Argumente ist von zwei $-Zeichen eingerahmt.

```
:info $Mit diesem Funktionsblock können Sie einen Doppelklick auf eine Taste realisieren:\\
    Wenn Sie innerhalb von 0.8 Sekunden zweimal auf den Taster drücken, wird eine andere\\
    Aktion ausgelöst,als wie wenn Sie einmal drücken. Beide Aktionen können Sie mit \\
    diesem FunktionsblockMakro steuern$ \\
  $Name des Taster (zwecks Eindeutigkeit)$\\
  $Gruppenadresse, auf welcher der Taster Werte sendet$\\
  $Der vom Taster gesendete Wert (z.B. EIN oder AUS) $\\
  $Gruppenadresse für Telegramm bei einfachen Tastendruck$\\
  $Wert für Telegramm bei einfachen Tastendruck (z.B. EIN oder AUS oder 23%$\\
  $Gruppenadresse für Telegramm bei doppelten Tastendruck$\\
  $Wert für Telegramm bei doppelten Tastendruck (z.B. EIN oder AUS oder 23%)$
```

Lokale Variablen

Makros können lokale Variablen definieren, die nur in einem lokalen Kontext des Makros verwendet werden. Wenn ein Makro mehrere Male verwendet wird, werden die lokalen Variablen jeweils eigenständig in jeder Expansion des Makros verwendet. :var VARNAME@ definiert eine lokale Variable ist. Beachten Sie, dass das @-Zeichen am Ende des Namens obligatorisch ist, während VARNAME einen gültigen Variablennamen (Kombination aus Buchstaben und Zahlen und "_" Zeichen) darstellt.

Rückgabewerte

Jedes Makro hat einen Rückgabewert. Entweder ist es mit dem Makro-Befehlszeile :return *Expression* definiert oder wenn dies nicht definiert wurde, wird grundsätzlich die letzte Zeile vor dem :end -Befehl verwendet.

Es können beliebig viele lokale Variablen definiert werden (jede lokale Variable belegt einen der 65.500 Speicherplätze)

Im folgenden soll eine Funktion $\cosh(x)=\frac{e^x-e^{-x}}{2}$ als Makro so definiert werden, dass diese mit dem berechneten Rückgabewert direkt wie eine „eingebaute“ Funktion genutzt werden kann.

```
:begin cosh(x)
:info Berechnet die Cosh-Funktion
: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
```

Im obigen Beispiel wäre die Defintion von lokalen Variablen *sum@, p_ex@ u*nd *m_ex@* nicht wirklich notwendig, so dass wir kürzer schreiben können:

Kompakter Code

```
:begin cosh(x)
:info Berechnet die Cosh-Funktion
:return (exp(x)-exp(-x))/2f32
:end
```

Wie bereits erwähnt, ist das :return ebenfalls redundant (um die Kompatibilität mit Makro aus früheren Versionen zu erhalten), weswegen auch

:return wegoptimiert – schlechter zu lesen.

```
:begin cosh(x)
:info Berechnet die Cosh-Funktion
(exp(x)-exp(-x))/2f32
:end
```

möglich wäre. Wenn die letzte Zeile vor dem :end leer ist oder nur Leerzeichen enthält, wird kein Rückgabewert definiert. Grundsätzlich empfiehlt sich die Verwendung des :return Kommandos, alleine aus Gründen der besseren Lesbarkeit des Codes.

:return kann an beliebiger Stelle innerhalb des Makros platziert werden.

Folgender Code hat keinen Rückgabewert

Leerzeile von :end bedeutet, dass kein Wert zurückgegeben wird.

```
:begin cosh(x)
:info Berechnet die Cosh-Funktion
(exp(x)-exp(-x))/2f32
// Achtung : ES WIRD KEIN WERT ZURÜCK GEGEBEN!

:end
```

Wie eine eingebaute cosh-Funktion

Ein Makro, das in einer Makrolib steht, kann wie eine eingebaute Funktion (auch rekursiv) genutzt werden:

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

Beispiele

Gruppentelegramme senden

Beispiel 3: Ein Schalter und zwei Telegramme

Wird der Schalter „EIN“ gedrückt, soll die Leuchte einschalten und der Dimmer auf 80% gehen. Wenn er auf AUS geht, sollen die Lichter ausgehen.

Hintergrund
Der Schaltaktor erwartet einen Binärwert (EIN oder AUS) und der Dimmer einen Prozentwert. Dies bedeutet, der Schalter muss zwei verschiedene Telegramme auslösen. Über den Schalter können Sie jedoch nur ein Telegramm auslösen. Viele Schalter haben auch keine Möglichkeit, einen Prozentwert auf den Bus zu schicken.

Durch Angabe der Gruppenadresse und ihres Typs in einfachen Anführungszeichen kann auf beliebige Gruppenadressen geschrieben werden (S. 30).

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

Statt der manuellen Gruppenadresse kann auch eine importierte Gruppenadresse verwendet werden, falls Objekte aus der ETS importiert wurden (S. 15):

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

Leseanforderungen beim Start

Beispiel 4: Programmstart

Hintergrund
Wenn das Programm neu gestartet wird, werden alle Objekte mit 0 initialisiert (S. 33). Hat der Schalter 1/0/0 (bzw. ein verknüpfter Aktor) in obigem Beispiel allerdings bereits den Zustand EIN, so sendet der Schalter bei der nächsten Betätigung den Wert AUS. Dieser ist identisch zum aktuellen Zustand des Verarbeitungsobjektes, es werden also keine Schreibtelegramme ausgelöst. Erst bei der nächsten Betätigung, wenn der Schalter wieder EIN-schaltet, ändert sich der Interne Zustand und die Telegramme werden gesendet.

Beim Programmstart soll der aktuelle Zustand der Gruppenadresse "Schalter-1/0/0" abgefragt werden.

Um eine Funktion einmalig beim Programmstart auszuführen, wird die Funktion systemstart verwendet, die nach der Initialisierung einmalig von 0b01 auf 1b01 wechselt und ihre Abhängigkeiten aktualisiert (invalidiert).

Um den aktuellen Zustand einer Gruppenadresse abzufragen wird die Funktion read verwendet. Diese sendet bei Invalidierung eine Leseanforderung an die entsprechende Gruppenadresse.

Wichtig: Damit der Aktor antwortet, muss das Lesen-Flag in der ETS gesetzt werden.

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

Zur Vereinfachung kann zum Senden einer Leseanforderung beim Programmstart auch die Funktion initga verwendet werden.

Flurlichtsteuerung

Beispiel 5: Ein Bewegungsmelder, Schalter und Helligkeit je nach Uhrzeit

Wird der Schalter „EIN“ gedrückt, soll die Lampe einschalten und der Dimmer auf 100% gehen. Wenn er auf „AUS“ geht, sollen die Lichter ausgehen. Wenn der Schalter aktiv ist, soll der Bewegungsmelder deaktiviert werden.

Sendet der Bewegungsmelder ein EIN-Telegramm, soll der Dimmer auf

- 50% seiner Leuchtkraft gehen, wenn es nach 20:00 Uhr ist
- 30% seiner Leuchtkraft gehen, wenn es nach 23:00 Uhr ist
- 10% seiner Leuchtkraft gehen, wenn es nach 3:00 Uhr ist
- 100% seiner Leuchtkraft gehen, wenn es nach 7:30 Uhr ist

Die zeitabhängige Steuerung erfolgt mittels der Funktion htime (S. 124).

```
if (systemstart()) then {
                BewegungsMelder=AUS;
                read("Schalter-1/0/0") ;
                write("Lampe-1/1/1",AUS);
                write("Dimmer-1/1/2"u08,0%);
} endif
// Variablen
Schalter="Schalter-1/0/0"
BewegungsMelder=“Bewegungsmelder-1/2/0“
Dimmer=100%
// Der Schalter
if (Schalter==EIN)  then {
                write("Lampe-1/1/1",EIN);
                write("Dimmer-1/1/5",EIN);
                write("DimmerWert-1/1/2",100%);
} endif
if (Schalter==AUS) then {
                write("Lampe-1/1/1",AUS);
                write("Dimmer-1/1/2"u08,0%);
} endif
//Bewegungsmelder
if (htime(20,00,00)) then Dimmer=50% endif
if (htime(23,00,00)) then Dimmer=30% endif
if (htime(03,00,00)) then Dimmer=10% endif
if (htime(07,30,00)) then Dimmer=100% endif

if  (BewegungsMelder==EIN) and (Schalter==AUS) then write("Dimmer-1/1/5",EIN);
write("DimmerWert-1/1/2",Dimmer) endif
if  (BewegungsMelder==AUS) and (Schalter==AUS) then write("Dimmer-1/1/5",AUS) endif
```

Beispiel 6: Ein Treppenhauslicht

Es soll ein Treppenhauslicht (3 min Schaltzeit) realisiert werden soll. Der Schalter liefert dabei immer nur einen EIN-Impuls (es wird also nicht mehr ausgeschaltet).

Beim Systemstart soll das Licht ausgehen. Der Schalter liefert abwechselnd EIN und AUS Telegramme. Nach Schalterdruck („Schalterstellung“ sei beliebig) soll die Lampe angehen und automatisch nach 3 Minuten wieder ausgehen.

Variante 1: Bei erneutem Schalterdruck während der 3 Minuten Einschaltzeit soll der Timer nicht erneut starten

Variante 2: Bei erneutem Schalterdruck während der 3 Minuten Einschaltzeit soll der Timer erneut starten

Variante 1:

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

Die Funktion event (S. 171) reagiert auf Gruppentelegramme. Sie überprüft nicht, ob sich die Nachricht ändert, welchen Inhalt sie hat oder welchen Typ. Sobald eine Nachricht eintrifft, geht sie für einen Verarbeitungszyklus auf EIN. Damit wird die Bedingung der if-Anweisung wahr und die Anweisung ausgeführt.

Die Funktion after (S. 132) verzögert den Eingang (EIN) um die Zeit, die im zweiten Argument angeben wird. Der Rückgabewert ist also EIN – Impuls. Grafisch lässt sich das mit Abbildung 43 recht anschaulich darstellen. Das zweite Argument ist vom Typ ganzzahlig, vorzeichenlos mit 64 Bit. Wir benötigen daher den Datentyp u64. Dieser Wert gibt die Verzögerungszeit in ms an.

