LabVIEW Kurzlehrgang

Von ziemlich leicht bis ganz schön schwierig.
Speziell für Neulinge, aber auch Geübte..

LabVIEW praxisgerecht und richtig anwenden

IT-Methoden verstehen

Fehler methodisch vermeiden

Struktur logisch und funktional gliedern.

Gedacht für die Uni-interne LabView Schulung

von Christof Ermer 07.09.2023

Version 1.3 - Wird schrittweise erweitert.

Status: kleine Fehler möglich, Themenreihenfolge je nach Bedarf

NUR INTERN AG Schüller Physik NAS Server_
Beispiel VIs liegen auf
Share_AGS NAS Server dem Server der AG Schueller
Share_AGS → V:\LabViewKurs\Lab-View-Kurzlehrgang-Exampels
.. und je nach Anpassung, Erweiterung auf meiner priv. Homepage (Christof Ermer)
https://homepages.uni-regensburg.de/~erc24492/
in einem ZIP File. (Downloaden) unter
https://homepages.uni-regensburg.de/~erc24492/ZIPs/ZIPS.html
Darin
LabView:
Zentrifuge_PWM_Sema.zip
Mnemonic_LabVersion.zip
Partikel-LaserSensor.zip

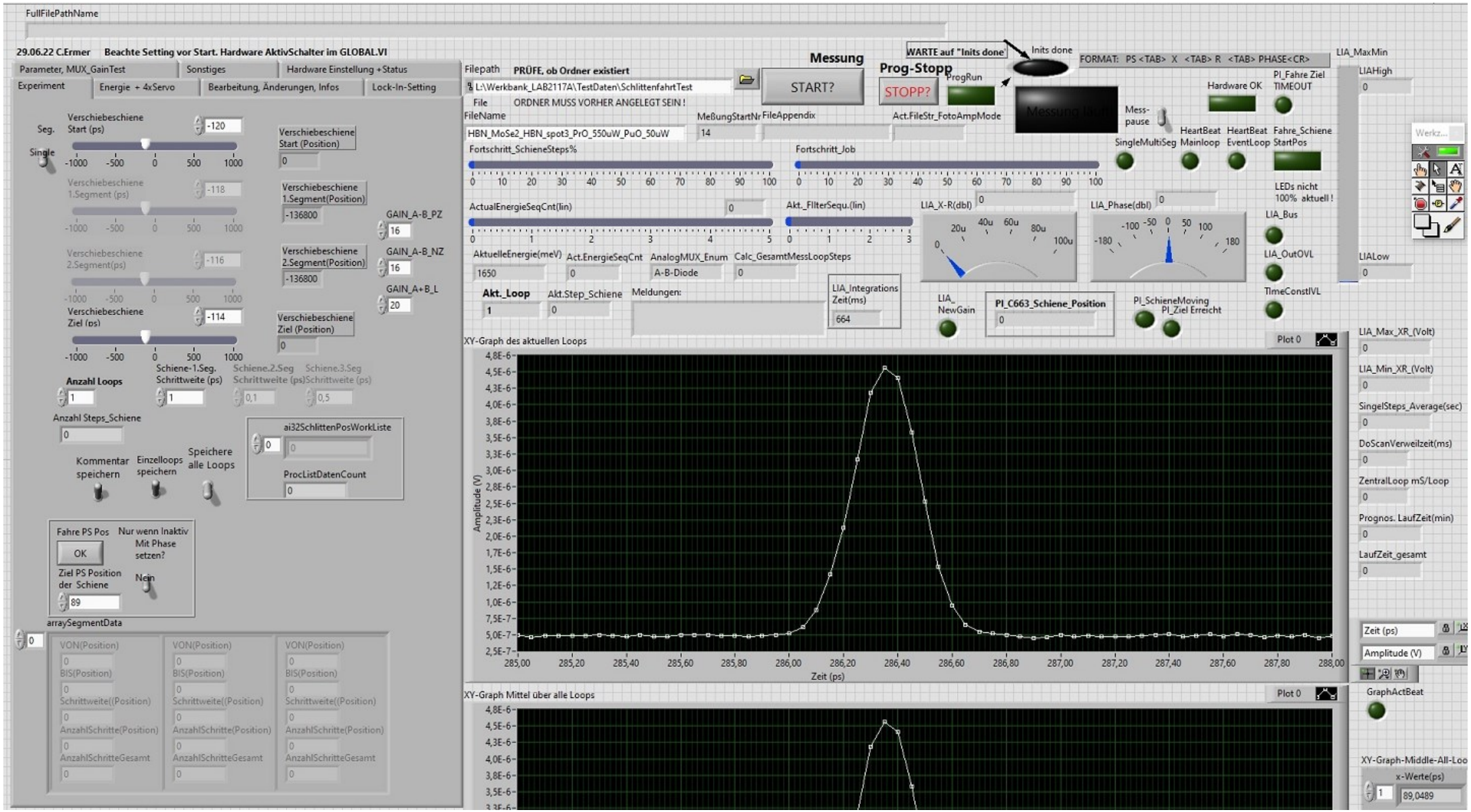
Von Anfang an strukturiert und richtig zu programmieren erspart einem schlechte, hartnäckige „Gewohnheiten“ wieder loszuwerden.

Eher früher als später zwingt die Realität über Strukturen nachzudenken.
Denn man wird von der ‚*gesparten Zeit*‘ durch die ‚*schnell mal gemacht*‘ Fehler sicher und zuverlässig überholt.

Das schöne ist, es wird überall mit Wasser gekocht. Nichts ist magisch.

oder…..
„kaum macht man es richtig,
schon geht es“.

So kann LabView aussehen. Eine typische Laborsoftware



LabView Programmierumgebung & IT – Strukturen

Der Sprachumfang, die Elemente einer **Programmierumgebun**g wie **LabView** ist enorm-
Ich sage bewusst nicht Programmiersprache. Diese sind eher textbasiert, LabView symbolbasiert.
Aber LabView erlaubt in einer grafischen Umgebung ebenso professionelle Programmstrukturen wie Textbasierte Programmiersprachen wie „**C**“.

Unabhängig von der Programmierung:
Ob ANSI-C, C++, C# Pascal, Python etc. sind Kenntnisse der Architektur von PCs und deren Komponenten erforderlich. Denn diese beeinflussen die Gestaltung der Software *entscheidend.*
*E*benso die Ablauforganisation bei der Bedienung der Hardware.
Daten-Schnittstellen wie COM-Ports etc.

Bsp. Sinnvolle Reihenfolge eine JEDEN Programmes!
START → mit Parameter Initialisierung. **Open Ports,** Start Definitionen, Ports, Flags Reset…
RUN → „**Main-Loop**“ Arbeitsschleife mit Plausibilitätschecks, Timeouts, kontrolliertem Abbruch etc..
STOP → **Close Ports**, Aufräumarbeiten, Formatierung der Werteübergabe -> Parameteroutput

Dazu später mehr. :

Worauf will ich hinaus:

LabView ist grafisch aufgenaut und dadurch anfänglöich deutlich leichter begreifbar.
Aber auch hier muss man IT typische Begriffe und **Strukturen** kennen.
If-Then, Bedingungs-While-Schleifen, From-To, Switch- Case, etc.
Auch die Frage, „Was sind Datentypen?“, ist relevant. Nicht anders als Zeilenorientierte.
Erkennbare Ablaufstrukturen: Starten, etwas tun, ordentlich beenden“ sollten vertraut sein.
Nicht anders wie in professionellen Compilersprachen Wie „C“ C++ C#.

Damit geht der Zeitgeist in Richtung „**Quick & Dirty**“, aber eben weit weniger steinig als **lange Lernzeitkurven** erfordernde professionelle Hochsprachen wie → **ANSI-C, C++, ADA, PL1, Assembler,** und auch Python
Phython ist beliebt, aber etwas unsauber im Umgang mit Schreibstil bzw. Struktur und Datentypen.

Die Lernkurve bis zu einem Sinnvollen Programm ist anfänglich angenehm kurz.
Dies verleitet jedoch weniger Tief über Programm Strukturen nachzudenken, als es bis vor wenigen Jahren für Textbasierte Programme erforderlich war.

Typische Beispiele: Basic, Python (simpel und tolerant) , aber leider auch IT-Technisch sehr schmutzig.
(….möchte jemand in einem Flugzeug sitzen, dessen Propeller von nicht sauber definierten Datentypen, mit definierten Größen, gesteuert werden).

1**. Fehler sind immer „Fehler**

2. Es gibt keinen Ersatz für geplante Ablauforganisation eins Programms

“- egal wie die Programmiersprache heißt und gestaltet ist!“!.

Zuerst das wichtigste: **LabView Virtual Instruments „VIs + Sub VIs“ RICHTIG sichern**.

(…heißt: den **jetzigen STATUS bewahren**)

1. LabView hat die ‚unangenehme Eigenschaft, geöffnete“ **VIs**“ in die gegenwärtige Version zu verändern.
Und schon hat man beim Beenden, Verlassen eine ungewollte Version gespeichert, die ALTE Version überschrieben.

2. VIs, **Sub VIs**, mit „**gleichen Namen**“, die noch im Speicher stehen, werden trotz „anderer“ Dateipfade aufgerufen, verändert, eingefügt. So gehen die Bezüge, Ladequellen, anstatt des aktuellen Ordners, schnell verloren, ersetz.
Das erzeugt oft unbemerkt Chaos inclusive falscher Links-Bezüge auf Sub VIs erzeugen.

Test: Verschiebe alle Ordner mit früheren Versionen auf einen sicheren Platz**.** Starte neu.

So offenbaren sich falsche Sub VI Bezüge

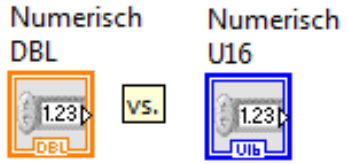
Lösung:

Meine Methode, gültige, lauffähige VIs- Strukturen zu sichern, ist ab einem gültigen Moment, ganzen Ordner mit Sub-Vis ZIP Files erzeugen. Und zwar mit Datum und Versionsnummer.
Diese *.zip dann evtl zusätzlich sogar auf externen Medium oder Server als SAVE_xxxx{Datum_Version}.zip ablegen.
Notfalls sogar mit Uhrzeit. um einen Entwicklungsstand zu sichern. → ***.so kann man sicher immer zurück***

3. Immer **ALLE VIs + LabView bewusst komplett beenden**. (auch LabView selbst), bevor man eine „Next Generation“ Kopie des Entwicklungsordners (mit allen Vis„) neu anlegt. **Jedoch Vorsicht beim Speichern** !
Nur so wird der Speicher zuverlässig geleert, neu zugeordnet und keine falschen Bezüge durch unvorsichtiges „Alles Speichern“ erzeugt. → Das aus Erfahrung, Lehrgeld und verlorener Zeit für Rekonstruktion….

Datentypen kennen und verstehen. Auch in LabView

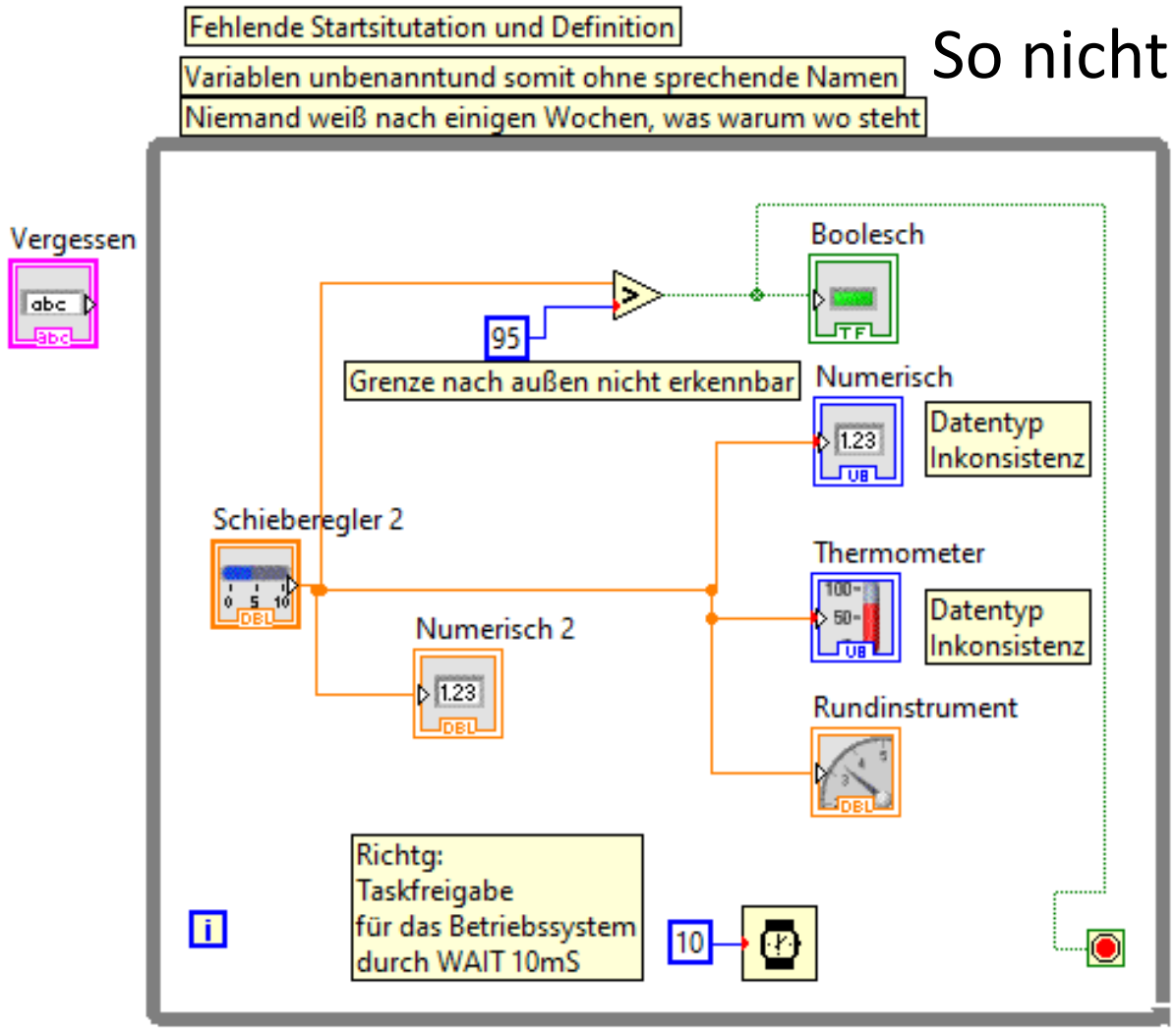
Beim Programmieren ist das Verstehen und die richtige Verwendung von **Datentypen** elementar. LV verführt etwas zum schlampigen Umgang, ……mit Folgen.
In der Praxis kommt es anfänglich vor, dass aus Unkenntnis von Datentypen ein Datentyp, der eigentlich für Kommazahlen dient, (**float** oder **DBL** genannt) z.B. als Zähler verwendet wird.

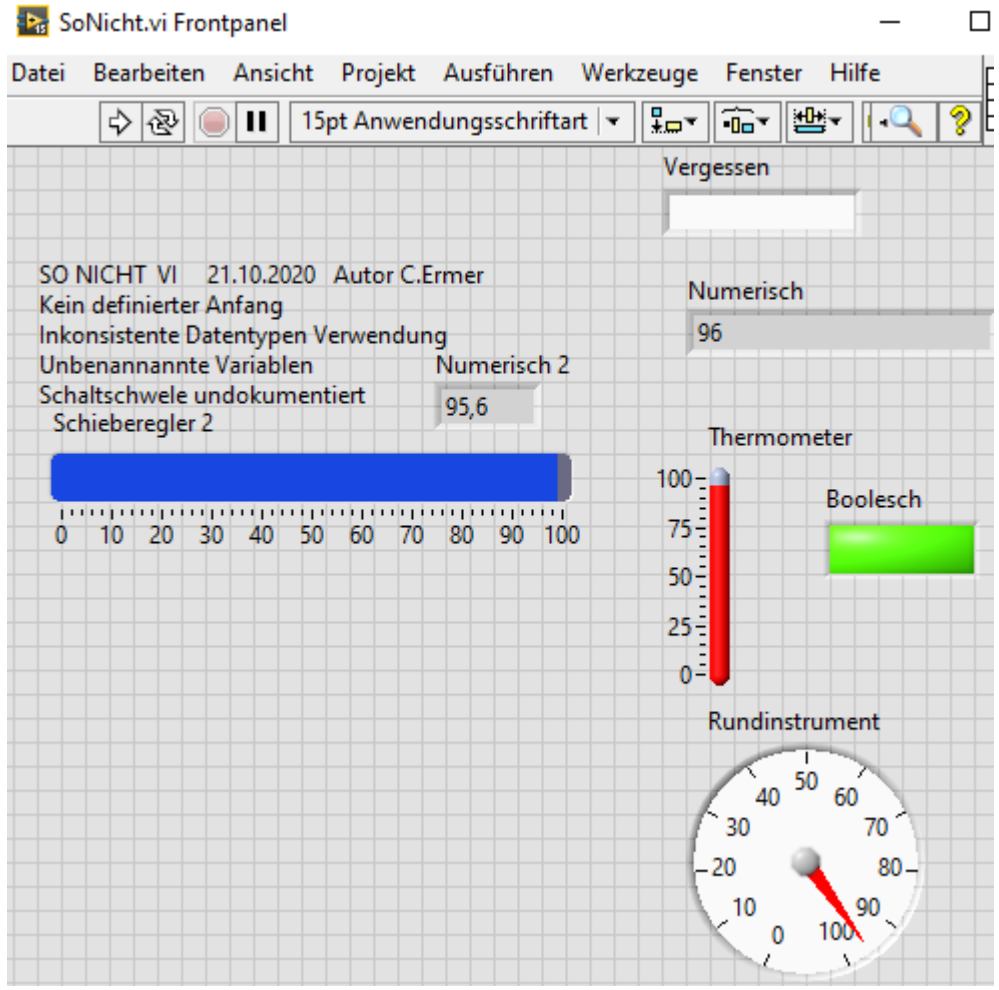


DBL ist zwar der **Standard Datentyp** für Zahlen**.**
Weil es damit immer ~irgendwie geht.
Dass ist aber nicht unbedingt guter Stil.
Auch wenn es „dennoch geht“ und es bei der heutigen PC Power egal ist wieviel Speicher unsinnig verbraucht wird.
Das wollen wir besser = „Vorzeigbarer“ machen. (→ Ziel ist Professionalität)
Deshalb sollten Datentypen vom Begriff und innerem Aufbau bekannt sein.

So sieht LV aus...

Nicht alles, was zwar funktioniert, ist praxistauglich, oder würdig, Programm genannt zu werden. „Quick & Dirty“ ist nur der erste Schritt





Versuchen wir es besser zu machen

Jetzt etwas ganz wichtiges:

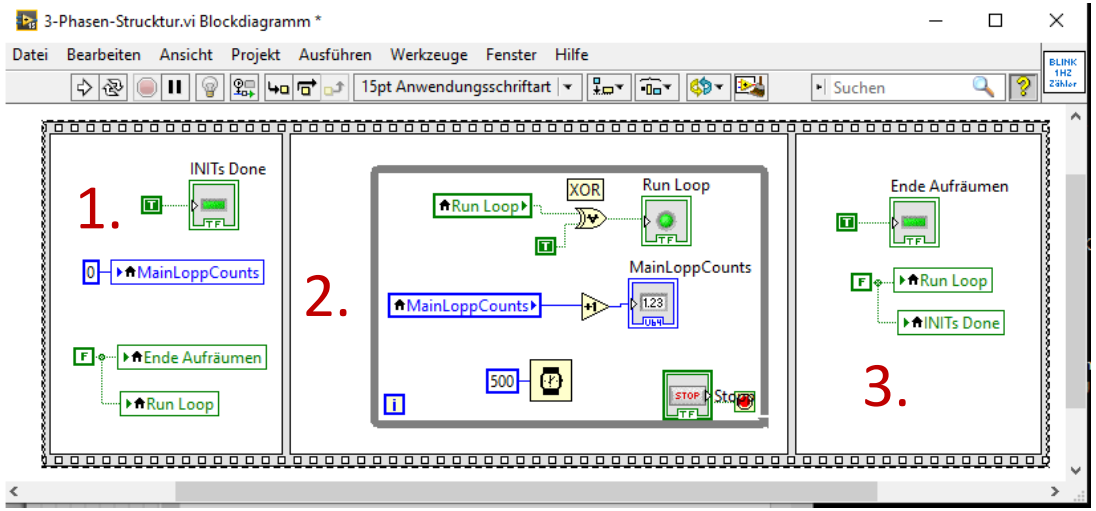
Beim Programmieren ist man verleitet „schnell“ etwas zu tun, und denkt nicht daran, hinter sich aufzuräumen.

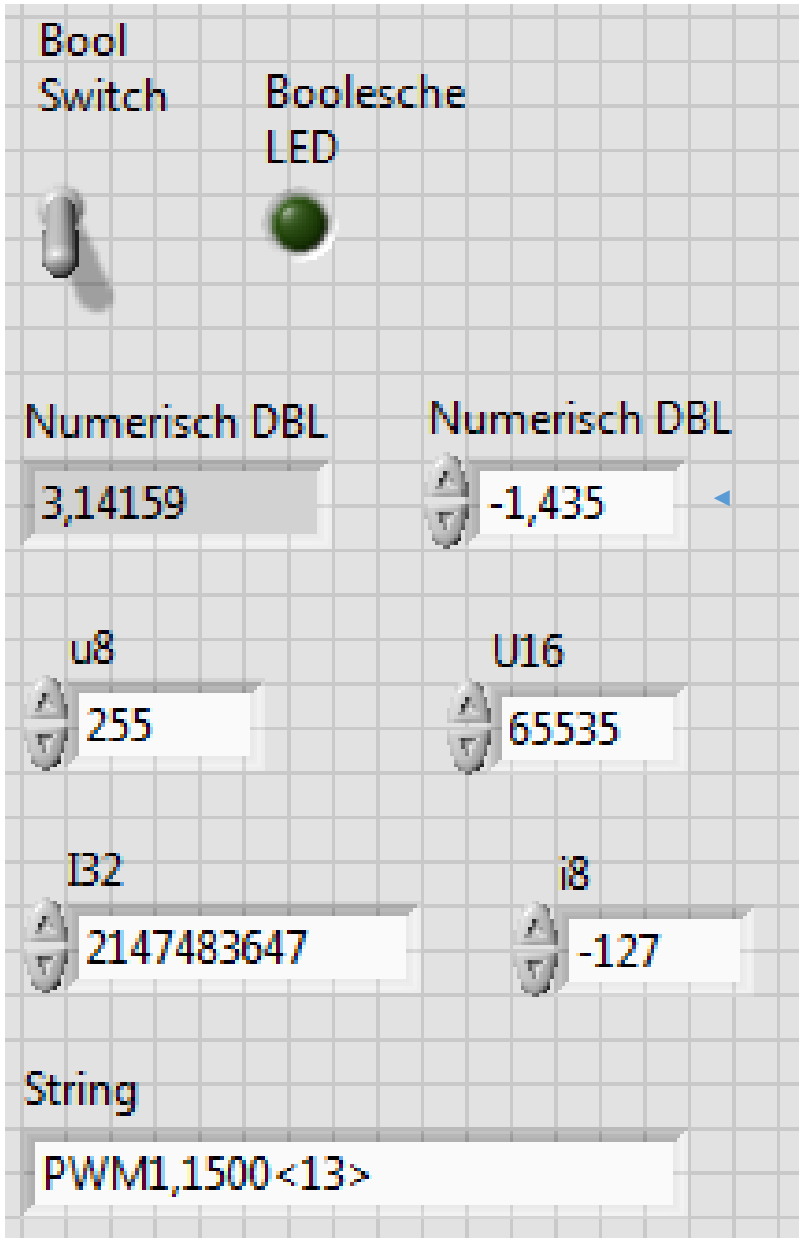
Heißt: Geöffnete Hardware kontrolliert wieder zu schließen.
Wie zum Bsp. Serielle Schnittstellen. Oder geöffnete Dateien etc.
Wir brauchen also eine einfache , dreigliedrige innere Struktur eines Programmes:

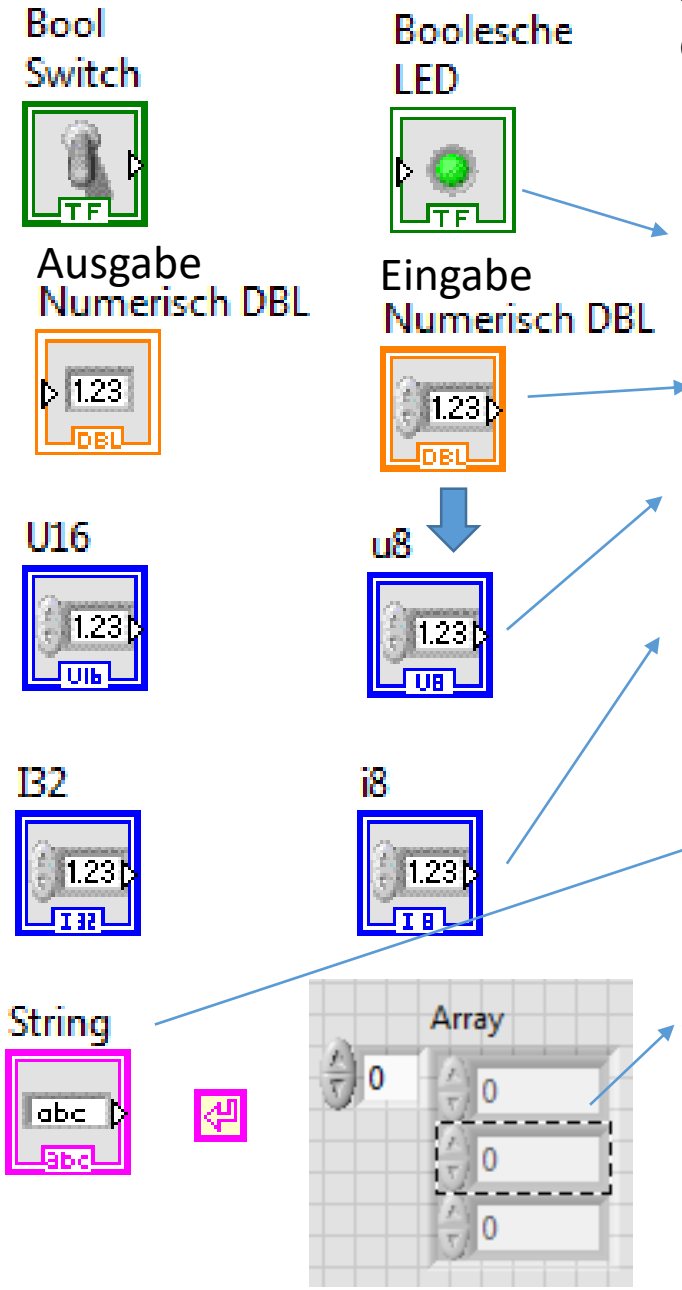
3 Phasen Standard Struktur eines Programmes.

Egal in welcher Programmiersprache.

1. *STARTEN-> Ports etc. Öffnen.*
2. *ABREITEN.*
3. *Ports etc. schließen. → STOP Programm.*







Sehen wir uns einmal ein paar elementare

Datentypen an….

Boolsche Variable (True/False)

Gleitkomma Zahlen (**Double**)

Vorzeichenlose Zahlen(**Unsigned** integer)

+/- Zahlen (**Integer** = signed)

Zeichenketten (**String**)

zudem gibt es:
Array (gleicher Datentyp)
Cluster sind Array aus zusammengesetzt aus verschiedenen Datentypen

Wertebereich von Datentypen

Diese ist durch die intern verwendete Bit Anzahl definiert.

Es ist ganz einfach. 2^N ergibt die Kombinationsmöglichkeit von 0en und 1sen.

2^8 Bit = 256 → **0..255** unsigned

2^{16} Bit = 65536 → **0..65535** unsigned usw.....

Bei **signed** wird <u>das *letzte Bit*</u> für das Vorzeichen verwendet.

Das halbiert den Zahlenbereich. Merke: Jedoch bei Minus anders als bei Plus.

Wertebereiche

signed integer unsigned integer

Wortbreite	Wertebereiche mit Vorzeichen	Wertebereich ohne Vorzeichen
8 Bit	-128 ... 127	0 ... 255
16 Bit	-32.768 ... 32.767	0 ... 65.535
32 Bit	-2^{31} ... $2^{31}-1$	0 ... 4.294.967.295
64 Bit	-2^{63} ... $2^{63}-1$	0 ... 18.446.744.073.709.551.615

Merke: Gezählt wird von 0 bis 9 , nicht von 1 bis 10.

Bsp. $2^8 = 256$ → Wertebereich ist unsigned **0..255**

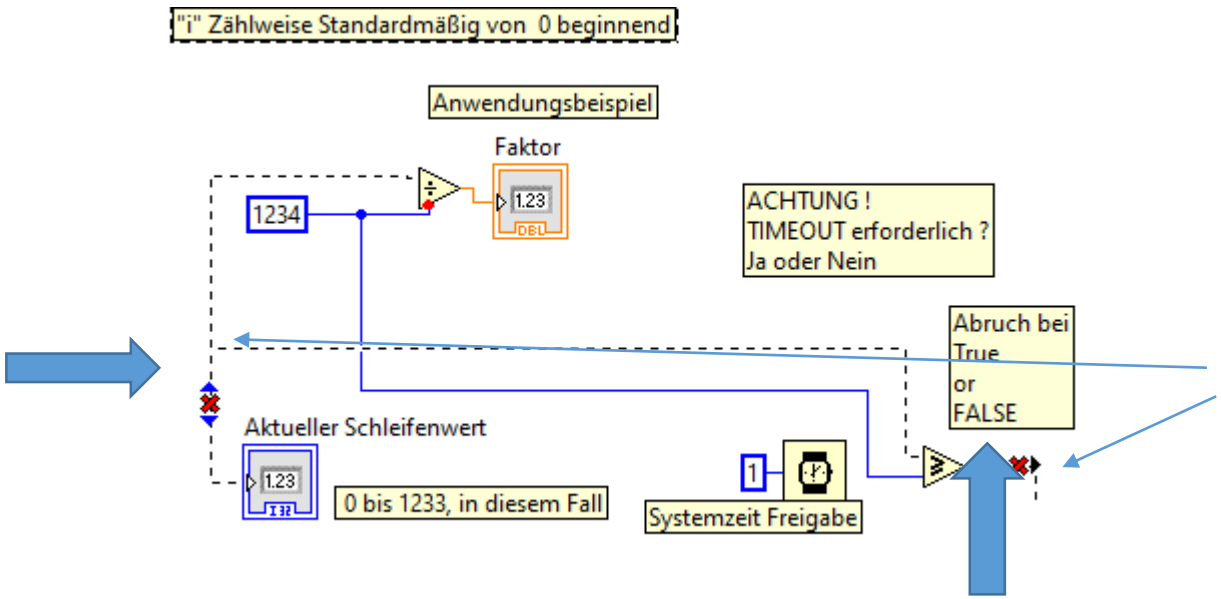
Datentypen hier z.B. Ansi-C, Mikrocontroller typisch

Die verwendete Bitbreite ist jedoch Maschinen bzw. Prozessor abhängig

Hier folgt eine Liste aller elementarer numerischer Variablentypen die in C zur Verfügung stehen:

Type	Keyword	Bytes	Range
character	char	1	-128 .. 127
unsigned character	unsigned char	1	0 .. 255
integer	int	2	-32 768 .. 32 767
short integer	short	2	-32 768 .. 32 767
long integer	long	4	-2 147 483 648 .. 2 147 483 647
unsigned integer	unsigned int	2	0 .. 65 535
unsigned short integer	unsigned short	2	0 .. 65 535
unsigned long integer	unsigned long	4	0 .. 4 294 967 295
single-precision floating-point (7 Stellen)	float	4	1.17E-38 .. 3.4E38
double-precision floating-point (19 Stellen)	double	8	2.2E-308 .. 1.8E308

Tastenkürzel, Wichtige Menüpunkte



Menü ANSICHT →

Werkzeugpalette

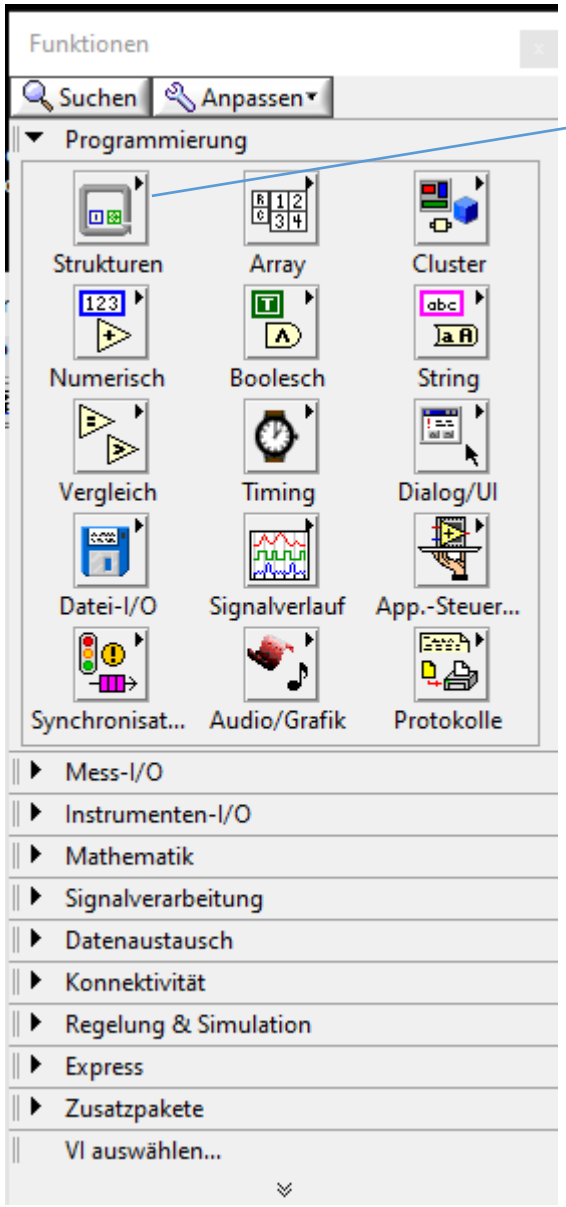
Elementpalette im Frontpanel

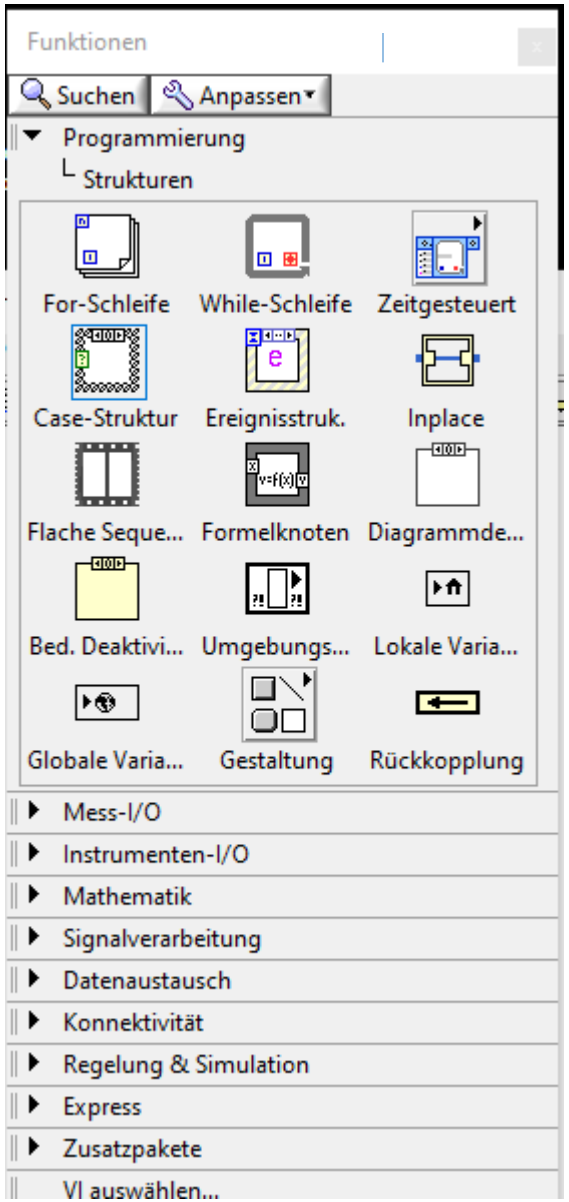
Funktionspalette im Blockdiagramm

Tastenkürzel:

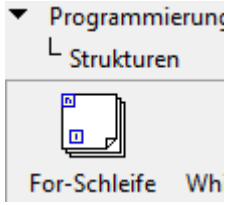
Kaputte Verbindungen löschen	**CTRl + B**
Umschaltung Front-Panel Block-Diagramm	**CTRL + E**

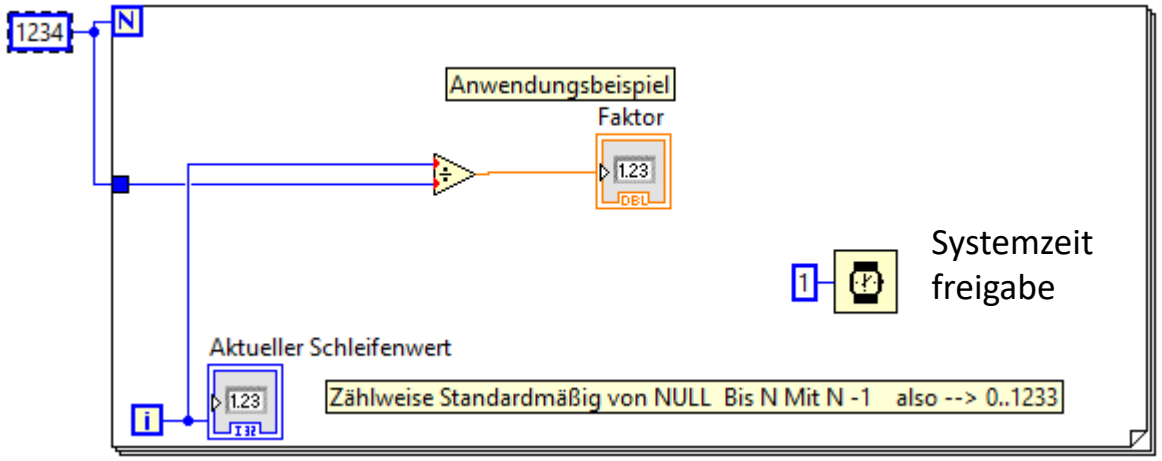
Elemente und Programm Struktur Elemente nutzen →Funktionspalette → Programmierung

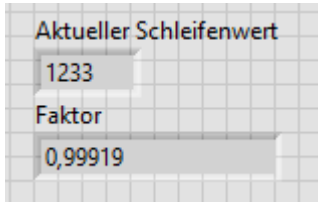




Strukturelement FOR Schleife







Der FOR Schleifen Inhalt wird wiederholt durchlaufen bis der Schleifenzähler „i“ die Zahl „N-1“ erreicht hat.

TRUE → i < N

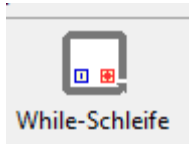
i = I + 1

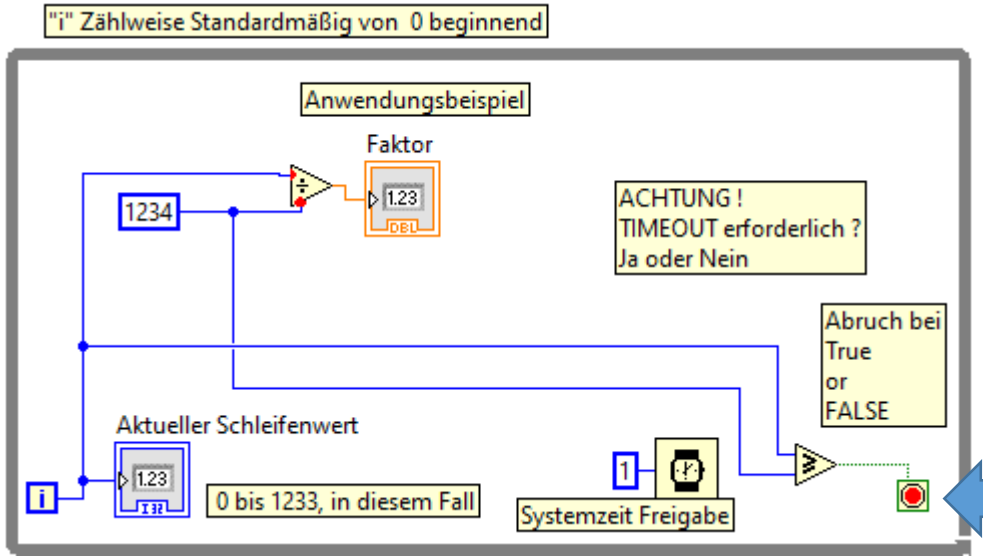
Startwert von i = 0 → i = 0..1233

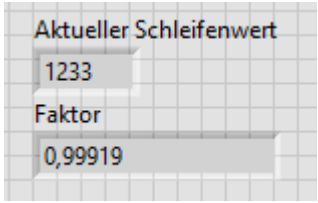
Bsp: N = 1234 Auch negatives N ist erlaubt

Die Schleife wird also 1234 mal durchlaufen.

Strukturelement WHILE Schleife







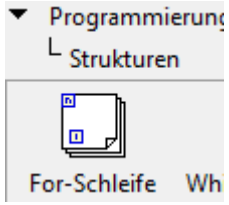
True or False

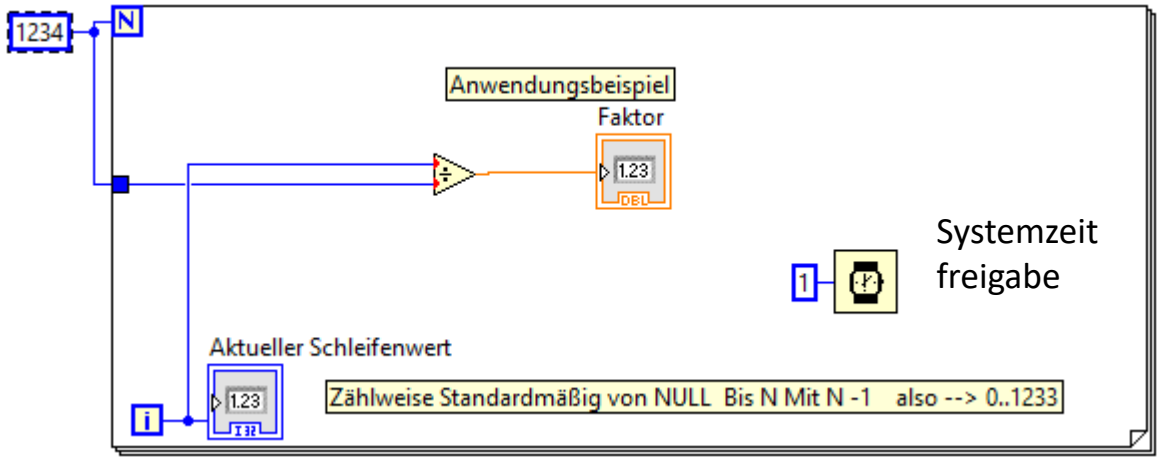
Der **While Schleife**n Inhalt wird wiederholt durchlaufen, bis die Bedingung TRUE oder FALSE erreicht ist

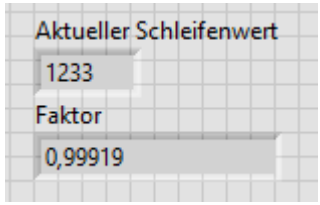
Startwert von i = 0
Bsp N = 1234 Die Schleife wird also 1234 mal durchlaufen.
i = 0..1233

Bedenke, dass das Programm hier solange verweilt, bis die Abbruchbedingung erreicht ist.
→ Timeout? verwenden?

Strukturelement CASE







Der FOR Schleifen Inhalt wird wiederholt durchlaufen bis der Schleifenzähler „i“ die Zahl „N-1“ erreicht hat.

