Enertex® EibPC und Command Fusion
- Ein Blick hinter die Kulissen

Enertex® EibPC

Die Ausgangssituation

Mit Hilfe von Command Fusion können grafisch anspruchsvolle Visualisierungen für das Apple iPhone oder Ipod generiert werden, z.B. http://knx-user-forum.de/87756-post5.html(vgl. Abbildung 1).

Abbildung 1: Eine Typische Visu mit Command Fusion

Ziel ist es, diese Visualisierung an den KNX-BUS anzubinden. D.h. jeder Knopf, Slider oder Textanzeige soll durch einen sogenannten Join („Verbindung“) mit einer Variablen des EibPC-Anwendungsprogramms oder KNX Busadressen verknüpft werden. Wird der Knopf in der Visu am iPhone betätigt, sendet der Join seine ID über LAN an den EibPC und dieser kann nun z.B. auf die entsprechende Gruppenadresse das EIN-Telegramm senden.

Empfängt der EibPC auf dieser Gruppenadresse ein Telegramm, weil z.B. ein Anwender den Lichtschalter betätigt, so sendet der EibPC diese Information an das iPhone über den Join, so dass die Anzeigeelemente am Telefon den aktuellen Status des Lichtaktors wiedergeben. (vgl. Abbildung 2).

Command Fusion kommuniziert dabei mit dem EibPC über eine TCP/IP Verbindung, wobei der EibPC als Server fungiert und die Daten in ASCII-Form übermittelt werden. Der EibPC stellt hier nun Makros bereit, mit denen in wenigen Klicks und dem Makroassistenten eine Konfiguration vorgenommen werden kann. Im folgenden soll dies kurz umrissen werden, um dann einen Blick hinter die Kulissen zu wagen: Wie macht man das im Detail: Wie verarbeitet man ASCII-Strings mit dem Enertex® EibPC?

Letztlich ist es Ziel, einen Prototypen eines Interpreters für TCP/IP, UDP und RS232 Verbindungen und der Verarbeitung von Datenströmen über dieses Schnittstellen aufzuzeigen.

Dies soll auch Anwendern die Möglichkeit geben, andere ähnlich gelagerte Problemstellungen auf die gleiche Art zu lösen.

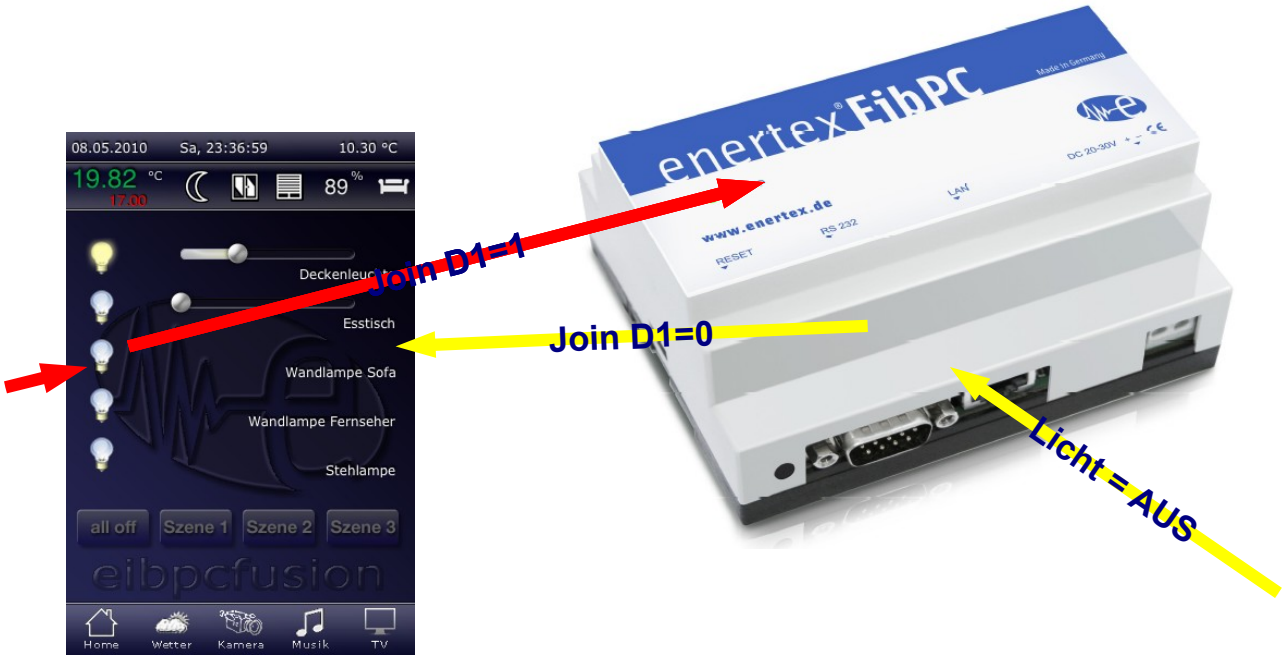

Abbildung 2: Grundsätzliche Kommunikation: Vom iPhone zum EibPC, vom KNX Bus zum EibPC und dann zum iPhone

Das bequeme Arbeiten mit Makros und Command Fusion

Als erstes zeigen wir, wie wir den EibPC mit dem Command Fusion (CF) auf einfache Weise konfigurieren. Zunächst muss mit dem guiDesigner die Oberfläche genieriert werden. Dabei trägt mandie IP Adresse des EibPCs im Projekt (guiDesigner) unter den Projekteinstellungen (Edit/Project Properties) im Feld "IP Address" ein. Als Port muss dort 4809 eingetragen werden.