Sie können damit Verzögerungen über Jahrzehnte hinweg einstellen. Wenn die Funktion after einmal gestartet wird, verarbeitet Sie nur einen Impuls ihres Eingangs. Dadurch entsteht die Abbildung 43 gezeigte Totzeit, die gleich der Verzögerungszeit ist. Im Beispiel nutzen wir eine Verzögerung von

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

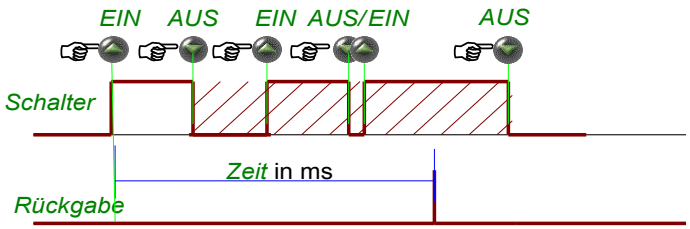


Abbildung 43: After-Funktion

Die Funktion after ist nicht nachtriggerbar, wie in Abbildung 43 durch die „Totzeit“ dargestellt wird. In unserem Fall (Variante 1) ist das ja so gewünscht. Das heißt, wenn after einmal gestartet wurde, werden sämtliche weitere Änderungen des Eingangs ignoriert (siehe Schraffur in der Abbildung).

Variante *2:*

Es soll der Timer für die Lichtschaltung immer wieder neu gestartet werden, wenn erneut der Lichtschalter betätigt wird. Dafür wird dir Funktion delay (S. 130) verwendet. Diese wird bei erneutem Aufruf den Timer neu starten („Re-Trigger“):

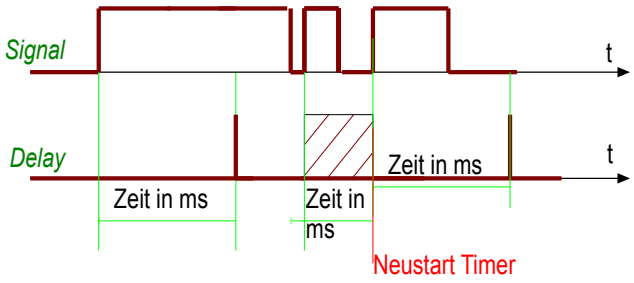


Abbildung 44: delay-Funktion

Damit ist unser Programm nur an einer Stelle zu ändern und wir müssen lediglich after durch delay ersetzten.

```
if systemstart() then write('1/1/1'b01,AUS) endif
SchaltVorgang=AUS
if event('1/0/0'b01) and (SchaltVorgang==AUS) then {
        SchaltVorgang=EIN;
        write('1/1/1'b01,EIN);
} endif
if (delay(event('1/0/0'b01), 180000u64)) then {
        write('1/1/1'b01,AUS);
        SchaltVorgang=AUS;
} endif
```

Zykluszeit

Eine häufig gestellte Frage der Anwender ist: *Wie lange dauert die Verarbeitung tatsächlich*? Grundsätzlich hängt das natürlich von der Programmgröße bzw. der Art der Programmierung und den auftretenden Ereignissen ab. Durch die „Validierung“ (s. S. 33) des Programms werden pro Zyklus nur diejenigen Teile des Programms aktiv, die sich auch wirklich ändern. Daher erfolgt im Normalfall die Verarbeitung in weniger als 1 ms, bei komplexeren Programmen in wenigen ms. Die Zykluszeit wird - abhängig vom Programm - durchaus schwanken. Es ist somit auch die minimale und die maximale Verarbeitungszeit von Interesse.

Im EibStudio kann nach jedem Zyklus eine Pausenzeit von bis zu 250 ms eingeräumt werden (S. 22ff), um asynchrone Funktionen auszuführen, z.B. Emails zu versenden, Webserveranfragen abzuarbeiten etc.

Um die Verarbeitungszeit zu berechnen, kann die Funktion afterc genutzt werden:

```
afterc(Variable {Typ b01}, Max{Typ u64}, Restzeit {Typ u64})
```

Diese Funktion wird wie die after-Funktion bei einem Wechsel der *Variable* (1. Argument) von AUS auf EIN getriggert: Der Rückgabewert ist nach Ablauf der vorgegebenen Zeit *Max* (2. Argument in ms) für einen Verarbeitungszyklus auf EIN. In jedem Zyklus von Beginn des Triggerimpulses von *Variable* wird dabei die *Restzeit* (3. Argument) wie ein Countdown-Zähler aktualisiert. Der Startwert von *Variable* ist *Max*. Die Änderung von *Restzeit* erfolgt immer exakt zu dem Zeitpunkt, an dem die Verarbeitung in einem Zyklus aktiv wird. Die Änderung von *Restzeit* ist also die Summe aus der eben genannten Pausenzeit plus der Verarbeitungszeit des vorangegangenen Zyklus. Damit lässt sich die Zyklusdauer berechnen, indem man mit systemstart einen afterc -Timer triggert und damit den Countdown von *Restzeit* startet, also z.B.

```
Max=10000000000000000u64
if afterc(systemstart(), Max, Restzeit) then { ..... } endif
```

Max wird dabei möglichst groß gewählt, um zu gewährleisten, dass das Ende des Countdowns möglichst nicht erreicht wird.

Mit dem Code

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

kann damit die minimale und maximale Zykluszeit mit einer Genauigkeit von ca. ± 1ms berechnet werden.

Einen Sonderfall gibt es noch zu berücksichtigen: Bei der Initialisierung des allerersten Programmdurchlaufs müssen alle Programmteile durchlaufen werden, die dann aufgrund der Validierung später „nur bei Bedarf“ ausgewertet werden. Daher kann die erste Verarbeitungsschleife durchaus mehrere Hundert ms benötigten, wenn das Programm eine Speicherauslastung von ca. 30% erreicht. Der Start des Countdownzählers muss daher verzögert werden, wenn man die Initialisierung des Programms als Sonderfall in der Messung der Zykluszeiten nicht berücksichtigen will.

Daher verzögert man den Impuls von systemstart beim Programmstart mit einem weiteren after Timer durch eine Verschachtelung:

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

Insgesamt schaut damit die Berechnung der Zyklusdauer wie folgt aus:

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

Max=10000000000000000u64
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
if change(Restzeit) then {
            MaxZyklusZeit=max(StoppZeit-Restzeit-PerformanceZeit,MaxZyklusZeit);
            MinZyklusZeit=min(StoppZeit-Restzeit -PerformanceZeit,MinZyklusZeit);
            StoppZeit=Restzeit;
} endif
```

Der Timer afterc nutzt das Argument Restzeit (s.o.) zur Speicherung der bereits abgelaufenen Timerzeit. Der Anwender muss daher darauf achten, dass verschiedene afterc Timer unterschiedliche Variablen zu dieser Speicherung nutzen:

```
// 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
```

Gleiches gilt für die Funktion

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

deren Timer – genau wie delay – durch jeden Wechsel der TriggerVariable (1. Argument) von AUS auf EIN erneut getriggert wird. Auch hier gilt, dass für Restzeit jeweils eine eigene Variable genutzt werden muss, da sich sonst die Timer gegenseitig stören.

Nach Ablauf des Timers, steht der Wert des 3. Arguments (Restzeit) auf 0u64, beim Triggern des Timers wird er auf den Wert von Max gesetzt. Wenn die Restzeit während einer aktiven Phase vom Anwender verändert, so wird damit die Ablaufzeit des Timers verändert.

```
RestZeit1=0u64
 if afterc(systemstart(),10000u64, RestZeit1) then {
      write('1/2/3'c14,$Timer1$c14)
} endif
if RestZeit1>1000u64 then RestZeit1=500u64 endif
RestZeit2=0u64
if delayc(systemstart(),13000u64, RestZeit2) then {
      write('1/2/3'c14,$Timer2$c14)
} endif
```

Im obigen Beispiel wird nur der afterc Timer verändert, die Restzeitvariable des delayc Timers ist unverändert.

Damit kann nun auch ein Timer angehalten werden, wenn beispielsweise das Ende des Ablaufs und die damit verbundene Aktion der if-Anweisung nicht mehr benötigt wird:

```
MyTrigger=AUS
RestZeit1=0u64
 if afterc(MyTrigger,10000u64, RestZeit1) then {
      write('1/2/3'c14,$Timer1$c14)
} endif
if MyTrigger== AUS then RestZeit1=0u64 endif
```

Wenn Im Beispiel *MyTrigger* auf EIN wechselt, wird der Timer gestartet, wenn *MyTrigger* vor Ablauf der Zeit auf AUS wechselt, so wird der Timer durch das Setzen von *RestZeit1=0u64* angehalten. Der then-Zweig wird dann nicht ausgeführt.

Will man den Timer vorzeitig beenden, aber den then-Zweig ausführen, so muss *RestZeit1=1u64* gesetzt werden. In diesem Fall erfolgt die Ausführung im nächsten Verarbeitungszyklus.

Warteschlange

Die Ereignis-basierte Verarbeitung im EibPC erfordert die Programmierung von sogenannten „Zustandsautomaten“. Das (abstrakte) Grundprinzip eines Zustandsautomaten ist, dass die Programmierung nicht sequentiell erfolgt, sondern dass abhängig von Ereignissen die Software einen bestimmten Zustand einnimmt.

Beim Datenaustausch mit einem anderen Gerät z.B. über TCP/IP Telegrammen kann man z.B. folgende Zustände definieren:

1. Daten des anderen Teilnehmers entgegennehmen
2. Daten an den anderen Teilnehmer verschicken
3. Daten des anderen Teilnehmers zwischenspeichern
4. Daten des anderen Teilnehmers auswerten
5. Verschiedene KNX Aktionen auf dem Bus ausführen

Jeder dieser Zustände ist, zumindest grundsätzlich, von den anderen unabhängig, d.h. der EibPC muss Daten entgegennehmen, während z.B. KNX Telegramme eintreffen. Darüber hinaus können verschiedene Zustände jeweils andere „antriggern“ bzw. kann das Eintreffen eines KNX Telegramms die Datenverarbeitung anregen.

Anwesentheits -Zustandsmaschine

Der Anwender möchten mit dem Makro `Bei_Sonnenuntergang_Gedeckelt_mitFreigabe` beim Sonnenuntergang bzw. spätestens zu einer im Makro bestimmten Zeit eine Gruppenadresse schalten.

In gleicher Weise soll das Makro `Bei_Sonnenaufgang_Gedeckelt_mitFreigabe` beim Sonnenunteraufgang genutzt werden.