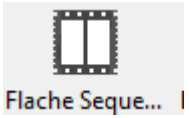
TRUE → i < N

i = I + 1

Startwert von **i = 0 → i = 0..1233 nicht 1234**

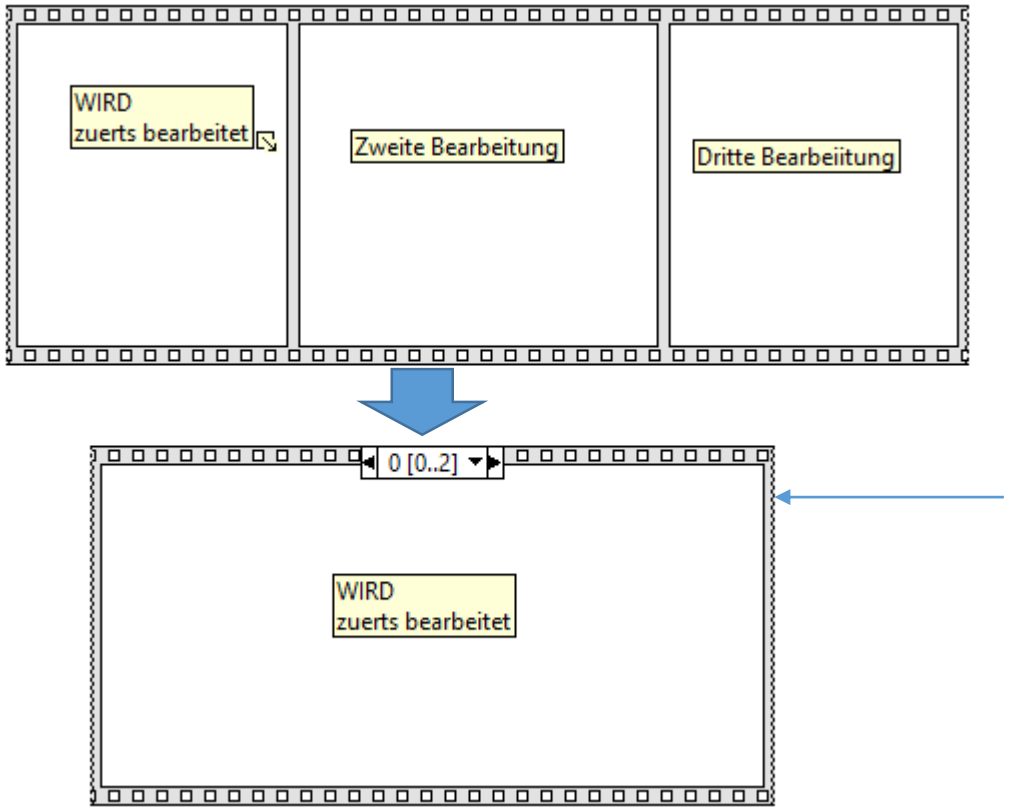
Bsp: N = 1234 Auch negatives N ist erlaubt

Die Schleife wird also 1234 mal durchlaufen.

Strukturelement 

Flache Sequenz = zwingende Reihenfolge der Bearbeitung organisieren

Metapher: Wie die Bilderreihenfolge auf einem Film



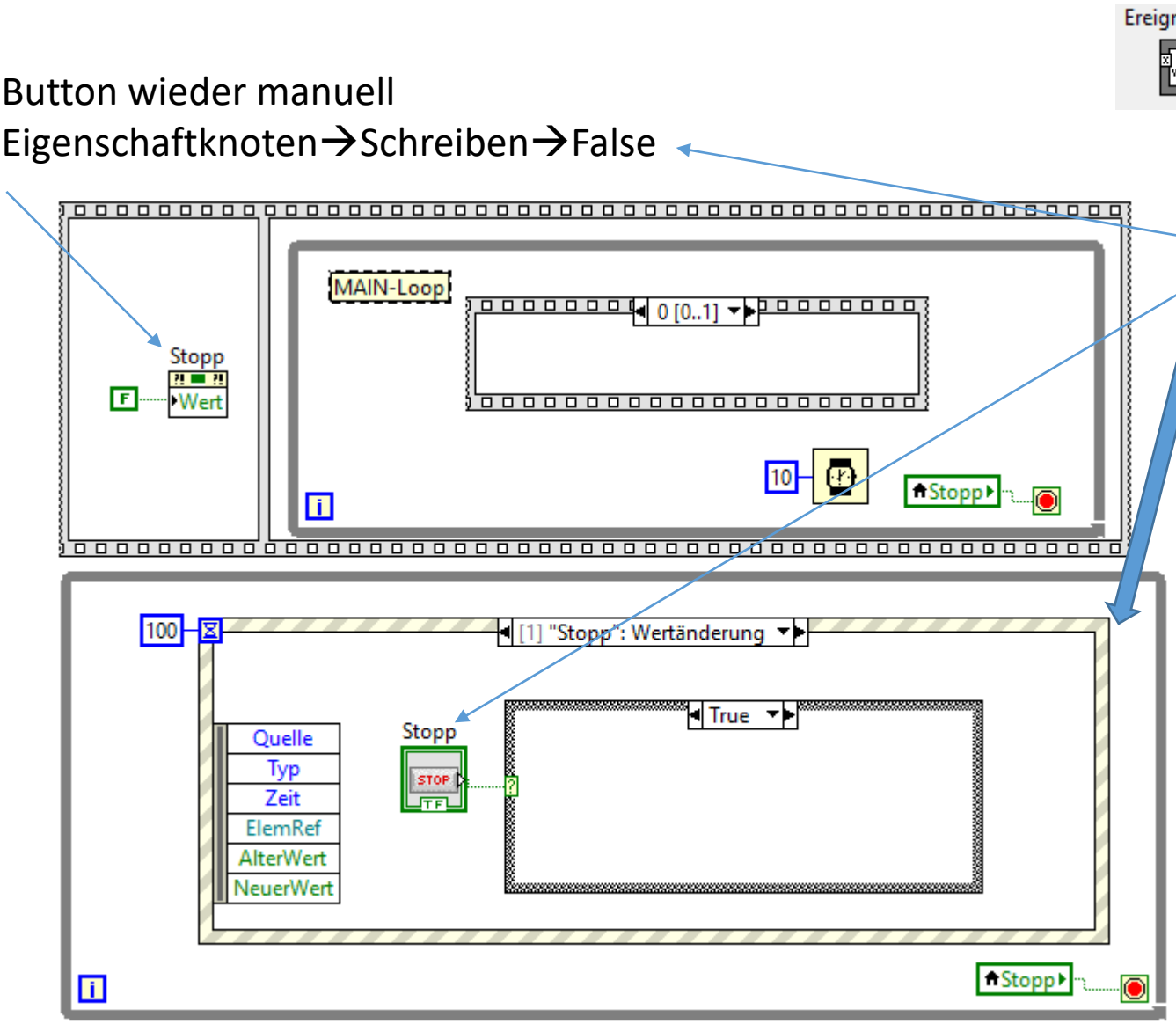
Ablauforganisation
Erzwingt eine bestimmte Reihenfolge der Bearbeitung .

1. Sequenz davor/danach einfügen
2. Beliebig viele möglich.
3. Jedoch ab einer gewissen Größe (>Bildschirmrand) ist durch „Gestapelte Sequenz ersetzen „Stacked sequenz“ empfohlen.

Stacked Sequenz: wird mit rechter Maustaste auf den Zähler aktiviert

Strukturelement EVENT

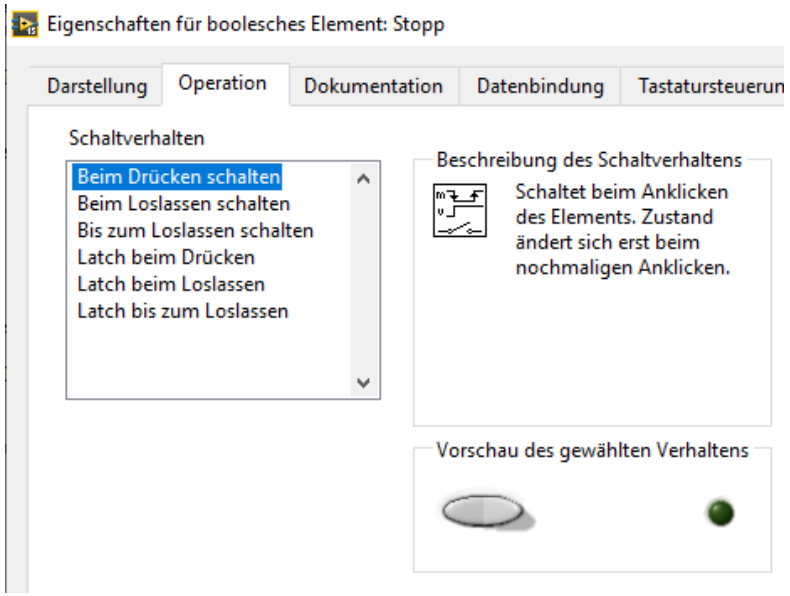
Button wieder manuell
Eigenschaftknoten→Schreiben→False

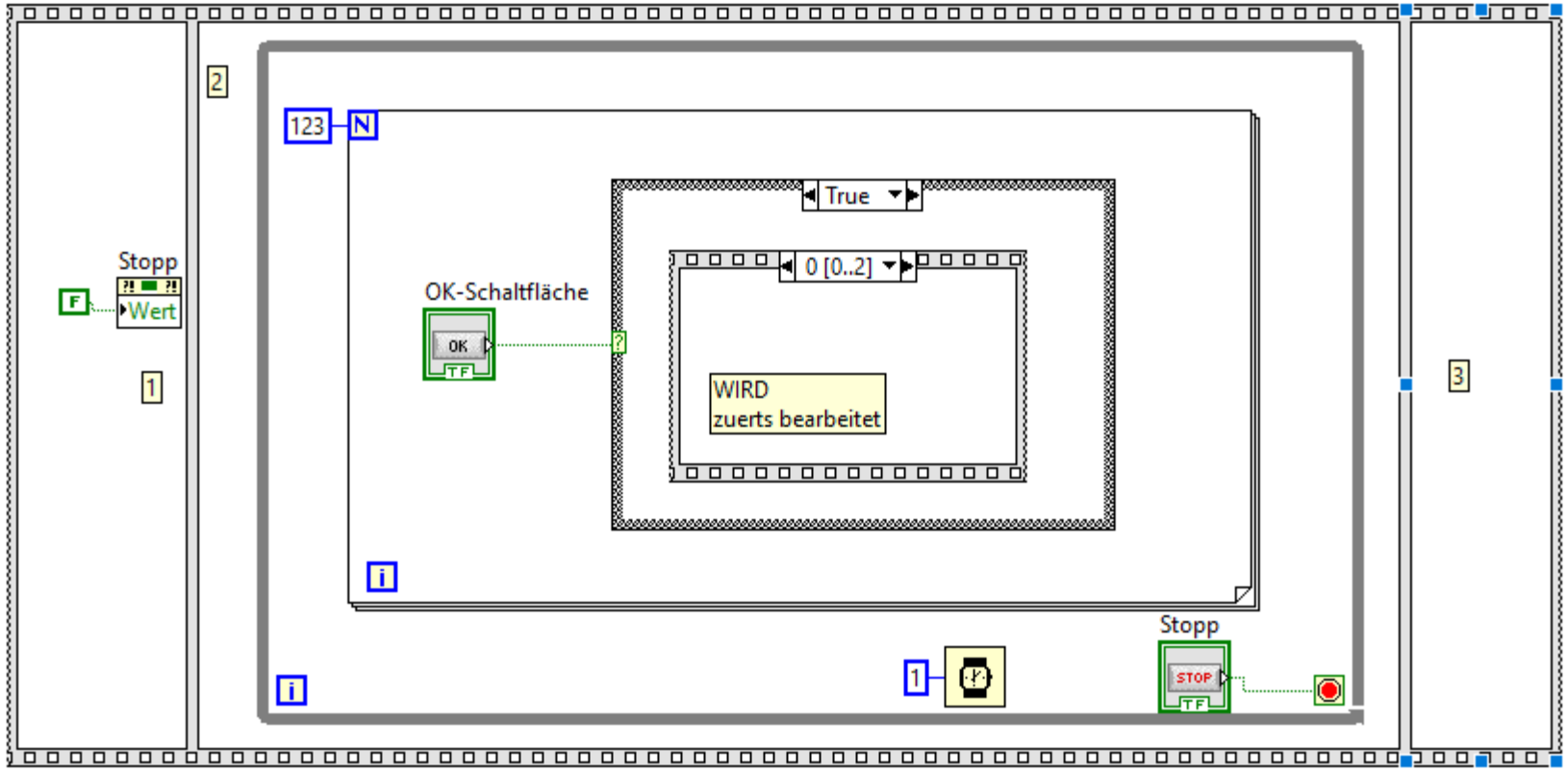


Ereignisstruk.

Ereignis Strukturen ermöglichen Quasi Parallele Verarbeitung und Vermeidung toter Bedienelemente.

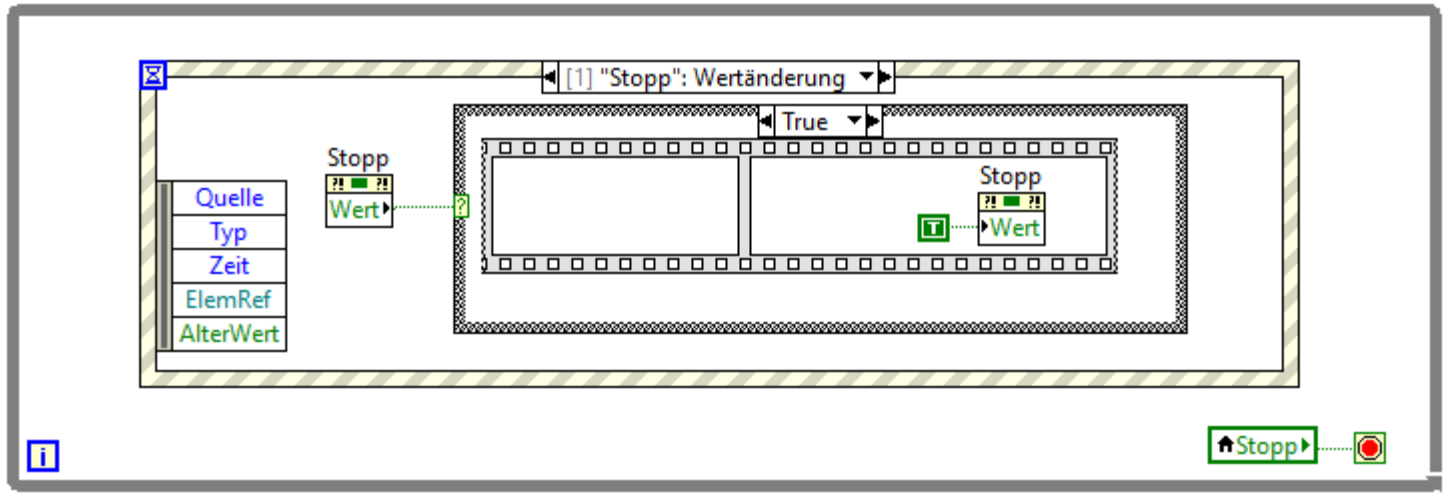
ACHTUNG: Buttons dürfen keine „Latsch“ Eigenschafft haben
Rechte Maustaste, Eigenschaften,



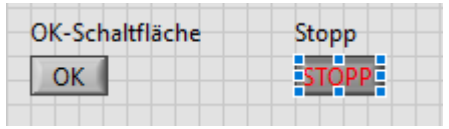


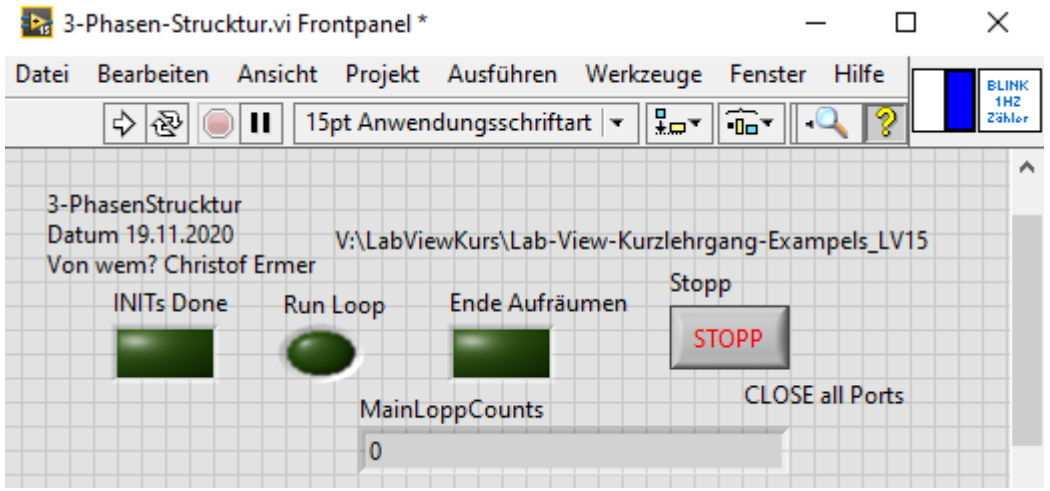
Kombination der wichtigsten Strukturelement zu einem sinnvollen Grund LabView Programm

Main Loop
Sequenz



EVENT Loop



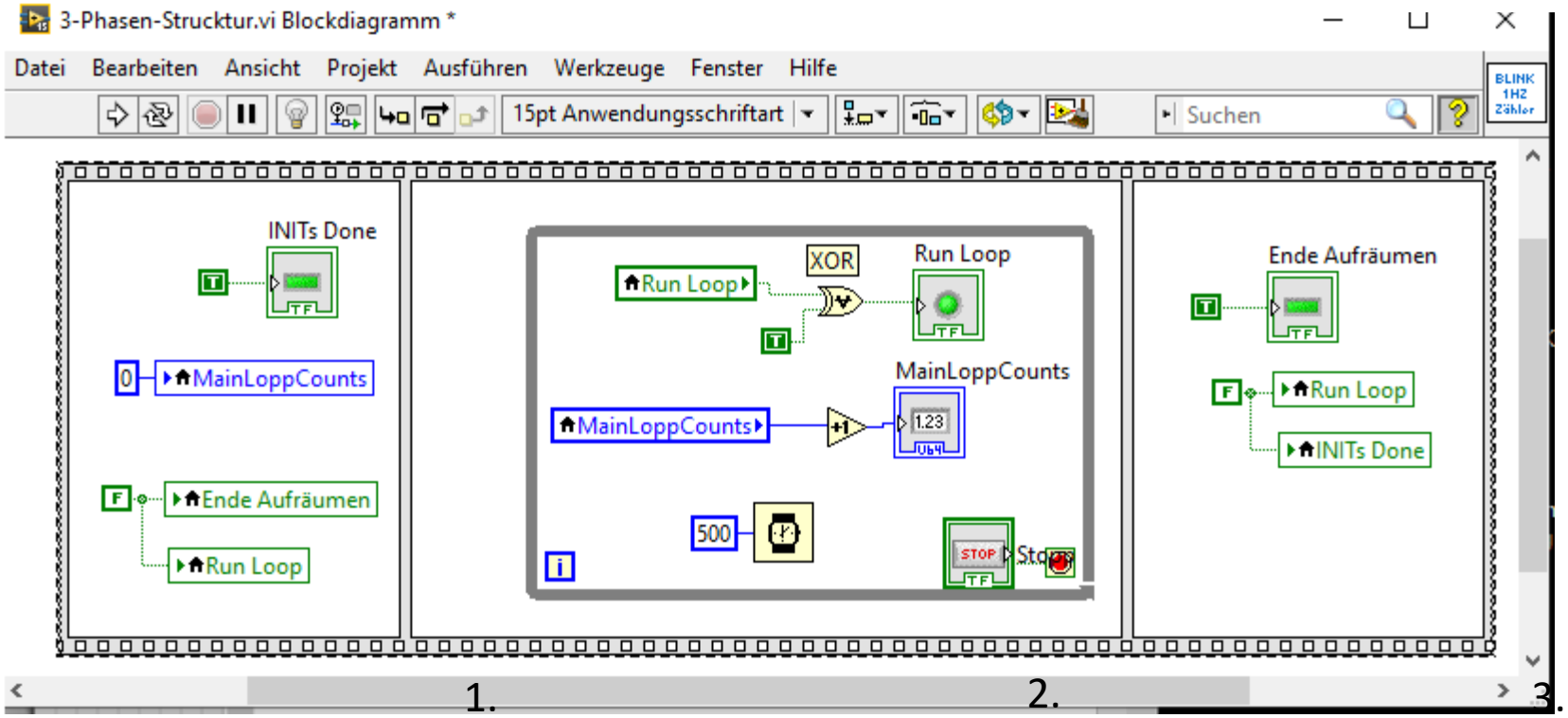


Jetzt etwas ganz wichtiges:

3 Phasen Standard Struktur eines Programmes.

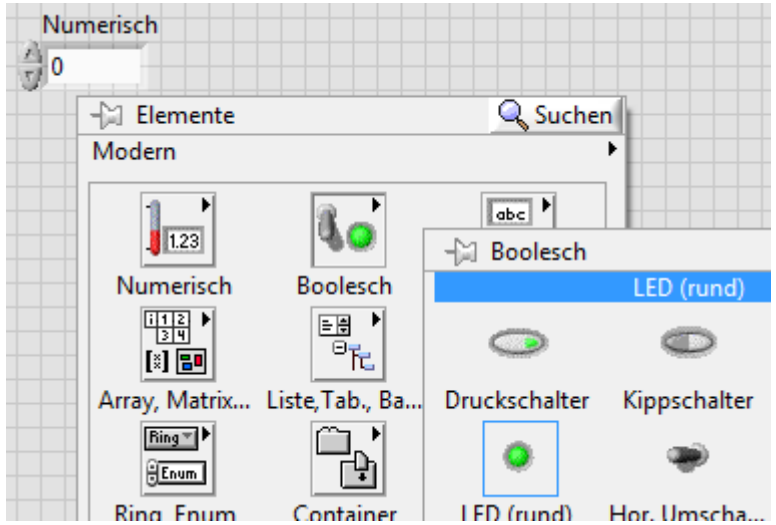
Egal in welcher Programmiersprache.

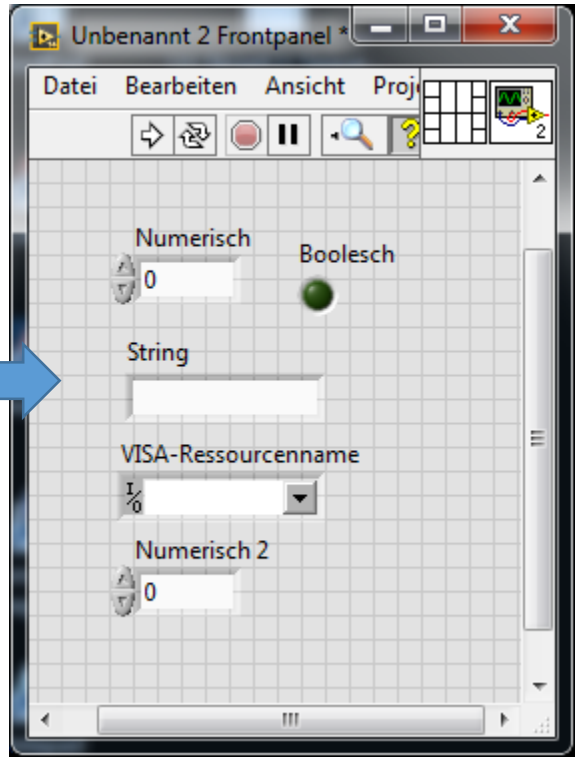
1. *STARTEN → Ports etc. Öffnen.*
2. *ABREITEN.*
3. *Ports etc. Schließen. → STOP Programm.*

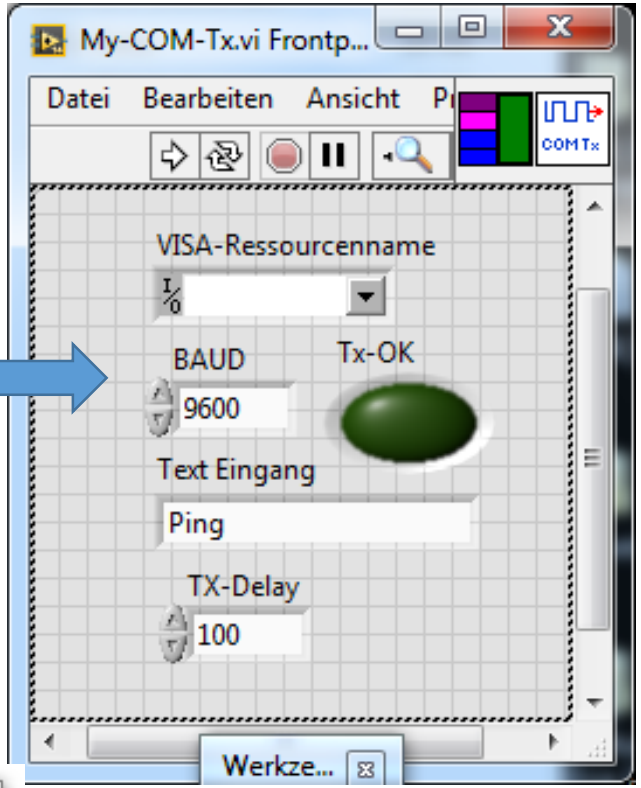


Im Einzelnen:

1. Schritt… Sammlung ‚voraussichtlich' benötigter Frontelemente





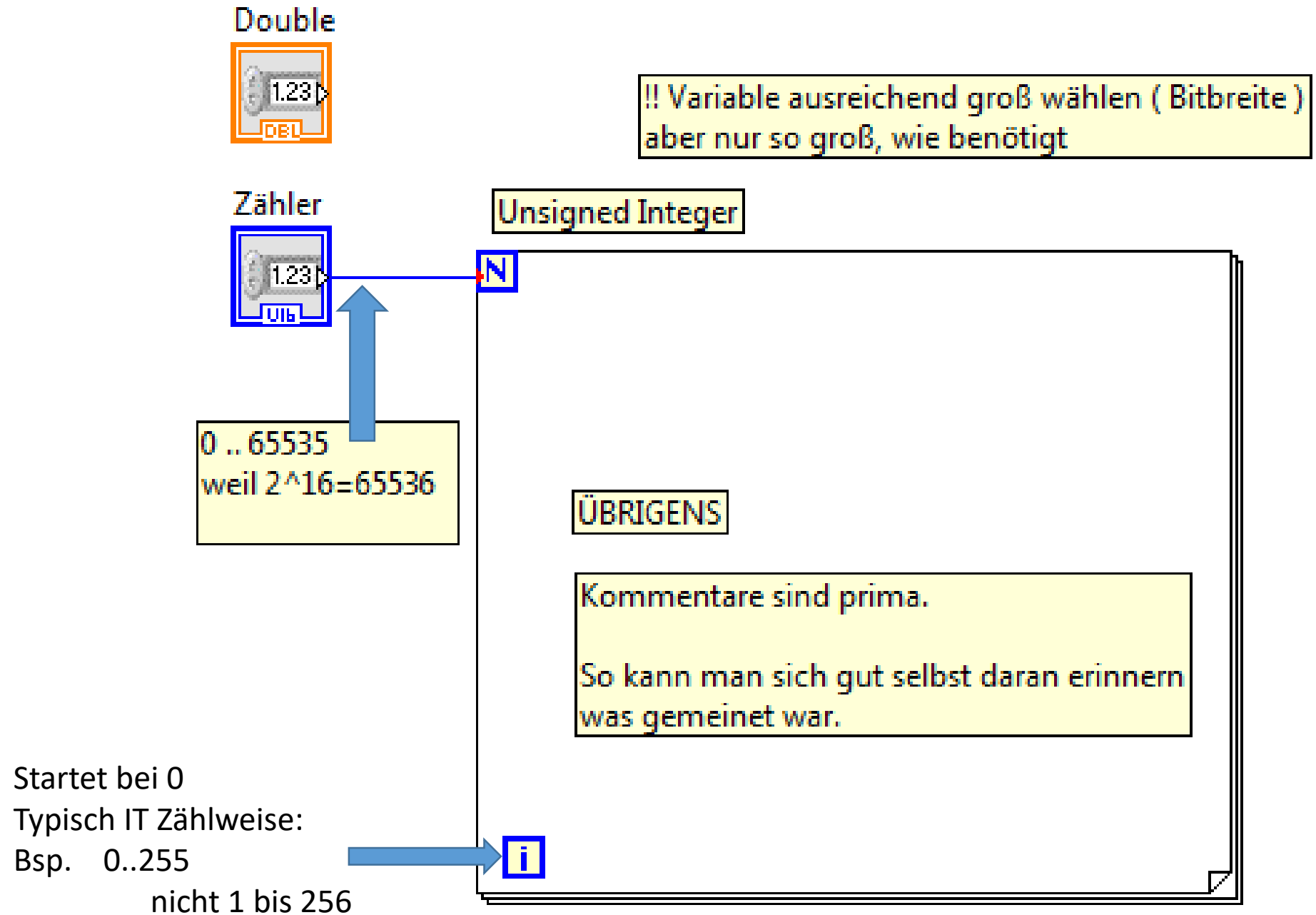


Einfügen
Mit „Right-Click" auf ein neues Frontpanel.

Elemente → Auswählen irgendwo ablegen.

Mit Pfeilsymbol auswählen, anordne Größe einstellen.

Mit „**A**" Symbol die Namen **sprechend** ändern, Möglichst praktische „Default Werte" eintragen..



Startet bei 0
Typisch IT Zählweise:
Bsp. 0..255
nicht 1 bis 256

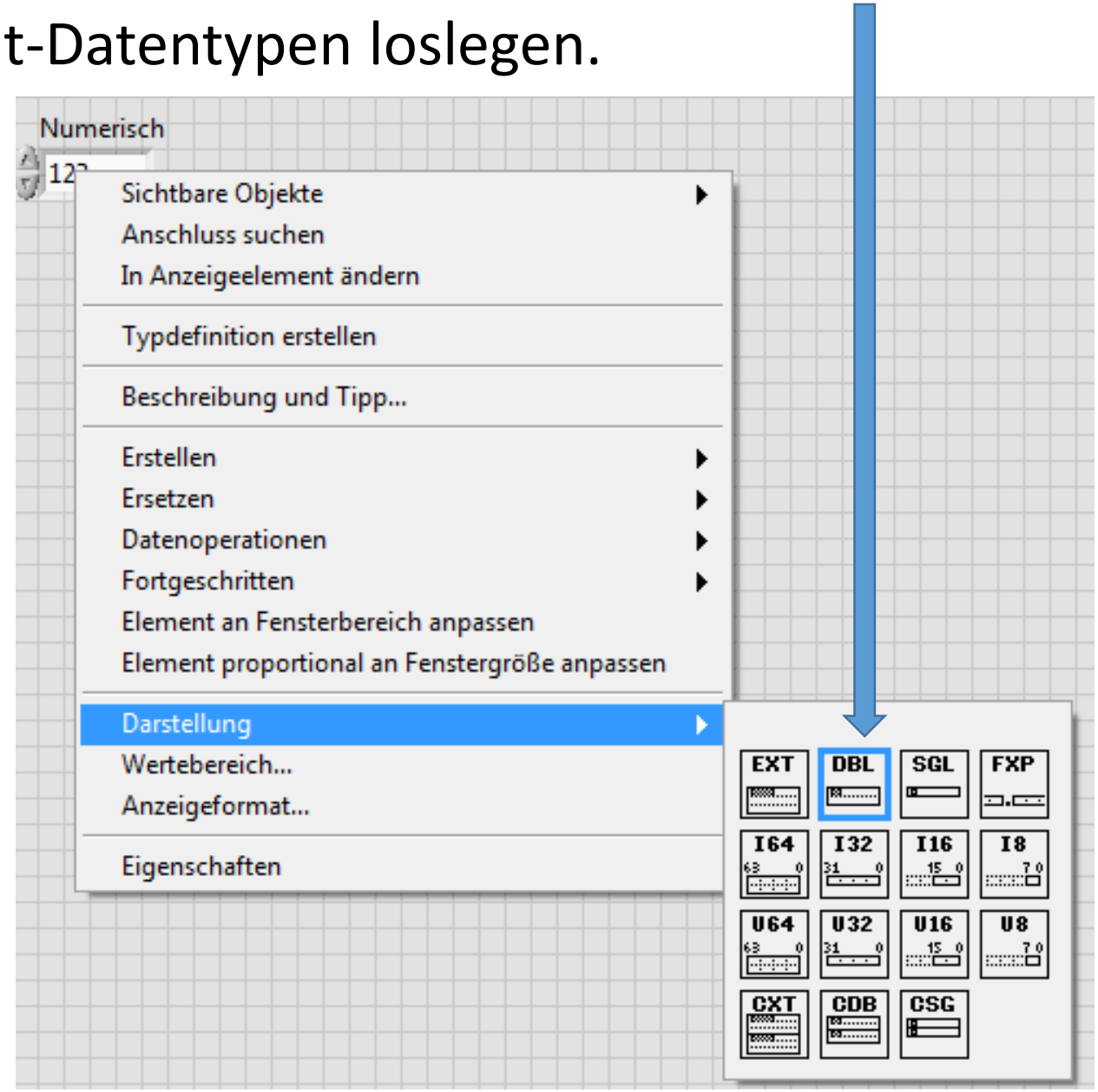
Jetzt: Datentypen der Frontelemente sinnvoll festlegen.
Nicht einfach mit den Default-Datentypen loslegen.

1. Im Frontpanel mit RECHTER Maustaste auf das Objekt klicken
2. Darstellung → Datentyp auswählen.

Datentypen (typisch)

Datentyp	Wortbreite	Speicher
short	8 Bit	1 Byte
int	16 Bit	2 Byte
long	32 Bit	4 Byte
longlong	64 Bit	8 Byte

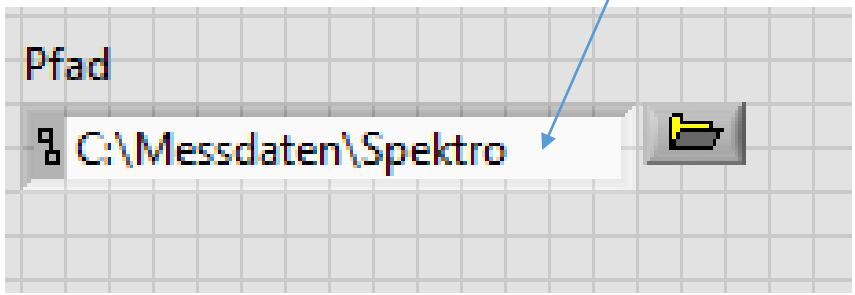
So heißen die Datentypen in Ansi-C

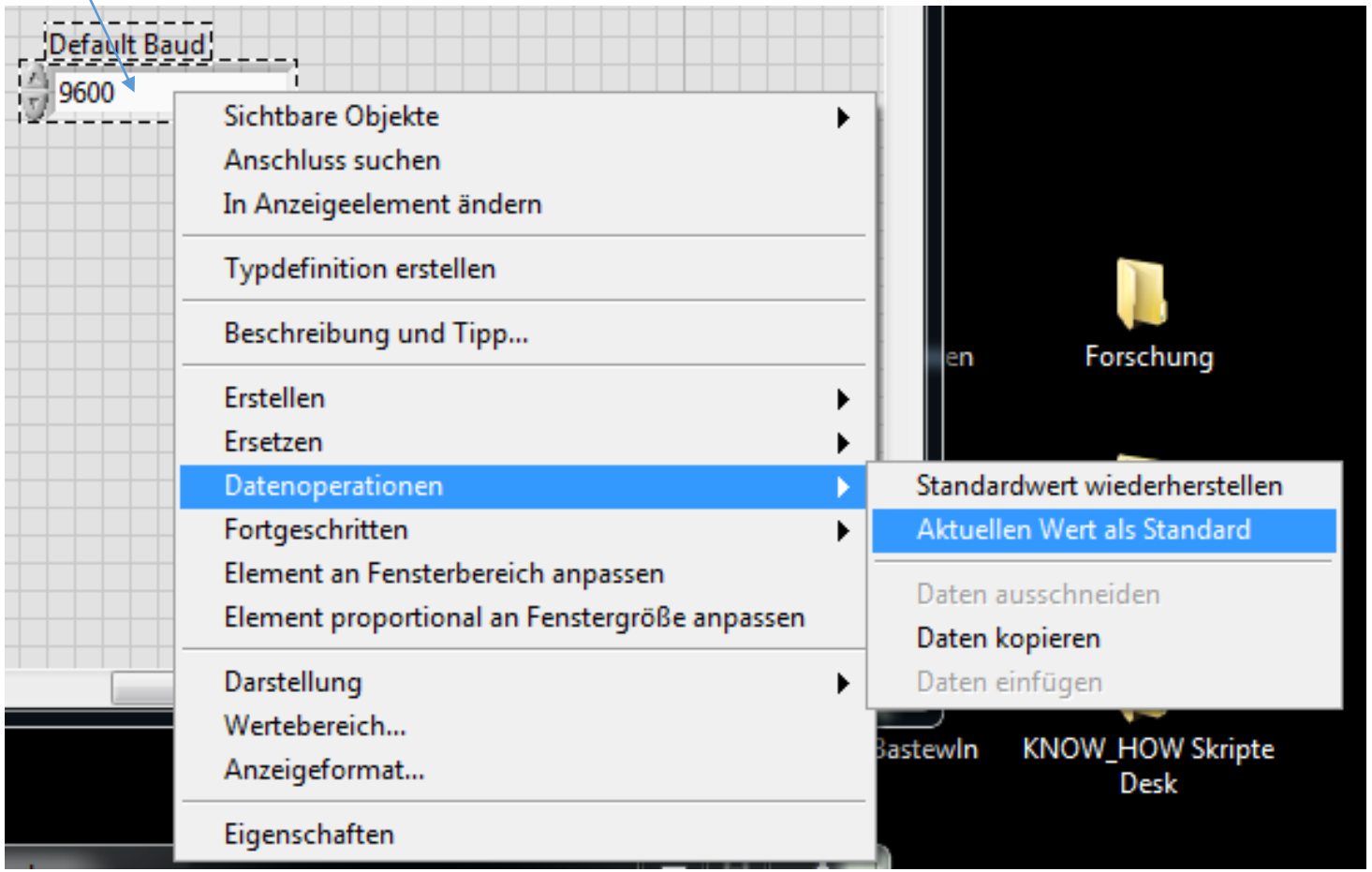


Startwerte festlegen

1. Im Objekt „Wert“ bzw. „Default Content“ eintragen (Vorher Datentyp sinnvoll festlegen)
2. Im Frontpanel mit RECHTER Maustaste auf das Objekt klicken
3. Datenoperationen →
4. Aktuellen Wert als Standard
5. Speichern.

Beispiel



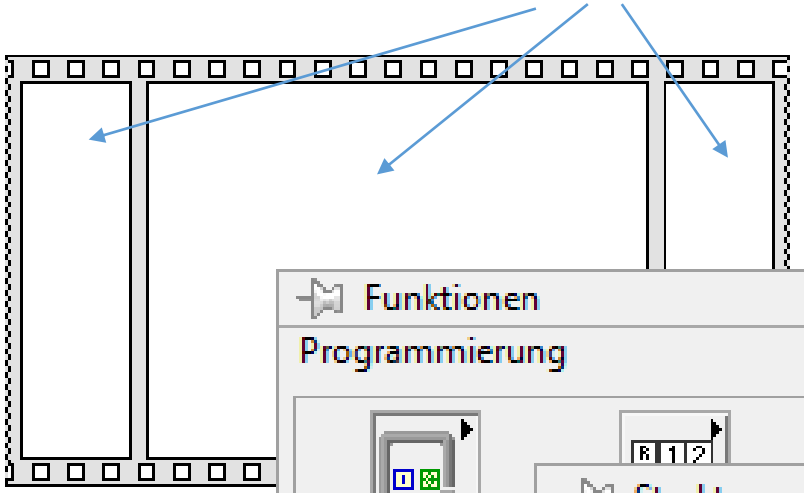


Programmfluss organisieren → 3 Phasen Organisation

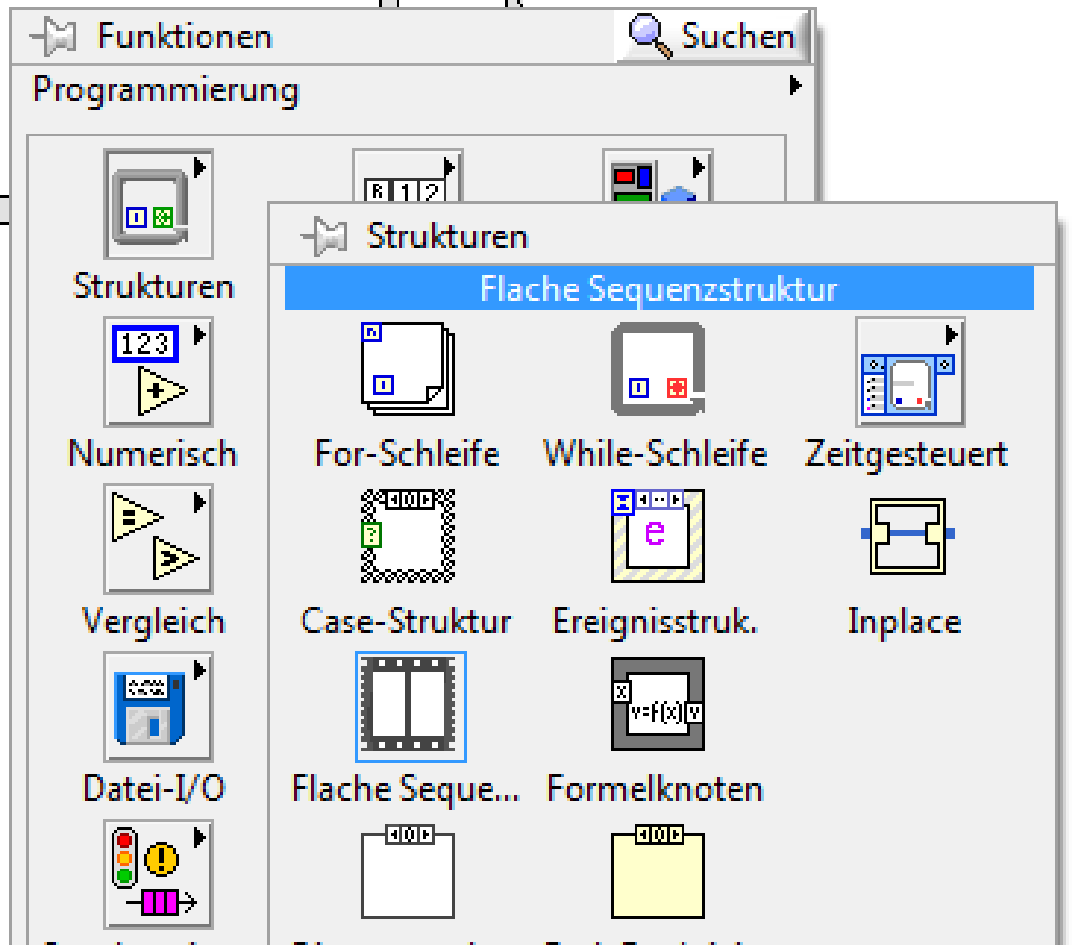
Ein **LabView VI**,

wie jedes andere Programm,

sollte **3 Phasen** haben.