Um die Visu auf Ihr iPhone hochzuladen, nutzen Sie guiDesigner (File-Upload Service). Dazu müssen Sie auf dem iPhone zunächst in den Einstellungen zum iViewer im Feld "File Url" das gleiche wie im Upload Service (Ihr PC) angezeigt eingeben und zunächst "Reload GUI" wählen. Anschließend einmal iViewer starten und bedienen. Am Ende der Session die Einstellung "Reload GUI" deaktivieren, d amit die GUI im iPhone gespeichert bleibt.

CommandFusion	EibPC an Command Fusion koppeln.
Join2GA	Command Fusion - Join für GA und separater StatusGA
Join2Dimmer	Command Fusion - Join für einen Dimmer
Join2Status	Command Fusion - Join für eine Variable anlegen
Join2Var	Command Fusion - Join für eine Variable anlegen
JoinToggle2GA	Command Fusion - Toggle-Join "Umschalter" für GA
JoinToggle2Var	Command Fusion - Toggle-Join "Umschalter"

Abbildung 3: Verfügbare Makros

Der eigentliche TCP/IP Server und das Messagehandling wird mit dem Makro CommandFusion angelegt. Dieses beinhaltet sämtliche Kommunikation mit dem Command Fusion auf dem iPhone, iPad oder Ähnliches.

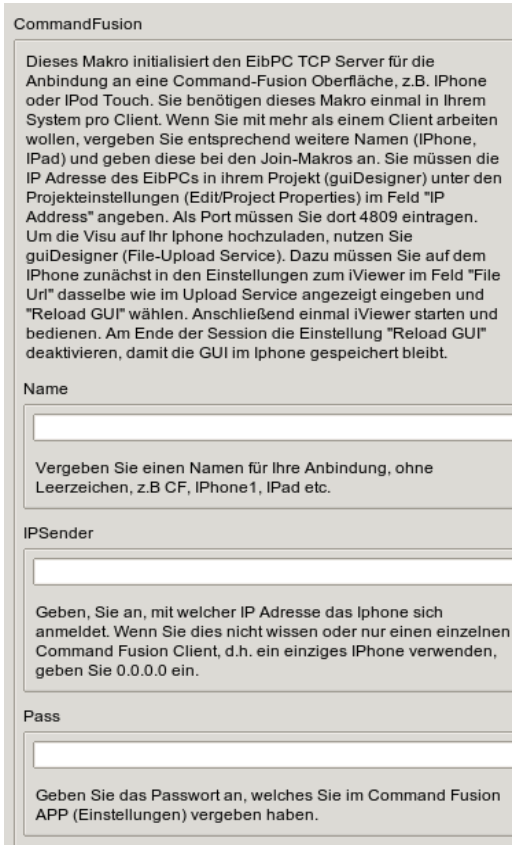

Abbildung 4: Parametrierung des Makros CommandFusion

Zunächst muss das Makro CommandFusion eingebunden werden ein. Grundsätzlich kann der Enertex® EibPC mit mehr als einen Command Fusion Teilnehmer arbeiten. Dazu muss allerdings die IP Adresse jedes Teilnehmers, also z.B. iPhone, iPad im Makro mit angegeben werden. Falls nur mit einem einzigen Teilnehmer gearbeitet wird muss keine IP angegeben werden bzw. einfach im Makro an der entsprechenden Stelle 0.0.0.0 vorgeben werden.

Im nächsten Schritt können nun die Makros genutzt werden:

- **Join2GA**

  Eine Gruppenadresse und eine Statusadresse direkt mit einem Join verknüpfen. Sendet CF an den EibPC so wird der Wert direkt auf den KNX Bus geschrieben. Die Rückantwort an das CF wird initiiert, wenn sich die Status-Gruppenadresse ändert.

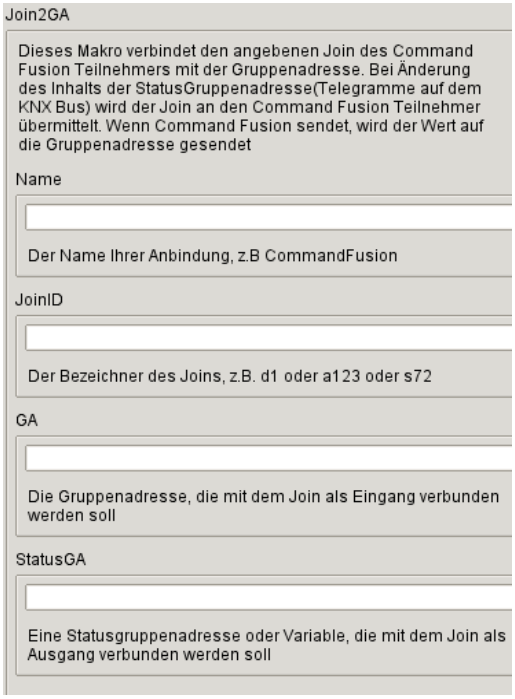

Abbildung 5: Parametrierung des Makros Join2GA

- **Join2Var**

  Eine Variable und eine Statusvariable direkt mit einem Join verknüpfen. Sendet CF an den EibPC so wird die Variable auf den Wert gesetzt. Wenn die Statusvariable sich im Anwendungsprogramm ändert, schickt der EibPC an das CF eine entsprechende Nachricht.

CF unterscheidet drei verschiedene Datentypen:

- „Digital“: Wert 0 oder1,
- „Analog“: 0 bis 65535
- „Seriell“: Text

Die Verknüpfung der Joins mit Gruppenadressen bzw. Variablen erreicht man mit den oben genannten Makros. *Diese Makros konvertieren auch gleich die Datentypen in den jeweiligen Zieldatentyp*. Der Datentyp des Joins wird anhand des ersten Buchstabens identifiziert (d für digital, a für analog und s für seriell). Wie die Joins im Command Fusion realisiert werden müssen, kann der guiDesigner Dokumentation entnommen werden.