```
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)
```

Die Makros werden mit der Freigabe-Variable *FreigabeVar* parametriert.

Die Freigabe wird hierzu in folgende Betrachtungszeiträume unterteilt:

- Tag-Modus: Sonnenaufgang bis Sonnenuntergang
- Früh-Modus: Zeitraum nach 00:00h und noch vor Sonnenaufgang
- Spätmodus: Nach Sonnenuntergang und nicht nach 0:00 Uhr

Der Anwender betätigt eine Gruppenadresse "Anwesenheit-8/1/1" (Typ b01, EIN==anwesend).

Die Freigabe-Variable *FreigabeVar* soll abhängig von den folgenden Zuständen geschaltet werden.

Zustand 1:

Beschreibung:

Frühmodus

Ziel:

Es soll kein Makro durchlaufen werden, unabhängig davon, ob "Anwesenheit-8/1/1" auf EIN oder AUS steht.

Damit muss *FreigabeVar* auf AUS gesetzt werden bzw. in diesem (AUS-) Zustand verharren.

Zustand 2:

Beschreibung:

Zeitraum nach Sonnenaufgang bis Sonnenuntergang

Ziel:

Wenn "Anwesenheit-8/1/1" auf EIN steht, soll *FreigabeVar* auf EIN gesetzt werden, die Makros werden also aktiv, wenn "Anwesenheit-8/1/1" auf AUS steht, soll *FreigabeVar* auf AUS gesetzt werden, die Makros werden deaktiviert.

Wenn die Gruppenadresse "Anwesenheit-8/1/1" verändert wird (Bustelegramm/User), soll der *FreigabeVar* unmittelbar deren Wert annehmen.

Zustand 3:

Beschreibung:

Spätmodus

Ziel:

Wenn "Anwesenheit-8/1/1" auf EIN steht, soll *FreigabeVar* auf EIN gesetzt werden, die Makros werden also aktiv, wenn "Anwesenheit-8/1/1" auf AUS steht, soll *FreigabeVar* auf AUS gesetzt werden, die Makros werden deaktiviert.

Dies kann nun direkt in ein Programm umgesetzt werden:

```
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
```

Besonders ist hier die Verwendung der Variable *TFrueh.* Diese ist über eine Verknüpfung von einer Schaltuhr um Mitternacht und einer nach Mittag realisiert. Es wird damit sichergestellt, dass *TFrueh* um 0:00 Uhr auf EIN und ab Nachmittag auf AUS steht.

Anwesenheitssimulation

In der Makrosammlung befinden sich Makros zur Anwesenheitssimulation. Das grundsätzliche Konzept dieser Makros soll im folgenden erläutert werden.

Bei einer Anwesenheitsimulation können zwei Zustände unterschieden werden:

1. Aufzeichnen

Während dieser Phase werden vorher ausgewählte Gruppenadressen aufgezeichnet. Meist sind das vom Bewohner ausgelöste Gruppentelegramme z.B. beim Betätigen von Schaltern. Die Aufzeichnung wird meistens über ein 2-Wochenintervall ausgeführt, wobei die Aufzeichnung kontinuierlich alte Werte überschreibt.

2. Abspielen

Wenn der Bewohner einer Liegenschaft z.B. in den Urlaub geht, sollen nun die Gruppentelegramme vom EibPC ausgelöst werden, sodass Außenstehenden den Eindruck einer Anwesenheit der Bewohner vermittelt wird. Dabei soll das Abspielen tages- und uhrzeitgleich erfolgen, sodass z.B. die Aufzeichnung von einem Samstag auch wieder am Samstag abgespielt wird.

Als Konsequenz wird benötigt:

- Ermittlung der Rohdaten der Telegramme
- Ermittlung der sendenden Gruppenadresse
- Ermittlung der Zeitpunkt des Eintreffens der Telegramm
- Aufzeichnen der Daten
- Versenden von Rohdaten zeit-versetzt auf den Bus

Ermittlung der sendenden Gruppenadresse

Für diese Aufgabe benötigt man die Funktion readrawknx:

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

Wenn ein beliebiges KNX-Telegramm am Bus beobachtet wird, aktualisiert die Funktion readrawknx ihre Argumente. In diesem Fall, werden die Argumente der Funktion mit Daten „gefüllt“. Die empfangenen Nutzdaten werden dann in das Argument *Sim_Data* kopiert, die Datenmenge (Bitlänge) kann mit der Variable *Sim_Len* abgefragt werden.

Bei Eintreffen eines Telegramms wird das Argument *Sim_IsGA* entsprechend gesetzt, d.h. handelt es sich um ein gewöhnliches Gruppentelegramm, so wird dieses Argument von readrawknx auf EIN gesetzt und *Sim_GA* enthält die Adresse selbst. Die Funktion readrawknx kann mit event verknüpft werden, um ein Eintreffen eines Telegramms verarbeiten zu können.

Mit den gewählten Definitionen

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

kann man nun das Eintreffen eines Telegramms wie folgt verarbeiten:

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

Dabei ist zu beachten, dass hier die Gruppenadresse *Sim_GA* als 16-Bit Wert berechnet wird. Um diese Adresse mit der gewöhnlichen Schreibweise vergleichen zu können, steht die Funktion getaddress zur Verfügung: Im folgenden Beispiel

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

ist nun MeinGA der 16-Bit Wert, welcher die Gruppenadresse repräsentiert und wie diese auch in *Sim_GA* kopiert wird. Damit lässt sich nun feststellen, auf welcher Gruppenadresse das eingetroffene Telegramm gesendet wurde.

Mit Hilfe der Variablen

```
Sim_GA=OFF
```

soll im folgenden das Aufzeichnen einer eingetroffenen Nachricht getriggert werden. Dazu werden für jede aufzuzeichnende Gruppenadresse im Code if-Abfragen hinterlegt. *Sim_GA* wird wie oben beschrieben von readrawknx ermittelt.

Code-Teil 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
```

Die beiden Betriebsarten Aufzeichnen/Abspielen werden über

```
Sim_Play=OFF
```

realisiert. Dabei soll bei *Sim_Play* = ON die (bestehende) Aufzeichnung abgespielt werden und bei OFF die Aufzeichnung starten.

Ermittlung der Rohdaten der Telegramme

Nun ist zu klären, wie die Rohdaten der Telegramme am Bus ermittelt werden können. Dazu

Code-Teil 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)*
256u32+convert(stringcast(Sim_Data,0u08,5u16),0u32)  endif;
     Sim_Next=ON;
  } endif;
}endif
```

Sim_RawData sind die Rohdaten im u32 Format. Wenn nur 1 Bit gesendet wurde, so sind 31 Bits „ungenutzt“. Die eintreffenden Daten werden von readrawknx in die String-Variable *Sim_Data* geschrieben. Diese sind grundsätzlich als Rohdaten zu betrachten und sollen dann in den u32 Bit Wert gewandelt werden. Die Anordnung der Daten in 4 Bytes (32 Bit) vereinheitlicht die Speicherung der Telegrammdaten und vereinfacht dadurch das Procedere (wie noch zu zeigen ist).

Zum Verarbeiten dieser Rohdaten im String *Sim_RawData* müssen nun die einzelnen Bytes als 1-Byte Integerwerte interpretiert werden. Dies geschieht mit Hilfe der Funktion stringcast.

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

Diese Funktion betrachtet die Bytes im String *src* ab der Byte-Postion *Pos. dest* gibt dabei den Zieldatentyp Umwandlung an, der somit die Anzahl der Bytes vorgibt, welche für die Umwandlung in das Ergebnis *X* festlegt. Anhand von Abbildung 45 soll das erläutert werden: Die Grafik zeige den String als Byteanordnung. An Position 3{u16} steht der Wert hexadezimal 0x74.

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

Abbildung 45: String src als Arrayfeld.

Eine Anweisung Z1=stringcast(*src,0,3u16*) wird eine Variable Z1 vom Datentyp u08 (Argument „0“) definieren. Der Wert wird aus *src* (Abbildung 45) an der Position 3{u16} gewonnen und ist damit im vorliegenden Fall 0x74 (dezimal 116). Eine Anweisung Z2=stringcast(*src,10u32,3u16*) definiert hingegen die Zahl 0x74a0e101 (dezimal 1956700417). Diese Anzahl der Bytes, die aus dem String entnommen werden, wird durch das Argument 10u32 gewonnen: Der Datentyp u32 ist 32 Bit lang und besteht aus 4 Bytes. Der Wert 10 von „10u32“ an sich, wird hier ignoriert. Die Reihenfolge der Bytes bleibt bei der stringcast Funktion unverändert.

Zurück zum Beispiel: *Sim_RawData* enthält die Daten des eintreffenden Telegramms in den ersten 4 Bytes. Die Reihenfolge der Bytes am Bus ist dabei unterschiedlich zur Byte-Reihenfolge des Linuxsystems des EibPCs. Um diese Daten verwenden zu können, muss die Bytereihenfolge umgedreht werden, d.h. das letzte Bit muss an die erste Stelle usw.. Diese Umordnung wird mit Hilfe der Multiplikationen mit 256, 65536 und 16777216 realisiert.

Die vorliegende Verarbeitung von Rohdaten beschränkt sich demnach auf max. 32 Bit breite Telegramme. Längere Datentelegramme können nicht aufgezeichnet werden, anderseits werden für 1 Bit Objekte sicherlich Bytes „verschwendet“, da alle Telegramme gleich behandelt werden. Dennoch stellt dieses Vorgehen gewissermaßen einen optimalen Kompromiss dar, da die Verarbeitung später einfacher wird.

Mit dem Code-Teil 2 sind nun die Daten in der u32 – Variablen *Sim_RawData* berechnet.

Ermittlung des Zeitpunkt des Eintreffens der Telegramme

Die Sendezeitpunkte der Telegramme muss relativ bestimmt werden, da eine zuvor aufgezeichnete Simulation relativ (zeitversetzt) zum Startpunkt der Simulation erfolgen soll:

Code-Teil 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
```

Beim Wechsel vom *Sim_Start* auf EIN initialisiert die erste if-Anweisung den String TimeStamp. Zudem wird ein afterc-Timer (s.o.) gestartet. Dabei legt *Interval* fest, wie lange die Aufzeichung erfolgen soll, z.B. 1 Tag = 86400000ms. Diese Funktion aktualisiert bei jedem Schleifendurchlauf wie ein Countdown-Zähler die Variable *Timer*. Diese zählt also relativ zum Startzeitpunkt verstrichene Zeit in ms runter. Im String *Timestamp* wird der Start auf Position 0 geschrieben, wobei hier aus Gründen der Einfachheit die maximale Aufzeichnungsdauer auf 32 Bit (49 Tage) begrenzt ist.

Aufzeichnen der Daten

Wenn nun mit Code-Teil 1 festgelegt wird, dass die eintreffende GA aufgezeichnet werden soll (*Sim_MyGA* auf EIN), so werden die Daten in den String *Data* und die Gruppenadresse in den String *Recoder* gespeichert. Da die Gruppenadressen nur 16 Bit breit sind, kann die Bitlänge dann gleich in diesem Array mit abgelegt werden. Für die Speicherung von Rohdaten in einem String verwendet man stringset.