FILM Symbol. Ablauf = „Bild für Bild“ Metapher

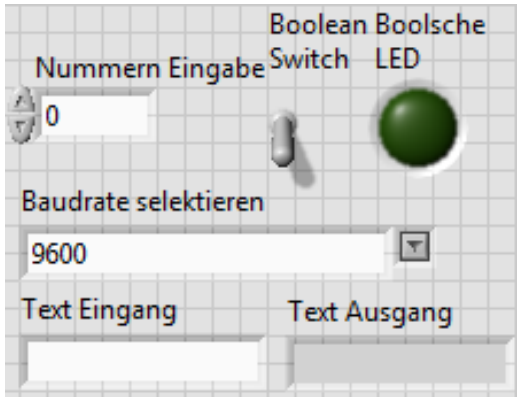


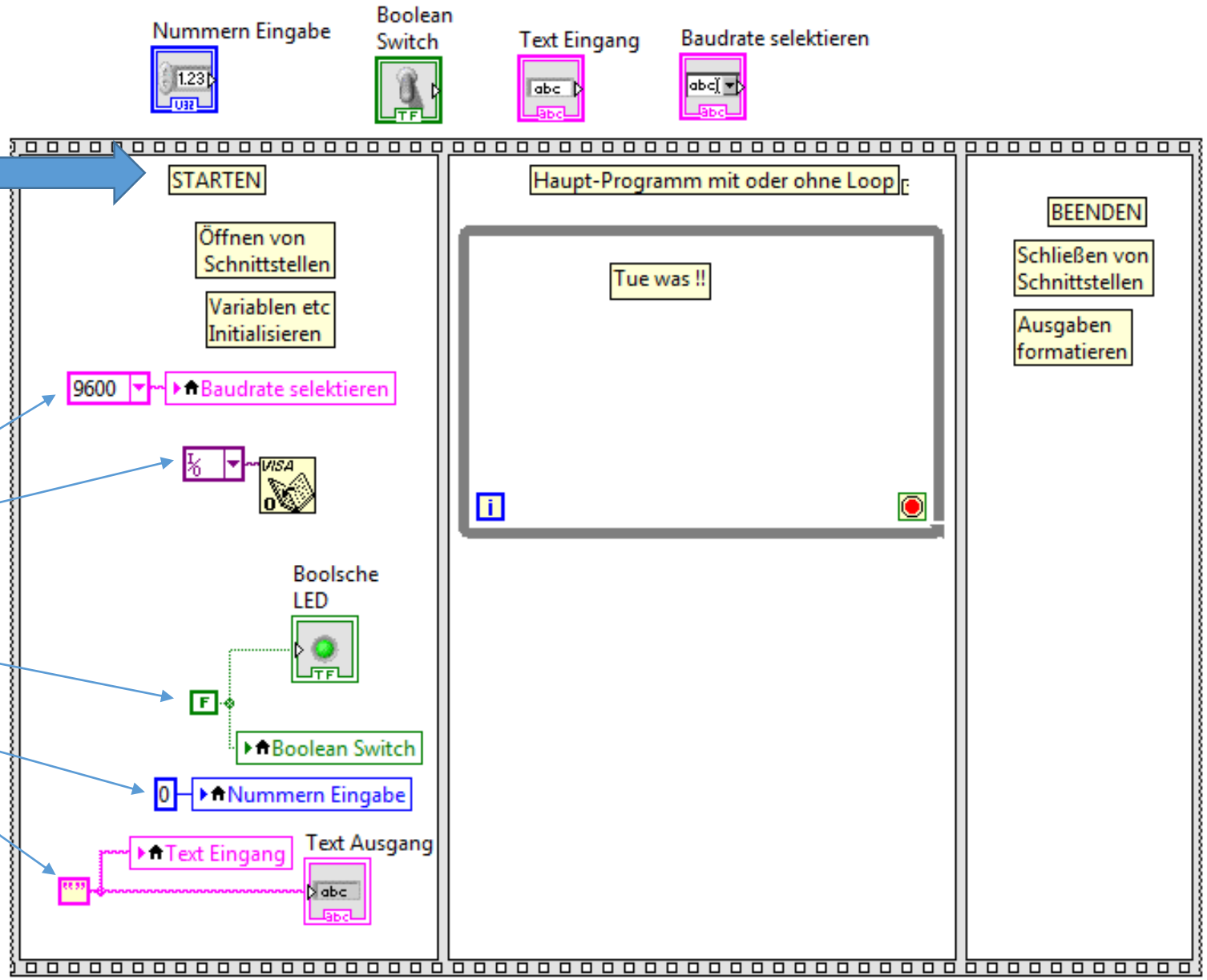
1. **Programm Start:**
Startpositionen Initialisiere.
(*True/False* Schalter), Defaults und Variablen definieren,
Flags Setzen
WICHTIG: **Schnittstellen öffnen**.

2.**Hauptprogramm-Funktion**
Programmfluss mit oder ohne Programmschleife

3**. Programm beenden**.
Werte zur Ausgabe formatieren.
WICHTIG: **Schnittstellen (Ports) schließen.**

Phase 1: „Programm starten“ vorbereiten





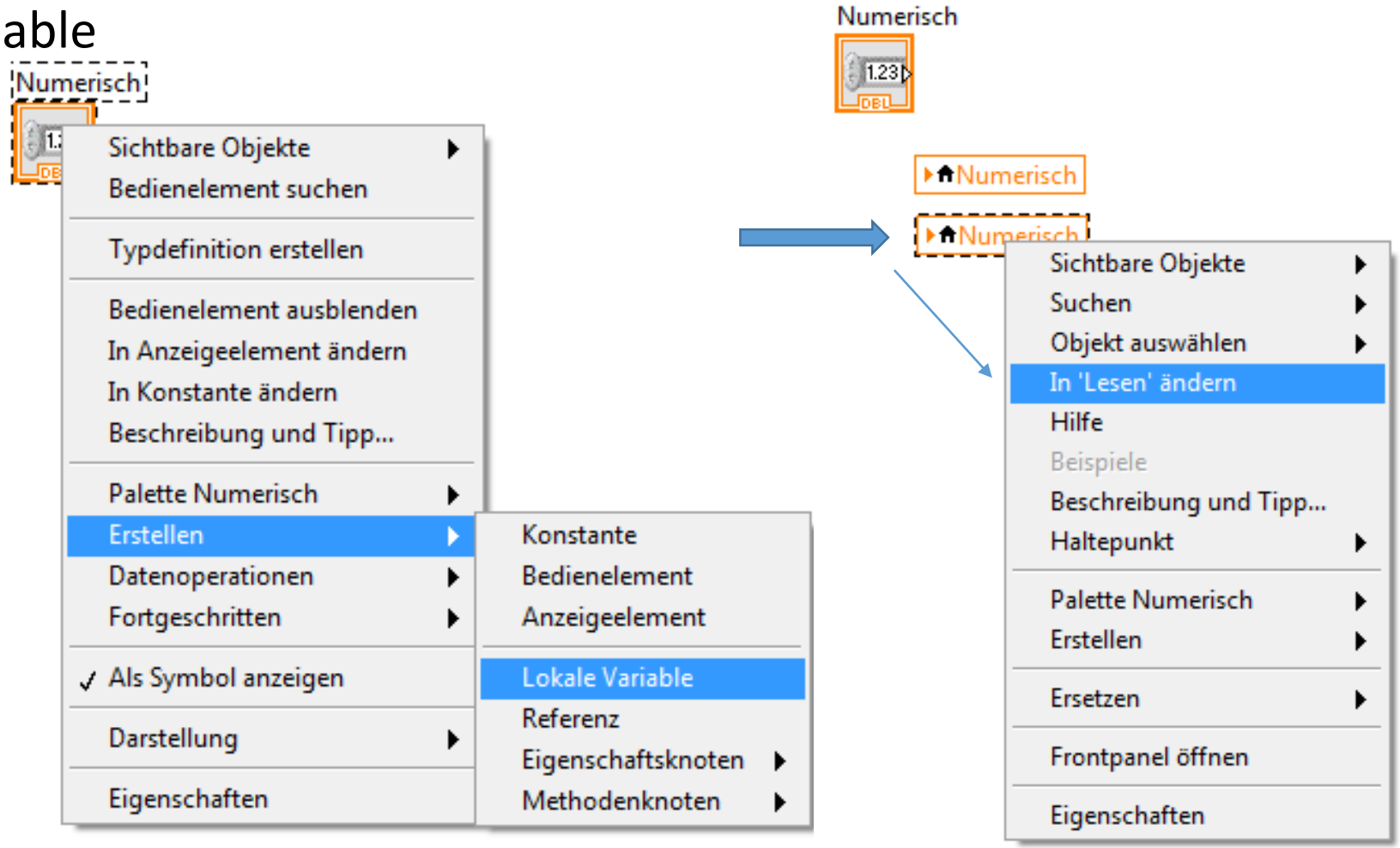
Mit Konstanten oder „Lokalen Variablen“ können Zuweisungen für Datenquellen leicht realisiert werden.

Lokale Variable erstellen

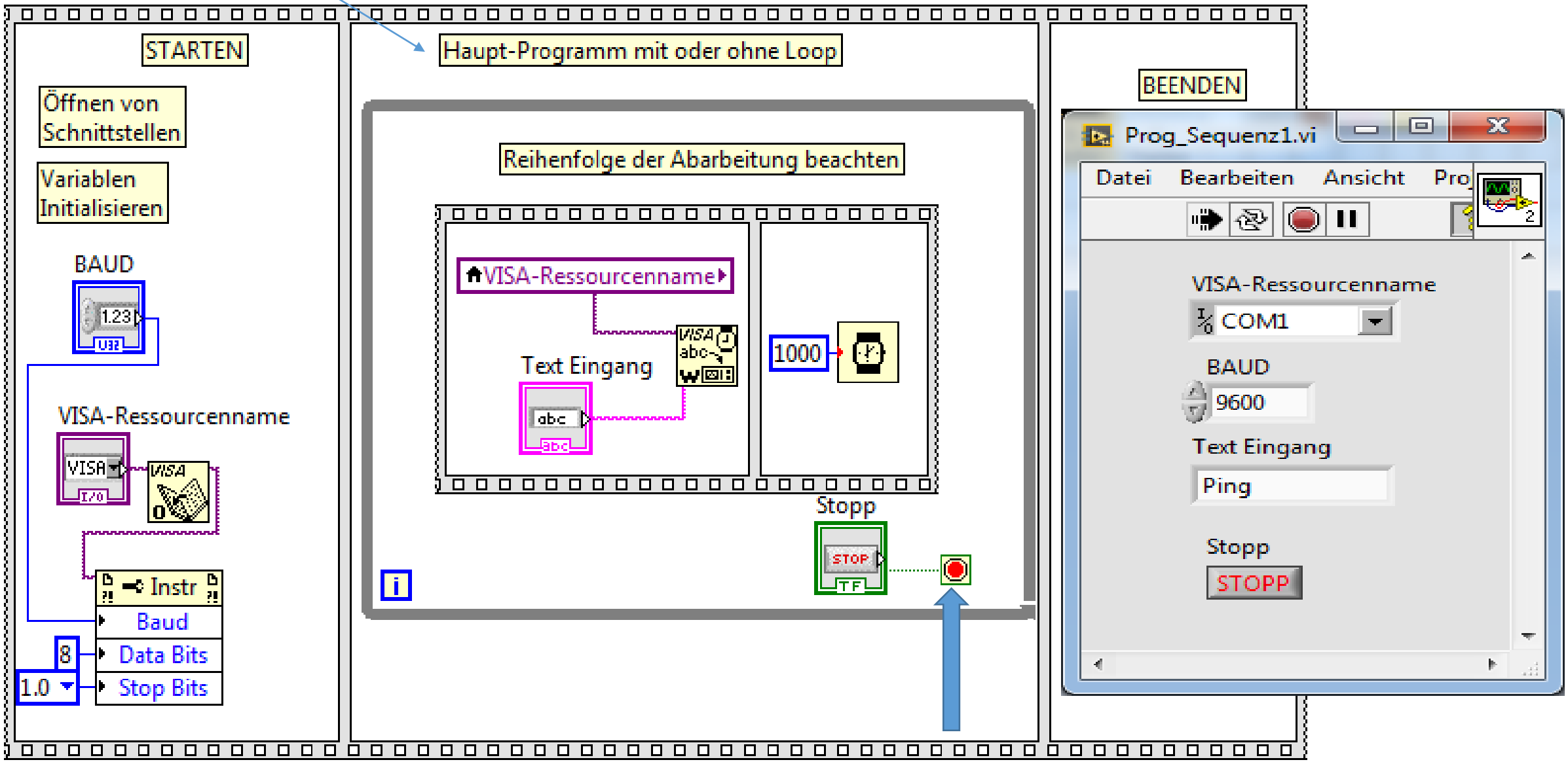
1. Im Frontpanel mit RECHTER Maustaste auf das Objekt klicken
2. Erstellen → Lokale Variable
3. Variable ablegen.

4. Bei Bedarf ändern: Variable Lesen oder Schreiben:

...wieder *„right click“* auf Variable
In ‚Lesen‘ ändern
Oder..
In ‚Schreiben‘ ändern
um die Datenrichtung auszuwählen



Phase 2: sinnvolle Ausführung des **VI** Zwecks



Das Schleifenmonster vermeiden……

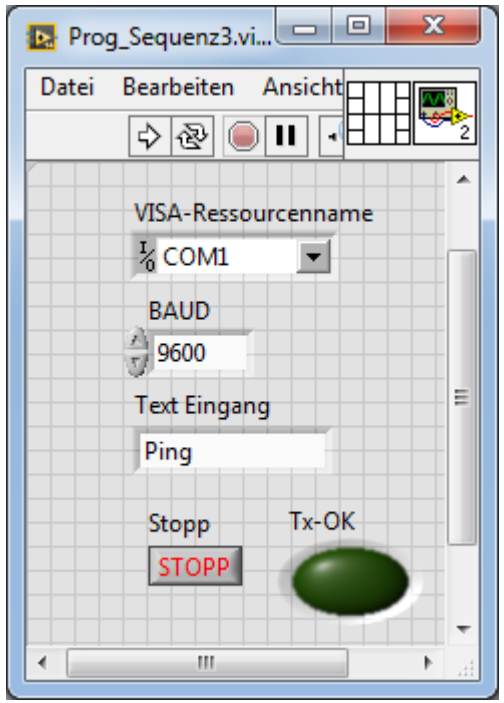
….durch Erstellung von mehreren **Unter-VIs** *‚virtual Instruments‘*

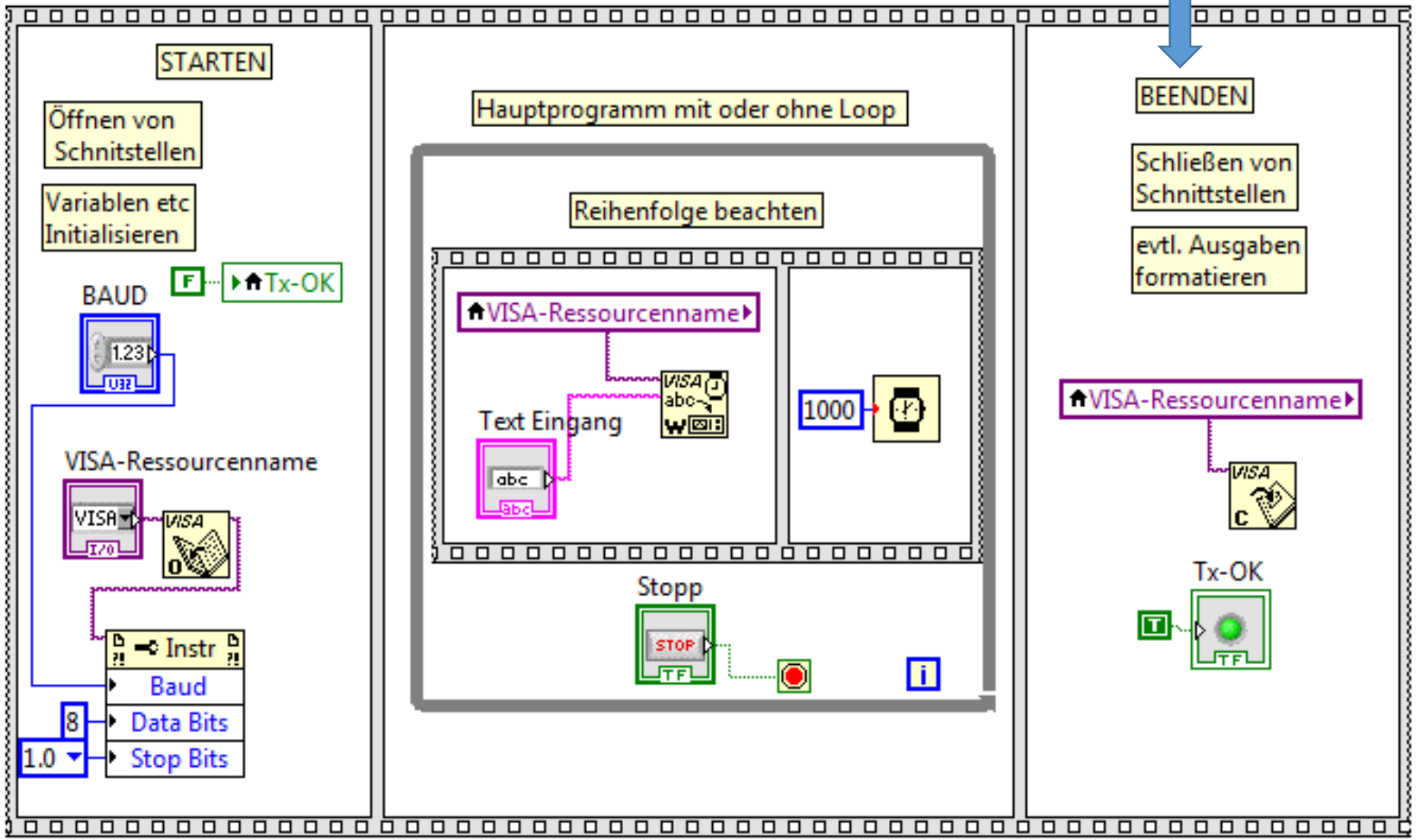
So kann man häufig gebrauchte Operationen in einem eigenen VI verarbeiten.

..dazu später mehr

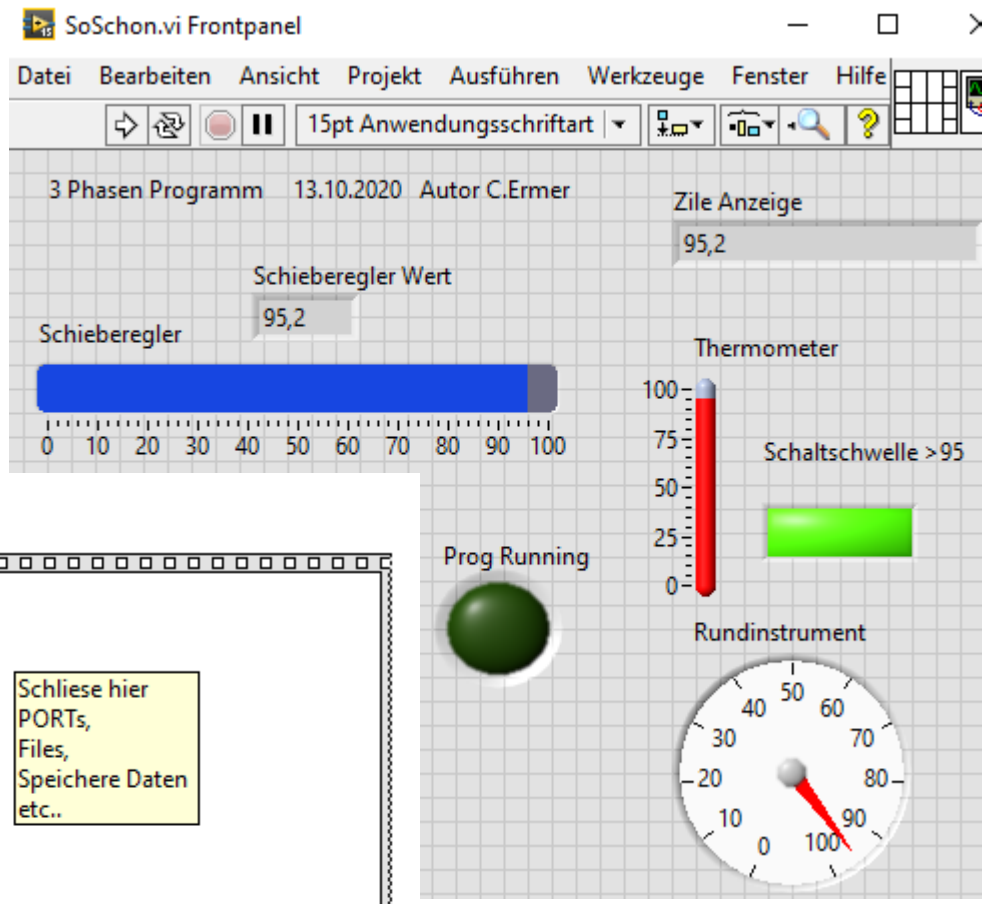
Phase 3: VI ordentlich beenden.

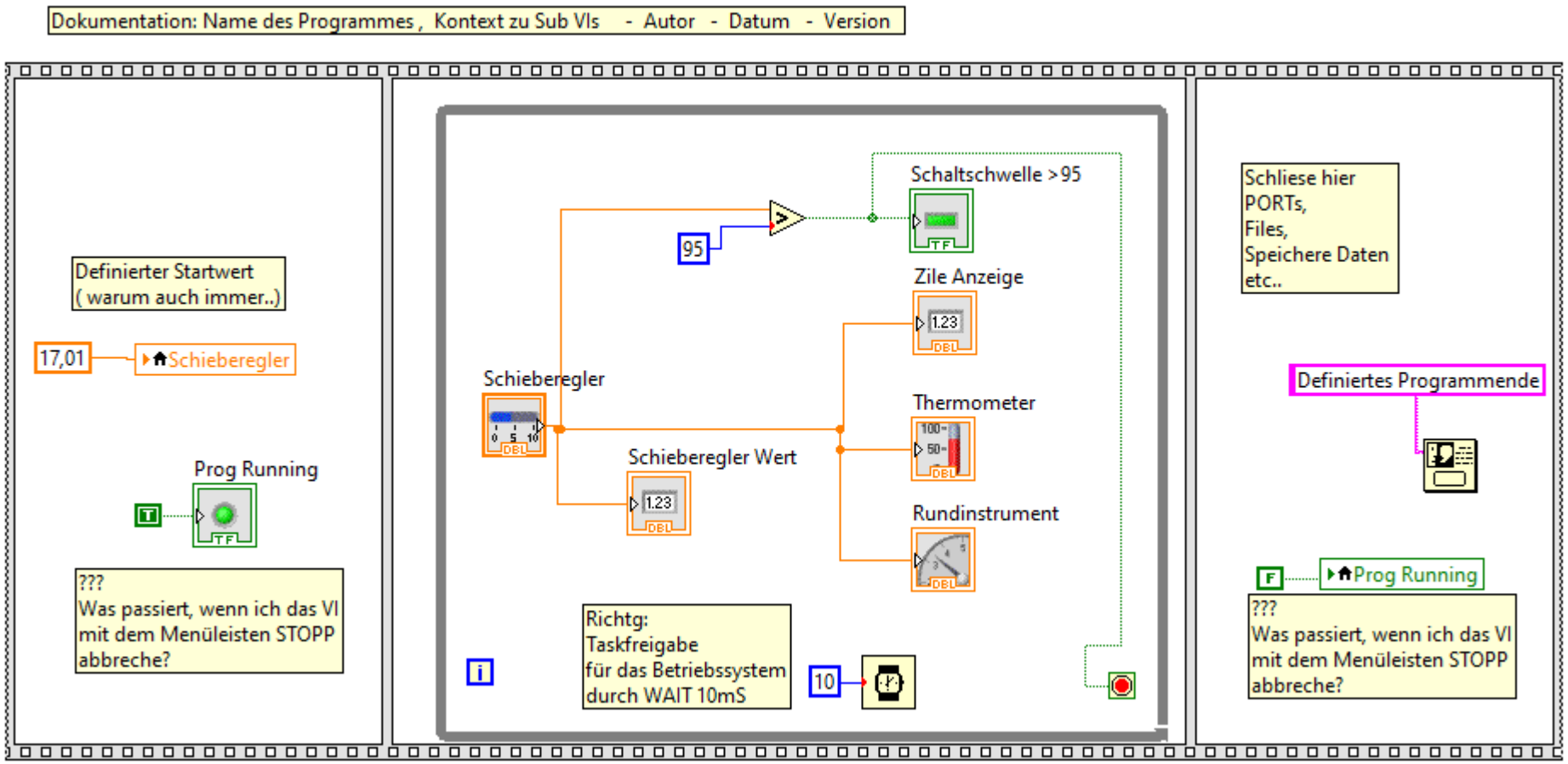
Am Schluss !
Alle Schnittstellen schließen bzw. Aufräumen und Ausgabedaten formatieren





Schon besser:
3 Phasen Start – Run –Stop
keine Datentypfehler in den Zuweisungen
Elemente mit sprechenden Namen
Kommentiert

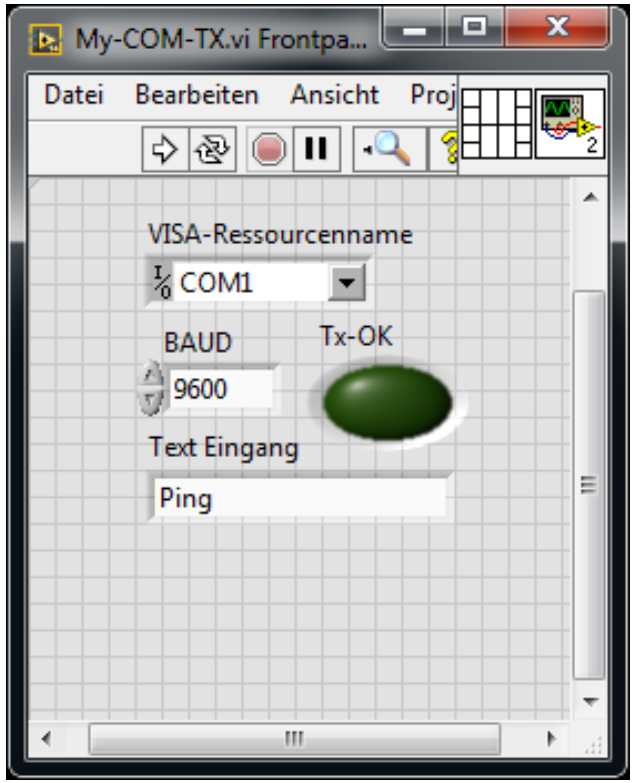


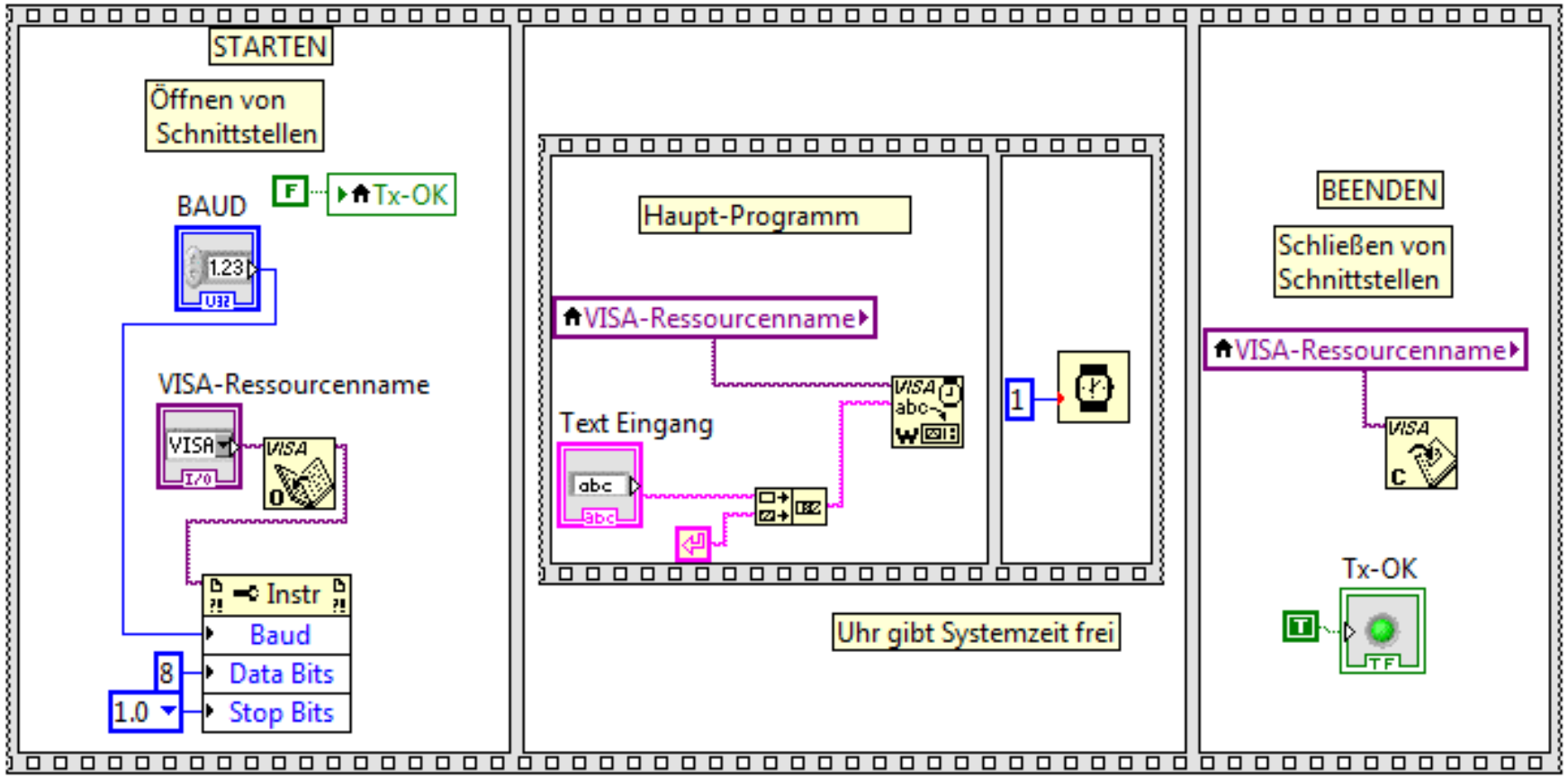


UNTER-VIs = Unterprogramme erstellen.

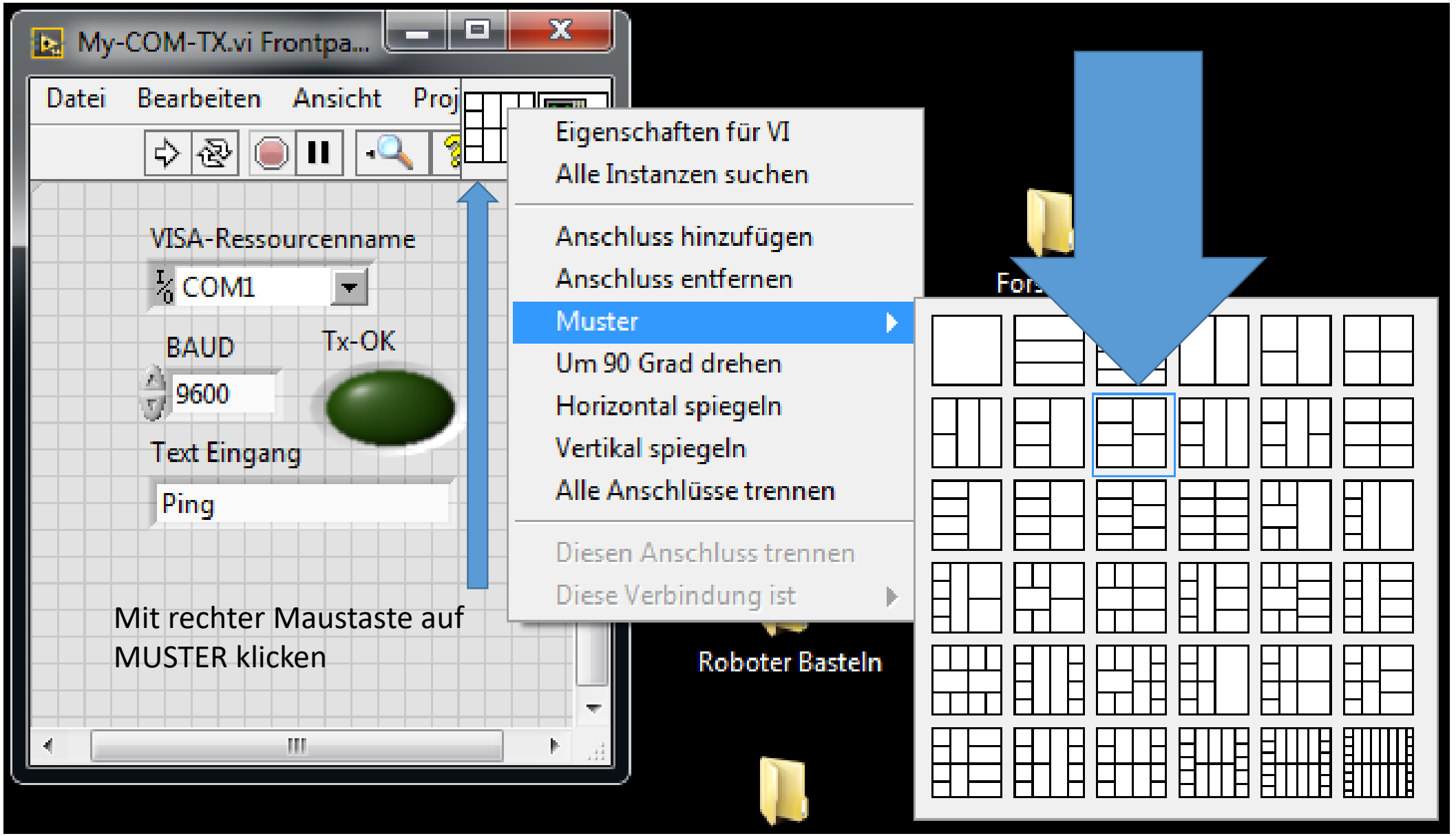
Es ist Sinnvoll, häufige Routinen einmal als eigenes, lauffähiges VI zu erstellen

Wir wollen mal ein serielle COM Sende VI erstellen



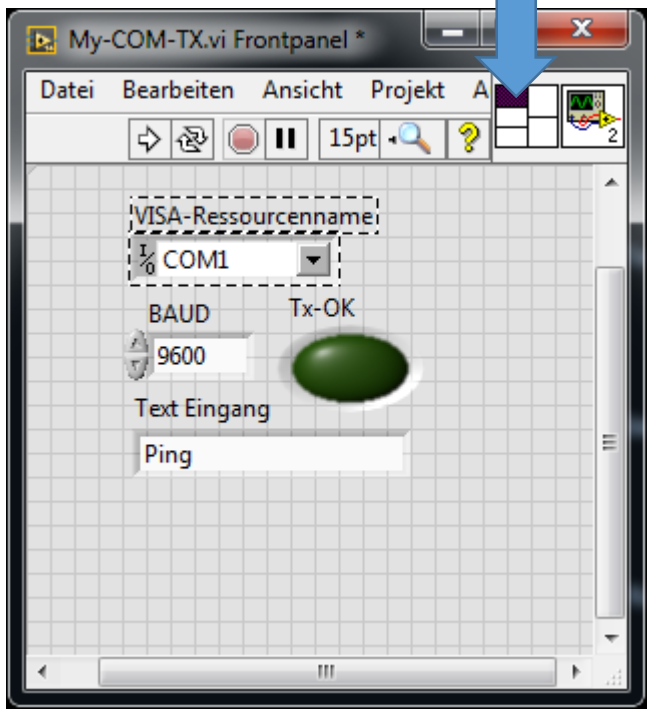


Anschluss Muster des VIs auswählen bzw. bestimmen

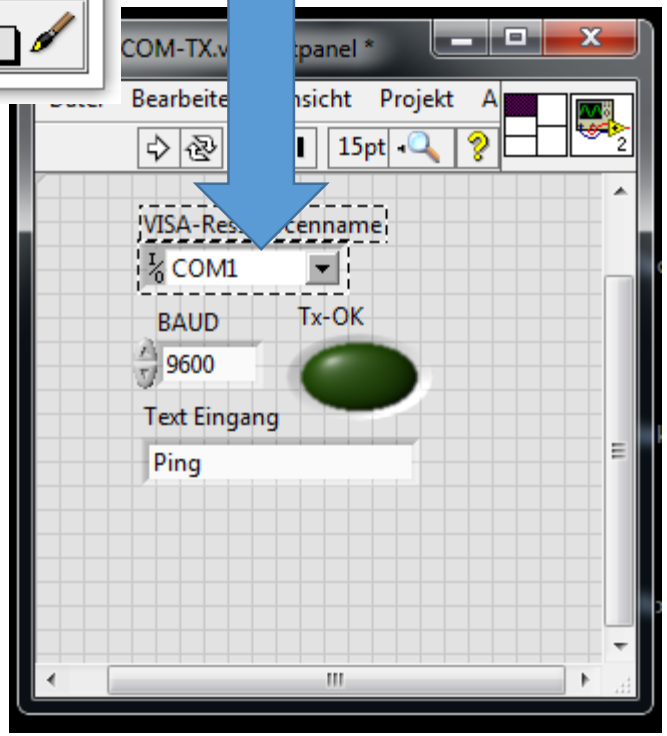


Verbinden der gewünschten Kontaktstellen im Muster mit den zugehörigen Datenobjekten… dann Speichern.

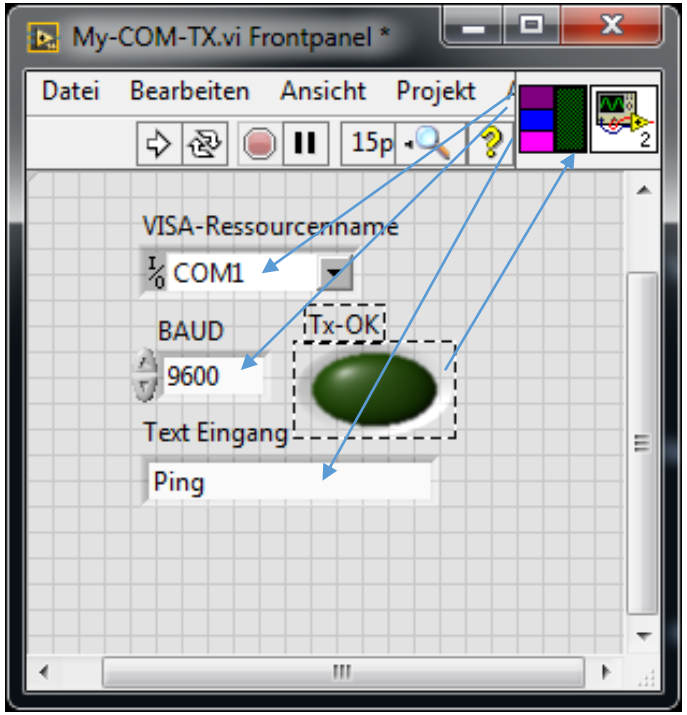
Mit Garnrolle Tool auf Anschluss…



…dann auf Übergabeobjekt

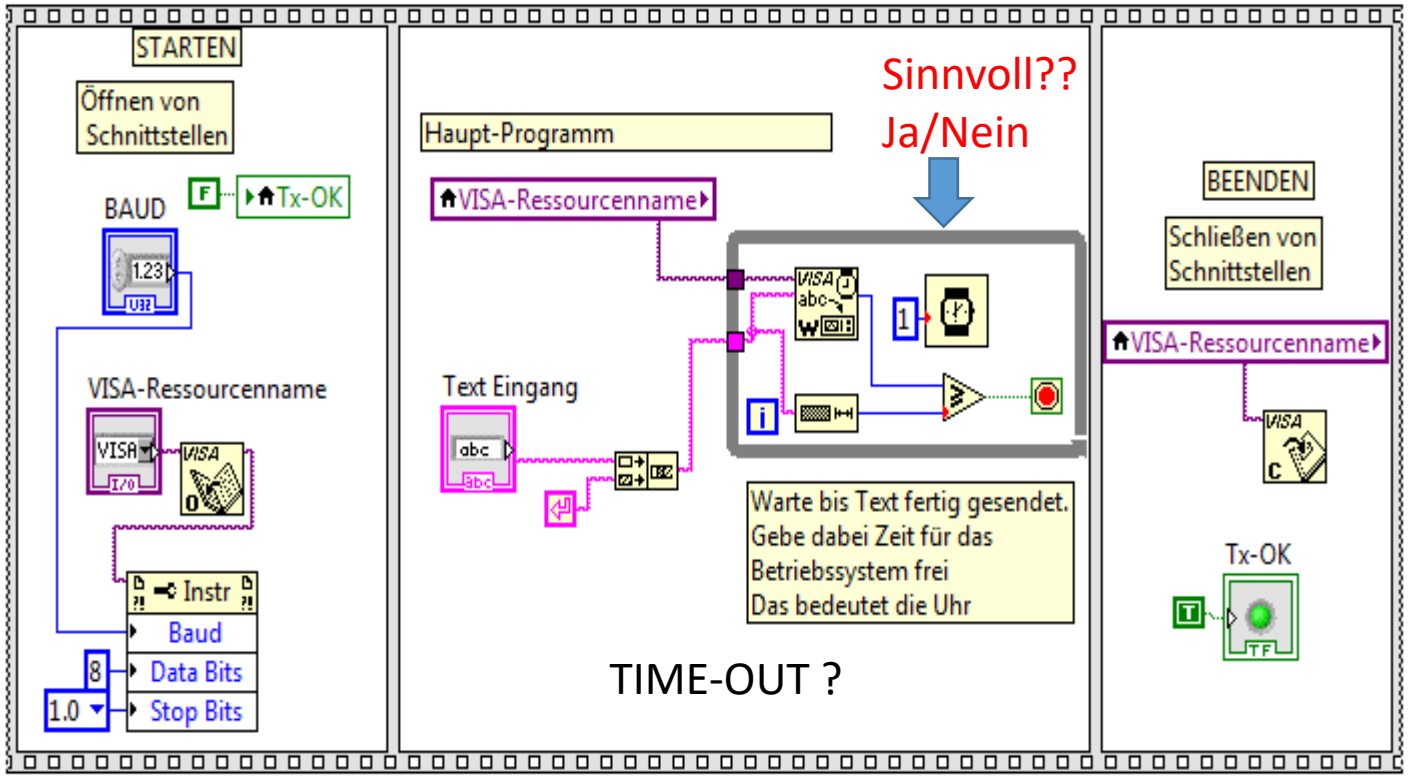


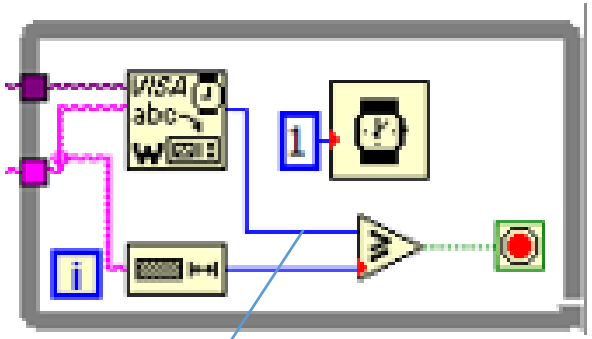
Alle benötigten Objekte mit Anschlüssen des Musters verbinden. (vernähen)



Lerneffekte….durch tieferes Verständnis der Abläufe

Serielle Daten brauchen Zeit für die Übertragung.
Man könnte auf die Idee kommen ein automatische Wartezeit bis zum Sendeende zu programmieren.
Doch ist das so..? 1. Erst muss man rausfinden **wie** die „Write“ Funktion arbeitet und was die Ausgaben bedeuten. 2. Es ist wichtig immer die Datenflussrichtung im zeitlichen Kotext im Auge zu behalten.
→Was geschieht in welcher Reihenfolge? Dies kann man mit dem „Film“ schon gut organisieren