Eine Besonderheit gibt es noch bei digitalen Joins, die mit Press-Release Knöpfen arbeiten. Hier wird beim Drücken der Wert 1 und beim Loslassen der Wert 0 gesendet. Dabei möchte man ggf. nur eine Zuweisung nutzen. Dies kann mit Hilfe des Makros mit

- **Join2ToggleVar**

  Eine Variable und eine Statusvariable direkt mit einem Join verknüpfen. Sendet CF an den EibPC, so wird die Variable auf den Wert gesetzt, wenn dem Join der Wert EIN gesendet wurde. Wenn die Statusvariable sich im Anwendungsprogramm ändert, schickt der EibPC an das CF eine entsprechende Nachricht.

- **Join2ToggleGA**

  Eine Gruppenadresse direkt mit einem Join verknüpfen. Sendet CF an den EibPC so wird die Gruppenadresse invertiert auf den Bus geschrieben (steht sie auf EIN wird sie auf AUS) gesetzt.

- **Join2Status**

  Eine Statusvariable (oder Gruppenandresse mit einem Join verknüpfen. Sendet CF an den EibPC, so wird dies ignoriert. Wenn die Statusvariable sich im Anwendungsprogramm ändert, schickt der EibPC an das CF eine entsprechende Nachricht.

- **Join2Dimmer**

Eine Gruppenadresse und eine Statusgruppenadresse direkt mit einem analogen Join verknüpfen. Hier wird intern eine Skalierung des analogen Joins vorgenommen, so dass direkt eine Dimmergruppenadresse verwendet werden kann.

Mit Hilfe dieser Makros kann nun die Visualisierung mit dem CommandFusion auf einfach Weise wie in Abbildung 6 gezeigt erreicht werden.

```
[Macros]
CommandFusion(CF,0.0.0.0,EibPC)
Join2Var(CF,a612,Test,Test)
Join2GA(CF,d601,"Licht2-0/0/2","Licht2-0/0/2")
JoinToggle2Var(CF,d601,MyVar,MyStatus)
JoinToggle2GA(CF,d604,"Licht1-0/0/1","Licht1-0/0/1")
[MacroLibs]
//Makro-Bibliotheken
EnertexCommandFusion.lib
```

Abbildung 6: Eine beispielhafte Konfiguration im EibStudio

Hinter den Kulissen – der Scheduler

Voraussetzung

Schauen wir nun etwas tiefer in die Makros und die Kommunkation. Command Fusion sendet einen textbasiereten Datenstrom über das TCP/IP Protokoll an den EibPC, wobei dieser den TCP Server darstellt. Dass der EibPC als solcher TCP/IP Server arbeiten kann, ist ab Firmware v1.200 möglich.

Das Protokoll von Command Fusion - Ein ASCII-basierter Datenstrom

Ist die Verbindung als solche aufgebaut, verschickt Command Fusion Text-Strings. Jede Textnachricht ist mit einem Endzeichen abgeschlossen. Das Endzeichen hat den Wert 0x03, welches lt. ASCII Standard auch End-of-Text (EOT) genannt wird und kein „darstellbares“ Zeichen ist. Ein beispielhaftes Kommando ist in Abbildung 7 dargestellt, wobei wir EOT als senkrechtes „Zeichen“ dargestellt haben.

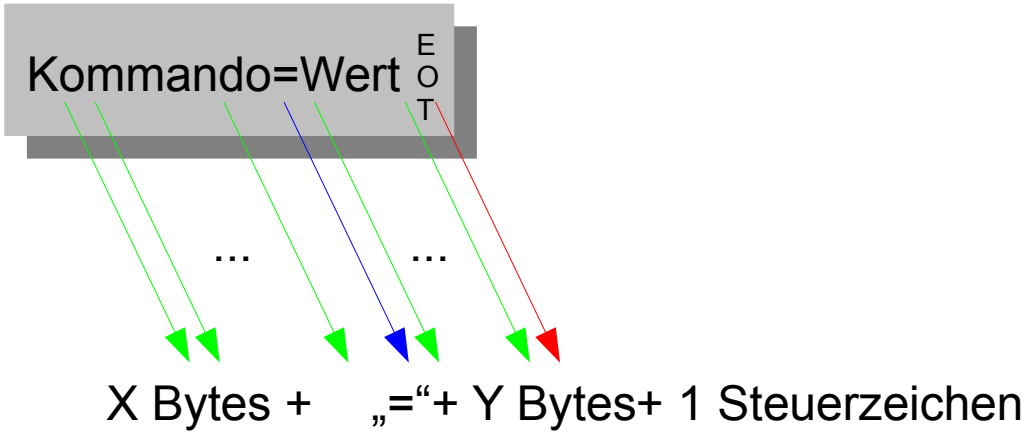

Abbildung 7: Ein beispielhafter Datenstrom an den EibPC

Command Fusion baut dabei die Kommandos nach dem Schema Komandotext „=“ Wert plus 1x EOT auf. In Abbildung 8 ist das Passwort-Kommando dargestellt, welches nur aus dem Buchstaben „p“ und dem Passwort „EibPC“ besteht. Dieses Kommando wird von Command Fusion nach dem Verbindungsaufbau an den EibPC geschickt, wobei das eigentliche Passwort als „Wert“ fungiert.

Alle Antworten, die an das Command Fusion geschickt werden, müssen ihrerseits genauso aufgebaut sein, insbesondere auch mit dem EOT-Marker abgeschlossen werden. Auf das Passwort muss der EibPC dann mit p=ok{EOT} antworten.