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

Diese Funktion schreibt in den Zielstring *dest* an dessen Speicherpostion *Pos* den (binären) Inhalt von *src*.

Code-Teil 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
```

Dadurch das Zeitstempel, Daten und Gruppenadressen 32 Bit breit abgespeichert werden, ist die Position eines Telegramms in diesen Strings gleich, was die Verarbeitung vereinfacht. In einen c1400 String können damit 350 Telegramme aufgezeichnet werden. Mit Hilfe von 65k Strings können 16341 Telegramme aufgezeichnet werden, im vorliegenden Fall wurde mit c4000 demnach der Telegrammspeicher auf 1000 festgelegt. Die Funktion capacity zeigt an, wie viel Bytes der String maximal abspeichern kann.

Nach Ablauf der vorgegeben Zeit wird in Code-Teil 3 die Aufzeichnung neu gestartet. Dabei werden die erst gespeicherten Werte neu überschrieben, die alten bleiben erhalten, was ggf. stören kann. Daher wird im obigen Code-Teil 4 ein eventuell aus einer vergangenen Aufzeichnung bestehenden Zeitstempel gelöscht.

Abspielen einer Aufzeichnung

Das Abspielen der Aufzeichnung ist relativ einfach. Es werden hierzu lediglich die Gruppenadresse und die Rohdaten auf den Speicher „geladen“ (Strings) und diese auf den Bus geschrieben. Dabei muss zunächst der Timer aus Code-Teil 3 neu gestartet werden. Die aktuelle Countdown-Zeit in *Timer* wird mit dem Zeitstempel in *Timestamp* verglichen und bei Unterschreiten der Zeit das Schreiben veranlasst:

Code-Teil 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
```

Dabei müssen die Datentypen aufgrund der Verwendung der Rohdaten nicht beachtet werden. Lediglich die Länge der Telegramme ist auszuwerten, so dass sie denen der Aufzeichnung entsprechen.

Die Makrobibliothek EnertexAnwesenheit.lib ist auf diese Weise realisiert.

In der Bibliothek wird die Aufzeichnung in kleinere Tagesintervalle zerlegt und beim Abspielen später wieder zusammengesetzt. Die Aufzeichnung beginnt dann jeweils zum nächsten Tagesintervall.

Encoding bei c14

Der KNX-Standard sieht vor, dass Geräte bei 14-Byte Textmeldungen („c14“-Typen) nur den ASCII Code umsetzen müssen. Dabei wird in einer Erweiterung optional ISO8859-1 erlaubt, die ihrerseits nur aus 1-Byte Zeichen bestehen (vgl. http://de.wikipedia.org/wiki/ISO_8859-1).

EibStudio speichert alle Programme in UTF-8-Codierung. Beim Übersetzen werden automatisch vom EibParser c14-Zeichenketten als ISO8859-1 codiert gespeichert.

String-Längen

Bei der Stringverarbeitung wird häufig auf die Verkettung zurückgegriffen, d.h. das „Aneinanderhängen“ von Zeichenketten.

So wird z.B. im Code

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

der String *s3* den Inhalt *Hallo Welt* haben. Die Datentypkontrolle im EibParser sorgt dafür, dass *s3* vom Typ c1000 ist. Der EibParser sorgt dafür, dass die Verkettung die Größe des längsten Strings aufnehmen kann, im vorliegenden Fall sind das für *s1+s2* 1000 Bytes. *s3* werden als Ergebnis der Verkettung *s1+s2* 1000 Bytes zugewiesen.

Wenn in *s2* bereits 950 Bytes an Daten vorliegen und *s1* seinerseits 90 Bytes belegt, so werden 40 Bytes bei der Verkettung „verloren“ gehen, da *s3* nur max. 1000 Bytes aufnehmen kann.

Im folgenden Code ist das ebenso zu beachten:

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

Auch hier wird die Verkettung *s1+s2* die Länge von 1000 Bytes, da sie sich aus aus zwei 1000 Byte-Strings zusammensetzt. Die Zuweisung auf den 2000 Bytes langen *s3* erfolgt erst nach der Verkettung. Da aber schon die Verkettungsoperation die Länge auf 1000 Bytes begrenzt hat, können hier Bytes „verloren“ gehen.

Im folgenden Code ist das anders:

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

Auch hier wird die Verkettung *s1+s2* die Länge von 1000 Bytes, da sie sich aus aus zwei 1000 Byte-Strings zusammensetzt. Die Zuweisung auf den 200 Bytes langen *s3* erfolgt erst als Ergebnis der Verkettung: Zunächst begrenzt die Verkettungsoperation *s1+s2* die Länge auf 1000 Bytes, die Zuweisung auf *s3* begrenzt dessen Länge auf 200 Bytes, also gehen ggf. 800 Bytes an Daten „verloren“.

Soll die Verkettung *s1+s2* in keinem Fall Daten verlieren, muss eine Dummy-Variable eingeführt werden:

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

Hier ist gewährleistet, dass *s1+s2+dummy* 2000 Bytes als Ergebnis aufnehmen kann. Daher wird auch die Verkettung 2000 Bytes an *s3* als Ergebnis liefern.

FTP Datenströme

Mit Hilfe von konfigurierbaren FTP Transfers können beliebige ASCII („Klartext“) Dateien auf einen externen FTP Server geschrieben werden. Die maximale Dateigröße beträgt 64 kB.

Vier Datenströme

Dazu können **vier** verschiedene Handles (=ID-Nummer des Transfers) angelegt werden, die - selbst per Warteschlange gepuffert - diese Dateien auf dem Server anlegen. Die Dateien werden per Timeout auch früher (und dann ggf. weniger Bytes) oder per flushftp() vom User initiiert geschrieben. Die Dateinamen werden von der Firmware automatisch nach Datum und Uhrzeit vergeben.

Im folgenden soll das Vorgehen beim Anlegen und Anwenden dieser FTP Auslagerung detailliert beschreiben werden.

Zunächst muss im Programm der Stream und sein Handle definiert werden. Hierzu wird die Funktion

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

benötigt (S. 193). Ein Handle bezeichnet eine eindeutige Zahl (ID) für einen Transfer und ist etwa gleichbedeutend mit einer Namensgebung.

Die ersten 3 Argumente dienen der Konfiguration des Tranfers: IP Adresse, Username und Passwort, dann folgt das Zielverzeichnis auf dem Server und ein Timeout Parameter. Mit dieser Anweisung reserviert man einen 64 kByte großen Puffer. Die Übertragung des Puffers erfolgt dann, wenn entweder der Puffer vollends befüllt wurde (dazu unten mehr) oder die Anzahl *timeout* Sekunden seit dem letzten Transfer verstrichen sind.

Konfiguration des Transfers

```
// 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)
```

Während des Betriebs müssen nun die Daten in den Puffer geschrieben werden. Dazu wird

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

Mehrere Strings werden in einer Textzeile zusammengefasst

benötigt. Die Funktion erlaubt beliebige Strings als Argumente, da die Zieldatei ebenso nur eine Textdatei darstellt. Etwaige Daten in Form von Zahlwerten müssen daher mit Hilfe der convert-Funktion umgewandelt werden. Dabei wird am Ende der Datenübermittlung von sendftp ein LF-CR (Zeilenumbruch für Windows geeignet) eingefügt. Ein Aufruf von sendftp kann mehrere Teilstrings übergeben, aber nicht mehr als 1400 Bytes insgesamt übernehmen. Demnach ist die maximale Zeichenlänge 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(00) then {
  status=sendftp(Handle, Data1,convert(Nr,$$),Data2,convert(settime(),$$));
  Nr=Nr+1u16;
} endif
```

Wenn die Variable *status* auf 1 steht, war das Schreiben auf den Puffer des Transfers erfolgreich. Dies hat aber noch nichts damit zu tun, dass die Daten schon auf dem FTP Server angekommen sind.

Dazu muss der Status des FTP Datenstroms abgefragt werden.

Es steht hierfür

ftpstate(handle)

zur Verfügung.

Mit

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

kann nun folgender Status erfragt werden:

- Konfiguriert / fehlerfrei = 0
- die letzte Übertragung war fehlerfrei = 1
- der FTP Server war nicht erreichbar = 2
- das Passwort/User nicht erlaubt = 3
- Das Zielverzeichnis nicht existent und konnte nicht angelegt werden = 4
- Die Warteschlange hat einen Überlauf (=5); dies kann nur entstehen, wenn vorher die Übertragung nicht erfolgreich war.
- Handle nicht definiert = 6

Wenn es für die Verarbeitung von Bedeutung ist, den Füllstand des Datenstrompuffers zu ermitteln, kann dies mit Hilfe von

ftpbuffer(handle)

ftptimeout(handle)

in Erfahrung gebracht werden. Die erste Funktion gibt die Anzahl der genutzten Bytes im Puffer, die zweite die verstrichene Zeit seit dem letzten Transfer an.