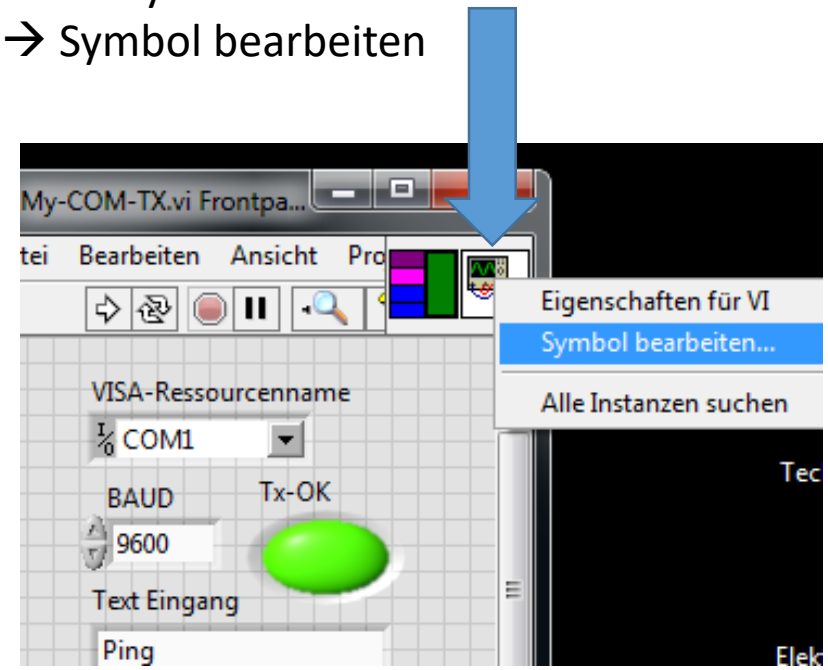


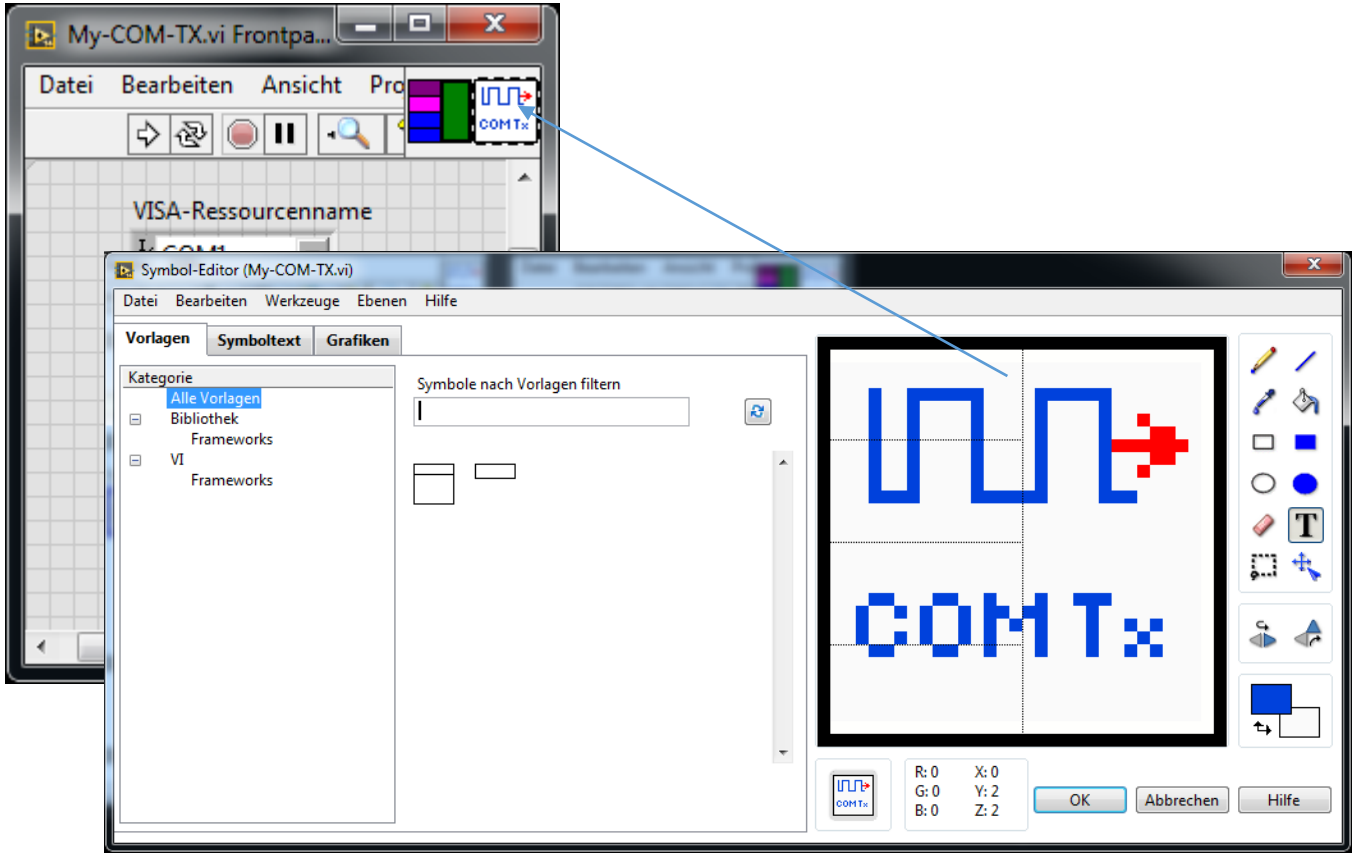
Gesendete Zeichen Zähler

Diese Loop sollte warten bis alle Zeichen gesendet sind. Tatsächlich wird die Schleife nur einmal ausgeführt und ist somit unsinnig. „Write“ wartet von sich aus bis alle Zeichen gesendet sind. →PRAXISTEST

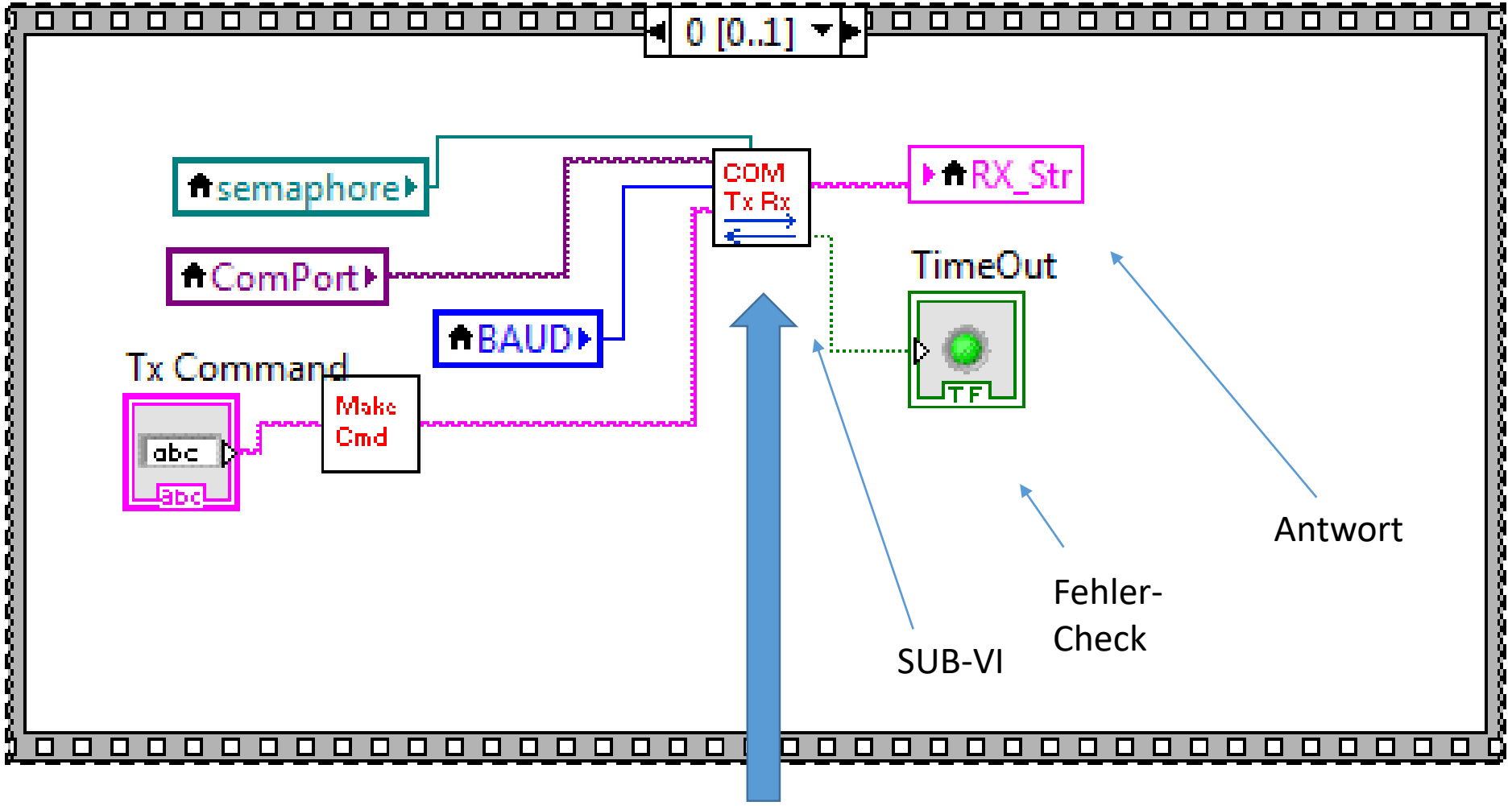
Symbol bearbeiten, und so Individualisieren

Das Standardsymbol ist
wenig aussagekräftig.
Es ist sinnvoll es dem
Zweck des VIs anzupassen
RC → Symbol
→ Symbol bearbeiten



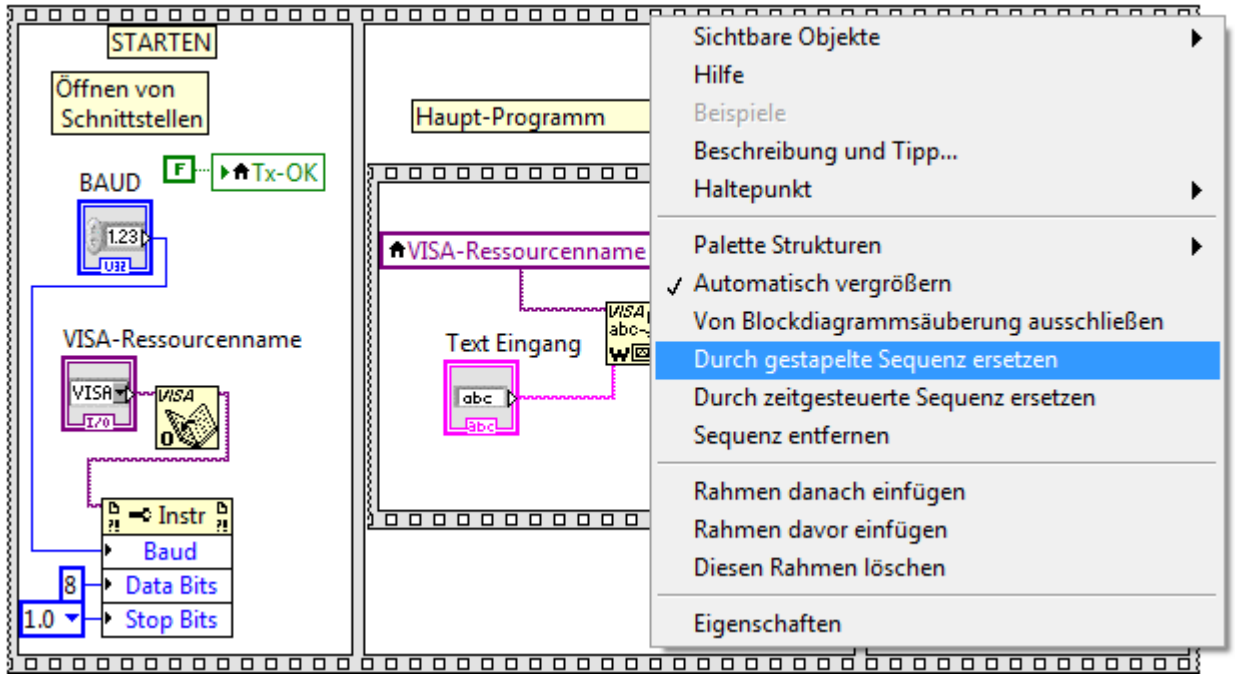


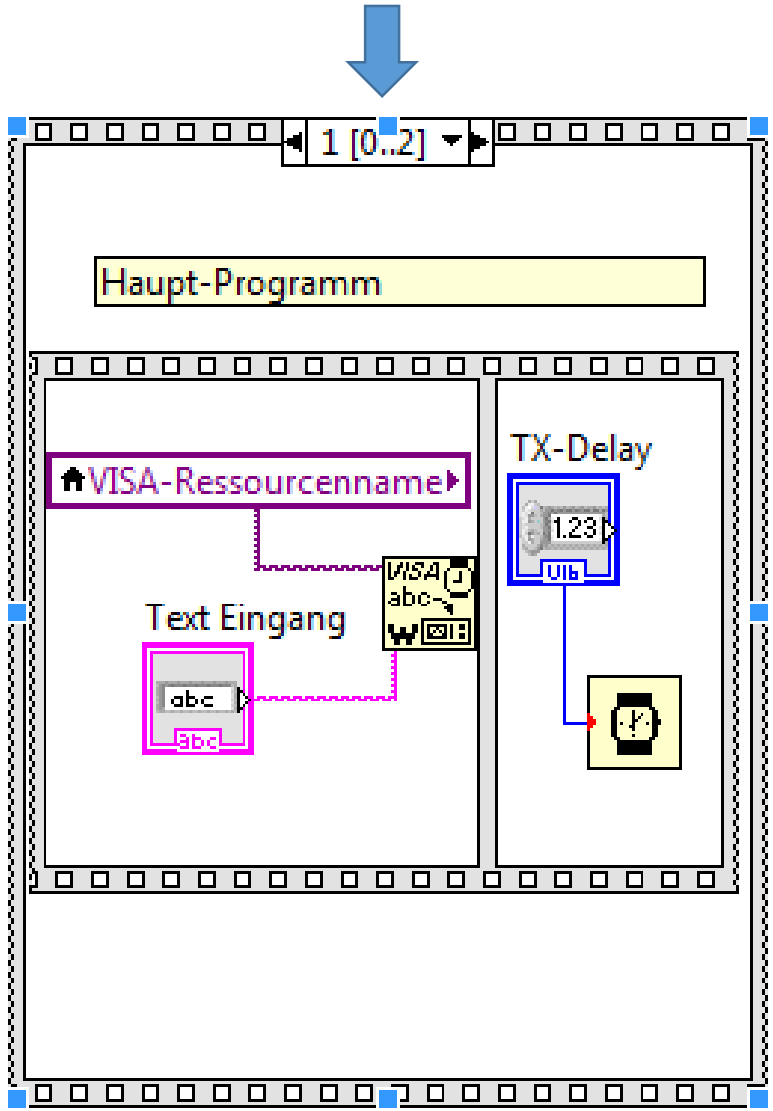
Einbinden von Unter-**VI**s
Hier eine Sequenz für „textbasierte“ Frage-Antwort via COM



Wenn es eng wird….. Komprimierung der Film-Ablauf Sequenzen durch „gestapelte Sequenz ersetzen“

MIT „*Right-Click*" = **RC** → auf Film-Perforation klicken



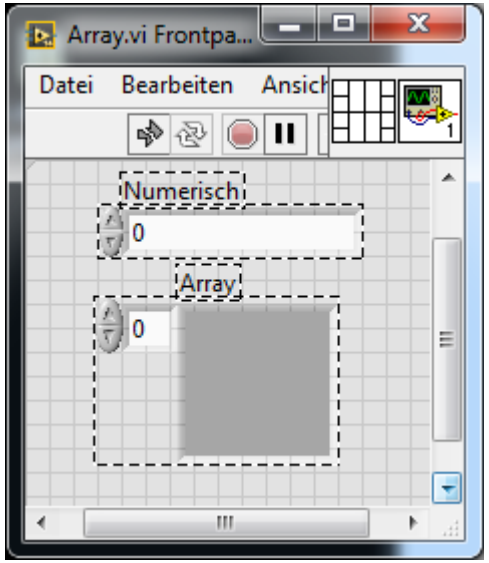


Jetzt ist wieder Platz..
Nur der Überblick wird schwieriger.
Immer wieder mal, die Sequenzen prüfen

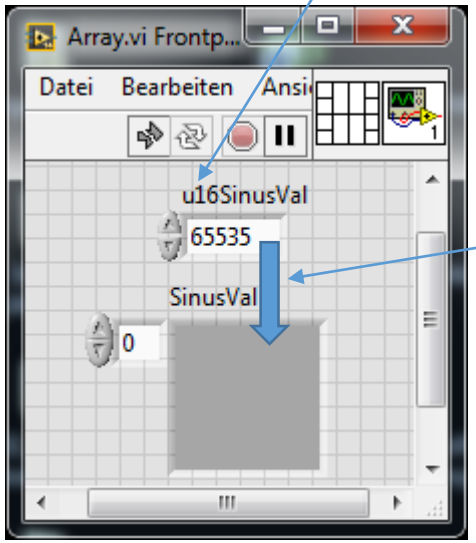
Arrays

Ich stelle jetzt noch ein paar zusammengesetzte Datentypen vor.
Eine Aneinanderreihung des selben Datentyps nennt man Array.

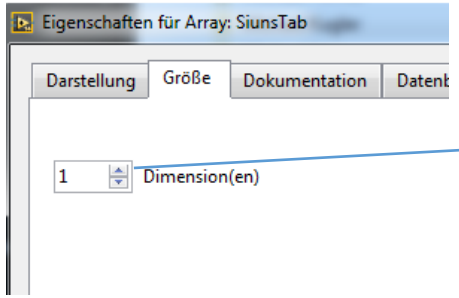
Step 1: Array Element ablegen



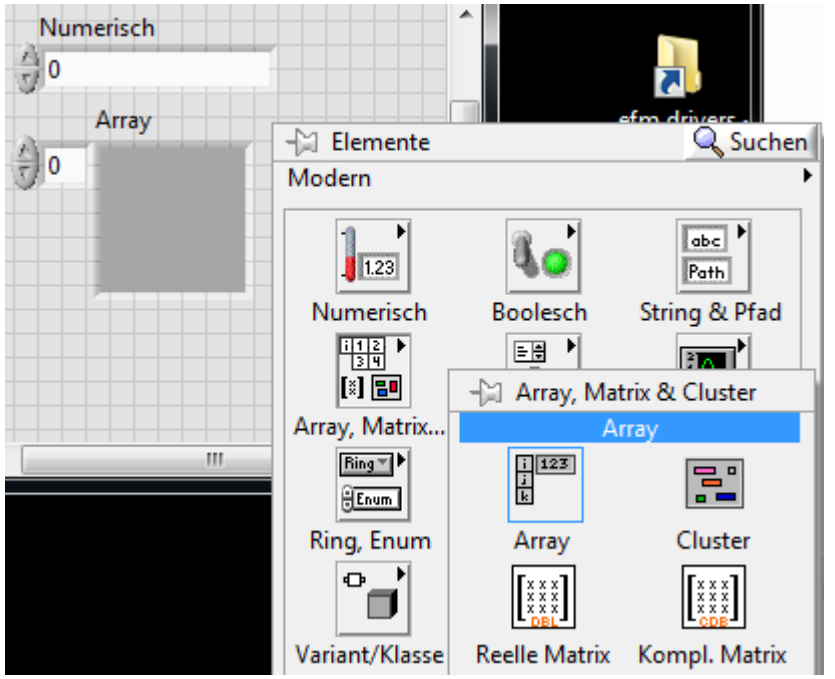
Step 2: Array Element typisieren und benennen (u16Val)



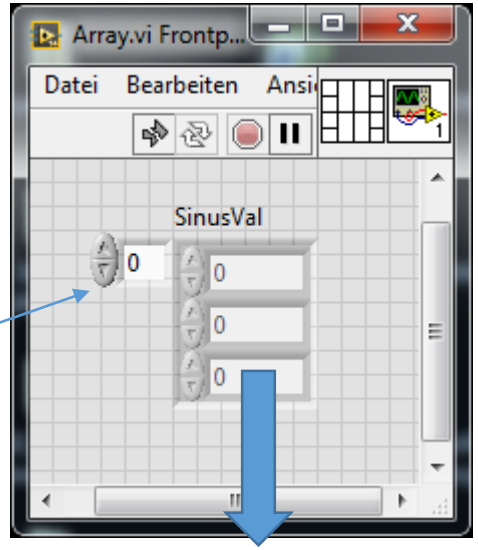
Step 3: Element in das Array hineinschieben



Index

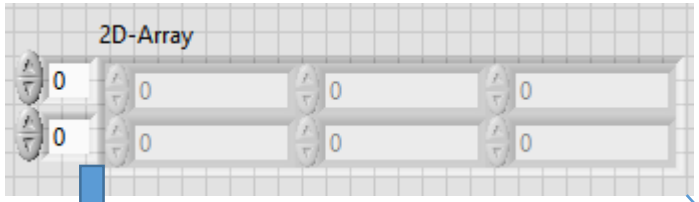


Step 4: Array durch ziehen nach unten vergrößern



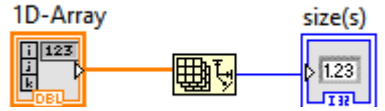
Beispiele:

2..x.D Arrays erzeugen

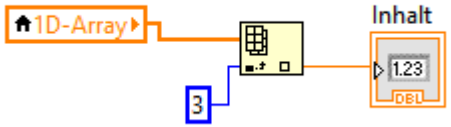


Hier ziehen

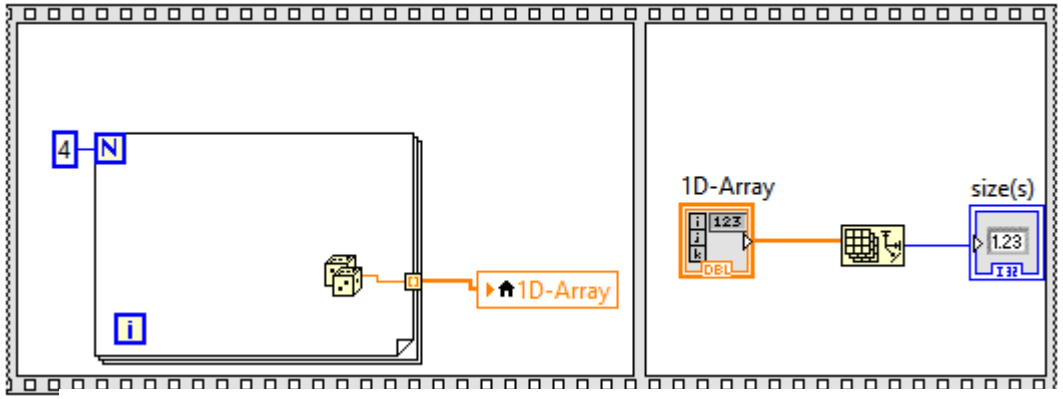
Fenster anpassen

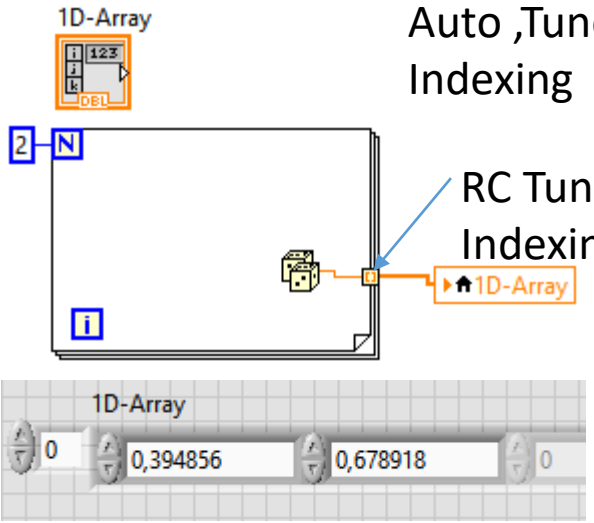


Size



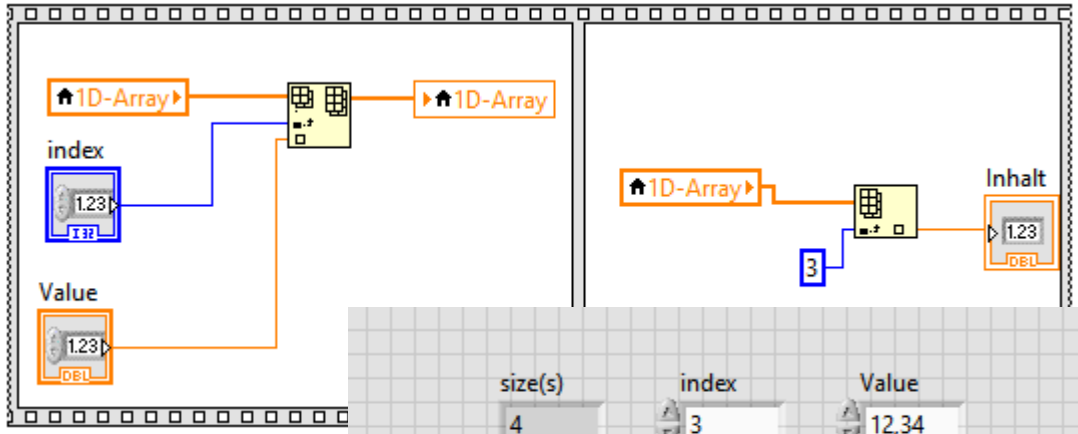
Inhalt auslesen

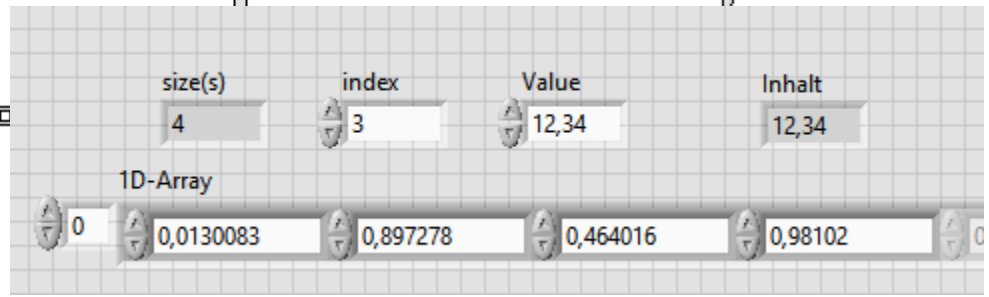


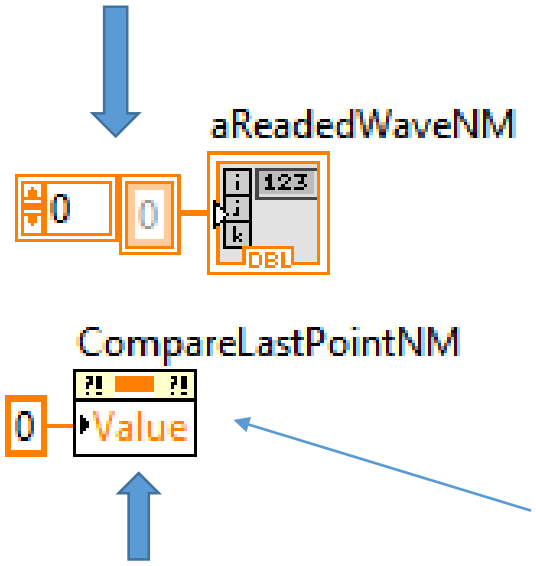


Auto ‚Tunel‘
Indexing

RC Tunel Mode:
Indexing

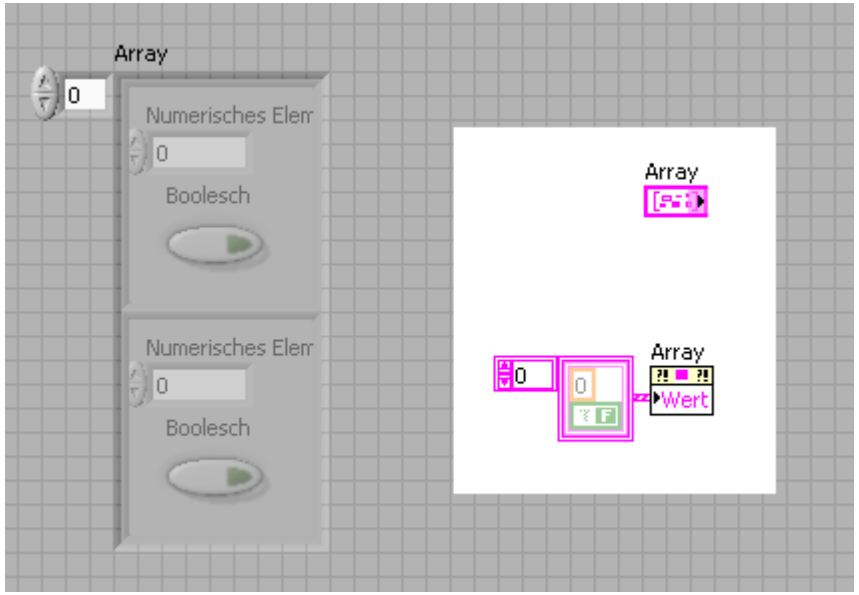






Array löschen: Right Click („RC“) mit Verbindungstool
Create Constant …ist automatisch leer

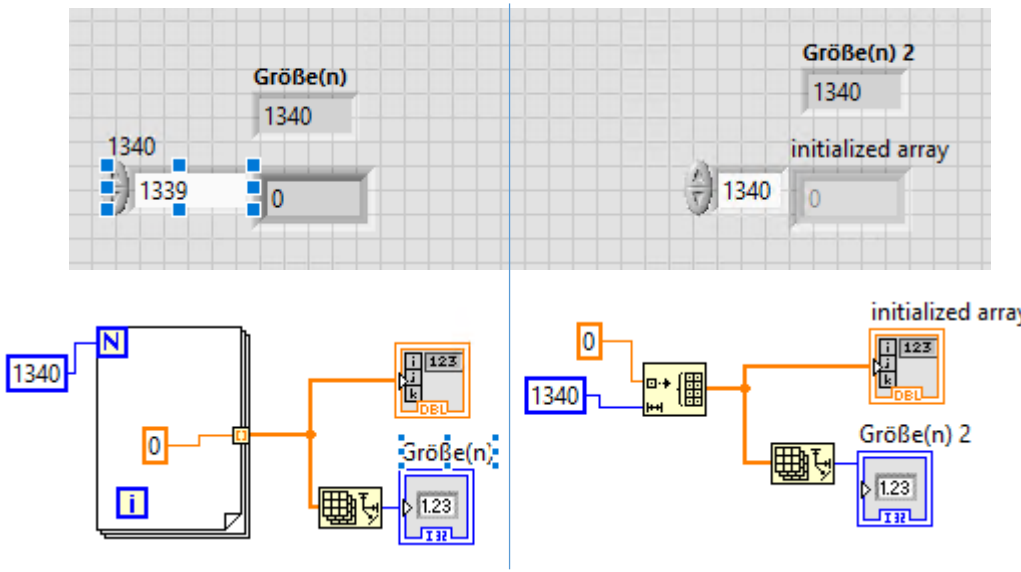
Diesen Tipp habe ich aus dem NETZ:
RE: Array leeren
Hi, probiere mal im Blockdiagramm rechte Maustaste auf dein Array - Erstellen - **Eigenschaftsknoten** - **Wert**, dann **R**echts**k**lick auf Knoten - alle in **Schreiben ändern**, dann Rechtsklick auf den Anschluss - **Erstellen** - **Konstante**



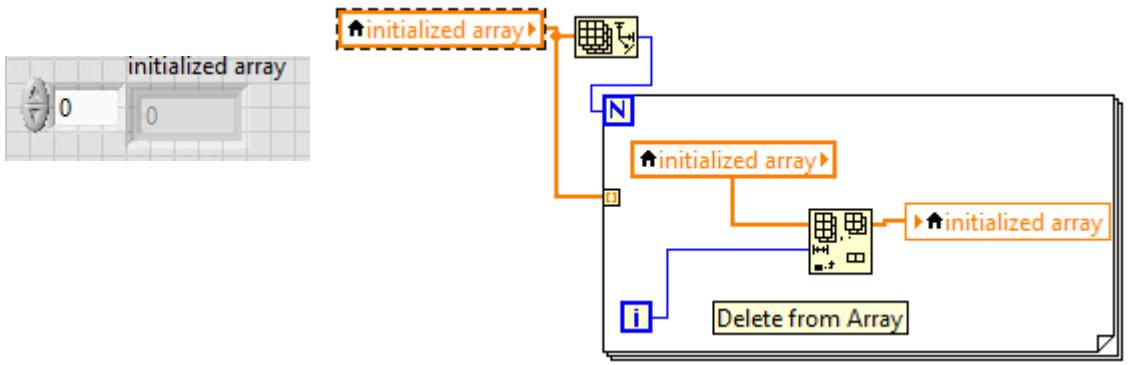
Im Object Fenster Rc -_> create → „Property note“
Hier kann man die Oject Eigenschaften steuern.
Daten, True False etc..
Genauso wie im Eigenschafts-Window.
Nun ist dies mit den ***Property-Nodes*** auch dynamisch im Programm möglich.

Weitere bespiele: Array löschen oder vorbelegen

Frage:? Sind Arrays vor der Benutzung sinnvoll Initialisiert oder geleert ?
Es gibt immer verschiedene Wege nach Rom.



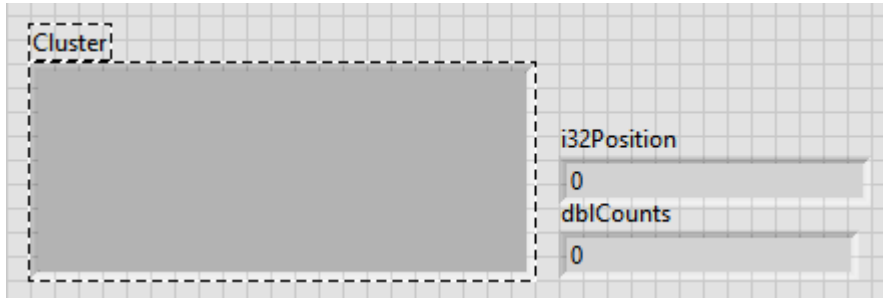
2 Wege ein vorinitialisiertes Array zu erzeugen.
(Der 2. Weg gefällt mir besser.)

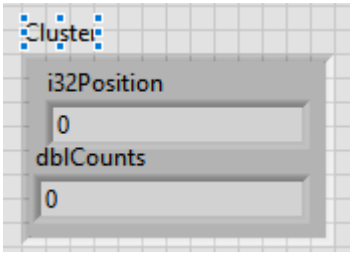


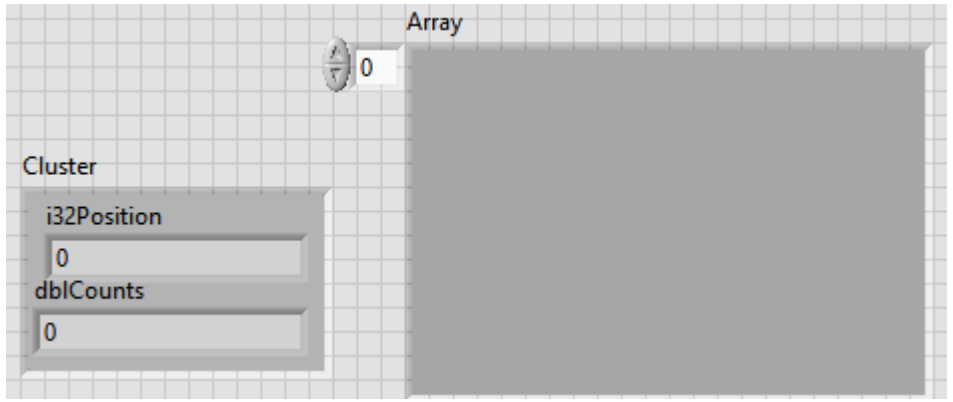
LÖSCHEN der Inhalte.
← oder →

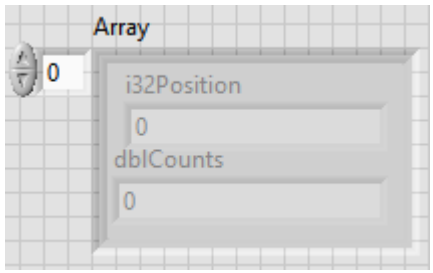
Hier mit „leerer“ Konstante

Cluster (struct in ANSI C)



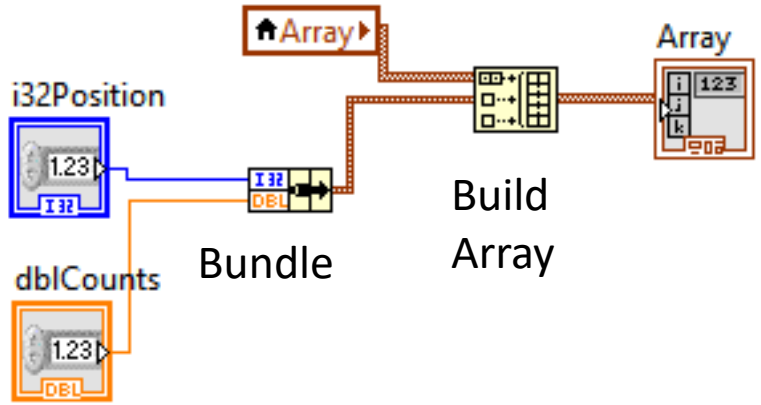




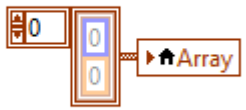


Werzeuge

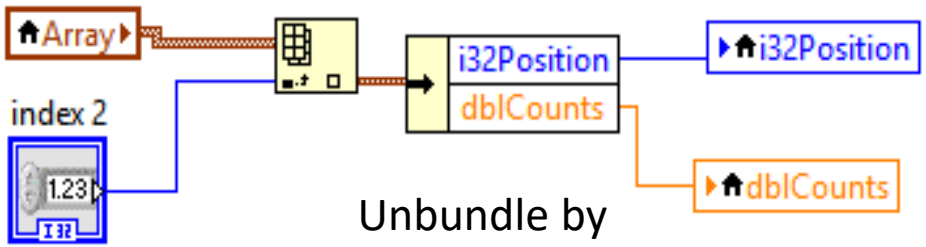
Concatenates mulitple arrays



Create Constant



1. Array Indexieren

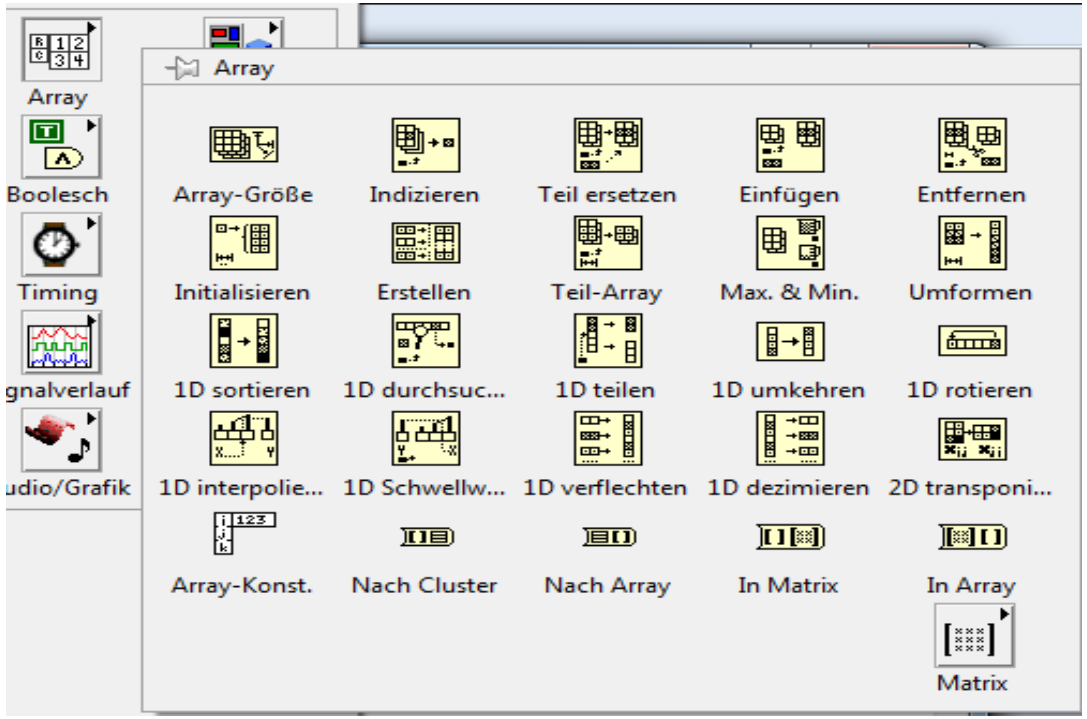


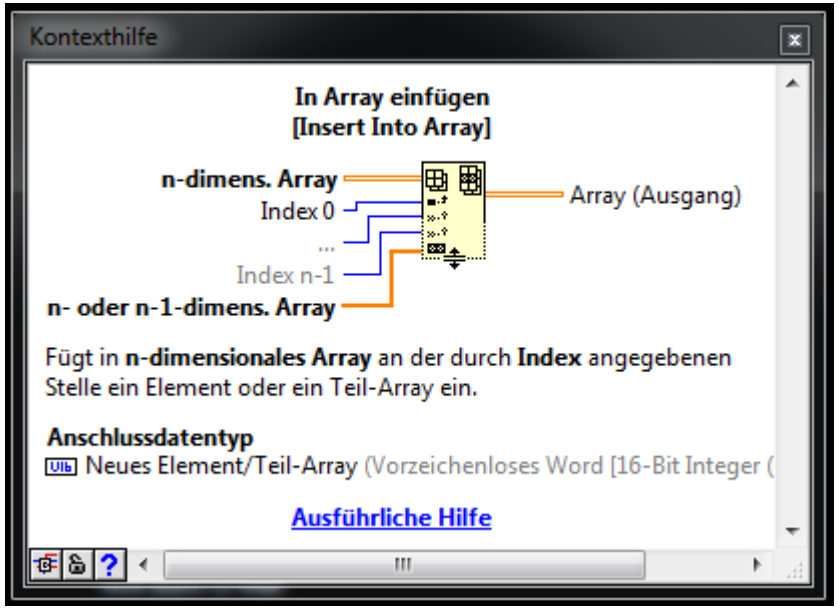
Unbundle by Name

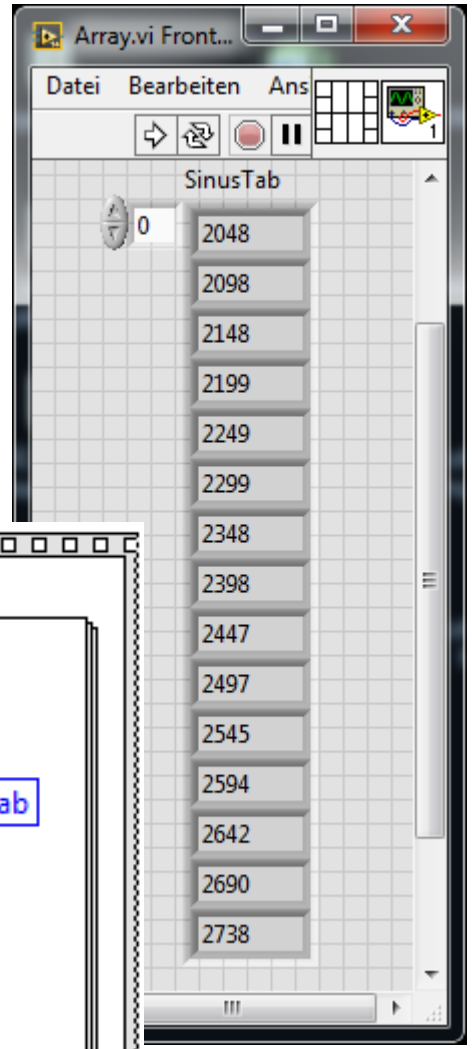
Array Operatoren

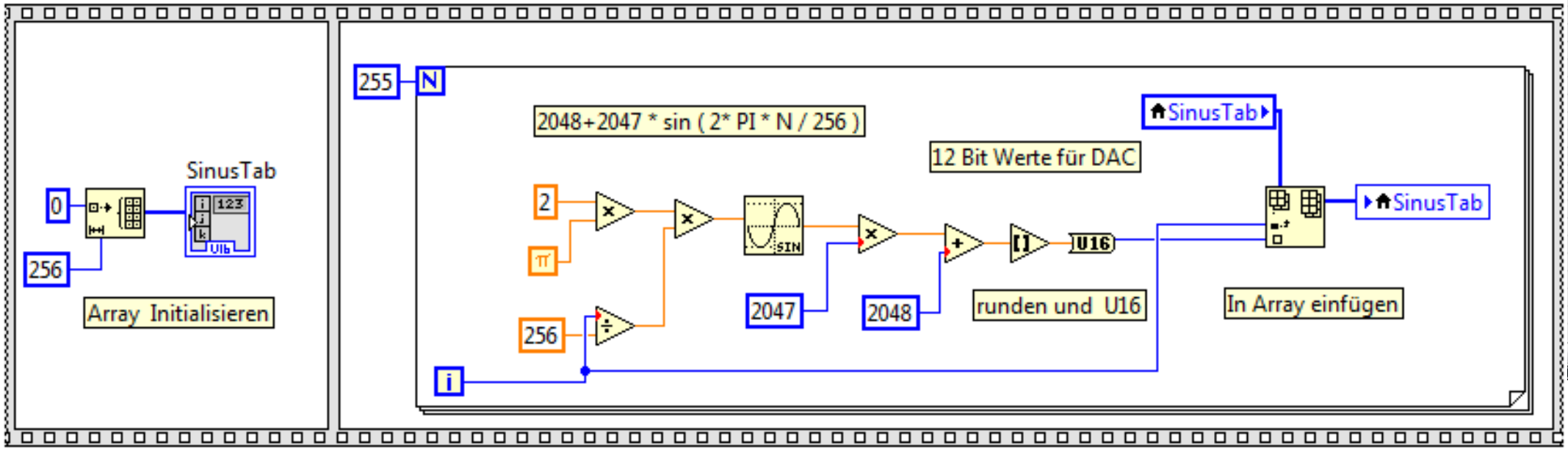
Kontexthilfe

Mit „ **?**“



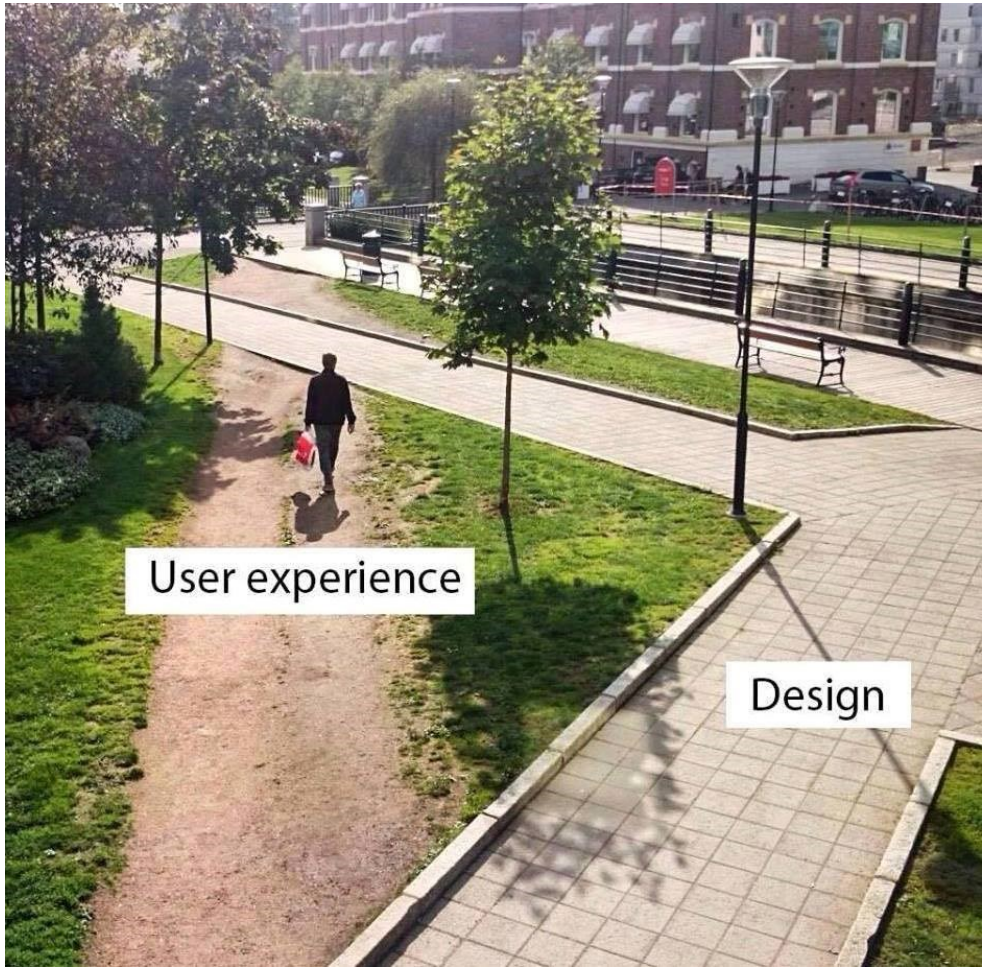






Ausflug in die Außenwelt der IT-Methodik-
Weil es eben ‚nicht alle‘ selbstverständlich kennen-

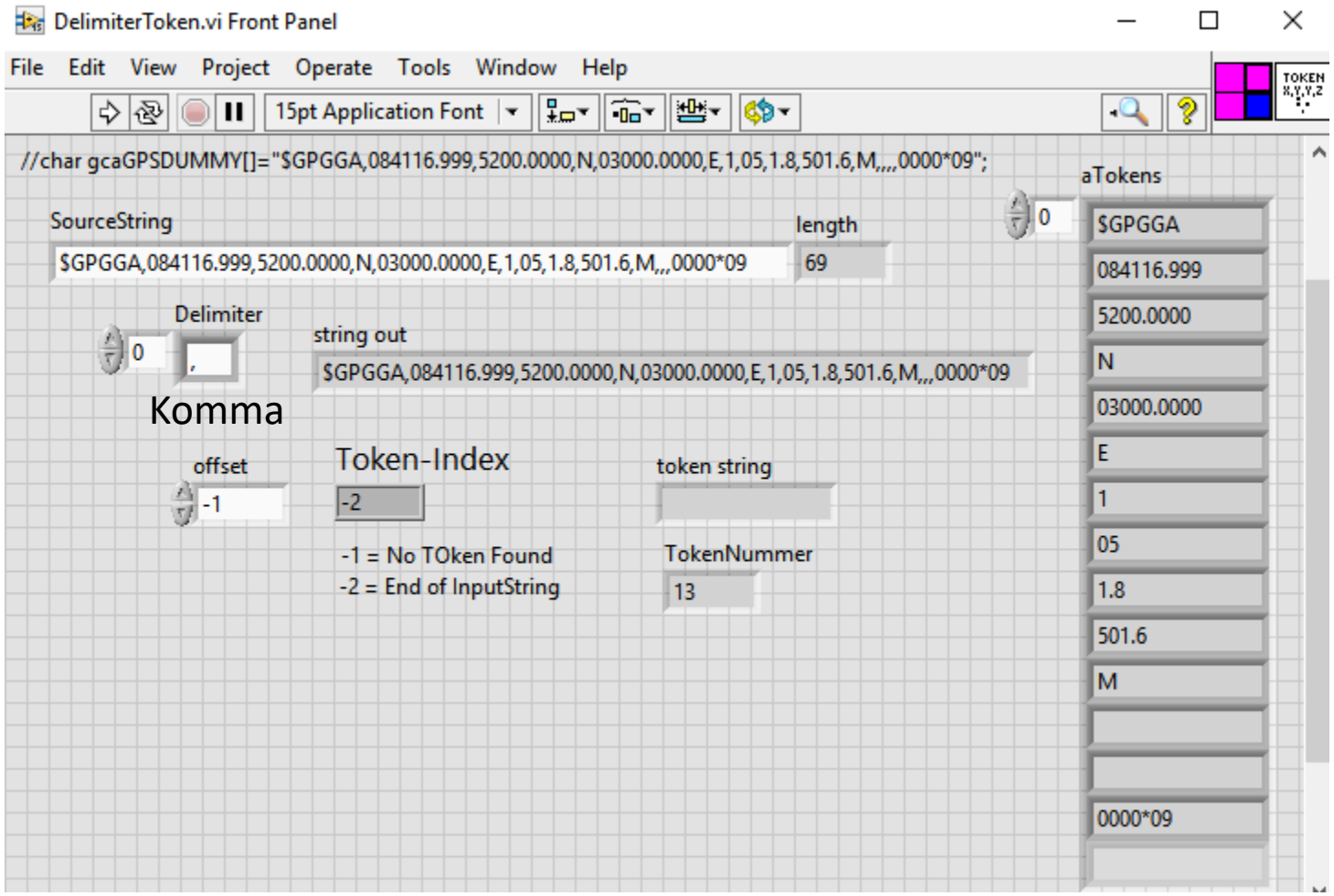
Es gibt keinen Ersatz für die eigene Erfahrung.