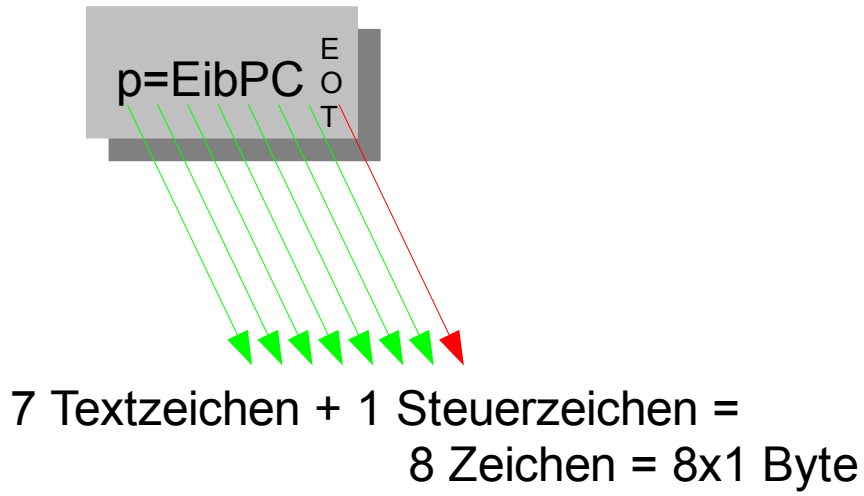

Abbildung 8: Ein beispielhafter Datenstrom an den EibPC

Command Fusion baut dabei die Kommunikation mit Joins nach diesem Schema auf:

- **dXX:** Digitaler Join, wobei XX für eine beliebige Nummer steht. Als Wert ist 0 oder 1 möglich.
- **aXX:** Analoger Join, wobei XX für eine beliebige Nummer steht. Als Wert ist 0...65365 möglich.
- **sXX:** Serieller Join, wobei XX für eine beliebige Nummer steht. Als Wert ist ein beliebiger Text möglich.

Schließlich gibt es noch das Lebenzeichen oder „Heart Beat Command“, welches alle 3 Sekunden von Command Fusion an den EibPC geschickt wird. Hier schickt Command Fusion h=0{EOT} und erwartet h=1{EOT} als Antwort.

Der Datenstrom erfolgt grundsätzlich kontinuierlich, d.h. es werden u.U. gleich mehrere Kommandos auf den Weg geschickt.

Verarbeitung

Aufgabenteilung

Damit wäre das Protokoll an sich geklärt, und nun die Frage: Wie machen wir das denn nun ganz konkret mit dem EibPC und seiner Programmierung?

Wir trennen die Aufgabe:

1. ***Nachrichtenwarteschlange***
   Wir benötigen einen Software-Teil, der Kommandos empfängt und den Datenstrom abspeichert. In unserem Fall dient ein String, im Programm „CF_MQueue“, als Warteschlange. Dabei müssen wir aufpassen, dass der String nicht mehr 1400-Zeichen lang ist. Diese Warteschlange muss völlig unabhängig von der weiteren Verarbeitung sein. Wenn eine neue Nachricht eintrifft, so muss dies erkannt werden und die Warteschlange erweitert werden, ohne dass die anderen Programmteile davon betroffen wären.
2. ***Kommando-Stapel***
   Wir benötigen einen Softwareteil, der den String verarbeitet, und in einzelne Teile zerlegt. Diese Teile stellen Kommandos (gespeichert im String „CF_Command“) und deren Werte (gespeichert im String „CF_Argument“) dar. Im Beispiel von Abbildung 8 wäre demnach CF_Command „p“ und CF_Arg gleich „EibPC“.
3. **Kommando-Verarbeitung**
   Nachdem das Kommando nun vorliegt, muss es verarbeitet werden. Hierzu müssen die verschiedenen Aspekte des Command Fusion Protokolls umgesetzt werden. Wenn das Kommando abgeschlossen bzw. verarbeitet wurde, soll dies dem Kommando Stapel mitgeteilt werden und das nächste Kommando verarbeitet werden.
   a) Systemkommandos: Kommandos, wie das Lebenszeichen, Passwortübermittlung etc. erforden keine Parametrierung durch den Anwender. Diese sind fester Bestandteil der Kommunikation.

b) Sollte ein Kommando nicht erkannt werden, so soll dieses verworfen werden.

c) Variablenzuweisungen/Joins: Dies ist das Ziel der Übung: Command Fusion fragt nach Variablen, bzw. schreibt über den EibPC auf den KNX-Bus.

Die Punkte 1., 2., 3.a) und 3.b). sind vom Anwender nicht mehr zu verändern, da diese nur vom Protokoll abhängen. Diese werden daher in einem Makro „CommandFusion“ zusammengefasst.

Die Vergabe der Verknüpfungspunkte mit dem Anwenderprogramm wird in den „Join2...“-Makros erledigt.

Die drei Aufgaben können als unabhängige Prozesse gesehen werden. Wir müssen nun von einem Prozess zum anderen eine Verbindung aufbauen, so dass die Warteschlange dem Kommando-Stapel mitteilen kann, wann denn ein neues Kommando eingetroffen ist. Die Kommandoverarbeitung muss ihrerseits vom Kommando-Stapel das nächste Kommando zur Verarbeitung gestellt bekommen und „getriggert“ werden. Diese Verbindungen legen wir mit Hilfe von zwei Variablen `CF_Next` und `CF_CommandCount` an. Dazu sehen wir in Abbildung 9 das Schema in der Übersicht.