```
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
```

Neben dem automatischen Schreiben der Daten auf den FTP Server, kann der Puffer auch mit Hilfe der Funktion

flushftp(handle)

durch Hochladen der Daten zum FTP-Server „manuell“ geleert werden:

```
// 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
```

Wenn nicht manuell geleert bzw. geschrieben wird, initiiert der EibPC den Transfer selbstständig. Der Transfer erfolgt, wenn der Puffer voll ist oder der konfigurierte Timeout (in Sekunden) seit dem letzten Transfer verstrichen ist.

Ändern des Zielpfads

Sie können auch den Zielpfad zur Laufzeit umkonfigurieren. Dazu sollten sie bereits offenStreams vorher rausschreiben („flushen)

```
changepath=OFF
newpath=$ftp/Telegramme2$
if changepath then {
        status=flushftp(Handle);
} endif
if changepath and status==0 then {
        Handle=ftpConfig(server,user,password,newpath,timeout)
}endif
```

Nutzung von eigenen Html-Code und Grafiken auf dem Webserver

Mit den weboutput-Element des Webservers kann der Anwender eigenen Html-Code in der Visualisierung anzeigen lassen. Im Ausgabefeld kann ein einfacher Text, aber auch ein komplexer HTML-Code dynamisch dargestellt werden.

Fehlerhafter bzw. ungültiger Html-Code im Weboutput kann den Seitenaufbau stören. Derartige Fehler werden nicht vom kostenlosen Support behoben. Arbeiten Sie hier mit Tools wie etwa http://www.quackit.com/html/online-html-editor/, um den HTML-Code zu testen.

Dazu muss zunächst in der Visu ein Weboutput angelegt und diesem ein eindeutiger Variablenname zugewiesen werden (S. 19).

Mit der Funktion

weboutput(*Variable*,Data)

werden die Daten auf das Feld geschrieben. Data ist hier ein String mit der maximalen Länge von 65534 Bytes (Typ c65534). Eine Besonderheit ist, dass dieser String auch ein gültiger Html-Code sein kann. Damit ist es möglich, eigene Formatierungen und Anzeigen zu dynamisieren.

Wir wollen nun zwei Weboutputs mit den Variablennamen *Out1*, *Out2* so beschreiben, dass eine Webseite wie in Abbildung 46 entsteht:

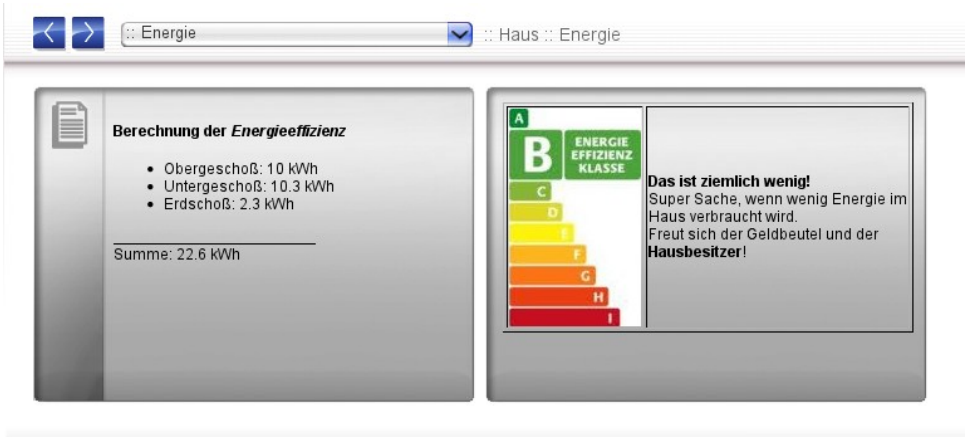


Abbildung 46: Gewünschte Ausgabe

Für die Erstellung des eigentlichen Html-Codes sei auf http://de.selfhtml.org verwiesen. Der HTML-Code für kann mit Hilfe der Webseite wie folgt vorgegeben werden:

```
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
```

Man beachte, dass der Code innerhalb der $-Zeichen nicht umgebrochen werden kann. Bei der Entwicklung empfiehlt es sich u.U., zunächst den HTML Code getrennt zu erstellen und zu testen.

Mit Hilfe einer anderen Abhängigkeit als if systemstart() kann der Text und die Formatierung jederzeit auch während der Laufzeit des Programms verändert werden.

Das zweite weboutput-Feld soll zudem mit einer eigenen Grafik ausgestattet werden. Zunächst ist eine .png, .jpg oder .gif-Datei in den EibPC hochzuladen (S. 24). Der Pfad der Grafik für den weboutput ist dann `/upload/` + Dateiname. Damit wird kann die Grafik samt etwas Text und HTML Formatierung mit folgender Anweisung initialisiert werden:

```
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
```

Die Ausgabe kann auch von aktuellen Werten, z.B. Zählerständen eines KNX Geräts abhängig gemacht werden, was im folgenden gezeigt wird.

Ein Energiezähler misst auf der Gruppenadresse "Energie-2/3/5“,"Energie-2/3/6“ "Energie-2/3/7“ vom Typ u32 den Verbrauch in Wh. Wir definieren zunächst 3 Variablen in kWh als String (c1400)

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

Um zwölf Uhr soll täglich der Wert angezeigt werden:

```
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
```

Je nach den tatsächlich übertragenen Werten, wird am Webserver etwa wie in Abbildung 47 dargestellt die Anzeige sein:

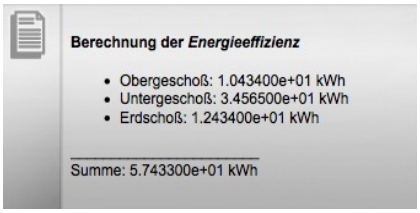


Abbildung 47: Dynamische Ausgabe

Im obigen Codeabschnitt wurde der HTML-String aus Teilstrings durch Verkettung („+“-Zeichen) gewonnen. Es ist wichtig sicherzustellen, dass die Verkettung die passende Stringlänge erzeugt. Die Funktion weboutput kann bis zu 65564 Bytes an das weboutput-Element übergeben. Die obige Verkettung besteht aber „nur“ aus $$ (= c1400) und einem c10000 String. Die Stringverkettung reserviert für das Ergebnis die Anzahl der Bytes, wie es das „längste“ String-Argument vorgibt. In diesem Fall sind dies 10.000 Bytes, gegeben durch den einen c10000 String im Code (vgl. oben).

Es sei an dieser Stelle nochmals angemerkt, dass Sonderzeichen aus mehreren Bytes bestehen können, wie schon auf S. 245 ausführlich beschrieben. Die Verkettung könnte theoretisch mehr als 10.000 Bytes als Ergebnis bringen, wenn die Strings jeweils die volle Länge ihrer Definition ausschöpfen. In diesem Fall würden die „überstehenden“ Zeichen nicht berücksichtigt bzw. schneidet die Verkettungsfunktion diese Zeichen vom String vor dem Kopieren in das Ergebnis ab. Es bleibt dem Anwender hier selbst überlassen, dies zu berücksichtigen.

Zurück zum Beispiel:

Den meisten Anwendern wird hier die Ausgabedarstellung der exponentiellen Fließkommadarstellung nicht gefallen. Daher sollten die Darstellung der Werte mit der Funktion stringformat noch lesbarer gemacht werden. Diese Funktion wandelt eine Zahl in einen String, wobei man führende Nullen, die angezeigte Genaugigkeit und Fließkommadarstellung parametrieren kann.

Argumente:

1. Wert (hier f32)

2. Umwandlung von F32 in Fließ-kommadarstellung: 4

3. Darstellung mit führenden Nullen: 4

4. Max. Länge: 8

5. Genauigkeit: 1 Stelle

```
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)
```

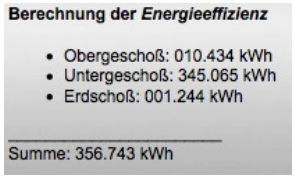


Abbildung 48: Formatierte Ausgabe

Visualisieren von Zeitreihen

Mit dem EibPC können auf einfache Weise Zeitreihen aufgenommen, permanent gespeichert und visualisiert werden. Die Speicherung erfolgt in TimeBuffern, die von TimeCharts dargestellt werden.

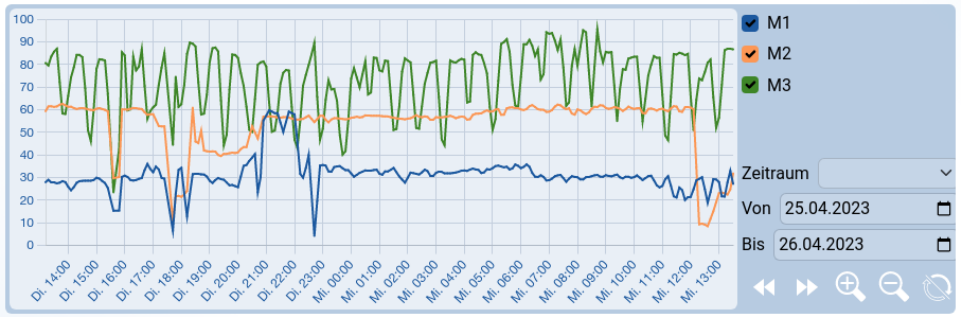


Abbildung 49: TimeChart Webelement EXTLONG

Abbildung 49 zeigt ein TimeChart mit drei von insgesamt vier möglichen Graphen und den Aktionsknöpfen. Der Anwender kann damit im TimeBuffer links und rechts scrollen, sowie zoomen. Außerdem kann der darzustellende Zeitbereich gewählt werden. Die Aktionsknöpfe sind Teil des TimeCharts selbst, so dass bei der Webelementprogrammierung kein weiterer Aufwand entsteht. Die Operationen werden auf alle Graphen angewandt und von allen geöffneten Visualisierungen dargestellt. Wurde der dargestellte Bereich des TimeCharts verändert, wird dies durch ein Hervorheben des Zurücksetzen-Knopfes dargestellt. Änderungen des TimeCharts durch das EibPC-Programm werden in diesem Zustand nicht mehr automatisch angezeigt, bis das TimeChart durch Drücken des Knopfes zurückgesetzt wurde (vgl. Abbildung 50).



Abbildung 50: Interaktives TimeChart

Zunächst müssen die darzustellenden TimeBuffer konfiguriert werden. Dazu betrachte man die folgende Definition (vgl. 221):

timebufferconfig(*TimeBufferID, MemType, Laenge, DataType*)

Mit dieser Funktion lassen bis zu 255 (ID 0 bis 254) TimeBuffer für die Aufzeichnung. *MemType* gibt an, ob der Speicher im Ring (0) oder linear (1) beschrieben wird (dazu unten mehr). Die Länge der max. Aufzeichnung der Zeitreihe wird mit *Laenge* (0u16 bis 65535u16) angegeben. Pro abgespeicherten Wert (s.u.) benötigt eine Zeitreihe 12 Bytes, unabhängig vom gespeicherten *DataType*. Daher empfiehlt es sich, diese Größe der TimeBuffer den tatsächlichen Bedürfnissen anzupassen: Eine Zeitreihe mit der maximalen Länge belegt 780 kB RAM.

DataType stellt eine repräsentative Zahl der Zeitreihe dar, z.B. 0f16 für 16-Bitzahlen oder 3% für u08 Werte. Die Zahl selbst wird nicht weiter verarbeitet und dient dem Compiler nur dazu, die Typinformation zu gewinnen. Wir wollen den TimeBuffer mit ID 0 für die Aufzeichnung der Temperatur-Gruppenaddresse 1/2/3 (vom Typ f16) nutzen, die ID 1 für die Stellgröße des Heizreglers 1/2/4 (u08).

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

Die Lesebarkeit des Codes wird erhöht, wenn wir im obigen Beispiel als letztes Argument die zu speichernde Variable oder Gruppenadresse angeben. Dies ist nicht zwingend notwendig, z.B. würde auch timebufferconfig(ChartBuffer1, MemType, Len, 2.2f16) oder timebufferconfig(ChartBuffer2, MemType, Len,2) den TimeBuffer richtig konfigurieren.

Nun müssen die Zeitreihen noch mit Daten „befüllt“ werden. Die Funktion

timebufferadd(*TimeBufferID, Daten*)

erledigt diese Aufgabe. Die Funktion schreibt den aktuellen Wert der Variable oder Gruppenadresse (*Daten*) sowie den Zeitstempel, der sich aus der Systemzeit des EibPC herleitet, in den Speicher der ausgewählten Zeitreihe. Damit besteht eine Zeitreihe genau genommen aus der Kombination Wert-Zeitstempel. Werte können bis zu 4 Byte lang sein, Zeitstempel benötigen intern 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

Abbildung 51:Aufbau der Zeitreihe (timebuffer)

Wie Abbildung 51 nahelegen soll, ist es nicht notwendigerweise so, dass die Werte im timebuffer im gleichen zeitlichen Abstand aufgenommen werden müssen, wenngleich dies beim Loggen von Energiedaten meist der Fall sein kann. Das Webelement mtimechart wertet den Zeitstempel automatisch richtig aus.

Wenn das Argument *MemType* von timebufferconfig als Ringspeicher definiert wurde, so wird beim Erreichen des letzten Wertes der Speicher wieder von vorne befüllt, d.h. der älteste Wert wird gegen den neuesten getauscht. Ist *MemType* als linearer Speicher definiert, so stoppt die Aufzeichnung, wenn der Speicher gefüllt ist.

Mit einer Zeitreihe verknüpfte Diagramme werden in der Visu automatisch aktualisiert, d.h. es können grundsätzlich die gleichen Zeitreihen in verschiedenen Diagrammen dargestellt werden. Um beispielsweise alle 15 Minuten einen Wert in den Puffer zu schreiben und die jeweils letzten 192 Werte in unserem Diagramm anzuzeigen, genügt folgender Code:

```
// Werte in den Buffer schreiben
if mtime(0,0) or mtime(15,0) or mtime(30,0) or mtime(45,0) then {
        timebufferadd(ChartBuffer0,"Temperature-1/2/3");
        timebufferadd(ChartBuffer1,"Controll-1/2/4");
} endif
```

Mit

timebuffersize(*TimeBufferID*)

kann der Füllstand eines Buffers jederzeit abgefragt werden.

Das mtimechart Webelement zeigt nun 192 Werte an, was einen Zeitraum von 2 Tagen gleichkommt. Unser Puffer hat Platz für 35040 Werte, was bei ¼ Stundenwerten einem Jahr Aufzeichnungsdauer entspricht. Abbildung 52 zeigt die Möglichkeiten für den Anwender, die vergangenen Werte darzustellen: Es kann ein Start- und Enddatum vorgegeben werden. Wenn in der Zeitreihe mehr als die im Webelement konfigurierte Anzahl von Werten in diesem Zeitraum abgespeichert werden, so passt das Diagramm die Anzeige so an, dass es Zwischenwerte ausblendet.

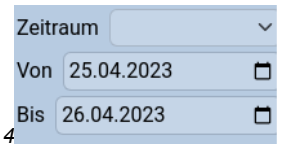


Abbildung 52: TimeChart Verschieben

Beispiel: Der Anwender stellt Zeitraum 4 Tage ein (z.B. 2013-07-11 bis 2013-09-13). Bei der hier gegebene Konfiguration sind im timebuffer (ID 0 und 1) 384 Werte abgelegt. Das Diagramm kann aber nur 192 Werte darstellen und lässt daher bei der Darstellung jeden zweiten Wert weg, so dass effektiv ½ Stundenwerte über 4 Tage angezeigt werden. Werteschwankungen, die nur im ¼ Stundenraster vorhanden sind, werden nicht mehr angezeigt. Die Zeitachse wird an die eingegebene Zeitdauer angepasst bzw. skaliert. Wenn der Anwender die Datumsfelder auf unterschiedliche Zeitintervalle parametriert, so wird die Achse so skaliert, dass die gespeicherten Werte vom ältesten zum neuesten Datum dargestellt werden.

Wichtig ist hier anzumerken: Alsbald der User ein Diagramm verschiebt oder skaliert, hängt er dieses vom Echtzeitwebserver ab, d.h. weitere Wertänderungen, die in die Zeitreihe (timebuffer) geschrieben werden, sind nicht mehr am Webserver sichtbar, bis der Zurücksetzen-Knopf gedrückt wurde. Dies betrifft aber nicht die anderen Elemente der Webseite.

Nachdem die Zeitreihe über einige Zeit im EibPC aufgenommen wurde, muss sichergestellt werden, dass diese auch beim Neueinspielen des Programms oder Neustart die Werte nicht verloren gehen. Die Funktionen

timebufferstore(*TimeBufferID*)
timebufferread(*TimeBufferID*)

sind für diese Aufgabe geschaffen (vgl. S. 222).

timebufferstore legt die Werte des TimeBuffers mit der *TimeBufferID* permanent in den Flashspeicher des EibPC ab, timebufferread liest einen abgespeicherten Puffer zurück. Zudem können die Werte mit EibStudio zur Sicherung der Daten heruntergeladen oder als CSV-Datei exportiert werden.

Somit speichern wir unsere Puffer alle 24 h auf folgende Weise:

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

Die Werte sichern wir beim Programmstart wie folgt zurück:

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

Weniger „Bedienkomfort“, speziell bei der Anwendung mit einem Touchpanel, aber dafür mehr Platz für die Darstellung bieten die Zeitdiagramme ohne Zeitwahl (vgl. Abbildung 53). In dieser Ausprägung erinnert das Diagramm an die mcharts bzw- mpcharts (vgl. S. 220 und S. 220), wobei auch hier die Zeitachse automatisch aus dem TimeBuffer geholt und skaliert wird.

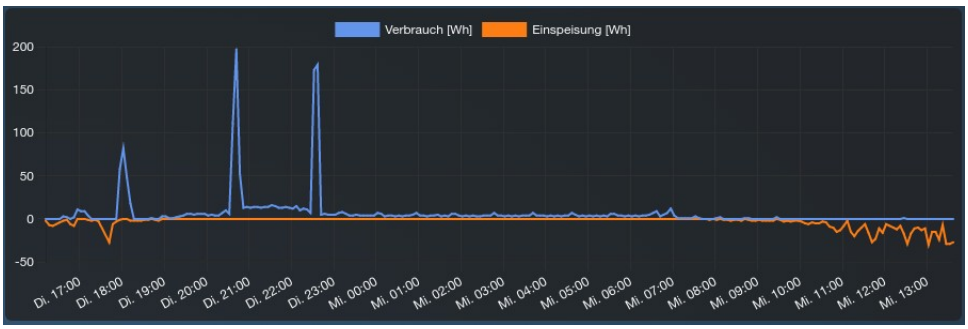


Abbildung 53: Standard Webelement

Mit der Funktion

mtimechartpos(TimeChartID,Graph,TimeBufferID,StartPos,EndPos)

(vgl. S. 223) kann auch vom Anwendungsprogramm die Anzeige ähnlich zur direkten Werteingabe in Abbildung 52 der Darstellungsbereich der mtimechart-Webelemente verändert werden.

mtimechartpos benötigt neben der ID und dem Graphenindex im mtimechart die Position des Wertebereichs der Daten im Puffer, an welcher der Wert steht. Wie in Abbildung 54 angedeutet, „nummeriert“ der EibPC jeden Speicherplatz von 0 bis n-1 durch. Dabei ist n die konfigurierte Pufferlänge, in Abbildung 54 sieht man einen Puffer mit der Länge 4000, der Startposition 0 und der Endposition 3999. Mit Hilfe von mtimechartpos greift man auf die entsprechende Stelle im TimeBuffer zurück, dabei ist Position 0 immer der älteste Wert im Puffer und Position n-1 (im Beispiel also die 3999) der aktuellste Wert im Puffer.

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

Abbildung 54:Aufbau der Zeitreihe (timebuffer) mit Index und Länge 4000

mtimechart wertet nicht den Index des Graphen aus, sondern den Wert des Zeitstempels selbst. Dabei sind die Zeitangaben StartZeit,EndZeit im Argument als utc-Millisekundenformat vorzugeben. Um dies für den Anwender einfach zu gestalten, kann auf die Funktion

utc(*Zeit*)

zurückgegriffen werden (vgl. 117). Diese wandelt eine Stringangabe der Form $2013-01-30 14:00:00$ in das utc-Millisekundenformat.

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

Wechsel der angezeigten Puffer eines mtimechart

Interessant ist die Möglichkeit, die im Webelement vorkonfigurierte Verknüpfung von Zeitreihe zum Graphen zu „lösen“ und im Graphen einen anderen Puffer zu darzustellen.

Dazu ein weiteres Beispiel: Wie in Abbildung 55 soll über ein mpshifter-Webelement eine Auswahl getroffen werden, die im aufgenommenen TimeBuffer dargestellt wird.

Das selbe Diagramm für unterschiedliche timebuffer verwenden: Hierzu wird mit der Auswahlbox unten links das Jahr gewählt. Das Anwendungsprogramm baut die Verbindung des Diagrammgrafen zum jeweiligen timebuffer auf.



Abbildung 55:Verändern der Darstellung während der Laufzeit

Im Webserver werden dazu die drei abgebildeten Elemente definiert, wobei der Shifter lediglich der Anzeige der aktuellen Uhrzeit dient. Beim Start des Anwendungsprogramms ist das Webelement auf den TimeBuffer mit der ID Chartbuffer3 verknüpft.

Wir definieren drei Zeitreihen (TimeBuffer),

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

die nun für jeweils 1 Jahr im ¼ Takt die Daten aufzeichnen:

```
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
```

Wenn nun der Anwender die Auswahlbox verändert, so soll der entsprechende TimeBuffer dargestellt werden:

Auswahlbox auswerten