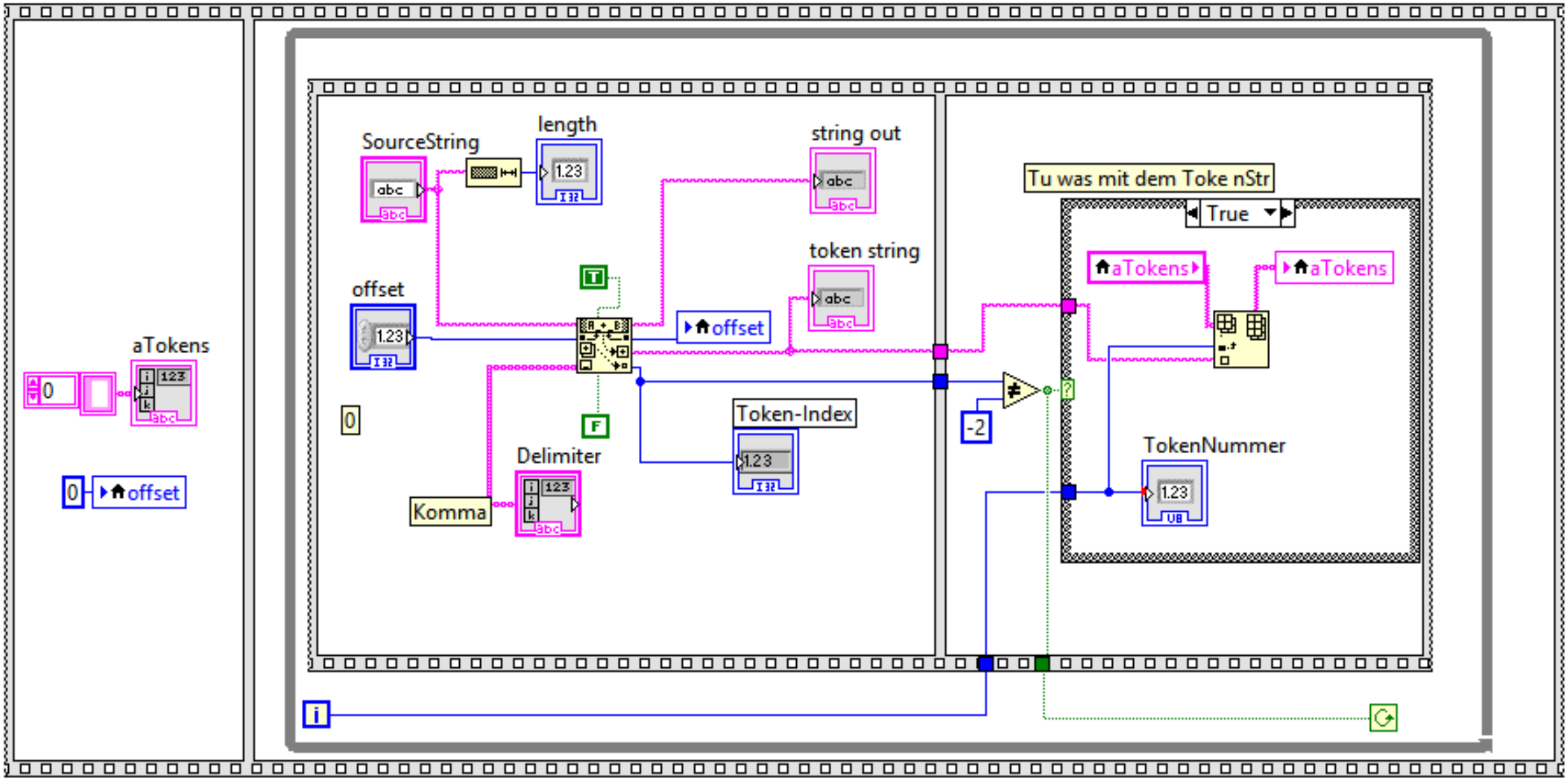
Übung: Stringverarbeitung, Bsp: **Delimiter Liste** Trennen. Die Elemente nennt man **Tokens**

So sieht es in Ansi-C aus: **„Delimiter Liste“ = Eine durch Kommas getrennte, textbasierte Datenliste**

char gcaGPSDUMMY[]=“$GPGGA,084116.999,5200.0000,N,03000.0000,E,1,05,1.8,501.6,M,,,,0000*09";



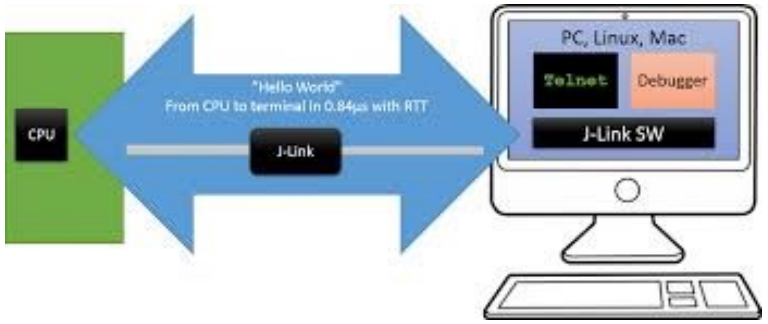
Trenne die durch (,)
Komma=*Delimiter* Elemente und lege diese in einem Array ab.
Hier könne diese weiterverarbeitet werden.
Hier ein Bsp. eines GPS Signals
→ $GPGGA *C*ode ……..


aTokens
0
0
offset
SourceString
length
1.23
string out
token string
offset
1.23
T
F
0
Delimiter
Komma
Token-Index
1.23
Tu was mit dem Toke nStr
True
aTokens
aTokens
-2
TokenNummer
1.23
i

…nicht die **Datenverarbeitung** ist das Problem,

sondern die Daten in den PC hinein

und wieder heraus zu bekommen.



Ein bewährtes Verfahren: **Ereignissteuerung**.

Einfacher **TEXT** ist eine universelle Sprache um die **unterschiedlichsten Geräte-Welten** wie PC (Win/Linux) und externe „Geräte aller Art“, wie Mikrokontroller **zu verbinden**.

Die Schriftzeichen sind in *uralten* **ASCII-Code Tabellen** definiert.

Es funktioniert nach einem **Frage – Antwort** Spiel. Praktisch!

Sende: **„PING<13>“** delay…..

Antwort: ← **„PONG<13>“**

(so etwas nennt man **asynchrone Kommunikation**)

Die <**13**> ist das **Carriage Return** Zeichen, kurz **CR. IN ASCII-Code Nr. 13**

Asynchrone Kommunikation

Auch „Gespräch“ genannt.
Irgendwas passiert irgendwann!
…ein „Gerät“ antwortet ungefragt, oder gibt nach einer unbekannten Zeit ein Antwort, ….oder auch nicht. Auf diese Fälle muss man reagieren bzw. es interpretieren.
Die Empfangsbereitschaft muss gut verwaltet werden.

Sender = Tx (Transmitter) --- Empfänger = Rx (Receiver)

Methode: **Master Slave**
Die beste Methode . Auf eine Frage, gibt es eine Antwort
„PING<13>“ (time ~30..100mS) → „PONG<13>“

Die Zahl **13** = **ASCII-Code „Carriage Return“** , zeigt das Ende eines Befehles an.
Der Anfang wird durch die Abwesenheit von „**NICHTS**“ definiert.
So wie wenn man in Natura (aus dem Hinterhalt) angesprochen wird.

Dann jedoch kommt die Antwort .
Wenn nicht, dann muss ein TIMEOUT Mechanismus für ein „WEITER GEHT ES“ sorgen.

ABLAUF . Sende (Tx) „PING<13>

Der Salve-Empfänger (Rx) hat die Zeichen empfangen:
P → I -→ N → G → 13
und sieht an der 13 , dass das Befehlswort vollständig ist.
DIESE 13 ist ein Vereinbarung, Es kann auch 10 = LF= Line Feed sein ..
Oder ein Zeichen „#“, oder was auch immer…

Das Gerät hat intern eingebaut, dass es auf PING mit PONG antwortet.
Es sendet (irgendwann in naher Zukunft (~<100mS):
P→O→N→G – und noch die →13.

WIR müssen also die „Emfpangsschnittstelle“ fragen: **(BytesOnPort)**
A: fragen ob Zeichen am Rx Empfang sind?
B: dann ein Zeichen davon lesen,
C: fragen ob dieses Zeichen ein „13“ ist
D: wenn nicht(13), dann das empfangene Zeichen an einen (Rx) Empfangsstring anhängen.
→ „Po“ wird zu „Pon“ und dann zu „Pong“
E: wenn **13(CR)** gekommen ist, dann ist alles OK, → Emfangsstrings auswerten……
F: wenn der Timeout (~> 1Sekunde) erfolgt, prüfen warum
… oder/aber weitermachen oder Meldung dazu ausgeben…..

Es gäbe sogar „echte **STX und ETX** Signale die im **ASCII-Code** reserviert sind.
STX = 2 (start of text)
ETX = 3 (end of text)

Aber um bequem mit **Terminal-Programmen** arbeiten zu können, die einen Wagenrücklauf als Signal kennen, nehmen wir die **13=CR**
„Ping<13>

{Zeitlicher Verlauf….}

1. P
2. Pi
3. Pin
4. Ping
5. Ping**<13>**

USASCII code chart

b_7 b_6 b_5 → / Bits ↓				0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4	b_3	b_2	b_1	Column → / Row ↓ 0	1	2	3	4	5	6	7
0	0	0	0	0 NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1 SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2 STX	DC2	"	2	B	R	b	r
0	0	1	1	3 ETX	DC3	#	3	C	S	c	s
0	1	0	0	4 EOT	DC4	$	4	D	T	d	t
0	1	0	1	5 ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6 ACK	SYN	&	6	F	V	f	v
0	1	1	1	7 BEL	ETB	'	7	G	W	g	w
1	0	0	0	8 BS	CAN	(	8	H	X	h	x
1	0	0	1	9 HT	EM	)	9	I	Y	i	y
1	0	1	0	10 LF	SUB	*	:	J	Z	j	z
1	0	1	1	11 VT	ESC	+	;	K	[	k	{
1	1	0	0	12 FF	FS	,	<	L	\	l	|
1	1	0	1	13 CR	GS	-	=	M	]	m	}
1	1	1	0	14 SO	RS	.	>	N	^	n	~
1	1	1	1	15 SI	US	/	?	O	_	o	DEL

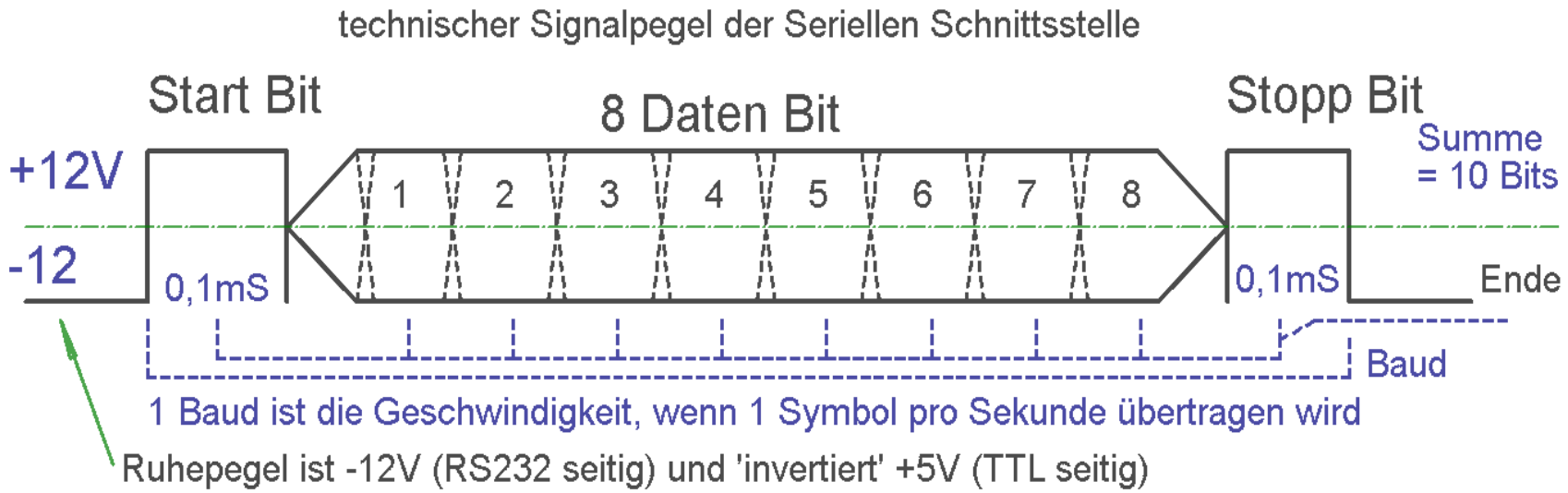
Wie bekommen wir ‚verständlich lesbare' und sonstige Daten aus LV hinaus und zu LV zurück?

Die Schnittstellen-Techniken sind vielfältig:
Spannungspegel gesteuert, Strompegel gesteuert, parallel, seriell……
Doch Historisch hat sich die alte **Serielle RS232** Übertragung bewährt. COM Schnittstelle genannt.
Signaltechnik: Der logische **TTL** Pegel wird invertiert

TTL 0V= +12V
TTL 5V = -12V

BAUD = 9600..**115200**, Kein Handshake, keine Parität, Kein Xon/Xoff

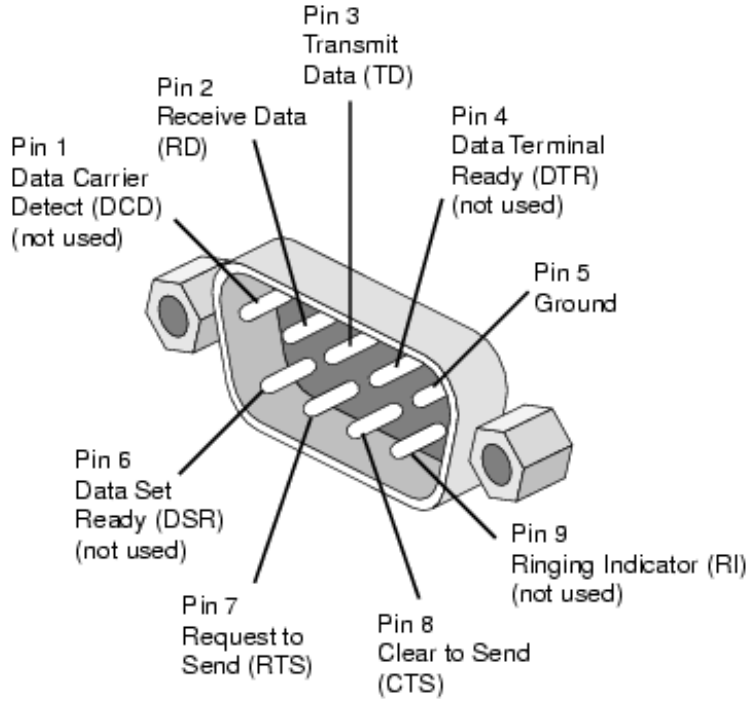
COM to USB
Converter sind heute jedoch üblich.
Dahinter steckt jedoch diese Methode.



Die serielle Schnittstelle

→ hier die PC Stecker Seite:

seltener Geworden, aber immer noch wichtig und notwendig zu verstehen.
Wenn auch nur als virtuelle serielle **COM-Schnittstelle**, meist mit dem FTDI232 Baustein, oder den weniger geliebten chinesischen Klonen wie **PL2303_Prolific**.



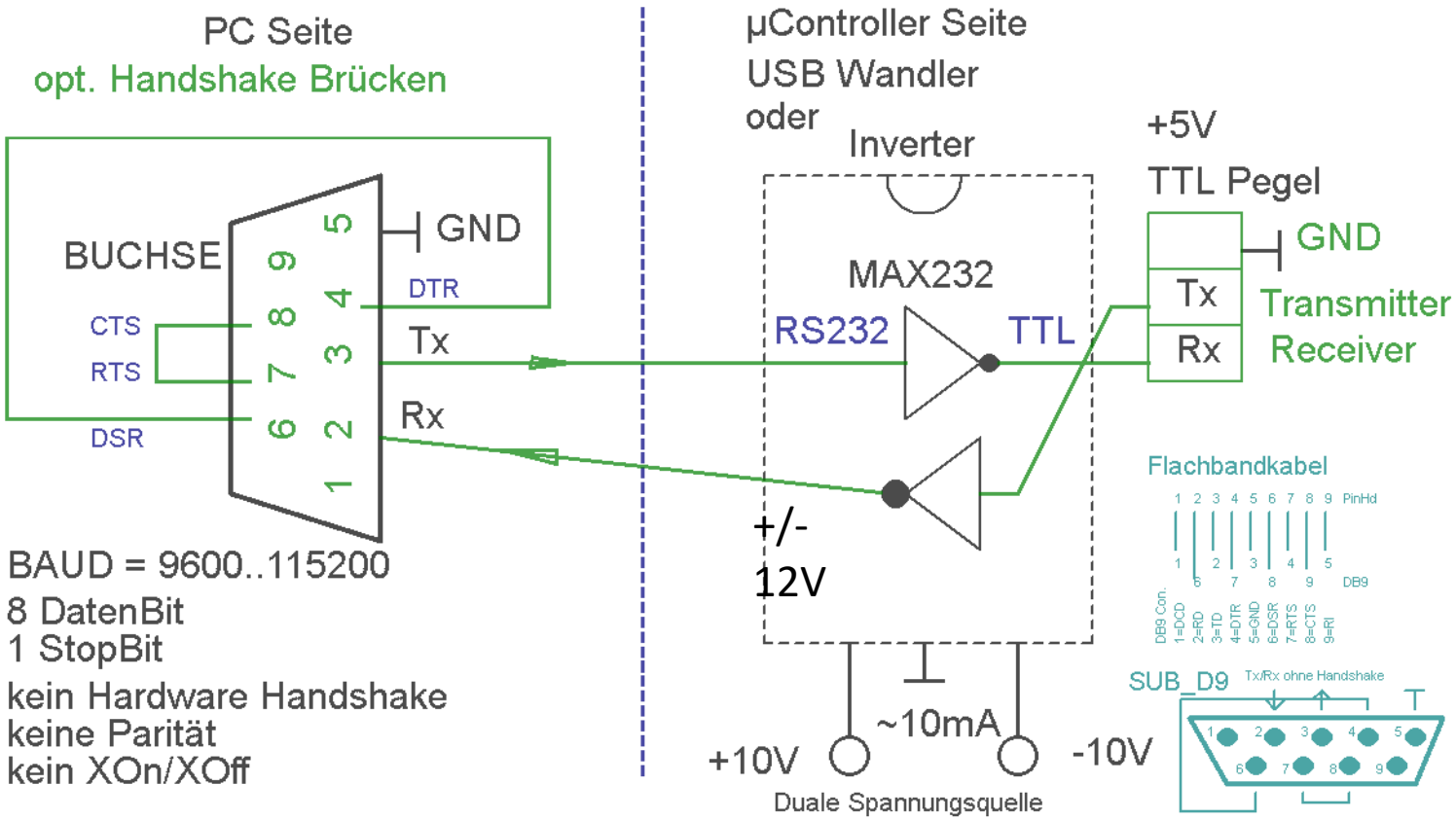
Wir brauchen meist nur
Pin 3 Tx = Transmit Data
Pin 2 Rx = Receive Data
Pin 5 GND = 0V

Achtung:
Pegelwandlung und Invertierung
zu TTL erforderlich

Selten muss man die
Handshake Leitungen brücken.
DSR-DTR
CTS-RTS

So sieht/sah es Hardwaretechnisch aus

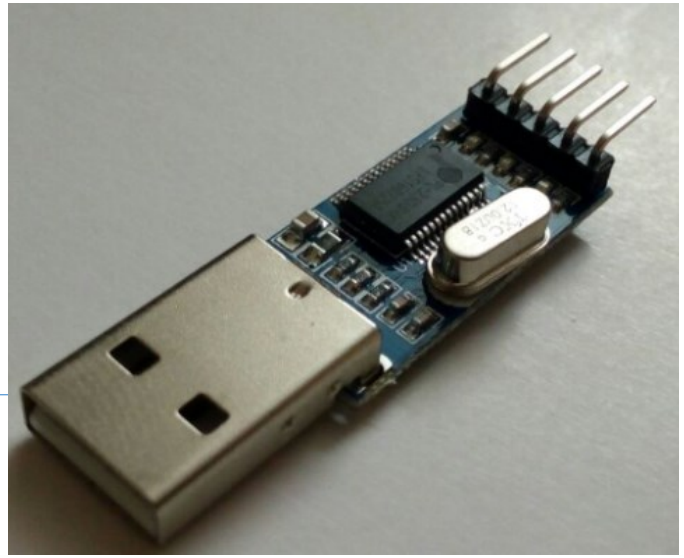
UART - TTL Signal → RS232



Heute (2020) hat man fast nur noch USB Simulationen von „Seriellen Schnittstellen“.
Bekannt sind *die*

USB TTL Wandler:

FTDI 232 (sehr gut und teuer)
CH340 (Shina)
Prolific PL2303 (China).



Merke:	TTL	RS232 (=invertiert)	USB
Logisch 0 =	5V	-12V	nicht direkt messbar.
Logisch 1 =	0 V	+12V	

Wir brauchen nur …..
Begriffe und Parameter für die Serielle COM Schnittstelle kennen…..
Es muss meist ‚nur‘ folgendes bekannt sein:

<u>Typische Baudraten:</u>

BAUDrate:

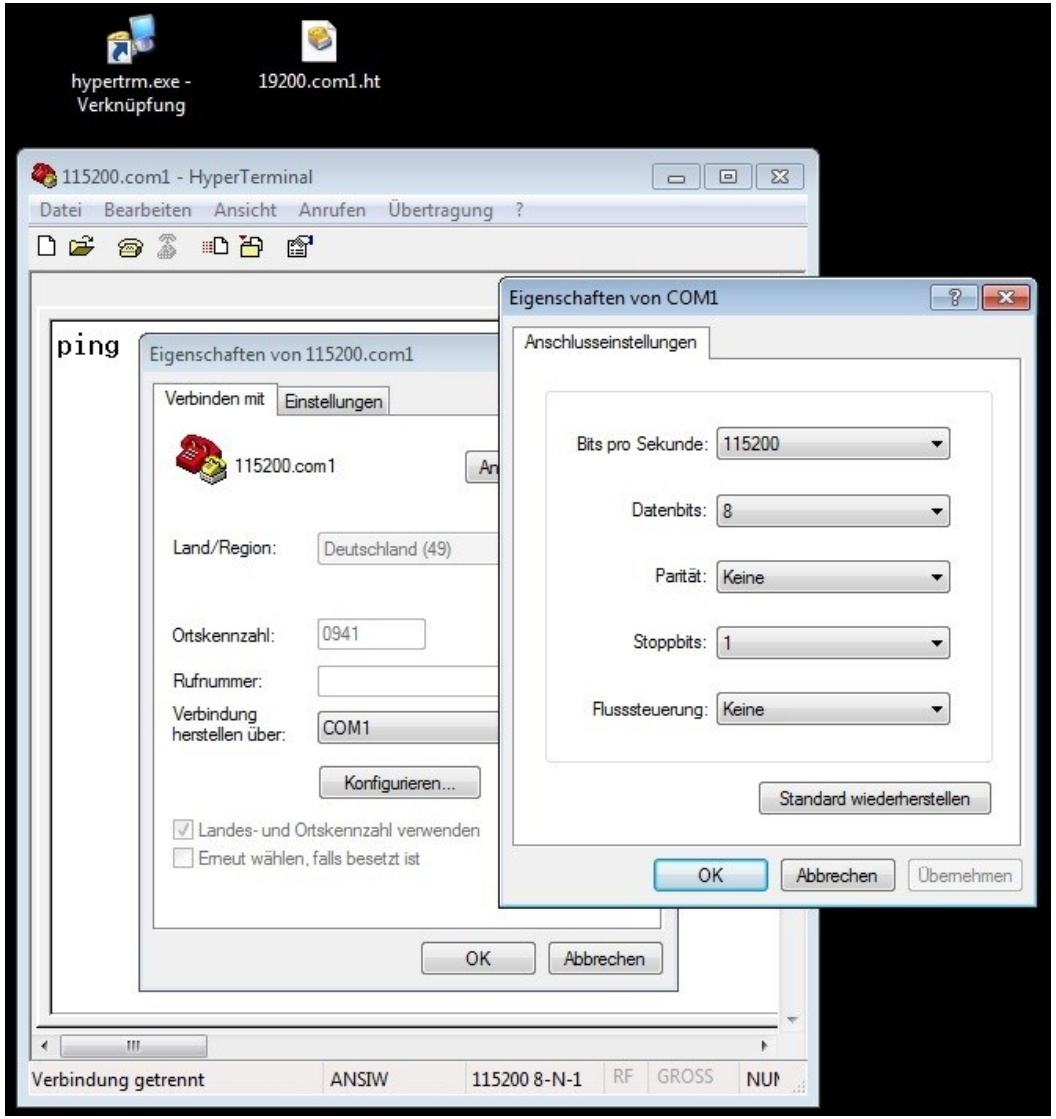
9600..
19200
57600
115200

Bitzahl: meist **8**

Stoppbits: meist **1**

Flussteuerung: **XON/XOFF →keine**
Handshake: **→ keiner**

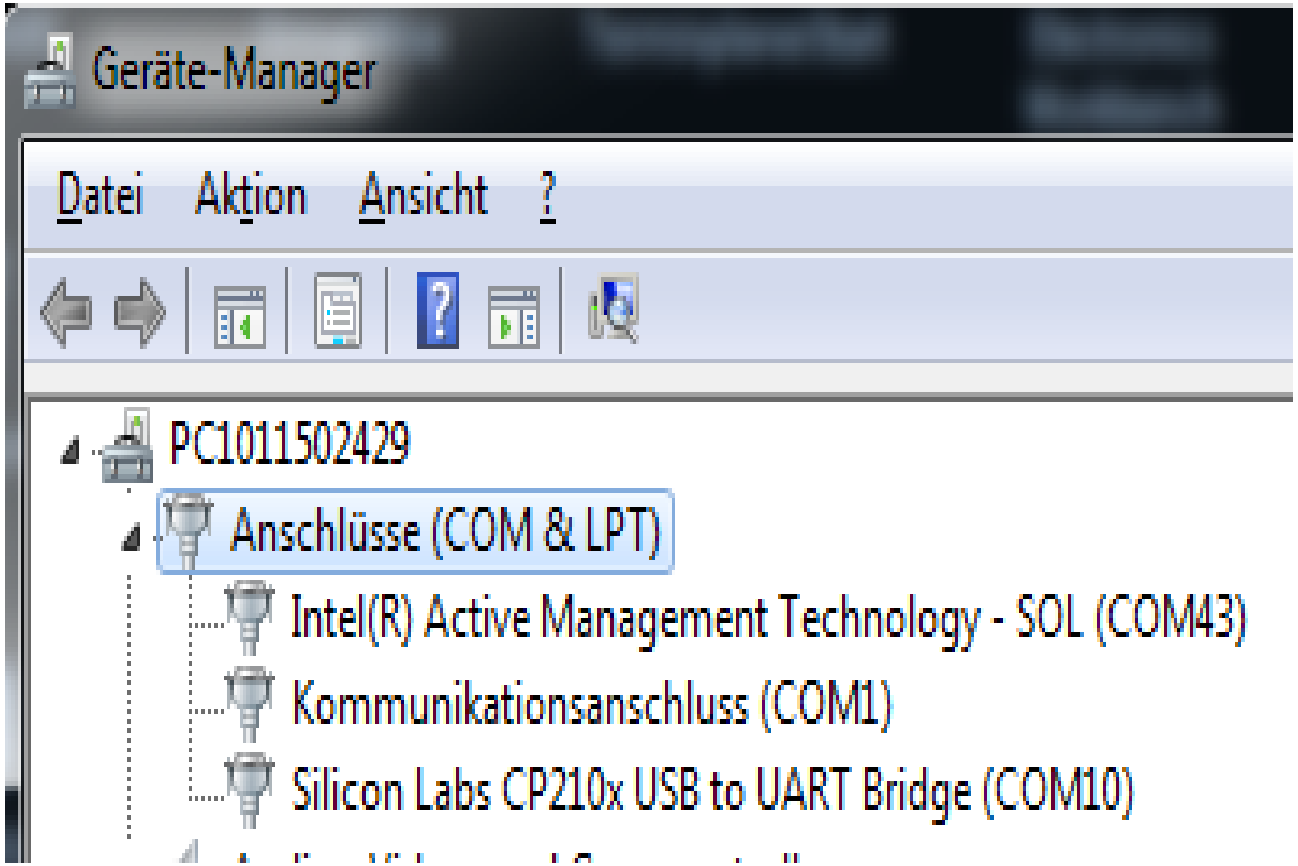
PRAXIS: XP-Hyper Terminal zum Text Tx-Rx über die Serielle Schnittstelle (UART)



Neuere Windows Versionen haben kein Terminalprogramm mehr „Onboard“. Aber es gibt viel Freeware und das alte **XP Hyperterminal** geht (noch) recht gut.
Nach etwas Gewöhnung und Starteinstellungen läuft es prima.
Dazu kann man mehrere Voreinstellungs-Dateien vorbereiten.
Bsp.: **115200.com.ht**
Parameter:

BAUD:	**9600..115200**
Datenbits:	**8**
Parität:	Keine
Stopbits:	**1**
Flusssteuerung:	**Keine**

PRAXIS: Datenübertragung aus LV am Beispiel COM = serielle Schnittstelle.

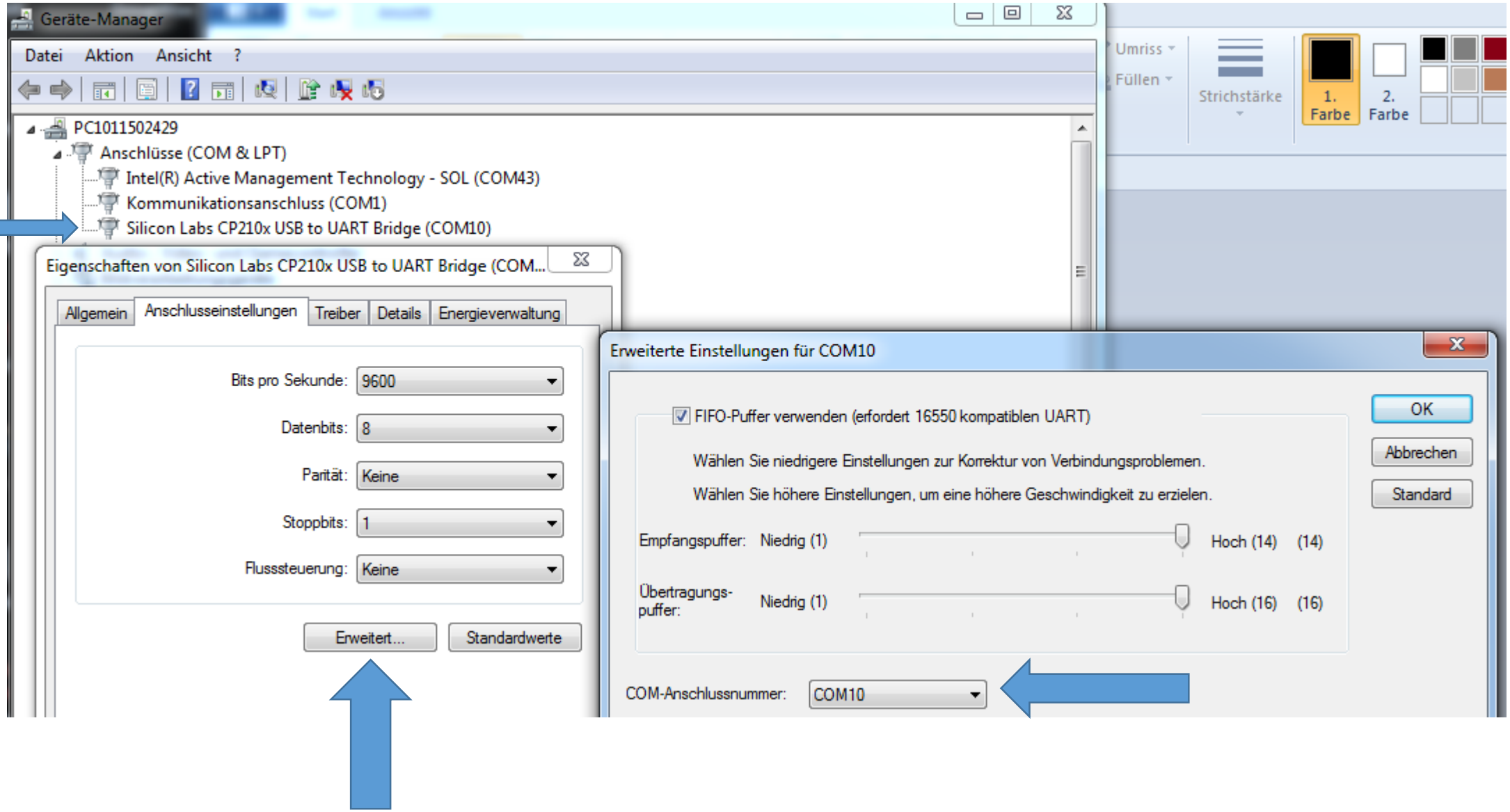


Die Schnittstellen finden sich mit
FENSTER +PAUSE →Taste
→ Gerätemanager
→ Anschlüsse (COM & LPT)
→ Darin ´finden sich die Nummern der vorhandenen seriellen COM-Anschlüsse. Diese können wie folgt geändert werden :

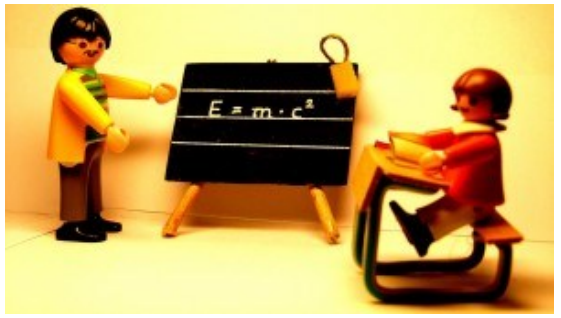
Lästig ist das rausfinden der richten COM-Nummer zum öffnen der Schnittstelle via

Ändern und Umbenennen absurd hoher COM-Nummern.. Doppelbelegungen jedoch genauestens kontrollieren.

R.C.



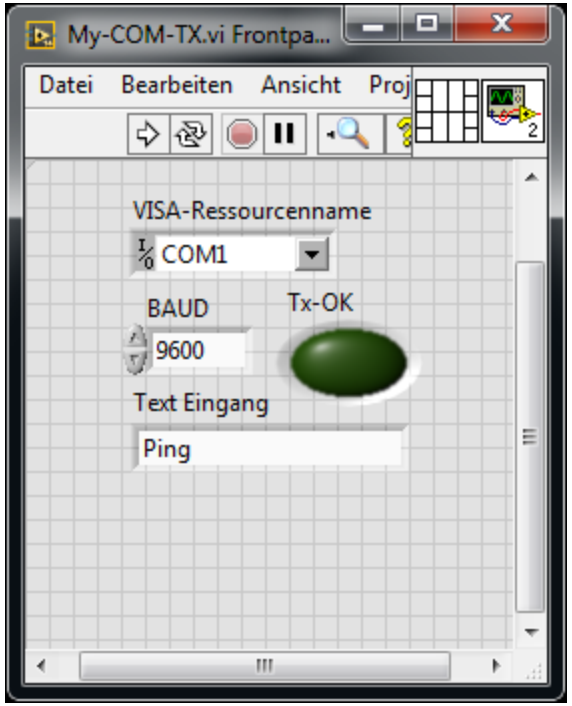
Zurück zu LabVIEW…..
Senden & Empfangen von Daten

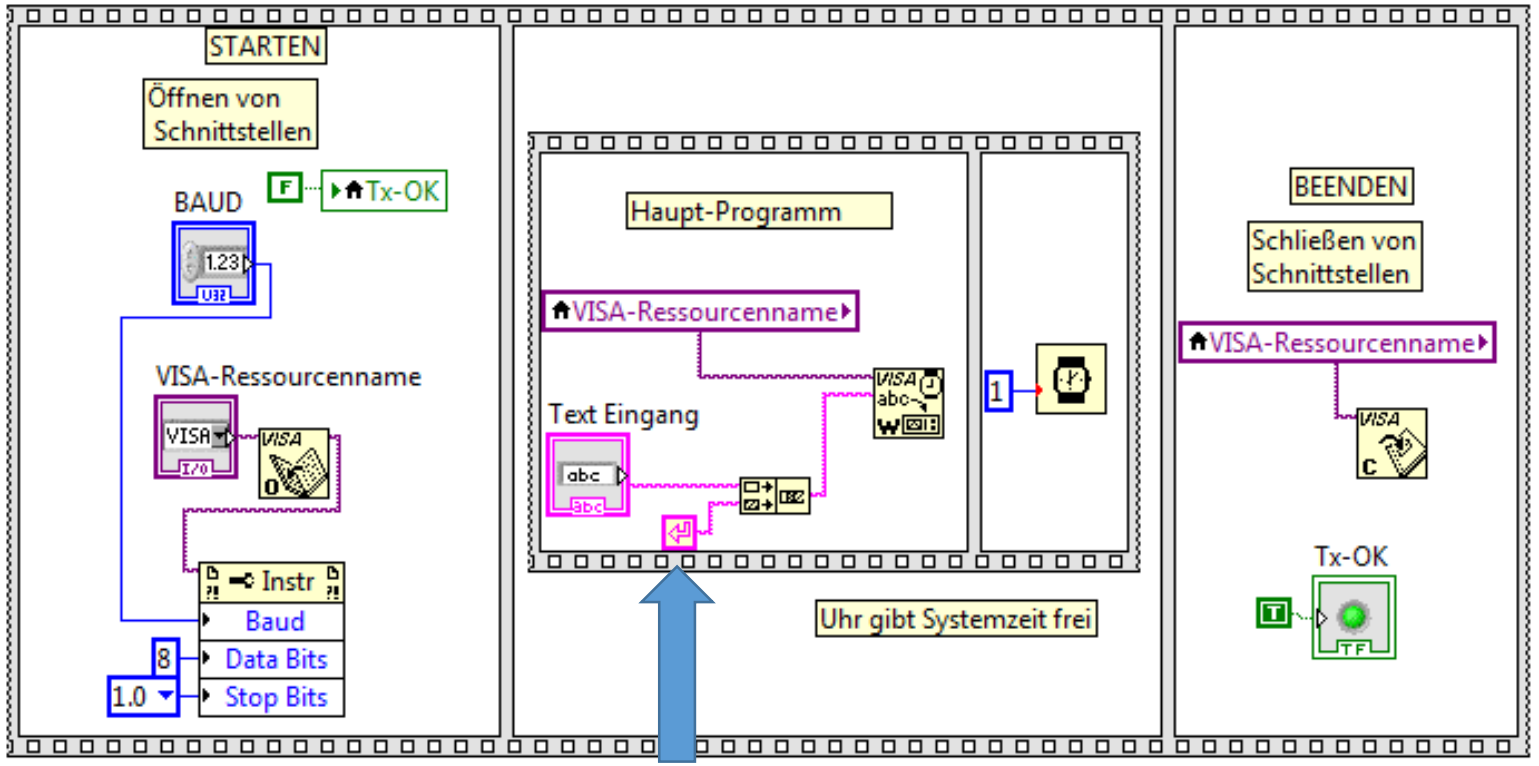


Es gleicht einer Frage-Antwort Situation. Man nennt dies ***Master-SLAVE***

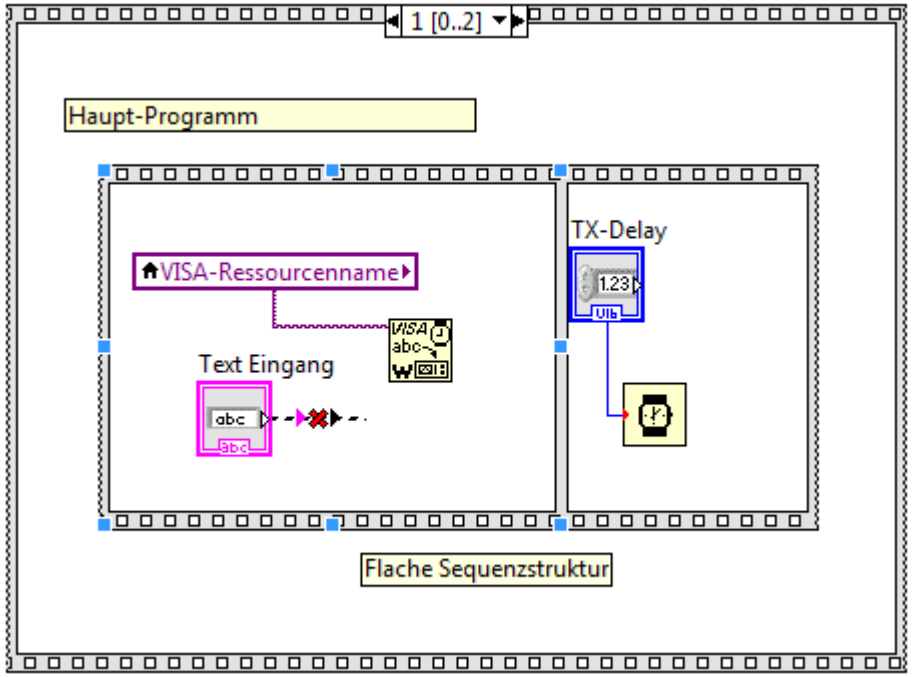
Senden von Text haben wir schon kennengelernt.
Es fehlt noch: „Antwort empfangen.“ … Und, Kommandos enden mit „***Carriage Return***“ = Ascii **13**

CR zu Verwenden ist jedoch nur eine Vereinbarung.