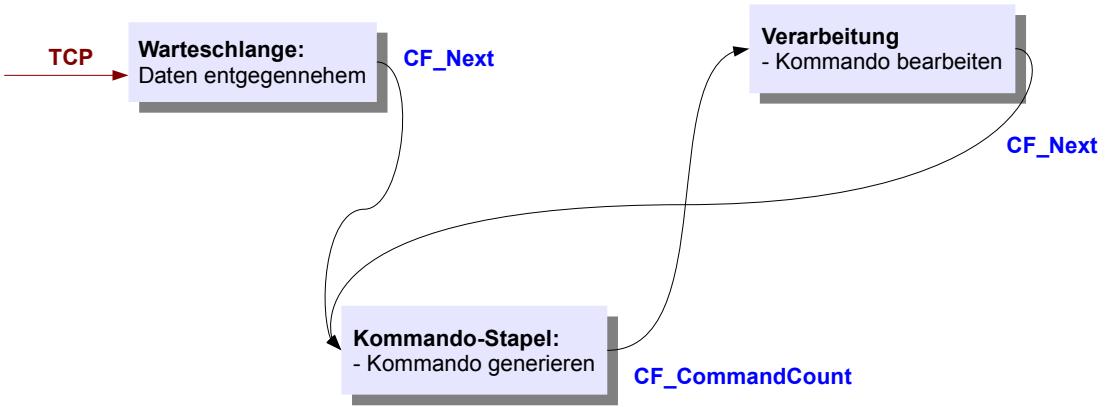

Abbildung 9: Die Synchronisierung der einzelnen Prozesse im Anwendungsprogramm

Etwas Theorie am Rande.

Diese Aufgabe ist mit Hilfe eines sogenannten Zustandsautomaten zu realisieren. Insgesamt ist das Vorgehen mit einem VHDL Entwurf zu vergleichen. Wir wollen an dieser Stelle nicht weiter auf diese Theorie eingehen.

Variablendefinitionen

```
// if you want to observe the actions please change this to ON
CF_Debug=OFF
// Do show if an unknown message occur
CF_Unknown=OFF
// Client
CF_IPC=0u32
CF_SenderPort=0u16
CF_Port=8019u16
CF_PASS=$EibPC$
CF_HeartBeat=$h=1$
CF_EndLimiter=$ $
// connection
CF_Connected=AUS
CF_Next=AUS
// Incomming Message
CF_Message=$$
// Queue for all Mesages
CF_MQueue=$$
// A single command
CF_Command=$$
CF_Arg=$$
CF_CommandCount=0
// Some String Data
CF_String=$$
CF_Send=$$
CF_Pos=0u16
CF_PosCount=0u16
CF_Init=OFF
// counts unknown commands
CF_error=0
// Heartbeat Message
CF_MHeartbeat=ON
CF_LastHeartbeat=OFF
// Portrait Message
CF_Portrait=ON
// EachMessage is limited by an Hex 03 Value (ASCII EOT)
IPSender=0u32
if systemstart() then {
        stringset(CF_EndLimiter,3,0u16);
        stringset(CF_HeartBeat,3,size(CF_HeartBeat));
} endif
```

Abbildung 10: Was wir alles brauchen

Bevor wir zeigen, wie das Ganze funktioniert, nehmen wir alle notwendig Variablendefinitionen in Abbildung 11 vorweg. Im weiteren wird darauf zurückgegriffen.

Debugging

Wie bei jeder Software-Entwicklung kommt man in die Verlegenheit, hin und wieder Variablen während des Entwicklungsprozesses zu debuggen. Enertex® EibStudio bietet hier den integrierten Debugger an. Allerdings ist im vorliegenden Fall die Verwendung des Debuggers schwierig, da die Protokollverarbeitung dynamisch so schnell vonstatten geht, dass man mit „Mausklicks“ keine Chance hat die Nachrichten und deren wunschgemäße Verarbeitung zu verfolgen.

Hier bedienen wir uns eines Tricks: Wir geben entsprechende Informationen mittels write auf eine nicht belegte Gruppenadresse, im vorliegenden Fall auf '1/2/3'c14, aus. Damit Enertex® EibStudio diese Informationen im Meldungsfenster anzeigen kann, bedarf es der Definition `Dummy='1/2/3'c14`, also einer Dummyvariablen, die weiter nicht genutzt wird. `CF_Debug` dient dann zur Steuerung der Ausgabe dieser Informationen. Die Gruppenadresse muss/sollte keinen Empfänger haben und dient dazu, dass im Meldungsfenster mitverfolgt werden kann, inwieweit überhaupt die Verbindung zustande gekommen ist und wie das System arbeitet.

Nachrichtenwarteschlange

Wir schauen uns nun das Ergebnis der Programmierung an, wie es im Makro definiert wurde (Abbildung 11):

```
//-------------------------------------------------------------
//
//    Message Handling in MQueue
//
//-------------------------------------------------------------
if event(readtcp(CF_Port,CF_IPC,CF_Message)) and (IPSender==0.0.0.0 or IPSender==CF_IPC)  then {
        if (size(CF_MQueue)+size(CF_Message))>=END then CF_error=1 endif;
        /* We insert an EndLimiter in Queue to make things easier */;
        if CF_Connected and size(CF_MQueue)!=0u16 then {
                CF_MQueue=CF_MQueue+CF_Message;
                if CF_Debug then {
                        write('1/2/3'c14,$OM:$c14+convert(CF_Message,$$c14))
                } endif
        } endif;
        if !CF_Connected or (CF_Connected and size(CF_MQueue)==0u16) then {
                CF_MQueue=CF_EndLimiter+CF_Message;
                /* Init Data if an incomplete Message was in Queue*/;
                CF_PosCount=0u16;
                CF_Pos=0u16;
                CF_CommandCount=0;
                if CF_Debug then {
                        write('1/2/3'c14,$NM:$c14+convert(CF_Message,$$c14))
                } endif
        } endif;
        /* Initiate Command Scheduler*/;
        if size(CF_Command)==0u16 then  CF_Next=!CF_Next endif
} endif
```

Abbildung 11: Der Aufbau der Warteschlange

Der EibPC arbeitet grundsätzlich als TCP/IP Server auf Port 4809.