```
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
```

Man erkennt, wie der Graph mit Index 0 des mtimechart auf die verschiedenen Timerbuffer über deren ID „umgelenkt“ wird. Dazu greifen wir auf die Funktion mtimechartpos zurück, die den Jahres-TimeBufferID jeweils mit dem Graphen 0 verknüpft.

Noch eine kleine Ergänzung für die Anzeige der Uhrzeit: Diese wird nun in der Visu sekundengenau angezeigt, da der Echtzeitwebserver diese bei jeder Änderung des „Sekundenzeigers“ entsprechend anpasst.

Ereignisse

Fehlercode	Bedeutung
ERR_PROC_OBJECT	Ein Objekt (eine Funktion) konnte nicht verarbeitet werden. Dies kann verschiedene, funktionsspezifische Ursachen haben. Bitte achten Sie auf weitere Fehlermeldungen.
ERR_PROC_OBJECT_MSG_OUT	Ein Ausgabeobjekt konnte nicht verarbeitet werden. Dies kann die folgenden Funktionen betreffen: 1 Schreibzugriff auf den KNX-Bus 1.1 settime 1.2 setdate 1.3 settimedate 1.4 write 1.5 read 1.6 writeresponse 1.7 scene 1.8 storescene 1.9 callscene 1.10 eibtelegramm 2 Netzwerkfunktionen 2.1 closetcp 2.2 connecttcp 2.3 ping 2.4 resolve 2.5 sendhtmlmail 2.6 sendmail 2.7 sendtcp 2.8 sendtcparray 2.9 sendudp 2.10 sendudparray 3. RS232-Schnittstelle 3.1 resetrs232 3.2 sendrs232 4. VPN-Server 4.1 closevpnuser 4.2 openvpnuser 4.3 startvpn 4.4 stopvpn Bitte überprüfen Sie, ob eine entsprechende Verbindung besteht.
ERR_PROC_REPETITIONS	Eine Endlosschleife ist erkannt worden. Die Verarbeitung wurde deshalb abgebrochen.
ERR_POW_OF_NEG_BASE	Bei der Verarbeitung der Funktion pow wurde der Fehler erkannt, dass die Basis negativ ist. Die Berechnung wurde deshalb nicht durchgeführt.
ERR_LOG_OF_NON_POS_BASE_OR_ARG	Bei der Verarbeitung der Funktion log wurde der Fehler erkannt, dass die Basis oder das Argument nicht positiv ist. Die Berechnung wurde deshalb nicht durchgeführt.
ERR_SQRT_OF_NON_POS_ARG	Bei der Verarbeitung der Funktion sqrt wurde der Fehler erkannt, dass das Argument negativ ist. Die Berechnung wurde deshalb nicht durchgeführt.
ERR_ASIN_OF_ARG_OUT_OF_RANGE	Bei der Verarbeitung der Funktion asin wurde der Fehler erkannt, dass das Argument außerhalb des Intervalls [-1; +1] liegt. Die Berechnung wurde deshalb nicht durchgeführt.
ERR_ACOS_OF_ARG_OUT_OF_RANGE	Bei der Verarbeitung der Funktion acos wurde der Fehler erkannt, dass das Argument außerhalb des Intervalls [-1; +1] liegt. Die Berechnung wurde deshalb nicht durchgeführt.
ERR_DIVISION_BY_ZERO	Bei der Verarbeitung einer Division wurde der Fehler erkannt, dass der Divisor gleich 0 ist. Die Berechnung wurde deshalb nicht durchgeführt.
ERR_EIBNET_IP_SETSOCKOPT_0	Es ist ein Fehler bei der Vorbereitung der Verbindung zu einer KNXnet/IP-Schnittstelle aufgetreten.
ERR_EIBNET_IP_SETSOCKOPT_1	s.o.
ERR_EIBNET_IP_SETSOCKOPT_2	s.o.
ERR_EIBNET_IP_SENDTO_0	Es ist ein Fehler beim Senden einer Nachricht an eine KNXnet/IP-Schnittstelle aufgetreten.
ERR_EIBNET_IP_SENDTO_1	s.o.
ERR_EIBNET_IP_SENDTO_2	s.o.
ERR_EIBNET_IP_SENDTO_3	s.o.
ERR_EIBNET_IP_SENDTO_4	s.o.
ERR_EIBNET_IP_SENDTO_5	s.o.
ERR_EIBNET_IP_TIMEOUT_SEARCH	Es konnte keine KNXnet/IP-Schnittstelle gefunden werden. Bitte überprüfen Sie, ob eine betriebsbereite KNXnet/IP-Schnittstelle mit demselben Netzwerk wie der EibPC verbunden ist.
ERR_EIBNET_IP_DISCONNECT_REQUEST_IN	Die Verbindung zwischen EibPC und KNXnet/IP-Schnittstelle wurde getrennt.

ERR_EIBNET_IP_DISCONNECT_REQUEST_OUT	s.o.
ERR_EIBNET_IP_TIMEOUT_CONNECTIONSTATE_REQUEST	s.o.
ERR_EIBNET_IP_E_CONNECTION_ID	s.o.
ERR_EIBNET_IP_E_DATA_CONNECTION	Die KNXnet/IP-Schnittstelle hat einen Fehler der Verbindung zum EibPC festgestellt.
ERR_EIBNET_IP_E_KNX_CONNECTION	Die KNXnet/IP-Schnittstelle hat einen Fehler der Verbindung zum KNX-Bus festgestellt.
ERR_EIBNET_IP_TUNNELLING_TIMEOUT_0	Eine Nachricht wurde erneut zur KNXnet/IP-Schnittstelle gesendet, weil ein Fehler aufgetreten ist.
ERR_EIBNET_IP_TUNNELLING_TIMEOUT_1	Die Verbindung zwischen EibPC und KNXnet/IP-Schnittstelle wurde getrennt.
ERR_EIBNET_IP_L_DATA_CON	Es wurde eine Bestätigung von der KNXnet/IP-Schnittstelle für eine an diese gesendete Nachricht empfangen.
ERR_FT12_LINE_IDLE_TIMEOUT_0	Es ist ein Fehler bei der Verbindung zur FT1.2-Schnittstelle aufgetreten.
ERR_FT12_LINE_IDLE_TIMEOUT_1	s.o.
ERR_FT12_SELECT	s.o.
ERR_FT12_INVALID_TELEGRAM	s.o.
ERR_FT12_READ	s.o.
ERR_FT12_RESET_REQ_IN	Die Verbindung zur FT1.2-Schnittstelle wurde zurückgesetzt.
ERR_FT12_STATUS_REQ_IN	Es wurde eine Statusanfrage von der FT1.2-Schnittstelle empfangen.
ERR_FT12_L_BUSMON_IND	Es wurde eine Nachricht vom KNX-Bus über die FT1.2-Schnittstelle empfangen.
ERR_FT12_FIX_LENGTH_END	Eine Nachricht von der FT1.2-Schnittstelle war fehlerhaft.
ERR_FT12_FIX_LENGTH_CHECKSUM	s.o.
ERR_FT12_VAR_LENGTH_LENGTH_0	s.o.
ERR_FT12_VAR_LENGTH_LENGTH_1	s.o.
ERR_FT12_VAR_LENGTH_START	s.o.
ERR_FT12_VAR_LENGTH_CHECKSUM	s.o.
ERR_FT12_VAR_LENGTH_END	s.o.
ERR_FT12_L_DATA_CON	Es wurde eine Bestätigung von der FT1.2-Schnittstelle für eine an diese gesendete Nachricht empfangen.
ERR_FT12_IN_BUFFER_FULL	Es ist ein Fehler bei der Verbindung zur FT1.2-Schnittstelle aufgetreten.
ERR_MEM_OBJECTS_COUNT	Obsolet in V3
ERR_MEM_OBJECT_OBJECT_TYPE	Obsolet in V3
ERR_MEM_OBJECT_CALC_TYPE	Obsolet in V3
ERR_MEM_OBJECT_BIT_LEN	Obsolet in V3
ERR_MEM_OBJECT_DATA_SIZE	Obsolet in V3

ERR_MEM_OBJECT_NAME	Obsolet in V3
ERR_MEM_OBJECT_EXPRESSION	Obsolet in V3
ERR_MEM_OBJECT_INPUT_COUNTER_0	Obsolet in V3
ERR_MEM_OBJECT_INPUTS_0	Obsolet in V3
ERR_MEM_OBJECT_DEPENDENCY_COUNTER_0	Obsolet in V3
ERR_MEM_OBJECT_DEPENDENCIES_0	Obsolet in V3
ERR_MEM_OBJECT_DEPENDENCY_COUNTER_1	Obsolet in V3
ERR_MEM_OBJECT_DEPENDENCIES_1	Obsolet in V3
ERR_MEM_OBJECT_NULL	Obsolet in V3
ERR_MEM_OBJECT_NO_ERROR	Obsolet in V3
ERR_MSGSND_ASYNC_SERIAL_0	Ein Fehler bei der Kommunikation mit der asynchronen seriellen Benutzer-Schnittstelle wurde festgestellt, weil eine interne Warteschlange nicht verfügbar war. Möglicherweise ist der EibPC mit dem aktuellen Anwendungsprogramm zeitweise überlastet.
ERR_MSGSND_ASYNC_SERIAL_1	s.o.
ERR_MSGSND_MSGOUT_0	Der Zugriff auf den KNX-Bus war nicht möglich, weil eine interne Warteschlange nicht verfügbar war. Möglicherweise ist der EibPC mit dem aktuellen Anwendungsprogramm zeitweise überlastet.
ERR_MSGSND_MSGOUT_1	s.o.
ERR_MSGSND_MSGOUT_2	s.o.
ERR_MSGSND_MSGOUT_3	s.o.
ERR_MSGSND_MSGOUT_4	s.o.
ERR_MSGSND_MSGOUT_5	s.o.
ERR_MSGSND_RESOLVE_0	Die Funktion resolve konnte nicht ausgeführt werden, weil eine interne Warteschlange nicht verfügbar war. Möglicherweise ist der EibPC mit dem aktuellen Anwendungsprogramm zeitweise überlastet.
ERR_MSGSND_INTERFACE_IN_0	Eine vom KNX-Bus empfangene Nachricht konnte nicht an das Anwendungsprogramm übergeben werden, weil eine interne Warteschlange nicht verfügbar war. Möglicherweise ist der EibPC mit dem aktuellen Anwendungsprogramm zeitweise überlastet.
ERR_MSGSND_INTERFACE_IN_1	s.o.
ERR_MSGSND_INTERFACE_IN_2	s.o.
ERR_MSGSND_MAIL_0	Eine E-Mail-Nachricht konnte nicht versendet werden, weil eine interne Warteschlange nicht verfügbar war. Möglicherweise ist der EibPC mit dem aktuellen Anwendungsprogramm zeitweise überlastet.
ERR_MSGSND_MAIL_1	s.o.
ERR_MSGSND_TCP_OUT_0	Eine TCP-Nachricht konnte nicht versendet werden, weil eine interne Warteschlange nicht verfügbar war. Möglicherweise ist der EibPC mit dem aktuellen Anwendungsprogramm zeitweise überlastet.
ERR_MSGSND_TCP_OUT_1	Eine TCP-Verbindung konnte nicht hergestellt werden, weil eine interne Warteschlange nicht verfügbar war. Möglicherweise ist der EibPC mit dem aktuellen Anwendungsprogramm zeitweise überlastet.

ERR_MSGSND_TCP_OUT_2	Eine TCP-Verbindung konnte nicht getrennt werden, weil eine interne Warteschlange nicht verfügbar war. Möglicherweise ist der EibPC mit dem aktuellen Anwendungsprogramm zeitweise überlastet.
ERR_MSGSND_TCP_IN_0	Eine empfangene TCP-Nachricht konnte nicht an das Anwendungsprogramm übergeben werden, weil eine interne Warteschlange nicht verfügbar war. Möglicherweise ist der EibPC mit dem aktuellen Anwendungsprogramm zeitweise überlastet.
ERR_MSGSND_UDP_OUT_0	Eine UDP-Nachricht konnte nicht versendet werden, weil eine interne Warteschlange nicht verfügbar war. Möglicherweise ist der EibPC mit dem aktuellen Anwendungsprogramm zeitweise überlastet.
ERR_MSGSND_UDP_IN_0	Eine empfangene UDP-Nachricht konnte nicht an das Anwendungsprogramm übergeben werden, weil eine interne Warteschlange nicht verfügbar war. Möglicherweise ist der EibPC mit dem aktuellen Anwendungsprogramm zeitweise überlastet.
ERR_MSGSND_PING_0	Die Funktion ping konnte nicht ausgeführt werden, weil eine interne Warteschlange nicht verfügbar war. Möglicherweise ist der EibPC mit dem aktuellen Anwendungsprogramm zeitweise überlastet.