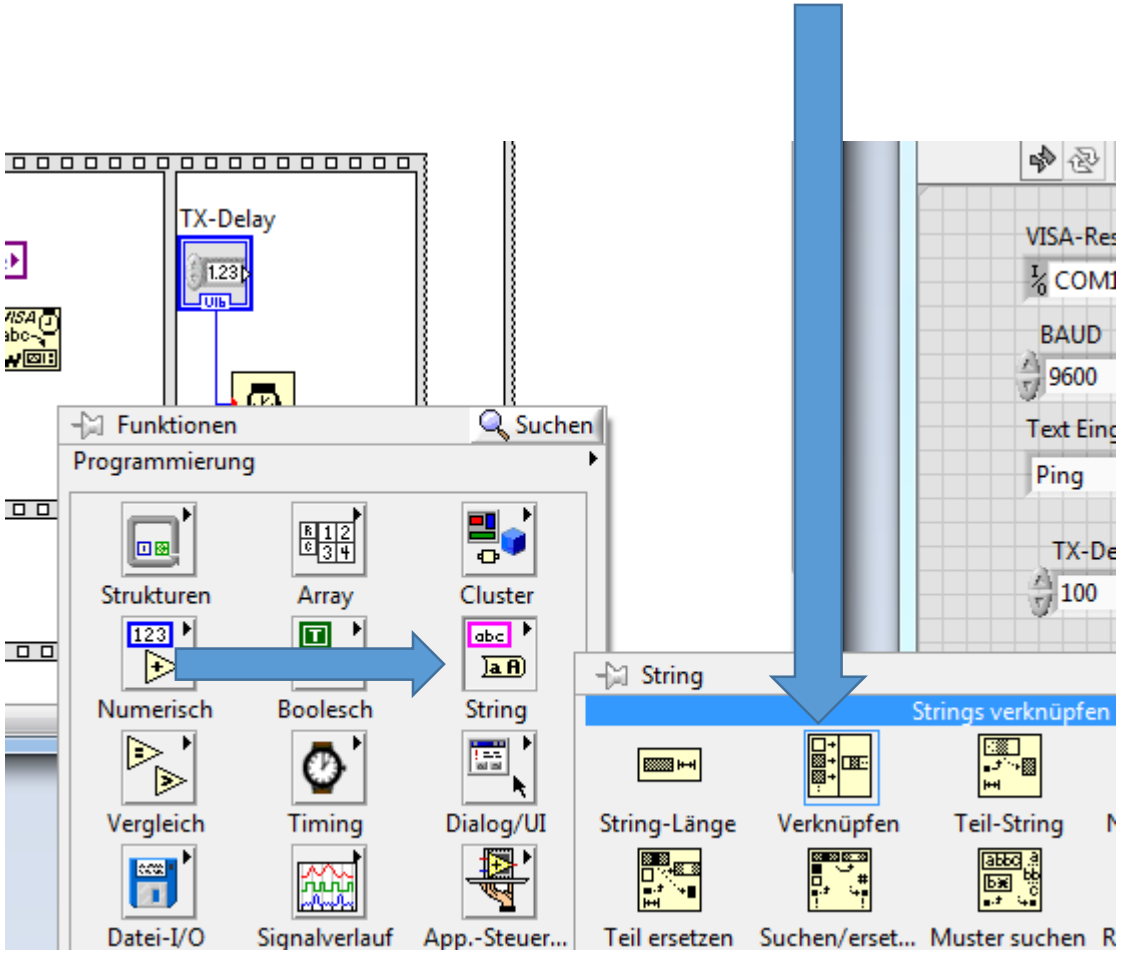




Jetzt noch **CR** anhängen

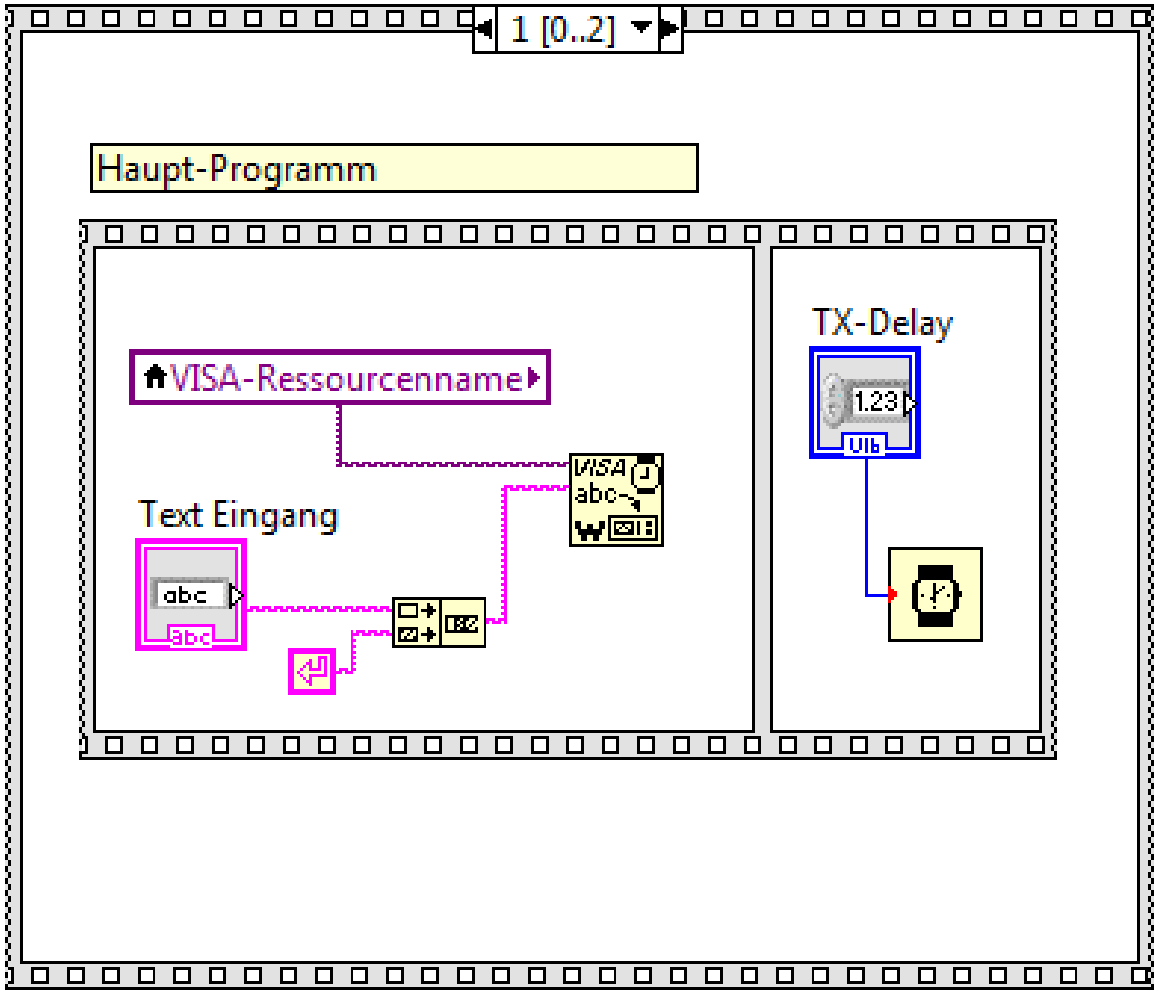


Vorbereitenden Arbeit: „Verbindung Text-Feld und VISA-Write lösen.
Den REST der „zerbrochenen Verbindung“ löschen mit **STRG+B.**



...dann, **RC** (right click) auf den Hintergrund

→ Funktionen
→ String
→ Verknüpfung auswählen und ablegen
→ Nochmals zu String
→ CR = Wagenrücklaufzeichen auswählen und ablegen.
→ Verbindungen herstellen (Nähen)

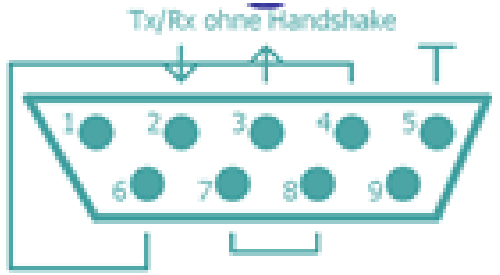


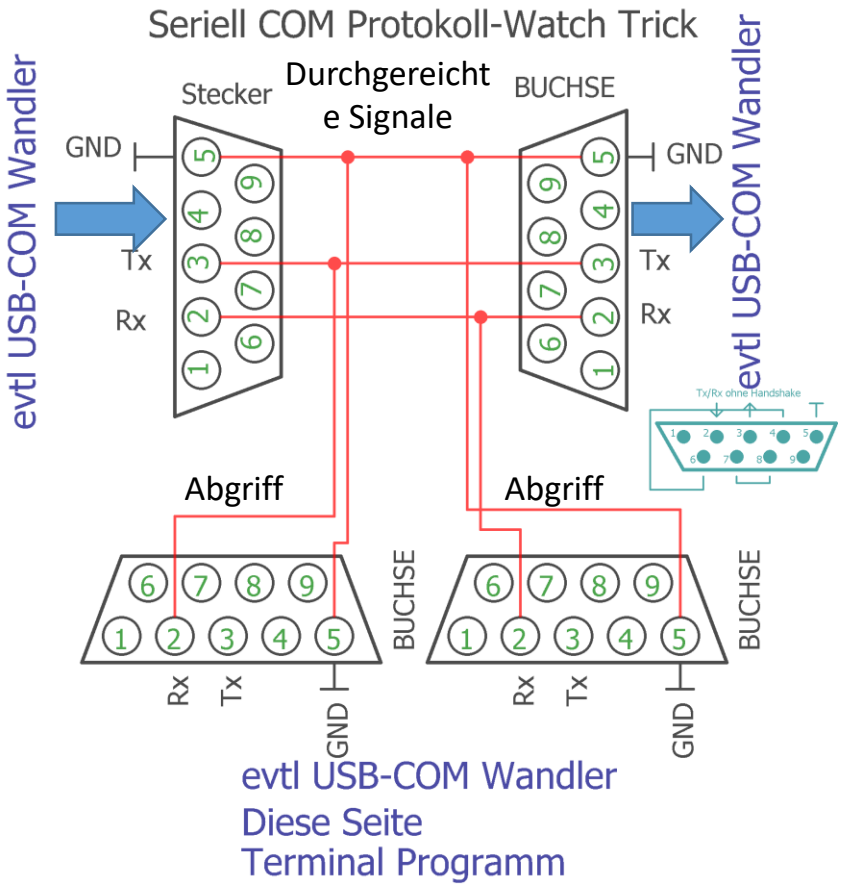
Fertig: Wir können jetzt senden.
Transmit = Tx-Data genannt.
Hier lohnt sich eine eigene **VI**.

Wir haben bis jetzt keine Ausführungs-Kontrolle. !

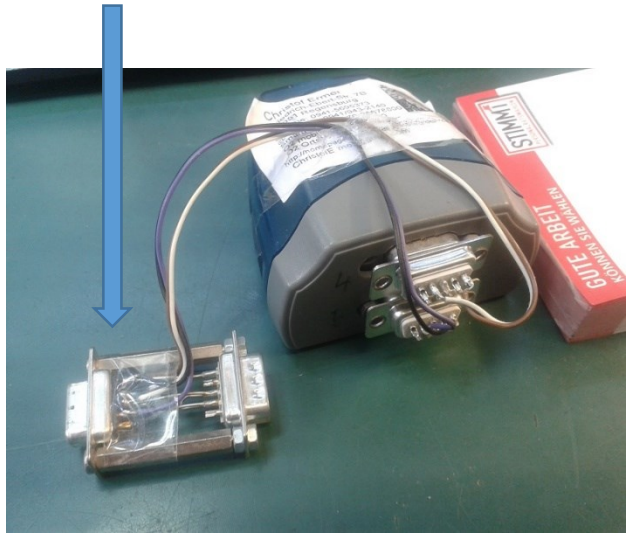
Zum Testen kann man den COM-Ausgang an der **SUBD-9** Stecker des PC oder USB Wandlers abgreifen und mit einem Adapter in ein Terminalprogramm einer weiteren **COM SUBD-9** Stecker zurückleiten.

Siehe Bild..





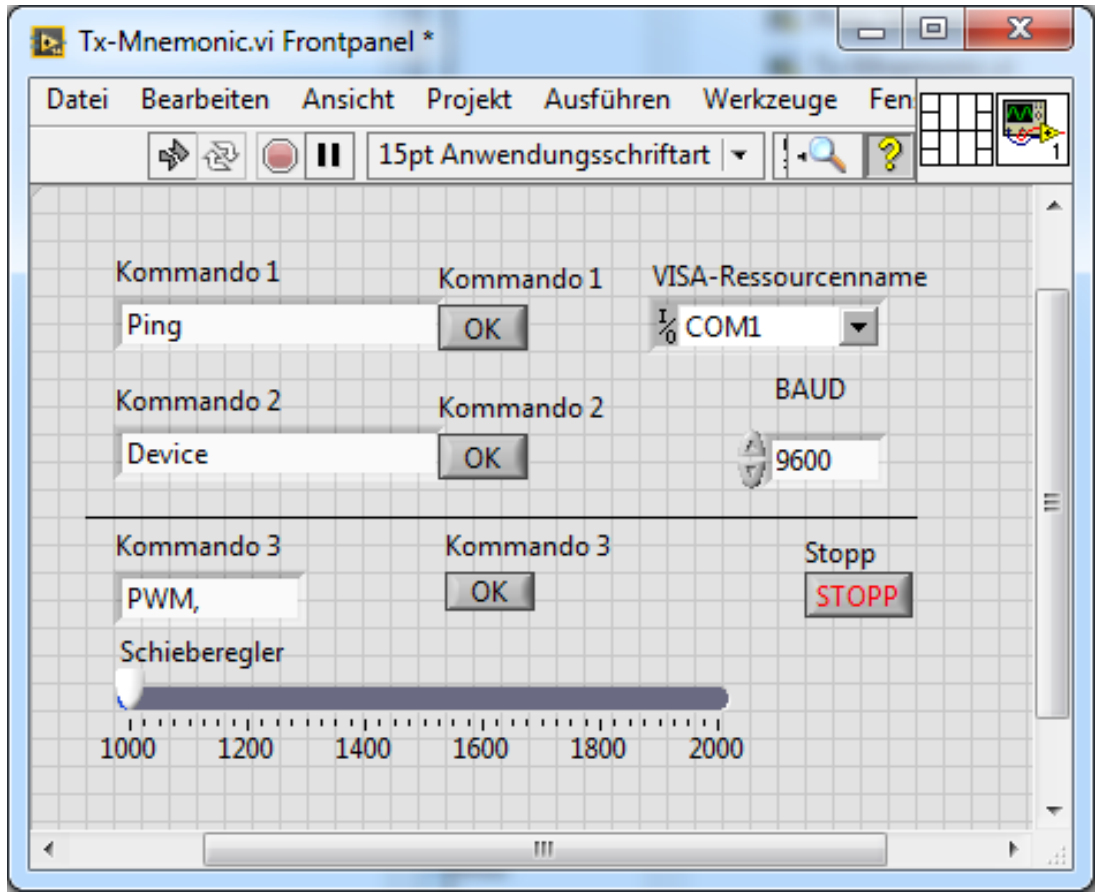
Debug Com Watch Trick. Billig und brauchbar.
Die seriellen 3-Leitungen Tx/Rx/Gnd mit Buchse-Stecker Kombination durchschleifen...
Dann die Tx, Rx, Signale auf zwei SUBD-9 Buchsen Pin-2 und GND Pin-5 leiten.
Mit w Terminal Programmen, wie Putty, mitlesen was gesendet wird

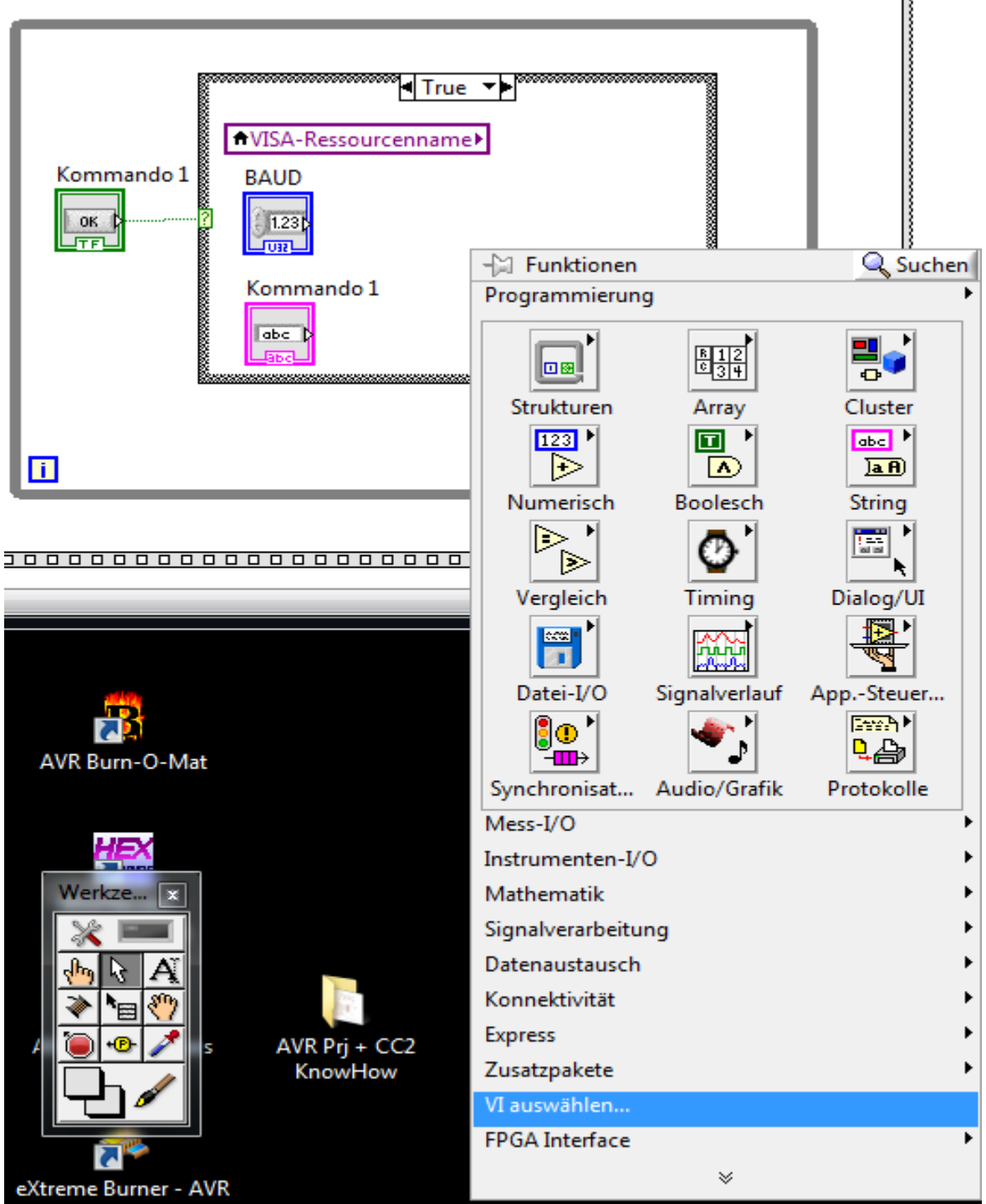


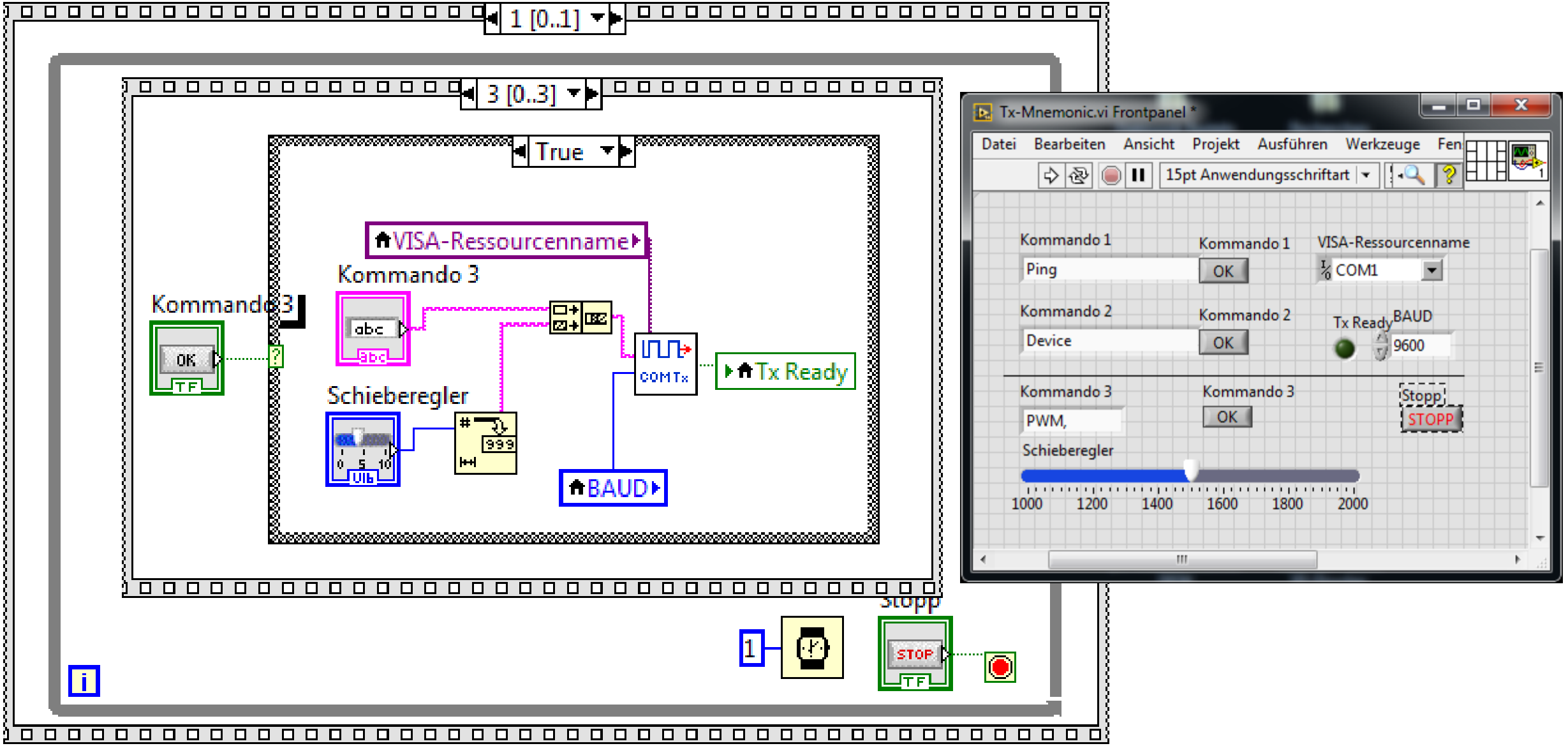
Anwendung der Sende bzw. Transmit Data = (Tx) VI

Erstelle eine Anwendung

Integriere das eigene Tx VI






1 [0..1]
3 [0..3]
True
VISA-Ressourcenname
Kommando 3
Kommando 3
OK
TF
abc
Schieberegler
U16
COMTx
Tx Ready
BAUD
1
STOP
TF
Stopp
Tx-Mnemonic.vi Frontpanel *
Datei
Bearbeiten
Ansicht
Projekt
Ausführen
Werkzeuge
15pt Anwendungsschriftart
Kommando 1
Ping
OK
Kommando 1
VISA-Ressourcenname
COM1
Kommando 2
Device
OK
Kommando 2
Tx Ready
BAUD
9600
Kommando 3
PWM,
OK
Kommando 3
Stopp
STOPP
Schieberegler
1000
1200
1400
1600
1800
2000

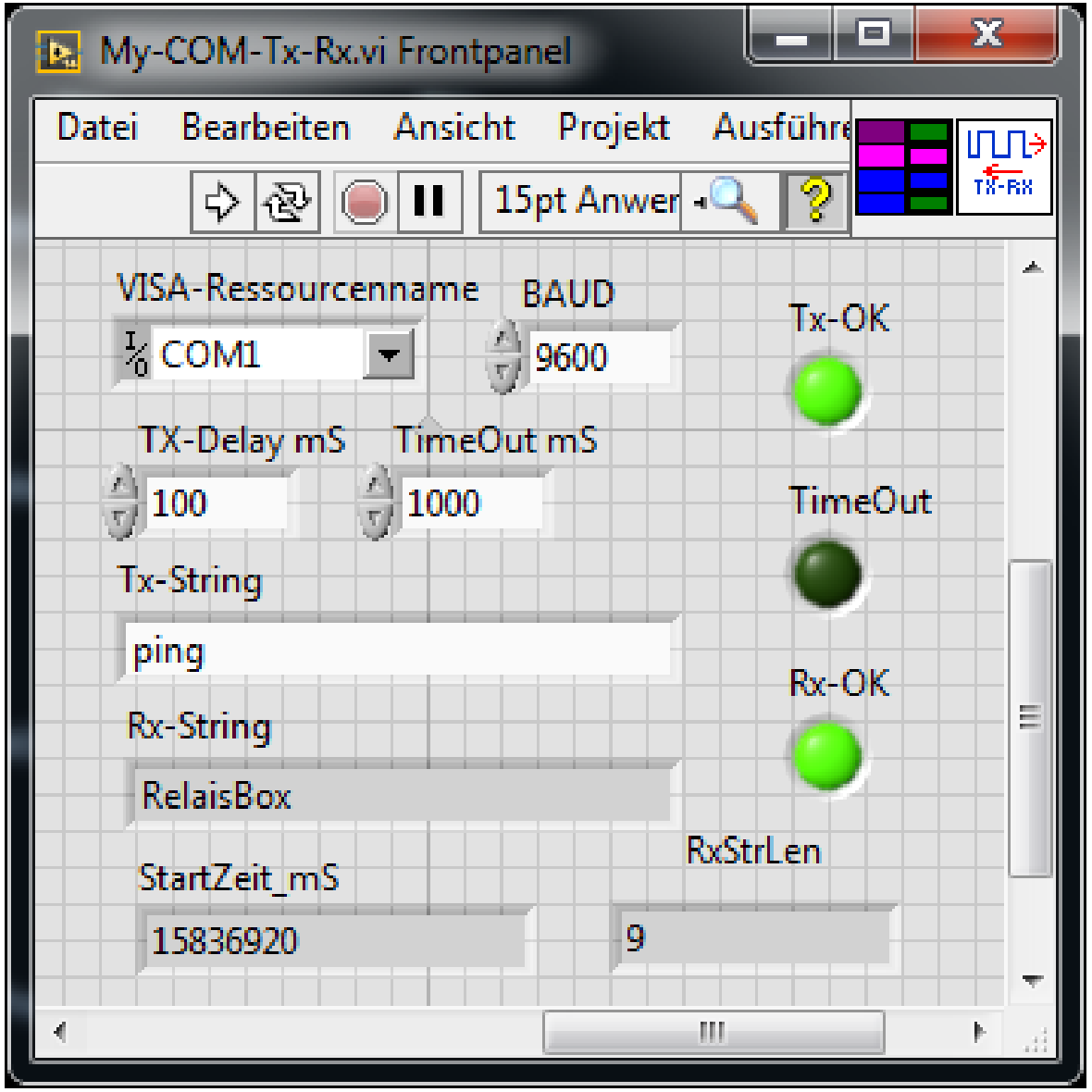
Asynchrone Ereignisse

Daten kommt zu beliebigen Zeiten an der seriellen Schnittstelle **(UART)** an.
Diese Zeichen werden der Reihe nach eingesammelt und der Reihe nach in einem **Rx-Buffer** ablegt.

Dazu gibt es eine Vereinbarung:
Dies geschieht so lange, bis ein Wagenrücklaufzeichen „Carriage Return“ **CR → 13** als Zeichen empfangen wird.

Dieses **CR/13** wird als „End-Of-Text Signal“ verwendet. **→ETX**
Wir haben aber kein „Start-Of-Text“ Signal → **STX** ??
Nun, das geht deshalb meist gut, weil man ja weiß, dass nach einem Ende ein Start kommen muss.

Jetzt brauchen wir eine Empfangsroutine Rx.
→Receive Data bzw. kombiniert Tx→ Rx

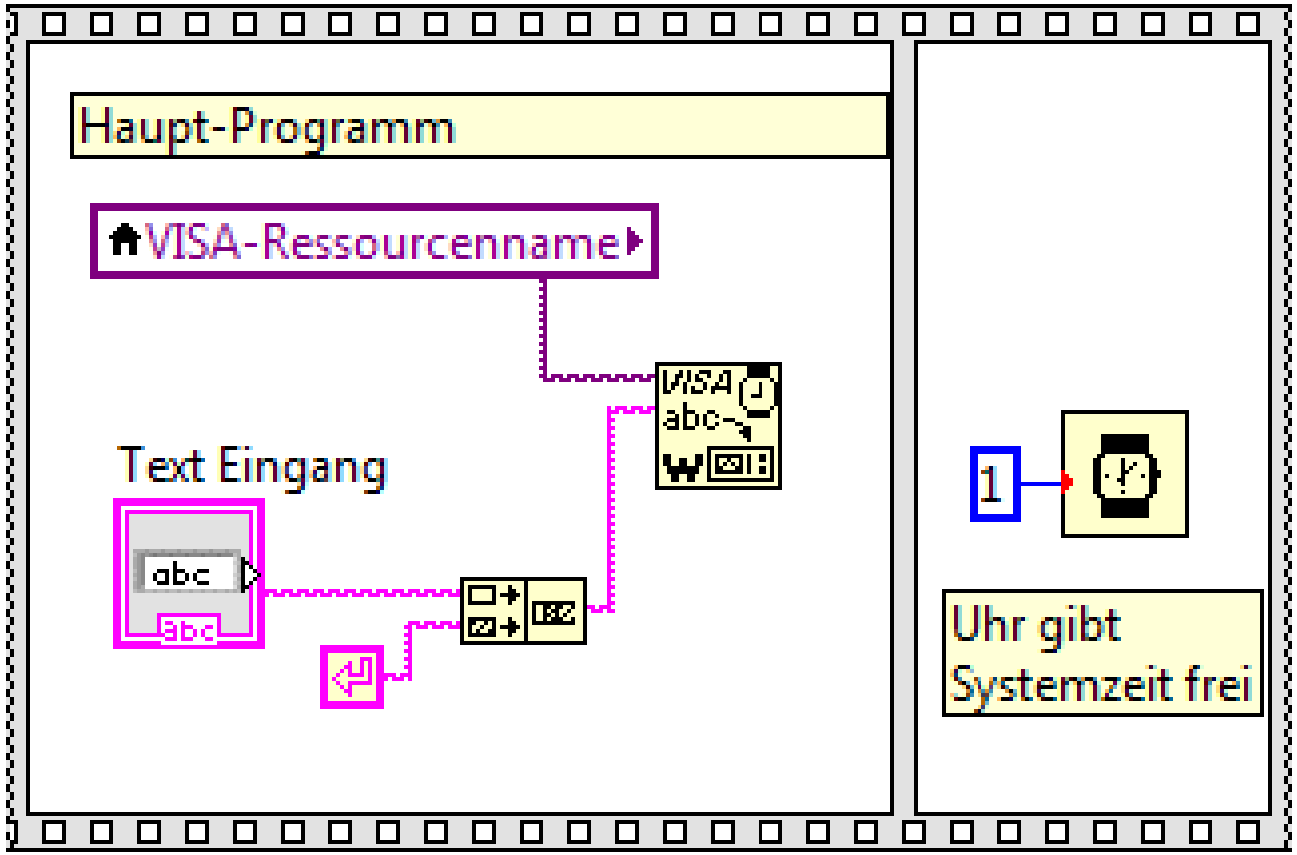


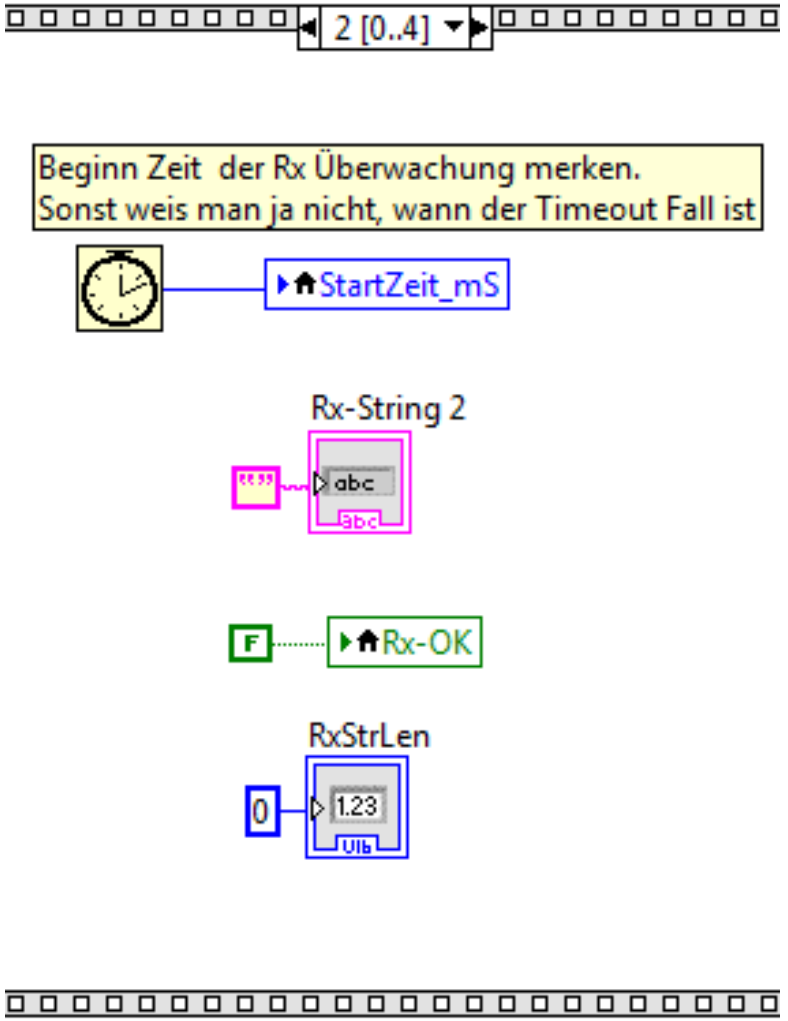
Da ja eine Antwort erst nach der Frage langsam eintröpfelt…, aber in naher Zukunft, muss das Programm sich die Startzeit merken um einen evtl. TIME-OUT generieren zu können.

Die hereinkommenden Zeichen werden gesammelt (CONCAT) bis das erwartete ENDE-Zeichen 1**3-CR** kommt.

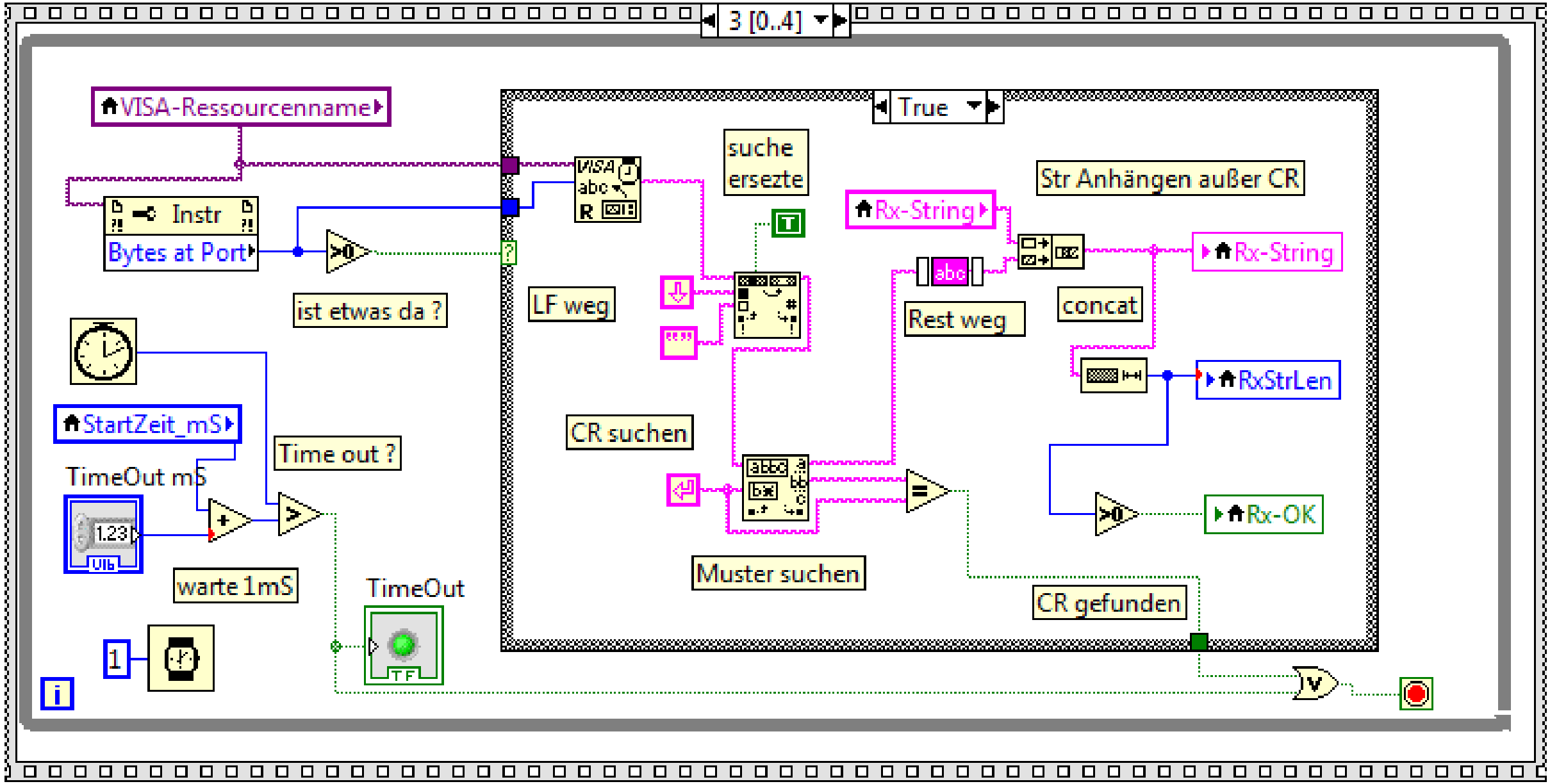
Rückgabe in Rx-String
mit Ok Flag oder mit Time-out

Bevor Empfangen werden kann, muss erst gesendet werden, …dann: Vorbereitend zum Empfang (Rx) Zeit merken sowie Inhaltsfelder und Flags löschen