Die erste Abfrage if event(readtcp wird aktiv, alsbald Command Fusion einen TCP Verbindung aufbaut und eine Nachricht schickt. Im Handbuch steht:

> *„Wenn ein beliebiges TCP/IP Telegramm an den Enertex® 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 unserem Fall kann Command Fusion demnach maximal 1400 Zeichen auf einmal übermitteln, was sicher ausreichend ist. Die Funktion *readtcp* füllt demnach den String `CF_Message`. Über die Abfrage mit event kann nun sichergestellt werden, dass man auf eine am Enertex® EibPC eintreffende Nachricht reagieren kann. *readtcp* schreibt auch die IP Adresse des Absenders in die Variable `CF_IPC`. Damit kann abgefragt werden, ob Anfragen seitens des Geräts mit der IP Adresse – im Code `IPSender==CF_IPC` – überhaupt angenommen werden. Wenn `IPSender` mit 0 vorgegeben wird, kann jeder Sender sich am Enertex® EibPC anmelden. Dies vereinfacht die Parametrierung, wenn man im eigenen LAN nur einen Command Fusion Teilnehmer hat.

Die Variable `CF_Connected` zeigt an, ob der Enertex® EibPC vom Command Fusion bereits das richtige Passwort bekommen hat und das Protokoll bereits aktiv ist. Diese Variable wird erst bei der Kommandoverarbeitung (s.h.) aktiv auf EIN gesetzt. Wenn dies der Fall ist, so wird nun die Warteschlange `CF_MQueue` weitergeführt durch das Anhängen der Nachricht `CF_Message`.

Grundsätzlich wäre dann die Warteschlange `CF_MQueue` wie in Abbildung 12 aufgebaut.

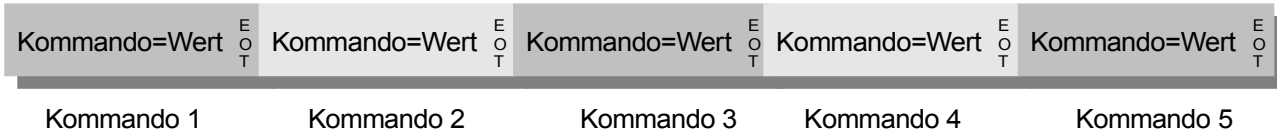

Abbildung 12: Nachrichten

Um uns die Sache später beim Auswerten der Warteschlange einfach zu machen, fügen wir, falls die Warteschlange leer ist, ein EOT (im Programm mit `CF_EndLimiter`) ein, sodass unsere Warteschlange tatsächlich wie in Abbildung 13 aufgebaut ist.

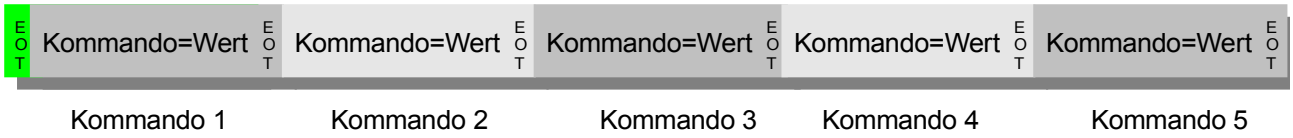

Abbildung 13: Nachrichten in `CF_MQueue`

Damit ist die Warteschlange aufgebaut und fertig. Der Programmteil läuft unabhängig von der eigentlichen Verarbeitung und ist rein ereignisorientiert. Will man ein ähnlich gelagertes Problem mit den UDP Telegrammen verarbeiten, so ist einfach an Stelle von readtcp die Funktion readudp zu verwenden.

Zu guter Letzt muss die Warteschlangenverarbeitung nun aber noch dem Kommando-Stapel mitteilen, dass er was zu tun hat. Hierzu toggeln wir die Variable `CF_Next`, falls nicht bereits ein Kommando verarbeitet wird, welches ggf. den String `CF_Command` mit einem Wert füllen würde.

Kommando-Stapel

```
//-------------------------------------------------------------
//
//    Next Telegramms in MQueue: Commands to schedule
//
//-------------------------------------------------------------
if after(change(CF_Next),1u64) then {
        CF_PosCount=CF_PosCount+1u16;
        CF_Pos=find(CF_MQueue,CF_EndLimiter,CF_PosCount-1u16)+1u16;
        CF_String=split(CF_MQueue,CF_Pos,find(CF_MQueue,CF_EndLimiter,CF_PosCount)-1u16);
        CF_Command=split(CF_String,0u16,find(CF_String,$=$,0u16)-1u16);
        CF_Arg=split(CF_String,find(CF_String,$=$,0u16)+1u16,END);
        if CF_Pos==EOS or CF_Pos==size(CF_MQueue) then {
                CF_MQueue=$$;
                CF_PosCount=0u16;
                CF_Pos=0u16;
                CF_Command=$$;
                CF_CommandCount=0;
                if CF_Debug then {
                        write('1/2/3'c14,$Ready$c14)
                } endif
        } endif;
        CF_CommandCount=CF_CommandCount+1;
        if CF_Debug then {
                if CF_Pos!=EOS and CF_Pos!=size(CF_MQueue) then{
                        write('1/2/3'c14,convert(CF_CommandCount,$$c14)+$N:$c14+convert(CF_Command+$=$+CF_Arg,$$c14))
                } endif
        } endif
} endif
```

Abbildung 14: Der Kommando Stapel

Fall die Variable `CF_Next` sich ändert, wird das nächste Kommando „geladen“. Damit ist auch dieser Programmteil rein ereignisorientiert implementiert und kann durch das Verändern dieser Variable getriggert werden.