ERR_MSGSND_TCP_OUT_3	Eine TCP-Nachricht ohne Nullterminierung konnte nicht versendet werden, weil eine interne Warteschlange nicht verfügbar war. Möglicherweise ist der EibPC mit dem aktuellen Anwendungsprogramm zeitweise überlastet.
ERR_MSGSND_UDP_OUT_1	Eine UDP-Nachricht ohne Nullterminierung konnte nicht versendet werden, weil eine interne Warteschlange nicht verfügbar war. Möglicherweise ist der EibPC mit dem aktuellen Anwendungsprogramm zeitweise überlastet.
ERR_MSGSND_ASYNC_SERIAL_2	Ein Fehler bei der Kommunikation mit der asynchronen seriellen Benutzer-Schnittstelle wurde festgestellt, weil eine interne Warteschlange nicht verfügbar war. Möglicherweise ist der EibPC mit dem aktuellen Anwendungsprogramm zeitweise überlastet.
ERR_EXIT_NCONF_0	Das Anwendungsprogramm wurde beendet. Dieser Vorgang wurde durch eine Aktion im EibStudio ausgelöst.
ERR_EXIT_NCONF_1	s.o.
ERR_EXIT_NCONF_2	s.o.
ERR_EXIT_NCONF_3	s.o.
ERR_EXIT_MAIN_0	Das Anwendungsprogramm wurde aufgrund eines internen Fehlers beendet.
ERR_EXIT_MAIN_1	Das Anwendungsprogramm wurde aufgrund eines internen Fehlers beendet.
ERR_EXIT_MAIN_2	Das Anwendungsprogramm wurde aufgrund eines internen Fehlers beendet.
ERR_EXIT_MAIN_3	Das Anwendungsprogramm wurde aufgrund eines internen Fehlers beendet.
ERR_EXIT_MAIN_4	Das Anwendungsprogramm wurde aufgrund eines internen Fehlers beendet.
ERR_LED_MUTEX_TRYLOCK	Obsolet in V3
ERR_READ_GROUP_ADDRESS	Eine Gruppenaddresse wurde mit initga parametriert, antwortet jedoch nicht auf die Leseanforderung.
ERR_ERRNO	Ein interner Fehler wurde erkannt. Der Fehlertyp kann durch den Hersteller anhand des Fehlercodes genauer bestimmt werden.
ERR_ASYNC_SERIAL_0	Es ist ein Fehler beim Zugriff auf die asynchrone serielle Benutzer-Schnittstelle aufgetreten.
ERR_ASYNC_SERIAL_1	s.o.
ERR_ASYNC_SERIAL_2	s.o.
TIMEBUFFER_DATATYPE_ERROR	Obsolet in V3
TIMEBUFFER_DATATYPE_ERROR	Obsolet in V3

TIMEBUFFER_DATATYPE_ERROR	Obsolet in V3
ERR_KNXNET_IP_L_DATA_CON	Das Bustelegramm wird nicht mit einem ACK bestätigt, weil kein Gerät mir der Gruppenadresse verknüpft ist.

Fehlermeldungen des Compilers

Fehlermeldung	Lösung
! Vorgabewert zu groß für gegebenen Datentyp in >xy< !	Wert muss mit Datentyp angegeben werden z.b. Helligkeit<2000u16
! Benutzten Sie Konvertierungsfunktionen: Datentyp der Parameter nicht gleich: >Var1+Var2< !	Var3=convert(Var1,Var2) + Var2
Syntaxfehler in Zeile:[17] >if (("KücheGesamt-1/1/9"==Ein) and wtime(23,00,00,00)) < Gültig bis Position: STOP--> and wtime(23,00,00,00))	Anweisung muss in einer Zeile stehen oder die Zeile muss mit ' \\' abgeschlossen werden. if ..... and \\ then ....
! Vordefinierte Variable kann nicht neu definiert werden in >EIN=1b01< !	Im EibParser sind Variablen vordefiniert, um das Erstellen eines EibProgramms möglichst einfach zu gestalten. Die Predefines werden im EibStudio im rechten Bildteil aufgeführt. Sie können nicht neu definiert werden.
Datentyp der Parameter nicht gleich: >sun()==1< !	Der Rückgabewert der Funktion ist binär. Eine Zahl ohne Datentyangabe ist immer ein 8bit Wert ohne Vorzeichen. Als Vergleichsoperator muss ein Binärwert angegeben werden. sun()==1b01
Syntaxfehler in Zeile:[13] >a=4,6e1f32< Gültig bis Position: STOP--> ,6e1f32	Als Dezimaltrennzeichen muss immer ein “.“ verwendet werden.
Syntaxfehler in Zeile:[21] >"Akt1-0/0/5"=after(a,5000u64)<	Eine direkte Zuweisung ist nur bei Variablen, nicht bei Adressen möglich. Das Schreiben einer Information auf den KNX-Bus wird mit der Hilfe der write-Funktion realisiert. write(„Akt1-0/0/5“, 1b01)
Syntaxfehler in Zeile:[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<	Mehrere Anweisungen in einer if-Anweisung müssen mit “;“ getrennt werden. if(a=EIN) then write(b=EIN); write(c=AUS) endif
Syntaxfehler in Zeile:[26] >write(ein,EIN)< Ungültiger Datentyp in >write(ein<	Die write-Funktion kann nur auf Gruppenadressen wirken (1. Argument), nicht aber Variablen.
Deklaration der Variable muss eindeutig sein in >u=convert(z,r)-r-e<	Jede Variable darf nur einmal deklariert werden. Eine weitere Deklaration bringt diese Fehlermeldung.
Falscher Datentyp in >cycle(0.5,5<	Es dürfen nur ganzzahlige Werte eingegeben werden.

Lizenzen

Der EibPC² verwendet Software die unter verschiedenen Lizenzen kostenlos zur Verfügung gestellt wird. Falls es von der entsprechenden Lizenz gefordert ist, wird der Quellcode auf Anfrage zur Verfügung gestellt.

Enertex® EibPC²

Betriebssystem: Debian Linux 9: Kernel 4.14.16

Enertex® EibStudio

Eine Liste der verwendeten Bibliotheken kann unter **Hilfe → Lizenzen** eingesehen werden.

Folgende Softwarepakete werden eingesetzt:

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"),
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 furnished 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,
FITNESS 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.

libmodbus

GNU LESSER GENERAL PUBLIC LICENSE
Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc.
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL. It also counts
as the successor of the GNU Library Public License, version 2, hence
the version number 2.1.]

Preamble

The licenses for most software are designed to take away your
freedom to share and change it. By contrast, the GNU General Public
Licenses are intended to guarantee your freedom to share and change
free software--to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some
specially designated software packages--typically libraries--of the
Free Software Foundation and other authors who decide to use it. You
can use it too, but we suggest you first think carefully about whether
this license or the ordinary General Public License is the better
strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use,
not price. Our General Public Licenses are designed to make sure that
you have the freedom to distribute copies of free software (and charge
for this service if you wish); that you receive source code or can get
it if you want it; that you can change the software and use pieces of
it in new free programs; and that you are informed that you can do
these things.

To protect your rights, we need to make restrictions that forbid
distributors to deny you these rights or to ask you to surrender these
rights. These restrictions translate to certain responsibilities for
you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis
or for a fee, you must give the recipients all the rights that we gave
you. You must make sure that they, too, receive or can get the source
code. If you link other code with the library, you must provide
complete object files to the recipients, so that they can relink them
with the library after making changes to the library and recompiling
it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the
library, and (2) we offer you this license, which gives you legal
permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that
there is no warranty for the free library. Also, if the library is
modified by someone else and passed on, the recipients should know
that what they have is not the original version, so that the original

author's reputation will not be affected by problems that might be 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.

Copyright (c) 2004, 2005 Metaparadigm Pte Ltd
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 furnished 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,
FITNESS 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.

Versionsgeschichte

38: 19.11.2023, Dipl.-Inf. F. Naumann

- Nummerierung an 1159-HB_EibPC2_DE angepasst. Änderungsverzeichnis wird ab jetzt analog geführt
- Anpassung von Icons an neue Visu
- MQTT eingefügt

39: 19.12.2023, Dipl.-Inf. F. Naumann

- Icons aktualisiert

40: 15.7.2024, Dipl.-Inf. F. Naumann

- Neue Funktionen httpapi, httpdownload, modbusmastertimings, rtsp, doorbird, queuelength, webdialog
- Neues Argument readmodbus, writemodbus
- Icons aktualisiert

41: 20.11.2024, Dipl.-Inf. F. Naumann

- Neue Abschnitte KNX Secure Sequenzzähler, KNX Data Secure, IoT-Geräte via MQTT
- Reihenfolge und Zuordnung der Abschnitte geändert

42: 28.10.2025, Dipl.-Inf. F. Naumann

- Neue Funktionen historybufferconfig, historybufferclear, historychart, replace, blackout, ready, webtts

43: 13.4.2026, Dipl.-Inf. F. Naumann

- pdisplay: verändert keinen link/plink
- display/webdisplay: verändert keinen webinput
- Neue Funktionen setwebinput, timeswitch
- Neues Visu-Element timeswitch
- Icon-Zustand für alle Visu-Elementdefinitionen
- Frame mit flexibler Größe fframe