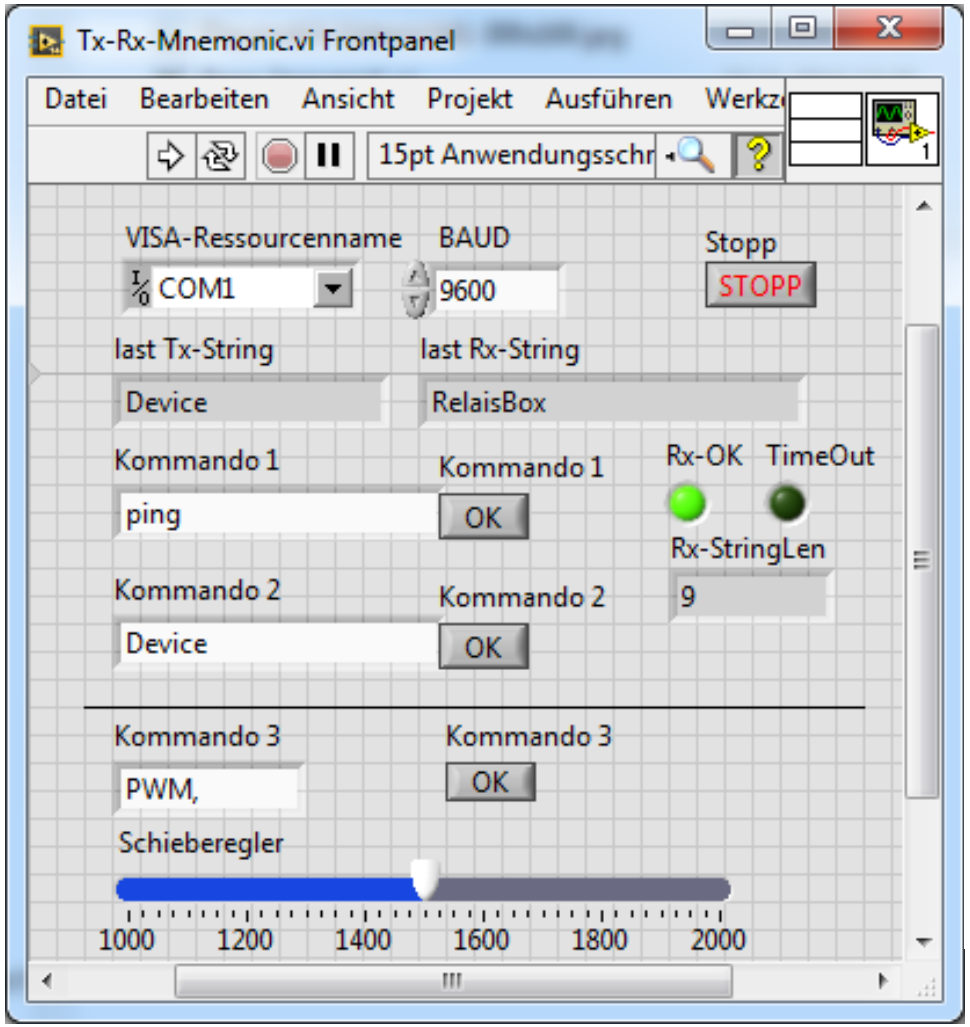


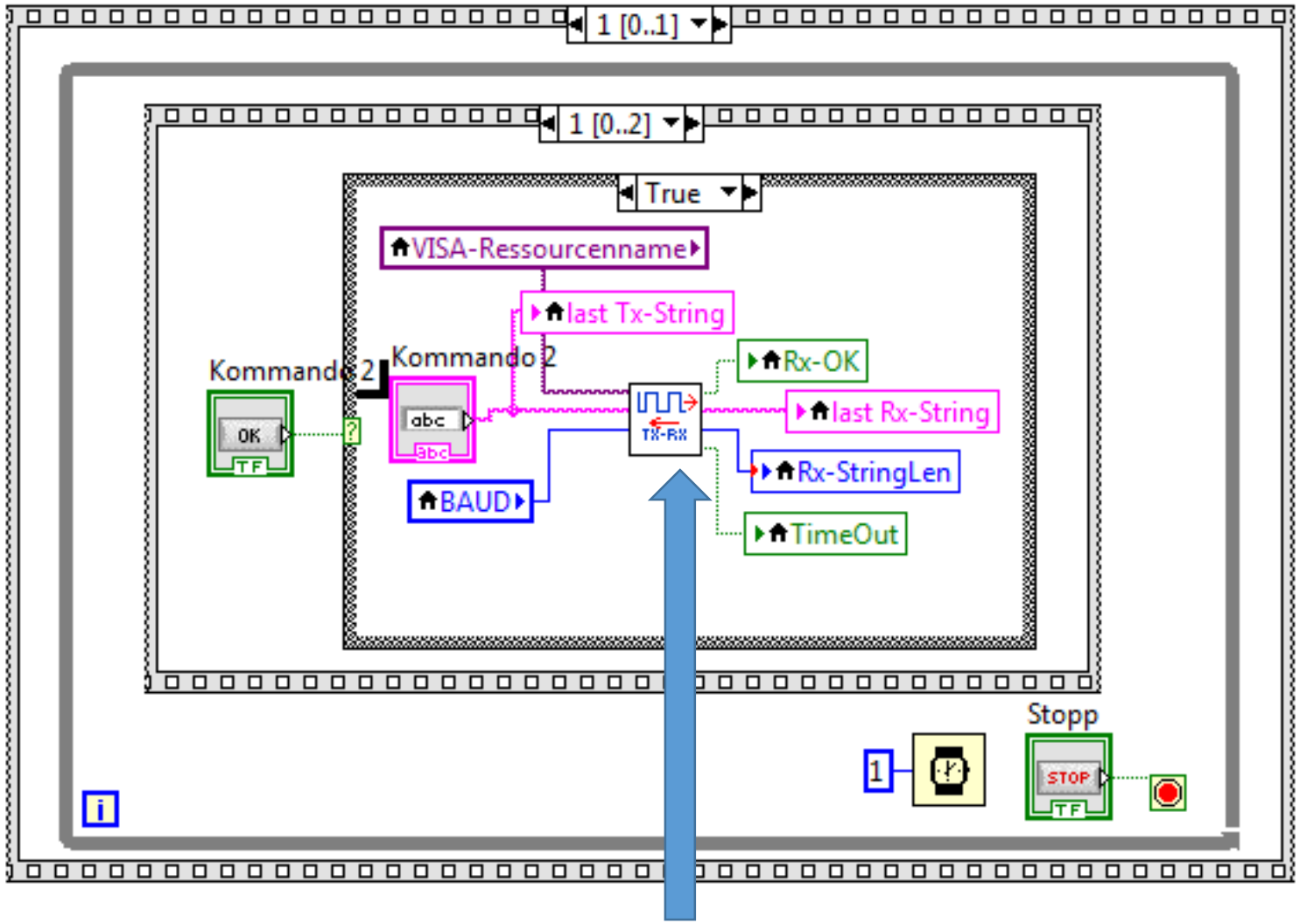


Asynchrones Einsammeln der zu empfangenden Zeichen bis zu CR = ASCII 13

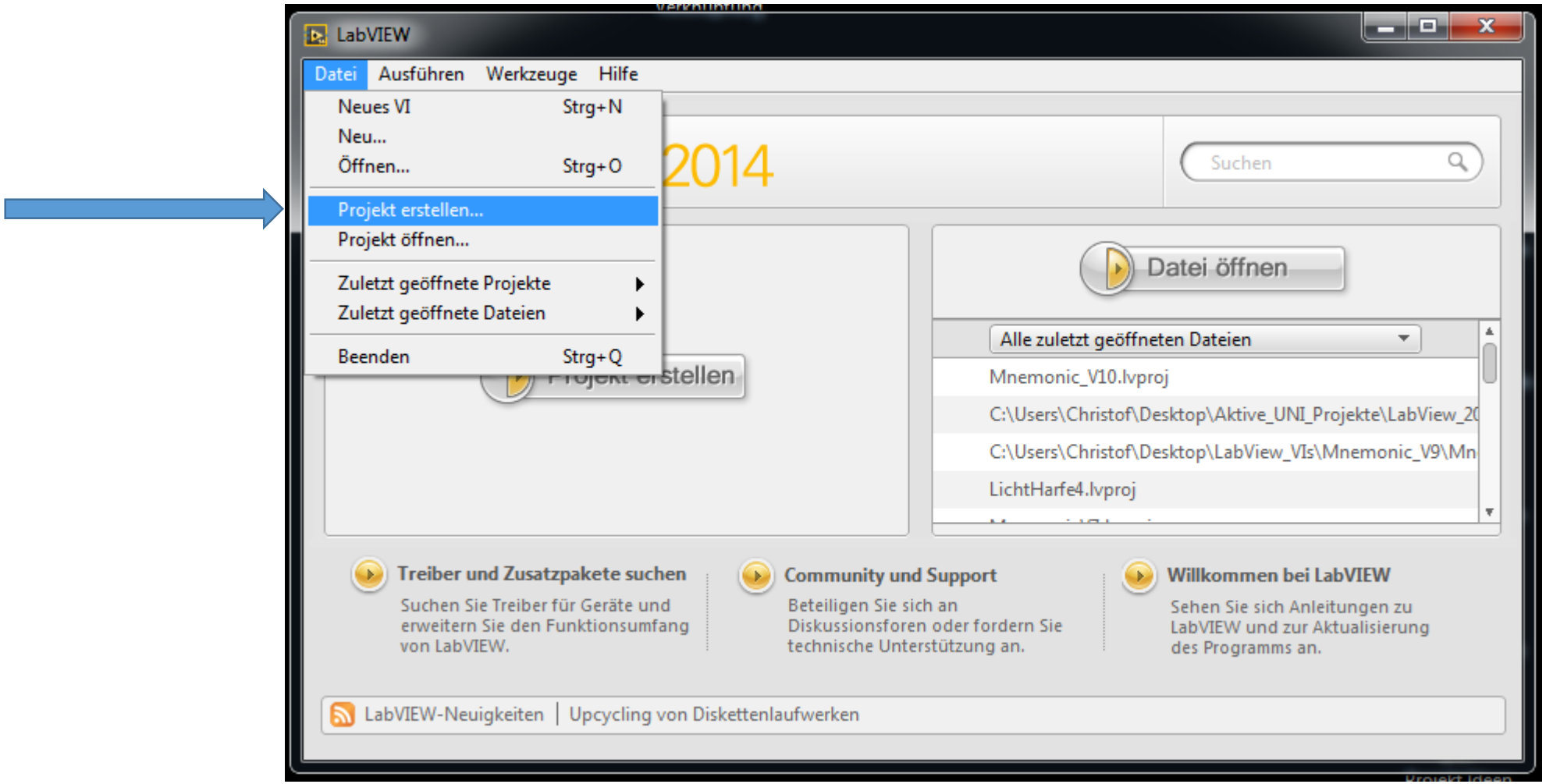


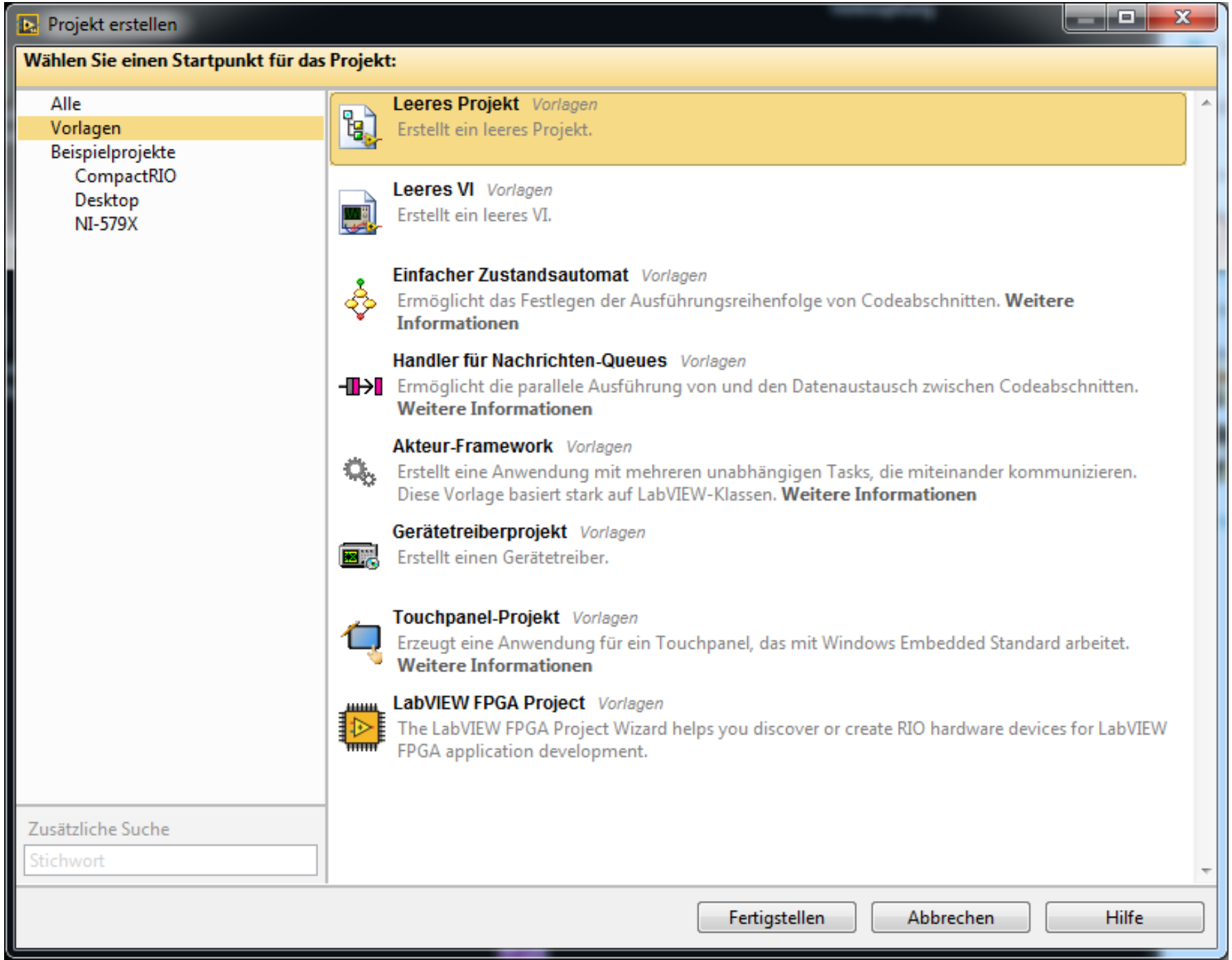
Einfügung eines VI-Moduls in eine eigene Anwendungs-VI

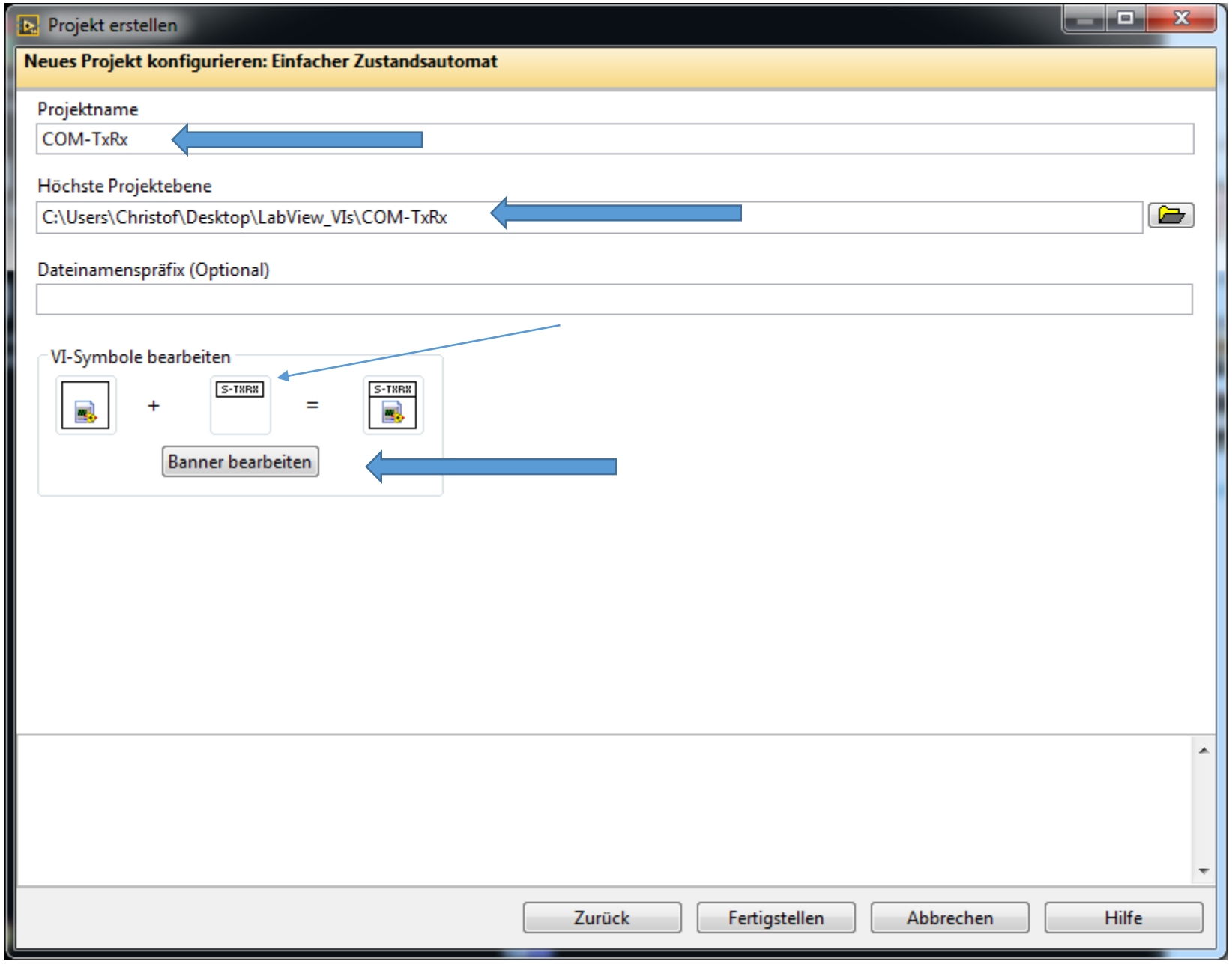




LV Projekte ermöglichen VI Verwaltung und echtes Kompiliererern in eine Runtime*.EXE



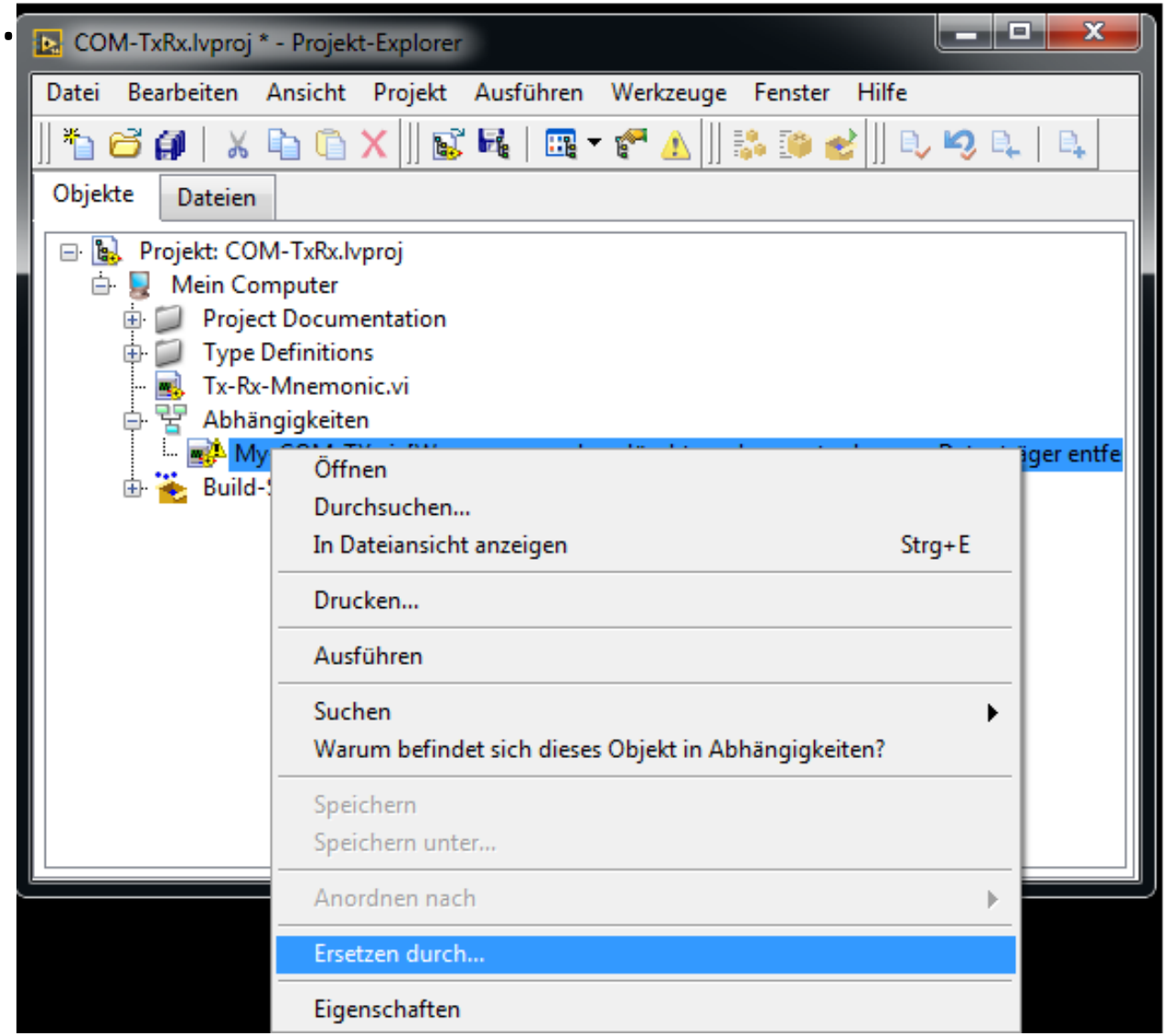

Projekt erstellen
Wählen Sie einen Startpunkt für das Projekt:
Alle
Vorlagen
Beispielprojekte
CompactRIO
Desktop
NI-579X
Leeres Projekt Vorlagen
Erstellt ein leeres Projekt.
Leeres VI Vorlagen
Erstellt ein leeres VI.
Einfacher Zustandsautomat Vorlagen
Ermöglicht das Festlegen der Ausführungsreihenfolge von Codeabschnitten. Weitere Informationen
Handler für Nachrichten-Queues Vorlagen
Ermöglicht die parallele Ausführung von und den Datenaustausch zwischen Codeabschnitten. Weitere Informationen
Akteur-Framework Vorlagen
Erstellt eine Anwendung mit mehreren unabhängigen Tasks, die miteinander kommunizieren. Diese Vorlage basiert stark auf LabVIEW-Klassen. Weitere Informationen
Gerätetreiberprojekt Vorlagen
Erstellt einen Gerätetreiber.
Touchpanel-Projekt Vorlagen
Erzeugt eine Anwendung für ein Touchpanel, das mit Windows Embedded Standard arbeitet. Weitere Informationen
LabVIEW FPGA Project Vorlagen
The LabVIEW FPGA Project Wizard helps you discover or create RIO hardware devices for LabVIEW FPGA application development.
Zusätzliche Suche
Stichwort
Fertigstellen
Abbrechen
Hilfe


Projekt erstellen
Neues Projekt konfigurieren: Einfacher Zustandsautomat
Projektname
COM-TxRx
Höchste Projektebene
C:\Users\Christof\Desktop\LabView_VIs\COM-TxRx
Dateinamenspräfix (Optional)
VI-Symbole bearbeiten
S-TXRX
+
=
S-TXRX
Banner bearbeiten
Zurück
Fertigstellen
Abbrechen
Hilfe

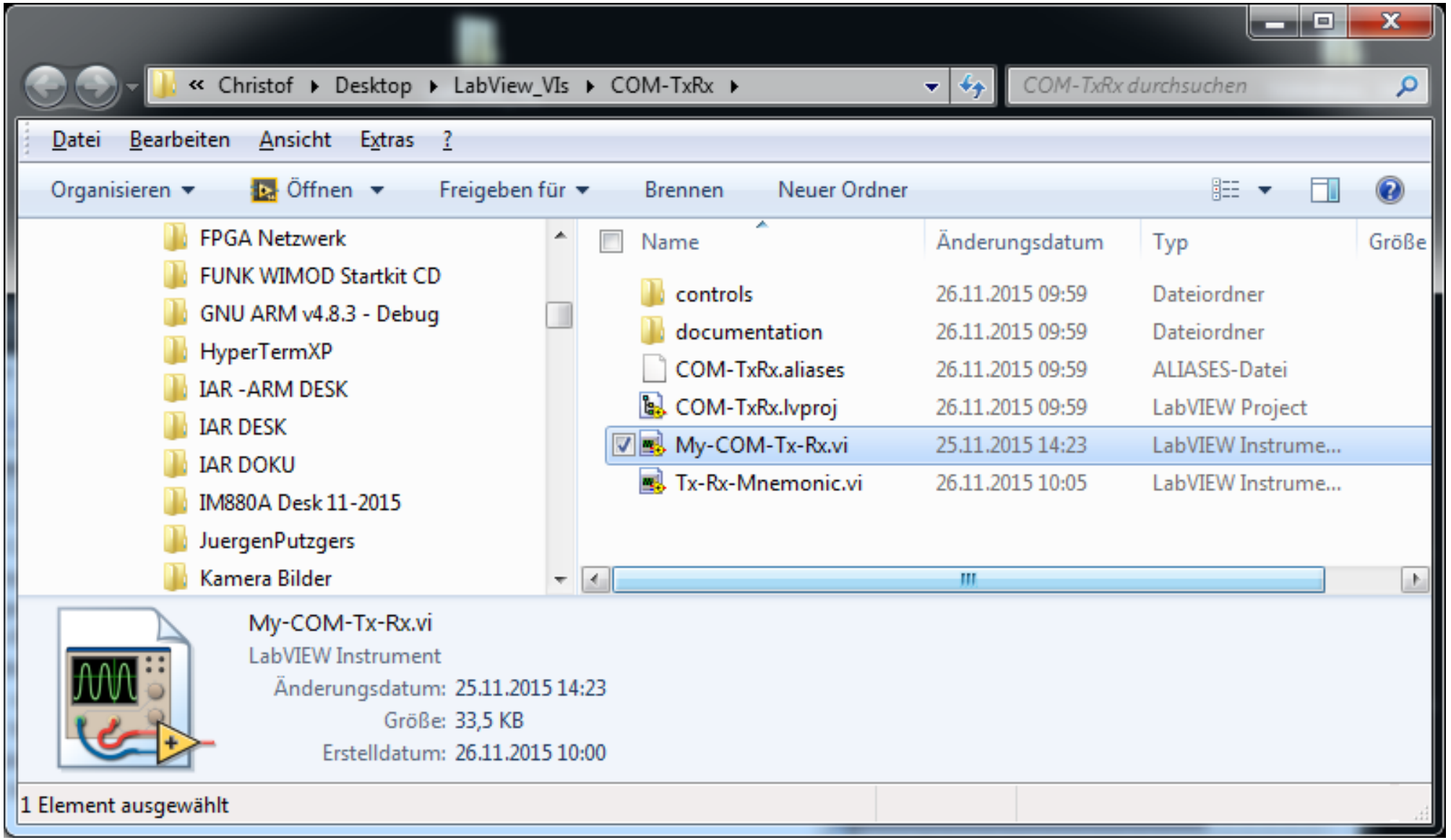
Vorhandene VIs in den Projekteordner kopieren und neue Abhängigkeiten zuweisen.

Das ist etwas unschön gelöst, dass LV nicht zuerst automatisch im Projekteordner sucht und die Dateien in die Abhängigkeiten zuweist.
Man muss diese manuell z.B. mit „Ersetzen durch..“ zuweisen
Sonst sucht LV diese VIs im originalen Ordner. Wahrscheinlich um Redundanzen zu vermeiden.

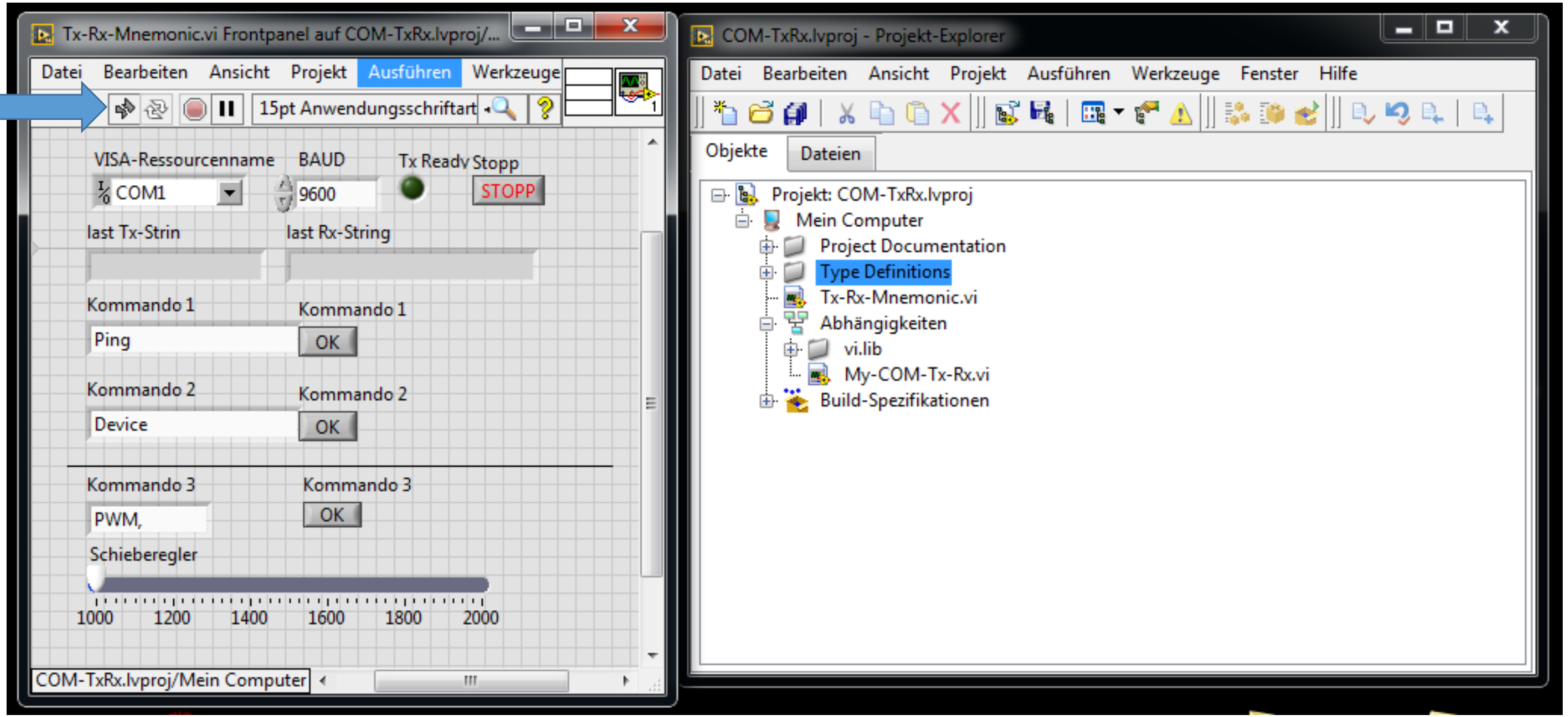
Evtl. Lösung: Jedes VI umbenennen mit einem Projekt Spezifischen Präfix.



So sammeln wir alle Daten im selben Ordner



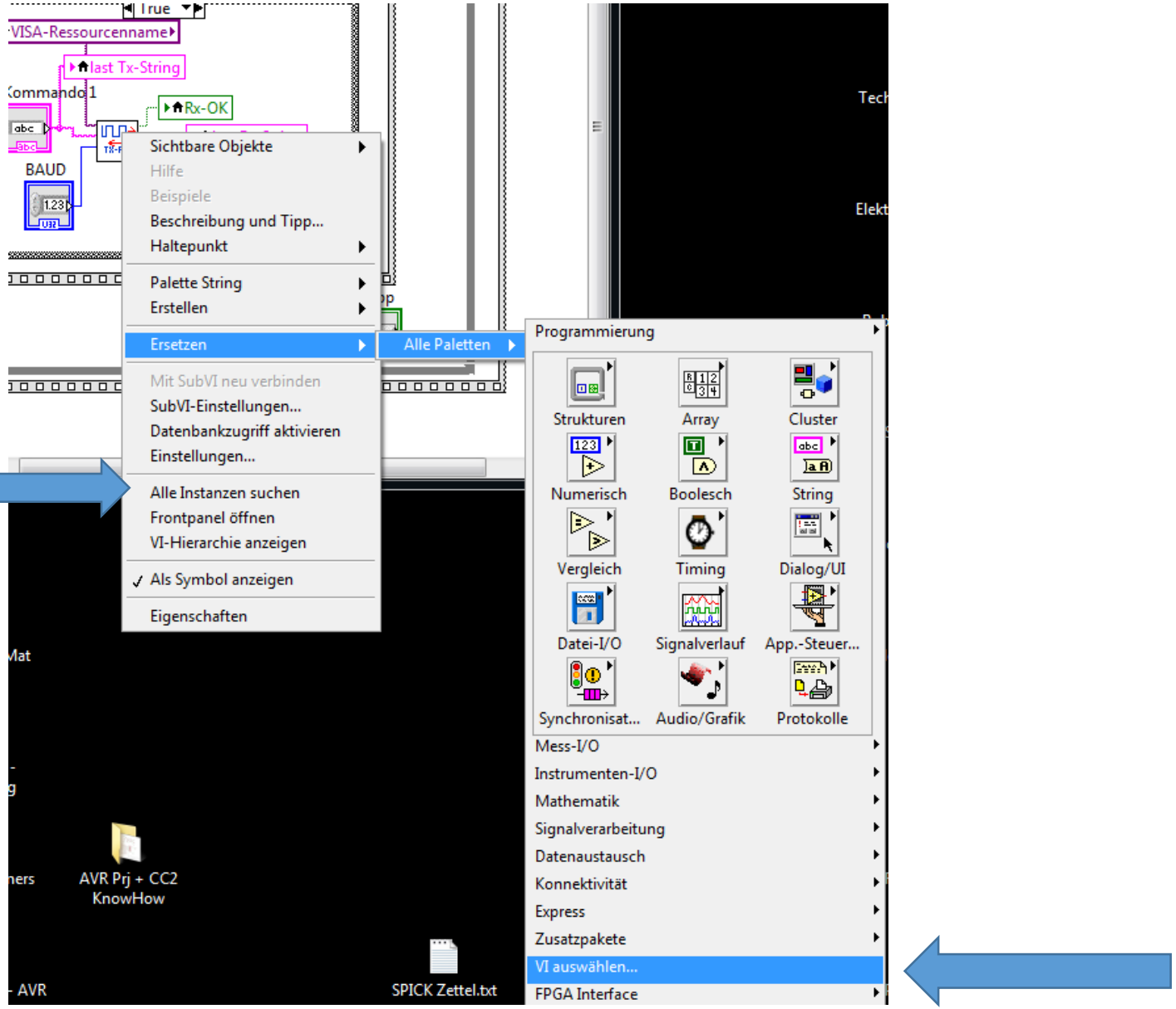
Leider kommt es dann vor, dass schon fertige VIs Fehler anzeigen. Das liegt wahrscheinlich an der internen Verlinkung der Unter-VIs an einen nicht mehr auffindbaren Ort. Diese müssen dann Zufuß neu zugewiesen werden



So kann man **un**-aufgefundene oder gebrochene VIs durch die richtige Version am richtigen Ort ersetzen.

RC→ auf Objekt oder VI
→**Ersetzen**
→**Alle Paletten**
→**VI auswählen**

Alternativ:
→**Alle Instanzen suchen** ausprobieren.



Formelknoten

Sehr leistungsfähig. Mit **ANSI-C vergleichbarer Syntax** können mathematische Ausdrücke direkter und deutlich beschrieben werden. Sehr schnelle Ausführung ! Ein und Ausgänge definieren und mit richtigen **Datentypen** versehen. Hier zahlt sich korrekte Datentyp Verwendung aus.

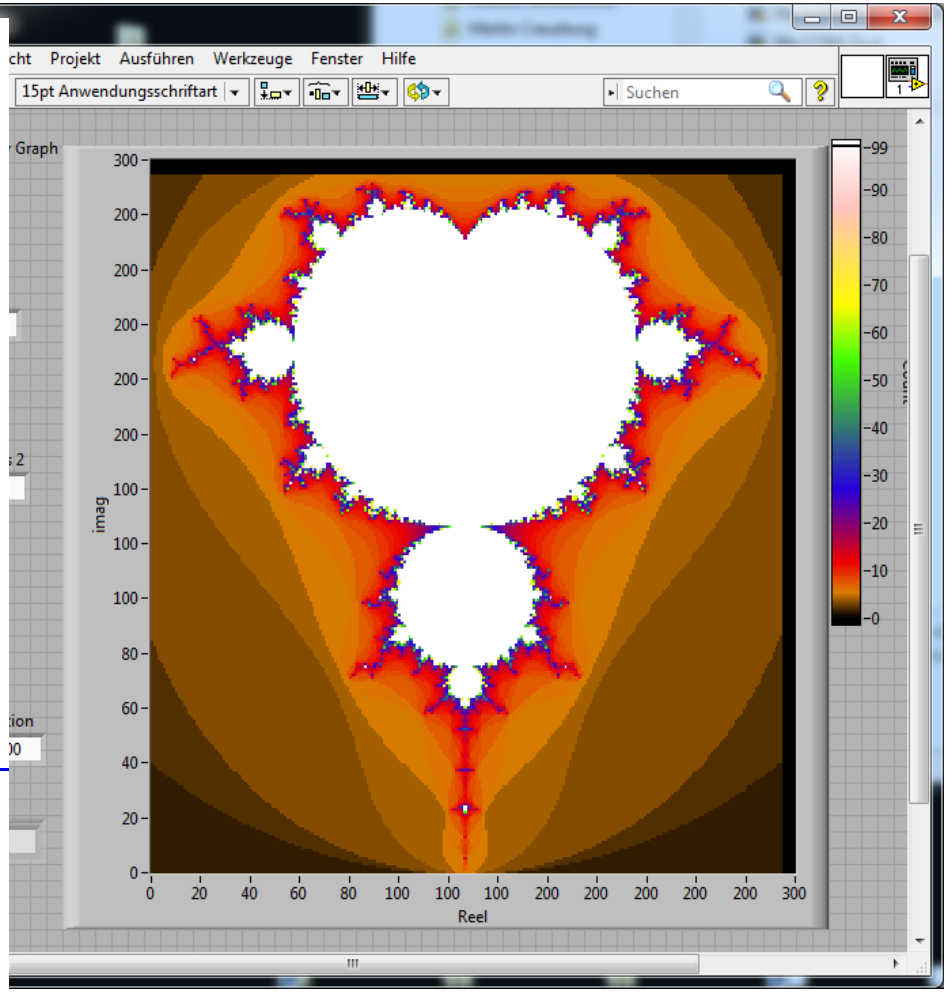
Im Beispiel Berechnung der Mandelbrot menge (Fraktal)

Komplexe rekursive Iteration:
Start: Z=0; C=R/i

$$Z_{n+1} = Z^2_n + C$$

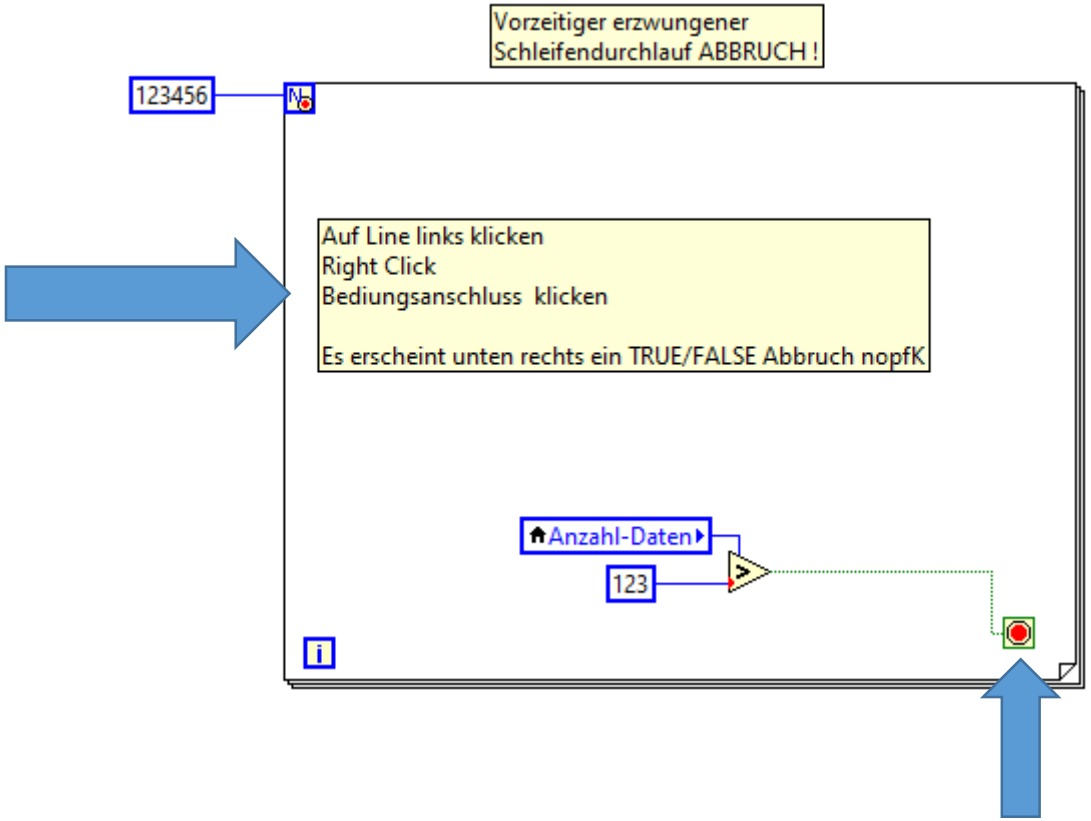
```
float64 RN;
float64 IN;
float64 Real;
float64 Imag;
float64 Betrag;
int32 IterNN;
IterNN =  0;
IN = 0;
RN =  0;

while( (Betrag < 4 ) && (IterNN < IterationM
{
Real = RN * RN;
Imag = IN * IN;
//Betrag = Real*Real + Imag * Imag;
IN = (IN *  RN * 2) + YC;
RN = Real -  Imag + XC;
Betrag = RN*RN + IN *IN;
IterNN++;
};
```

XC
YC
IterationMax
IterNN



TRICK KISTE

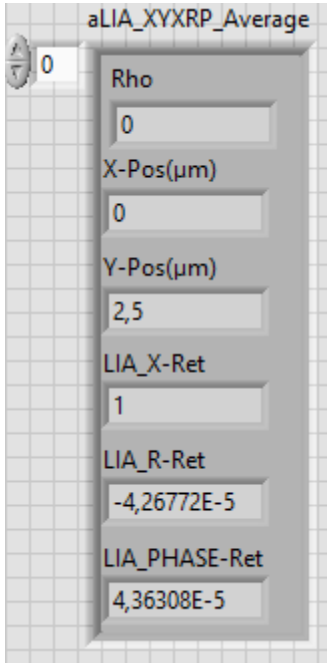
Um Programme auch in einer Schleife unterbrechen zu können, oder warum auch immer, kann man eine Schleife auch vor Beendigung der schleifenzahl unterbrechen.



Hier klicken, mit Rechter Maustaste

Es erscheint ein Abbruch Bedingungs Symbol

Ändern der Reihenfolge von Clustern Die angezeigte Reihenfolge ist nicht unbedingt die LOGISCHE



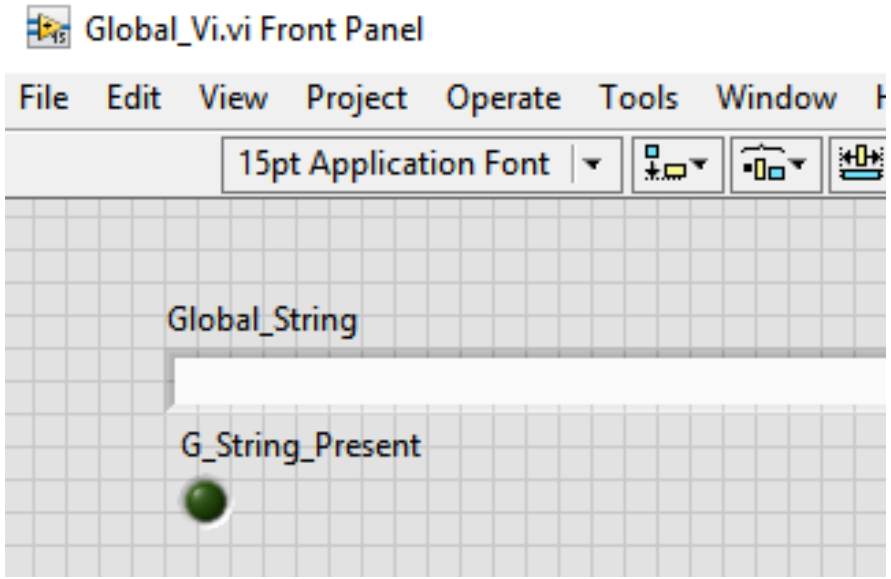
Klicke auf den Rand des Clusters

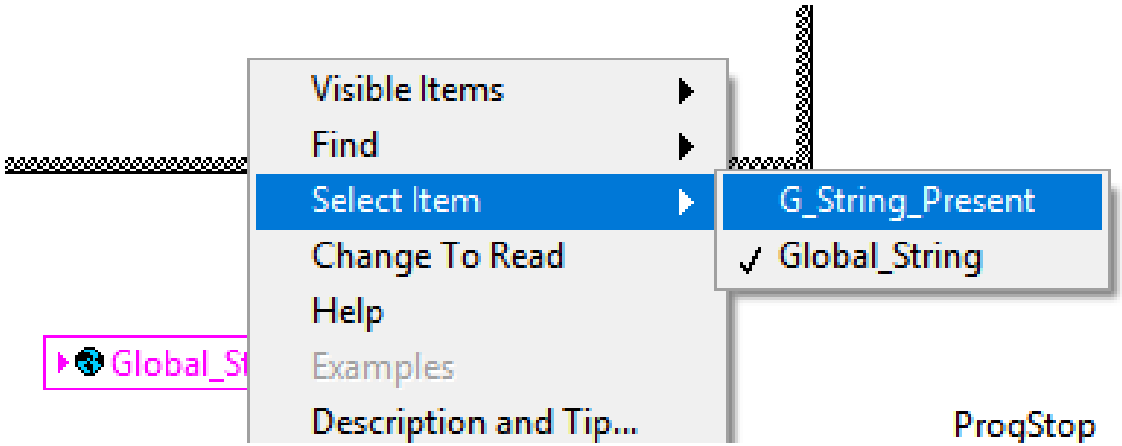
Klicken Sie mit der rechten Maustaste auf den Rand des Clusters und wählen Sie aus dem Kontextmenü die Option Elemente im Cluster neu ordnen aus. In der Symbolleiste und im Cluster werden folgende Änderungen angezeigt:
An jedem Cluster-Element wird ein weißes und ein schwarzes Kästchen mit einer Nummer angezeigt. Das weiße Kästchen zeigt die aktuelle Position eines Elements im Cluster an.
Die Nummern in den schwarzen Kästchen stimmen mit denen in den weißen Kästchen überein. Wenn Sie die Reihenfolge ändern, wird in den schwarzen Kästchen die neue Position des Elements angezeigt.
In der Symbolleiste über dem Cluster wird ebenfalls eine Nummer in einem schwarzen Kästchen angezeigt. Sie bezieht sich auf das nächste Element, das Sie ändern.
Der Cursor verwandelt sich in den Cursor "Cluster-Elementindex".
Zum Ändern der Reihenfolge gibt es folgende Möglichkeiten:
Klicken Sie auf ein Element im Cluster. Im schwarzen Kästchen wird die in der Symbolleiste angezeigte Nummer übernommen.
Geben Sie in das Feld Durch Klick setzen auf die neue Positionsnummer ein und klicken Sie das Element an.
Klicken Sie die Elemente in der gewünschten Reihenfolge an. Die Positionsnummer wird automatisch erhöht.
Klicken Sie anschließend auf die Schaltfläche OK, um die neue Reihenfolge zu bestätigen und den Bearbeitungsmodus für den Cluster-Elementindex zu verlassen. Wenn Sie die ursprüngliche Reihenfolge beibehalten möchten, klicken Sie auf Abbrechen.