Die Verwendung von `after` fällt auf: Die Verzögerung um einen 1ms-Zyklus bewirkt, dass die Verarbeitung des Kommando-Stapels und dessen Aufruf um einen Verarbeitungszyklus verschoben wird. So wird gewährleistet, dass die Antwort an das CF nicht zu schnell generiert wird und die Verarbeitung insgesamt ungestört abläuft.

Dieser Programmteil parst nun die Warteschlange, die wie in Abbildung 13 aufgebaut ist, von einem EOT zum nächsten und legt das zu verarbeitende Kommando in die Variable `CF_Command` und dessen Wertangabe in `CF_Arg` ab. Schließlich wird der Zähler `CF_CommandCount` erhöht. Dieser dient ausschließlich dazu, die nachfolgende Verarbeitung zu triggern. Man hätte wie bei `CF_Next` auch mit einer Variable vom Datentyp b01 arbeiten können (toggeln), allerdings bietet sich für Debugzwecke ein Zähler in der Ausgabe an. Der Überlauf, der auftreten kann, spielt keine Rolle, da die nachfolgende Verarbeitung lediglich auf Änderungen reagiert (man vergleich nochmals die Synchronisierung der einzelnen Aufgaben in Abbildung 9).

Kommando-Verarbeitung

Die Kommandoverarbeitung ist an sich leicht umzusetzen. Einzig zu beachten ist,

- dass erst wenn ein Passwort richtig übergeben wurde, die Warteschlange weiter bearbeitet werden soll. Dazu wurde die Variable CF_Connected eingeführt, die im weiteren Verlauf für die Verarbeitung der Joins immer abgefragt wird,

- dass das Heartbeat-Kommando (Lebenszeichen) immer alle 3 Sekunden übertragen wird – auch wenn der User „wie wild“ Slider hin- und herscrollt, so dass die Warteschlange schon mal 3 Sekunden bis zum völligen Abarbeiten benötigt,
- dass man die Verarbeitung stoppt, wenn das iPhone keine Lebenszeichen mehr schickt,
- und dass ein unbekanntes Kommando verworfen wird.

```
//-------------------------------------------------------------
//
//    Password Command
//
//-------------------------------------------------------------
if CF_Command==$p$ then {
        if CF_Arg==CF_PASS then {
                CF_Send=$p=ok $;
                stringset(CF_Send,3,size(CF_Send)-1u16);
                CF_Connected=EIN;
                CF_MHeartbeat==EIN;
                sendtcp(CF_Port,CF_IPC,CF_Send);
                if CF_Debug then write('1/2/3'c14,$Pass ok$c14) endif;
        } endif;
        if CF_Arg!=CF_PASS then {
                CF_Send=$p=bad $;
                stringset(CF_Send,3,size(CF_Send)-1u16);
                CF_Connected=AUS;
                CF_Init=AUS;
                if CF_Debug then write('1/2/3'c14,$NoP$c14+convert(CF_Arg,$$c14)) endif;
                sendtcp(CF_Port, CF_IPC,CF_Send)
        } endif;
        CF_Next=!CF_Next
} endif
```

Abbildung 15: Das Passwort

```
//-------------------------------------------------------------
//
//    Initializing Command
//
//-------------------------------------------------------------
if CF_Command==$i$ and CF_Arg==$1$ and CF_Connected then {
        if CF_Debug then write('1/2/3'c14,$Init$c14)  endif;
        CF_Init=EIN;
        CF_Next=!CF_Next
} endif
```

Abbildung 16: Initialisierung der Joins (siehe auch weiter unten)

```
//-------------------------------------------------------------
//
//    Heartbeat Command
//
//-------------------------------------------------------------
if CF_Command==$h$ and CF_Arg==$0$ and CF_Connected then {
        CF_MHeartbeat=EIN;
        sendtcp(CF_Port,CF_IPC,CF_HeartBeat);
        CF_LastHeartbeat=!CF_LastHeartbeat;
        if CF_Debug then write('1/2/3'c14,convert(CF_HeartBeat,$$c14))  endif;
        CF_Next=!CF_Next
} endif

if CF_Connected and delay(change(CF_LastHeartbeat),2800u64) then {
        sendtcp(CF_Port,CF_IPC,CF_HeartBeat);
        CF_LastHeartbeat=!CF_LastHeartbeat
} endif
```

Abbildung 17: Das Lebenszeichen

```
//-------------------------------------------------------------
//
//   Portrait Command
//
//-------------------------------------------------------------
if CF_Command==$m$ and CF_Connected then {
        if CF_Arg==$portrait$ then {
                CF_Portrait=EIN
        } endif;
        if CF_Arg==$landscape$ then {
                CF_Portrait=AUS
        } endif;
        if CF_Debug then write('1/2/3'c14,convert(CF_Command+$=$+CF_Arg,$$c14))  endif;
        CF_Next=!CF_Next
} endif
```

Abbildung 18: Das Portrait Kommando

```
//-------------------------------------------------------------
//
//   Disconnect
//
//-------------------------------------------------------------
// After 7 seconds without any Heartbeat kill connection
if cycle(0,7) then {
        if CF_MHeartbeat==AUS and CF_Connected then {
                CF_Connected=AUS;
                CF_Init=AUS;
                CF_MQueue=$$;
                CF_PosCount=0u16;
                CF_Pos=0u16;
                CF_Command=$$;
                if CF_Debug then write('1/2/3'c14,$Timeout$c14)  endif
        } endif;
        if CF_Connected then CF_MHeartbeat=AUS endif
} endif
```

Abbildung 19: Keine Antworten mehr vom iPhone?

Wie man erkennt, werden die Kommandos einfach mit Hilfe von If-Abfragen abgearbeitet. Was aber, falls ein Kommando gar nicht abgefragt (parametriert) wurde und einfach unbekannt ist? In diesem Fall würde die Verarbeitung stoppen, weil ja der Kommando-Stapel nicht durch ein Toggeln von `CF_Next` ein neues Kommando generiert.

Die Lösung für diese Aufgabe ist in Abbildung 20 angegeben: Man wartet einfach 100ms nachdem ein Kommando vom Stapel zur Verarbeitung gegeben wurde, was mit `change(CF_CommandCount)` zu erkennen ist.

```
//-------------------------------------------------------------
//
//    Unkown command: After 100ms each command must have been
//    processed otherewise it is cancled
//
//-------------------------------------------------------------
if delay(change(CF_CommandCount),100u64) and CF_Command!=$$ and CF_Connected then {
        CF_error=CF_error+1;
        if CF_Debug or CF_Unknown then write('1/2/3'c14,convert(CF_CommandCount,$$c14)+\\
                $U:$c14+convert(CF_Command+$=$+CF_Arg,$$c14)) endif;
        CF_Next=!CF_Next;
} endif
```

Abbildung 20: Unbekanntes Kommando verwerfen.

Join-Verarbeitung

Die Verarbeitung von Joins funktioniert auf gleiche Weise, wobei nun eben die Kommandos wie die Join Namen heißen. Um keine Probleme aufkommen zu lassen, werden die Daten, welche über die Wertangaben in `CF_Arg` übermittelt werden, einfach mit Hilfe von convert gewandelt.

```
//-------------------------------------------------------------
//
//    Join: Digital, Analog or Serial
//
//-------------------------------------------------------------
CF_join_s621=$Ein Tolles Programm$
if CF_Command==$s621$ and change(CF_CommandCount) and CF_Connected then {
        CF_join_s621=convert(CF_Arg,CF_join_s621);
        CF_Next=!CF_Next;
        if CF_Debug then write('1/2/3'c14,$Set:$c14+\\
                convert(CF_Command+$=$+CF_Arg,$$c14)) endif

} endif

// Answering
if change(CF_join_s621) and CF_Connected then {
        sendtcp(CF_Port,CF_IPC,$s621$+$=$+convert(CF_join_s621,CF_Send)+CF_EndLimiter);
        if CF_Debug then write('1/2/3'c14,$Send:$c14+\\
                convert($s621$+$=$+convert(CF_join_s621,CF_Send),$$c14)) endif;
} endif
// Initializing
if CF_Init and CF_Connected then {
       if convert(CF_join_s621,$$)!=$0$ and convert(CF_join_s621,$$)!=$0.0$ and convert(CF_join_s621,$$)!=$0.000000e+00$ then {
                sendtcp(CF_Port,CF_IPC,$s621$+$=$+convert(CF_join_s621,CF_Send)+CF_EndLimiter)
       } endif
} endif
```

Abbildung 21: Einen Join (CF_join_s621) verarbeiten

Eine Sonderstellung nimmt die Initialisierung der GUI des Command Fusion ein: Alsbald sich Command Fusion mit dem Enertex® EibPC verbindet, fragt es nach dem Initialisierungswert der verwendeten Joins. Dabei erwartet es aber nur Angaben von Werten ungleich null. Da wir der Einfachheit halber alles in einem Makro verarbeiten wollen – unabhängig vom verwendeten Datentyp – müssen wird mit Hilfe von convert alle Möglichkeiten abfragen.

Dabei gibt es noch die „ToggleJoins“ beim Betätigen von Press-Release Knöpfen. Hier wird beim Drücken der Wert 1 und beim Loslassen der Wert 0 gesendet. Um diese nutzen zu können wird hier immer nur der Wert 1 berücksichtigt und Variable einfach invertiert:

```
if CF_Command==$d603$ and change(CF_CommandCount) and CF_Connected then {
        if convert(CF_Arg,0b01) then {
                CF_join_d603=!CF_join_d603;
                if CF_Debug then write('1/2/3'c14,$Set:$c14+\\
                        convert(CF_Command+$=$,$$c14)+convert(CF_join_d603,$$c14)) endif
        } endif;
        CF_Next=!CF_Next;
} endif
```

Abbildung 22: Ein „ToggleJoin“ (CF_join_d603)

Makros

Wenn nun alles funktioniert, wird alles in Makros verpackt und wir können unserem Anwender die Sache recht einfach und bequem machen. Das Ergebnis kann in der Bibliothek „EnertexCommandFusion.lib“ gesichtet werden.

Fazit

Die Anbindung an das Command Fusion mit Hilfe von Makros ist für den Endanwender auf einfache Weise möglich, ohne dass dieser in die Tiefe der Programmierung einsteigen muss.

Die Verbindung von Joins und Variablen/Gruppenadressen ist mit einem einzigem Makro einfach herzustellen.

Mit Hilfe der gezeigten Aufgabenteilung in verschiedene Prozesse ist Framework gegeben, mit dem der Anwender nun weitere, ähnlich gelagerte Problemstellungen umsetzten kann.