Globales VI (singular)

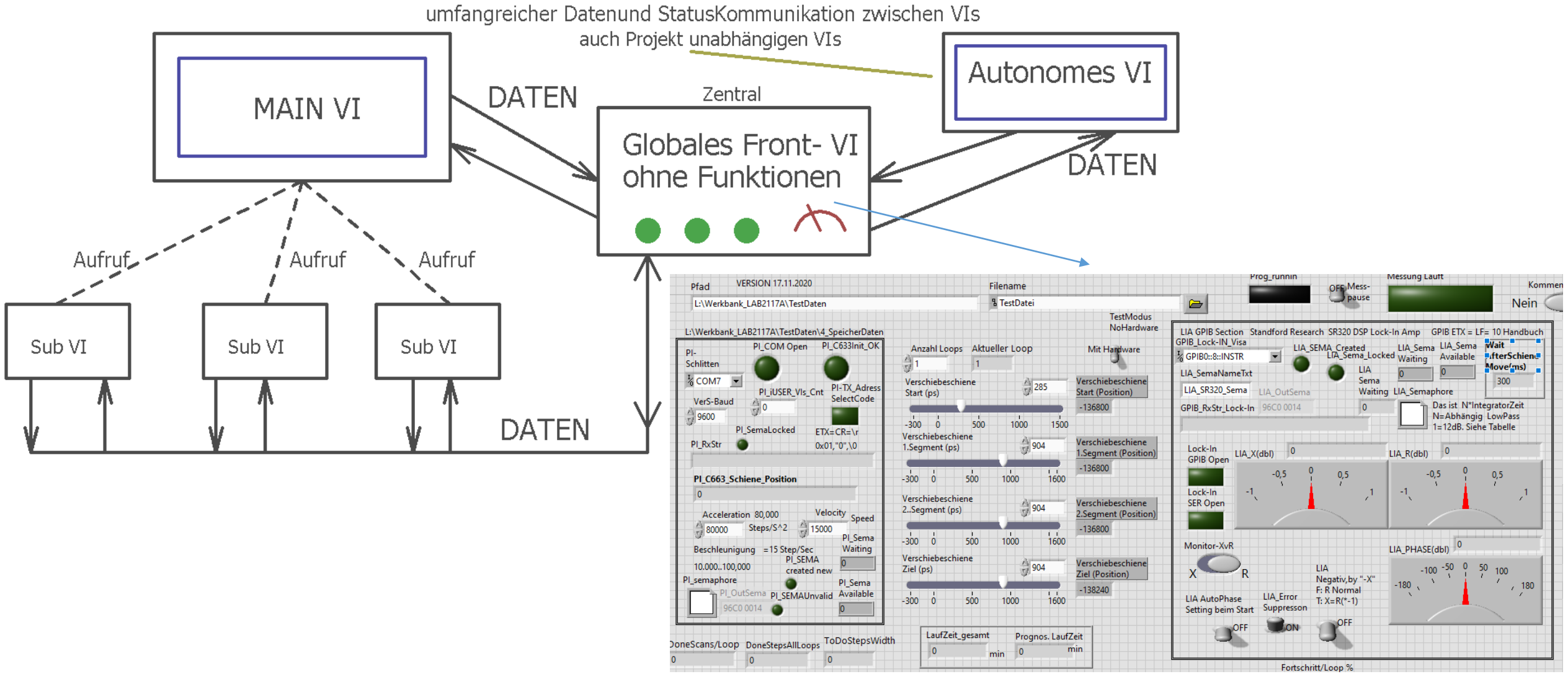
Um Daten zwischen **getrennt in while Schleifen laufende Vis** auszutauschen nutzt man eine „gemeinsame“ globale VI (Frontfenster mit Datenobjekten **OHNE** Schematic Fenster .
Aber… **Beide MAIN-Vis** teilen sich **das Globale Vi** um z.B. Strings oder Trigger Booleans (LED) abzulegen.
Erzeugung:
In einer Vi im Schematic Fenster→ Functionspalette → Structures → Globale Variable (unten rechts) → Abspeichern, Objekte hineinlegen. → Dann „im Main Programm“ **Right Click**→ Replace →All Pallettes → Select a Vi → „Gloval_VI.vi“ (Global VI Element importieren , → evtl. mit RC—“Select Item“ das Datenobjekt wechseln.. Quasi lokale Variable in Schreiben oder Lesen ändern etc..

Trick: Jetzt kann man z.B. Strings und Befehle an andere laufende Vis übergeben





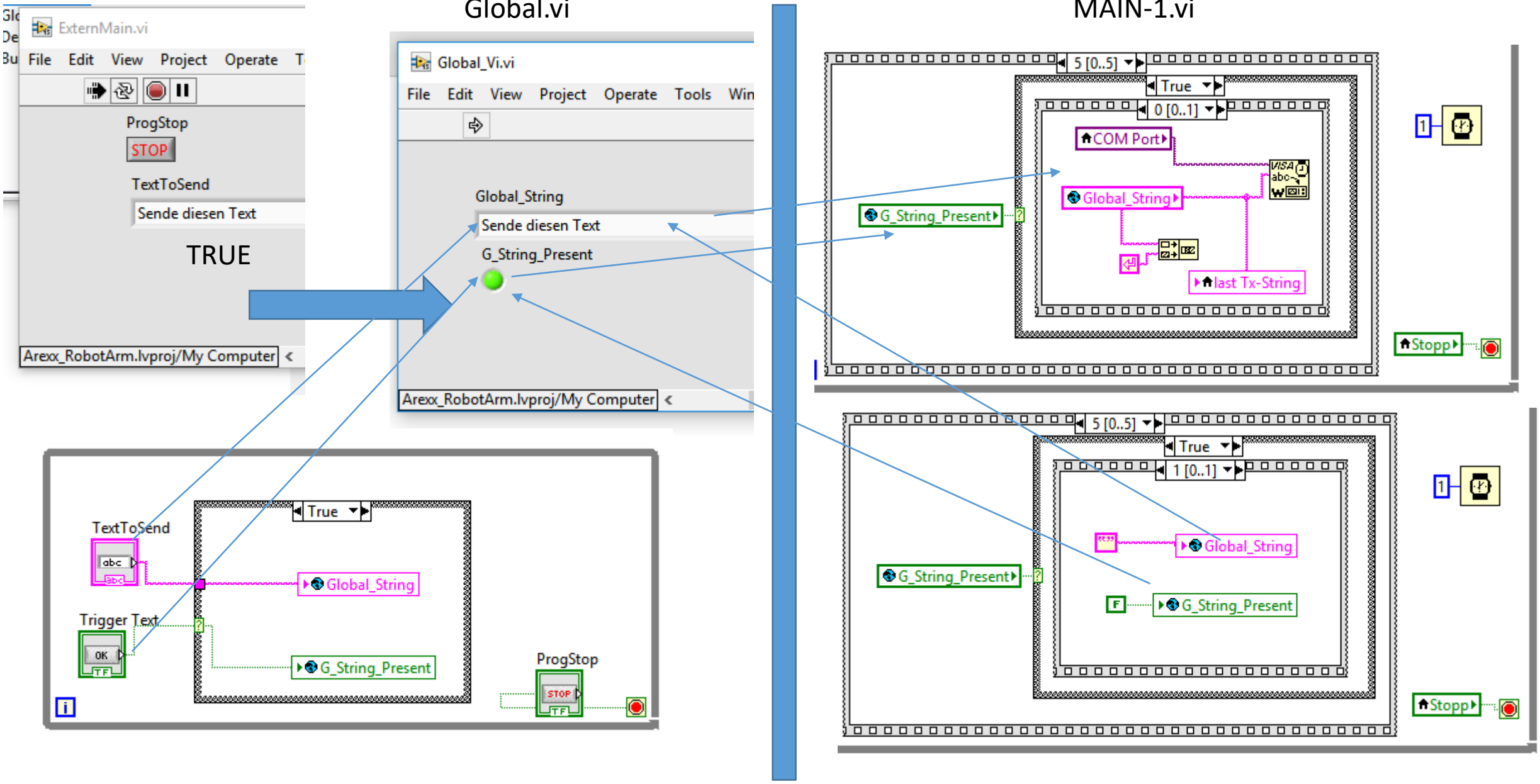
Globale VIs eignen sich zum Datenaustausch zwischen VIs (Arrays, Einstellungen, Anzeige und vor allem SYNCHRONISATION von Statusinformationen, wie → „Port Nummer / Open / Close“ usw.....



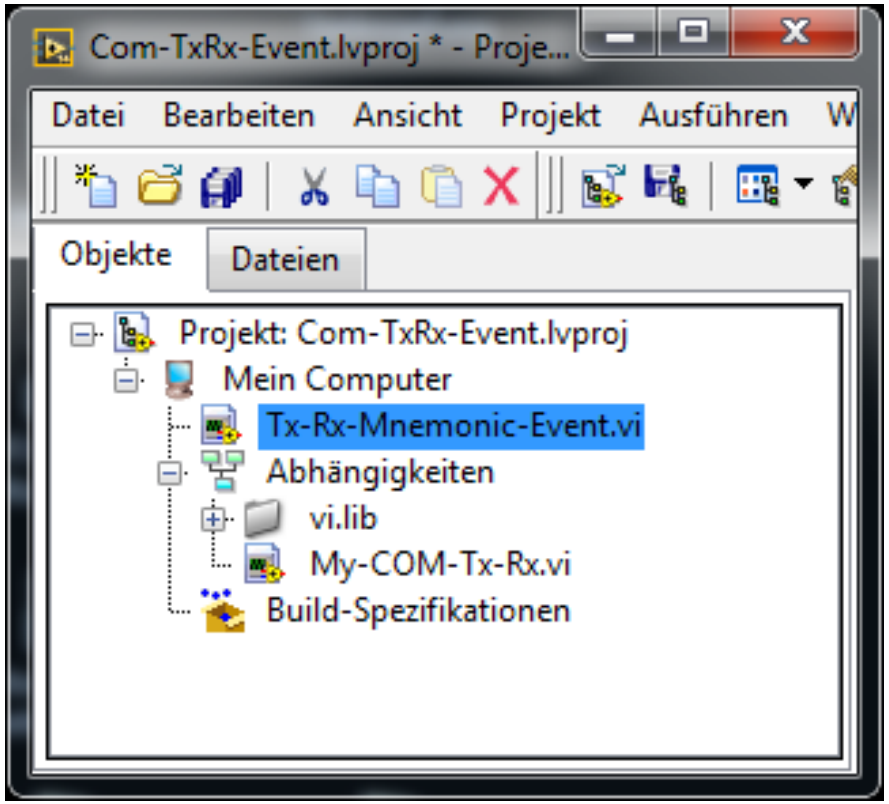
MAIN-2.vi

Global.vi

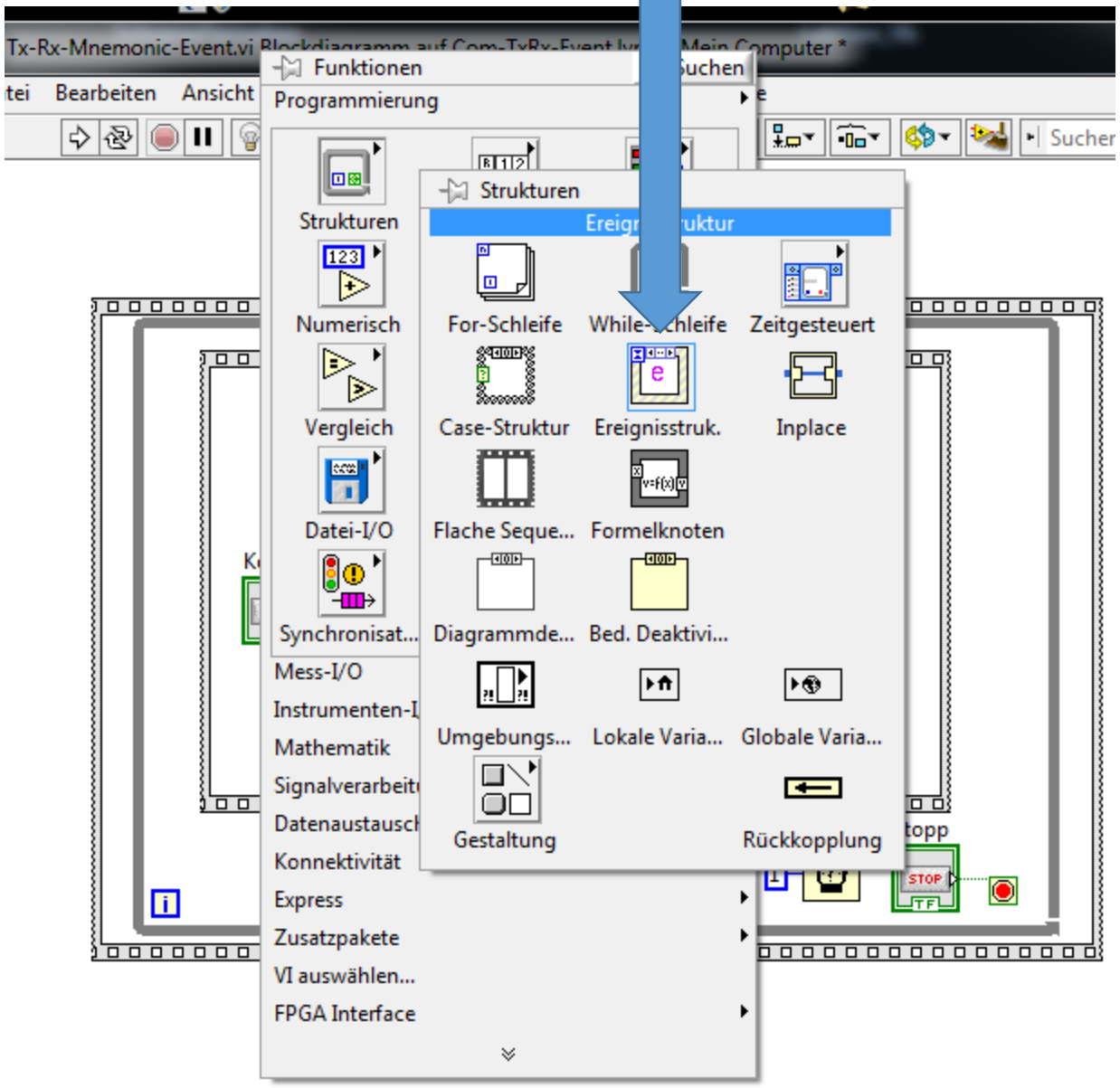
MAIN-1.vi



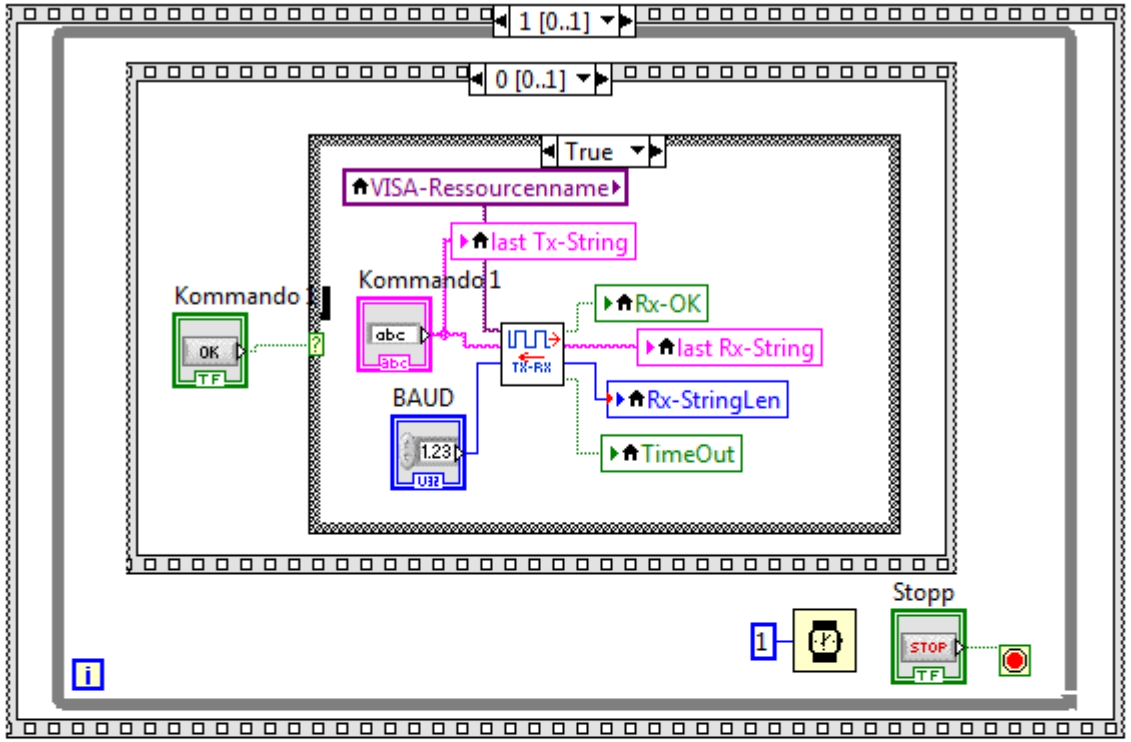
Event **oder** Ereignis Struktur

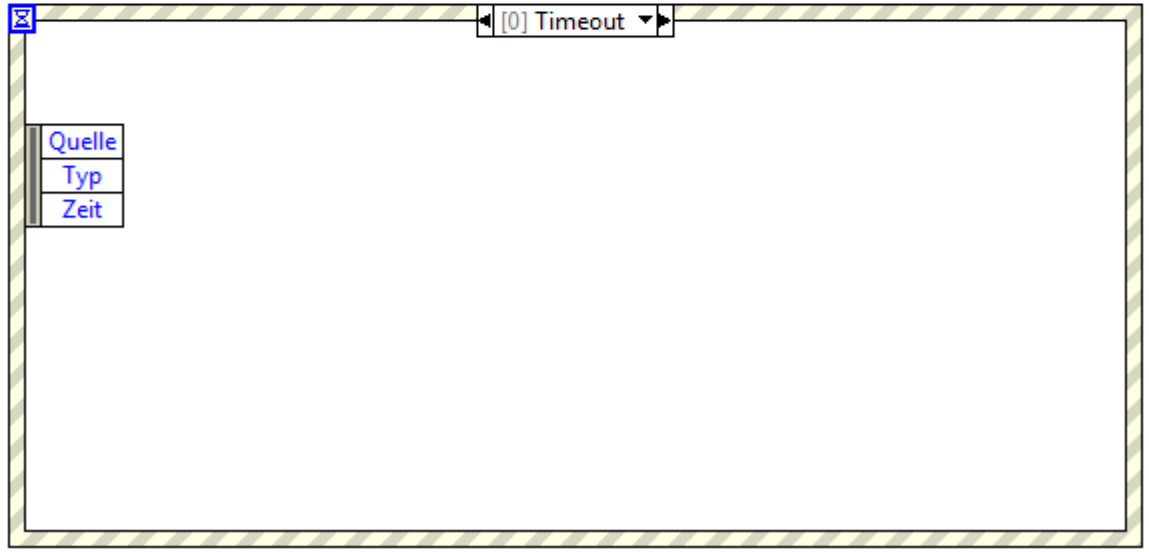


Ereinis-
struktur
Anlegen.



Neues Projekt anlegen. Dateien in neuen Ordner hineinkopieren. UMBENENNEN
Lauffähig Unter VIs einbinden

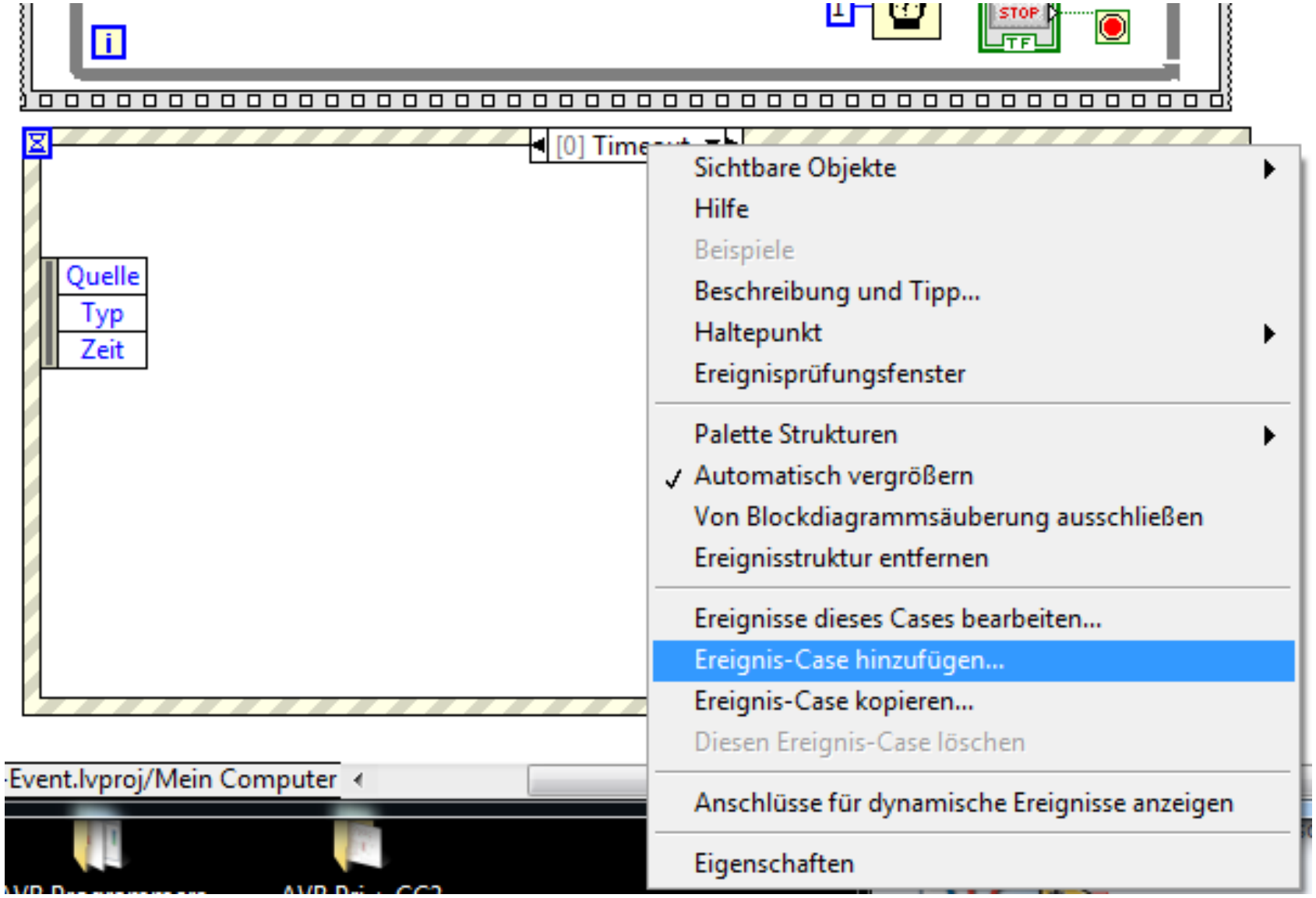




Ereignisstruktur unterhalb der VI Ablaufsequenz ablegen.

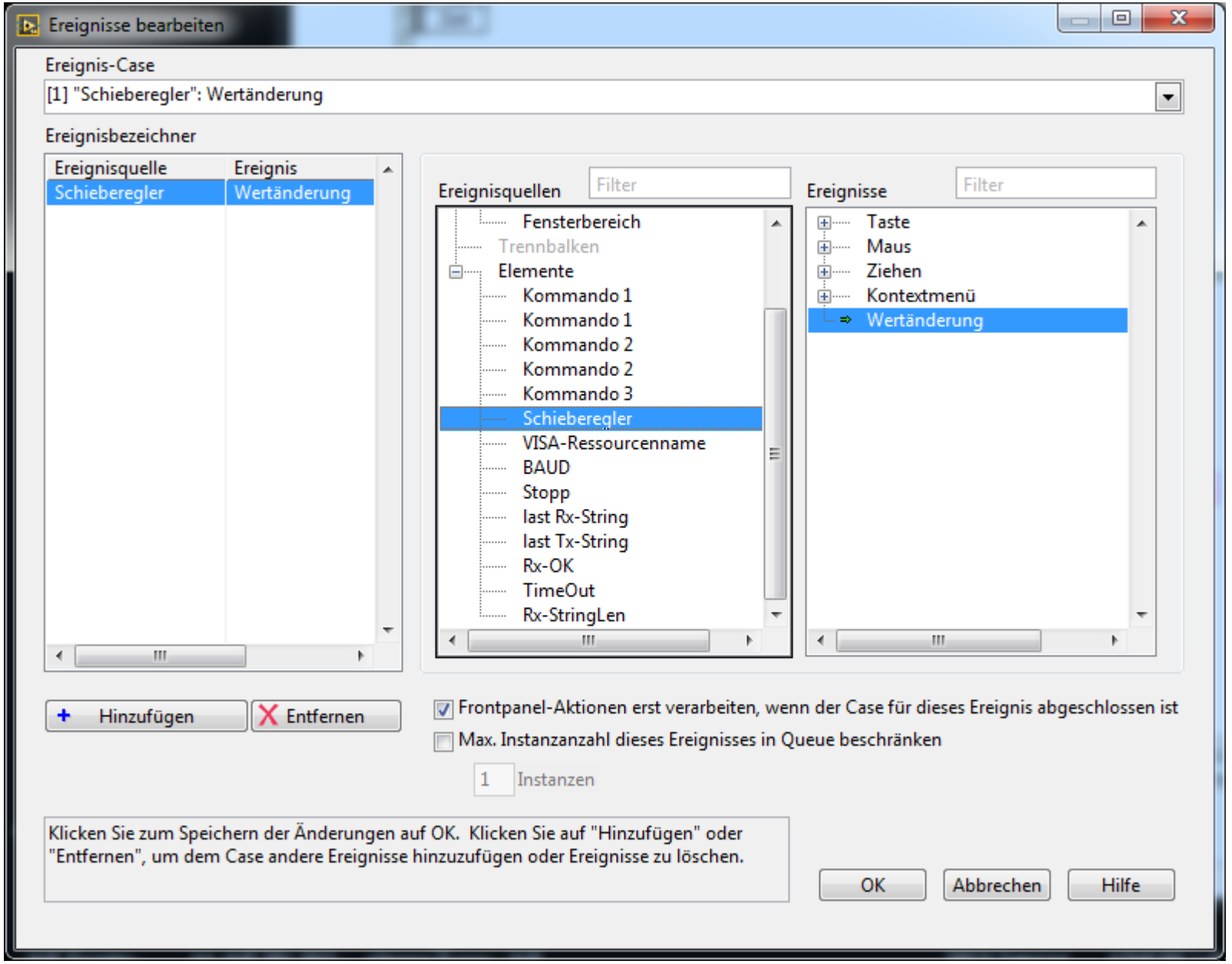
Für jedes Front Steuer Element wir ein eigenes Ereignis-Case erzeugt. Darin wird der Ereignisablauf abgelegt.
RC → „Ereignis-Case hinzufügen“

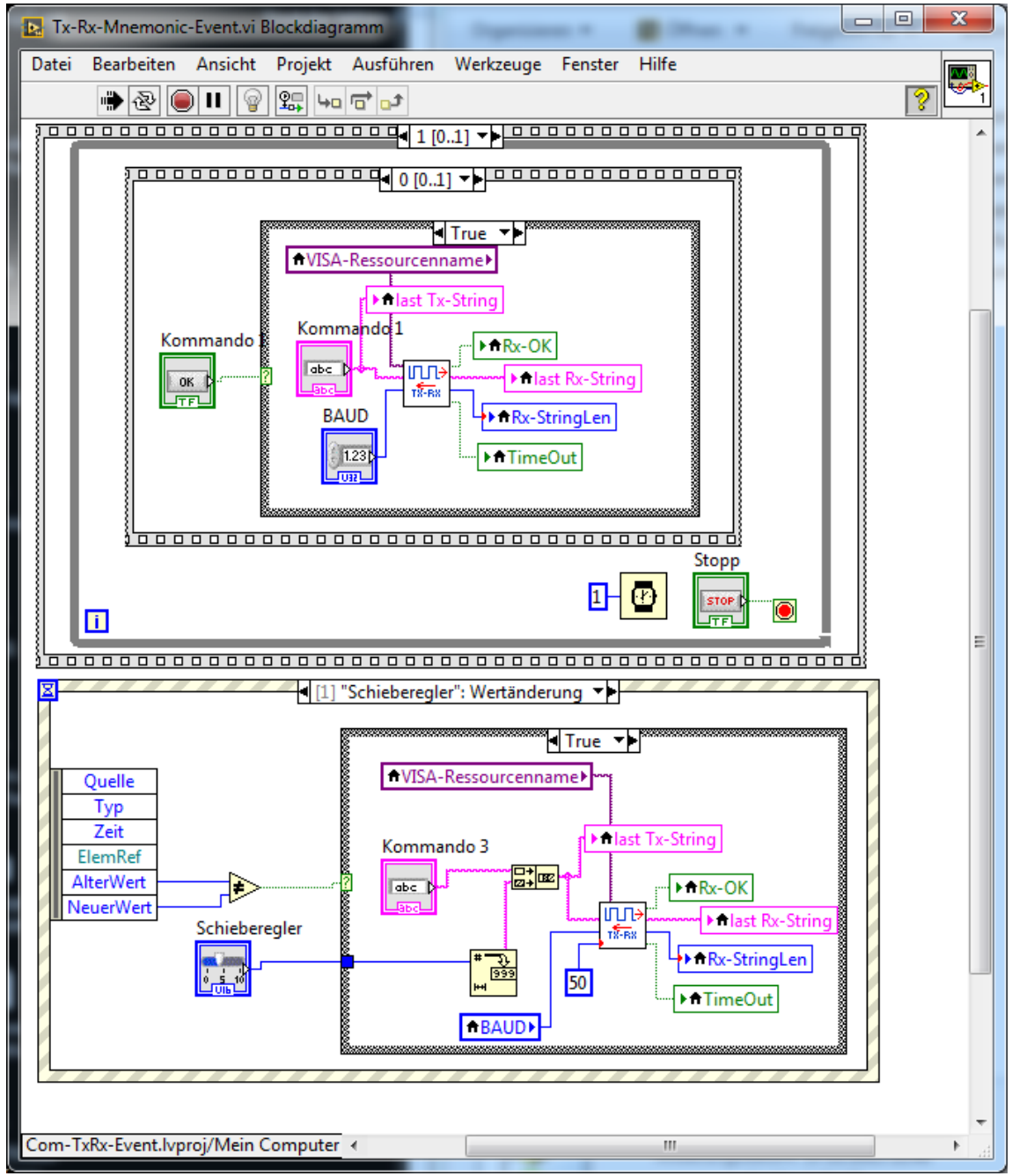
Wie… das lernen wir noch. Denn eine Besonderheit ist, dass z.B. Tastenereignisse besondere Behandlung erfordern. Und zwar wegen der enormen zeitlichen wie logischen Konsequenzen in Multitasking Systemen



Ereignisquelle hinzufügen

..dann
Selektiere Ereignisquelle
→ Schieberegler
..dann
..meist → Wertänderung.
→ OK





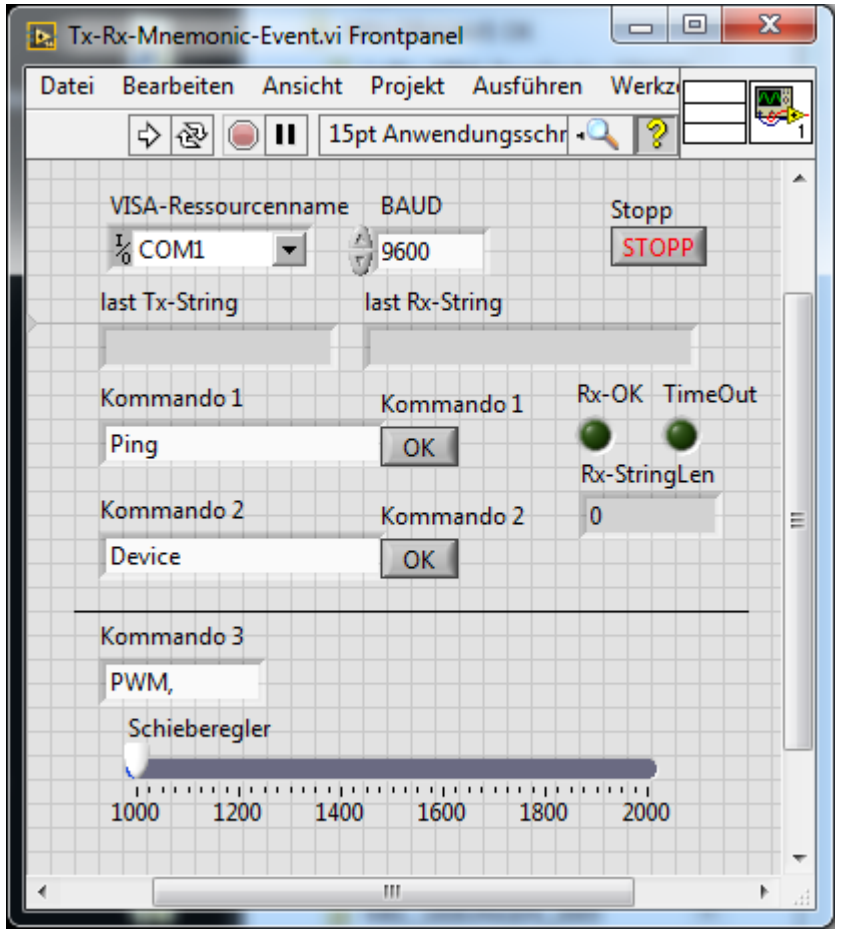
Wir füllen mal für den Schieberegler eine Ereignis-Case.
Es ergeben sich Konsequenzen für die Abarbeitung.
Das VI springt unmittelbar bei einer WERTÄNDERUNG in das zugehörige Case.
Dies ist quasi wie ein **Interrupt**.
Und das ist gefährlich für unbeendete Prozeduren, geöffnete Ports usw..

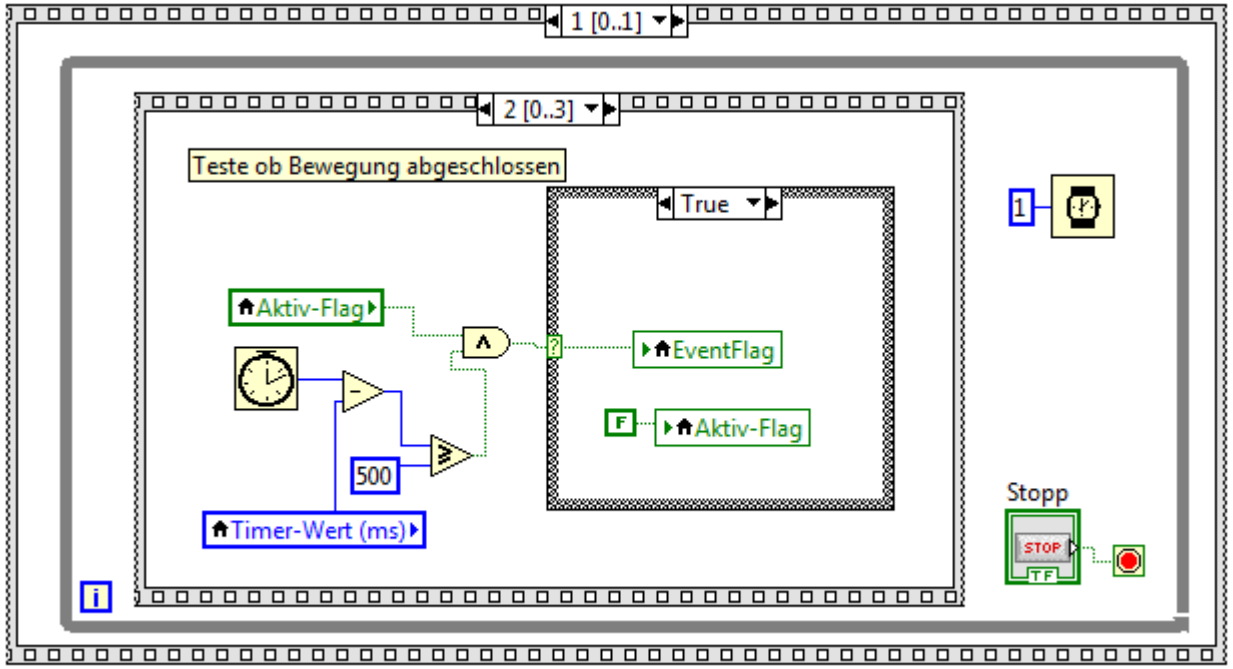
Hier sind Maßnahmen erforderlich.

Schrittweise Anpassen. Austausch von Rx-Rx Via durch eine VI mit nur Tx, um Delays zu vermeiden

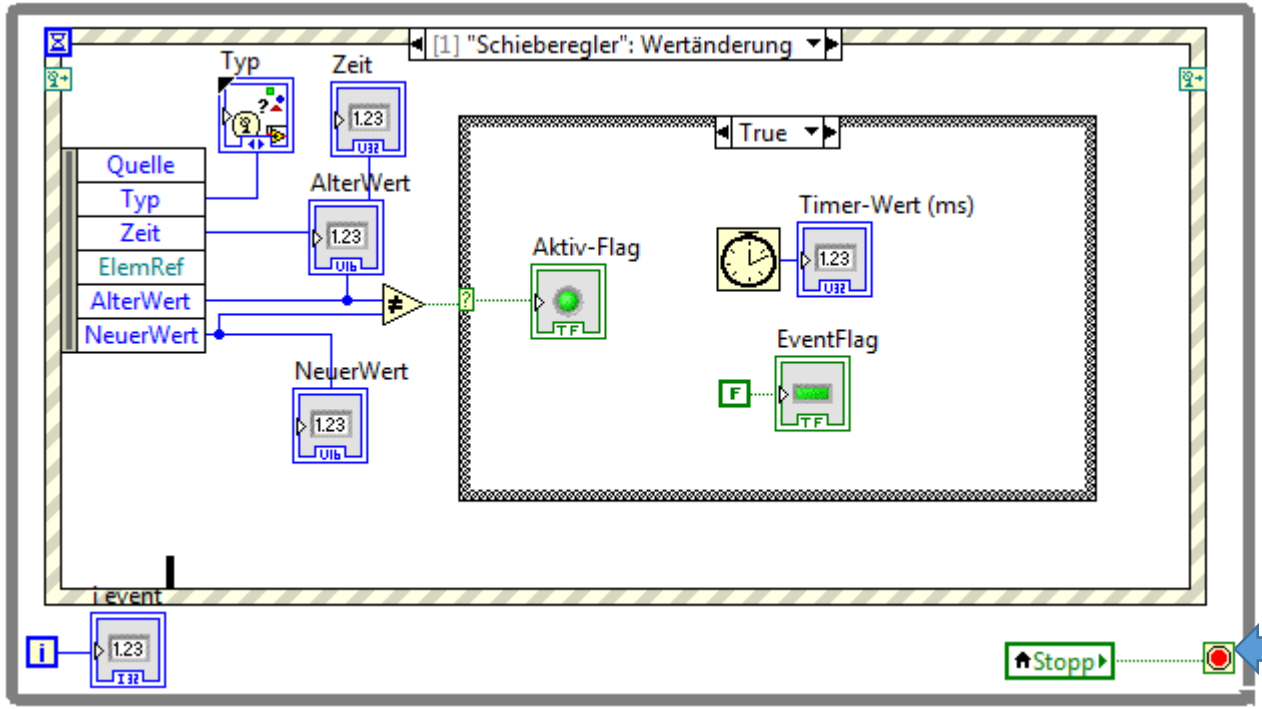
Wir müssen überlegen, ob die vorhanden seriellen COM-Routinen den Anforderungen in einer Multitasking Umgebung gerecht werden
….

Eventstruktur ebenfalls in eine WHILE-Loop verpacken



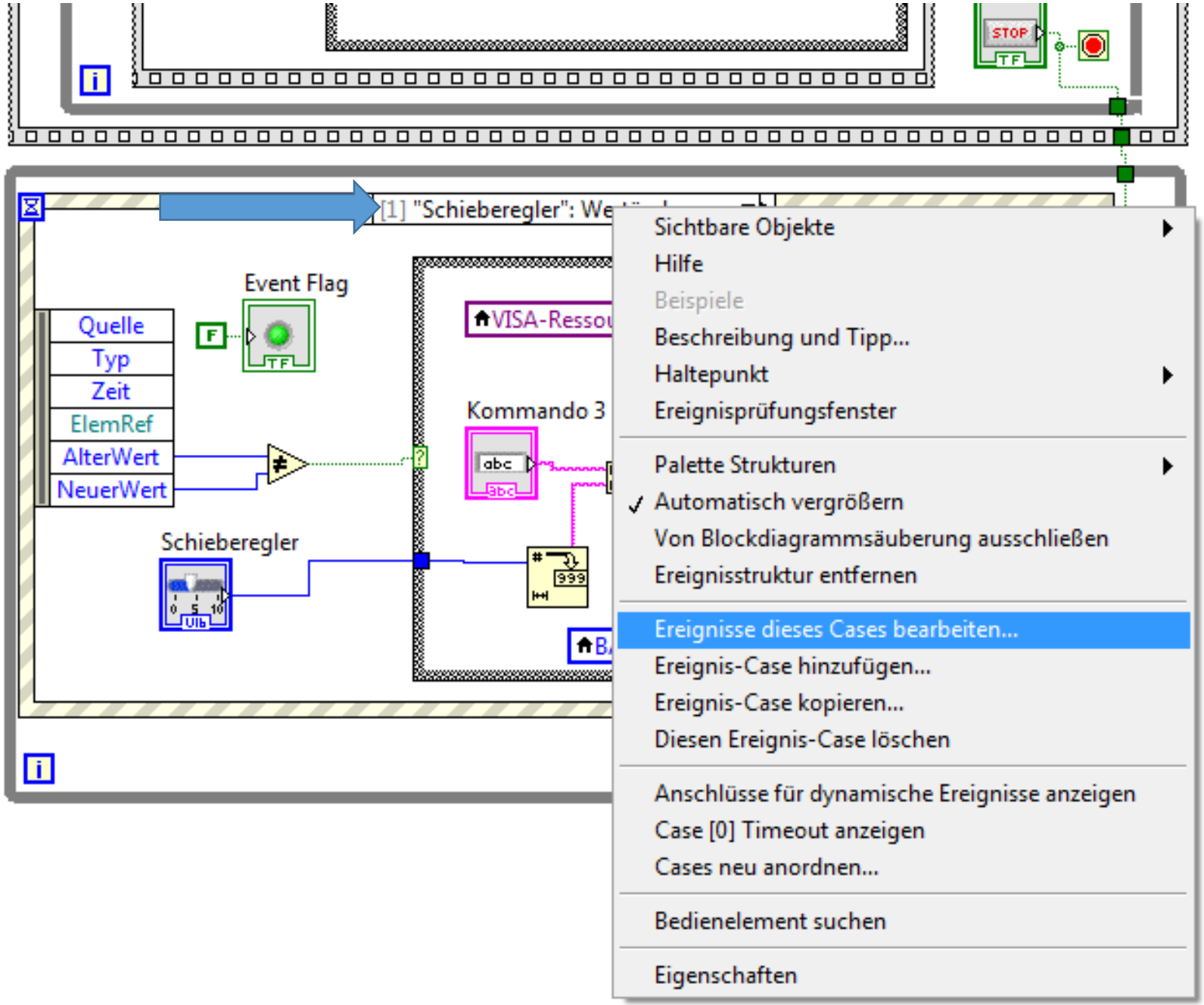


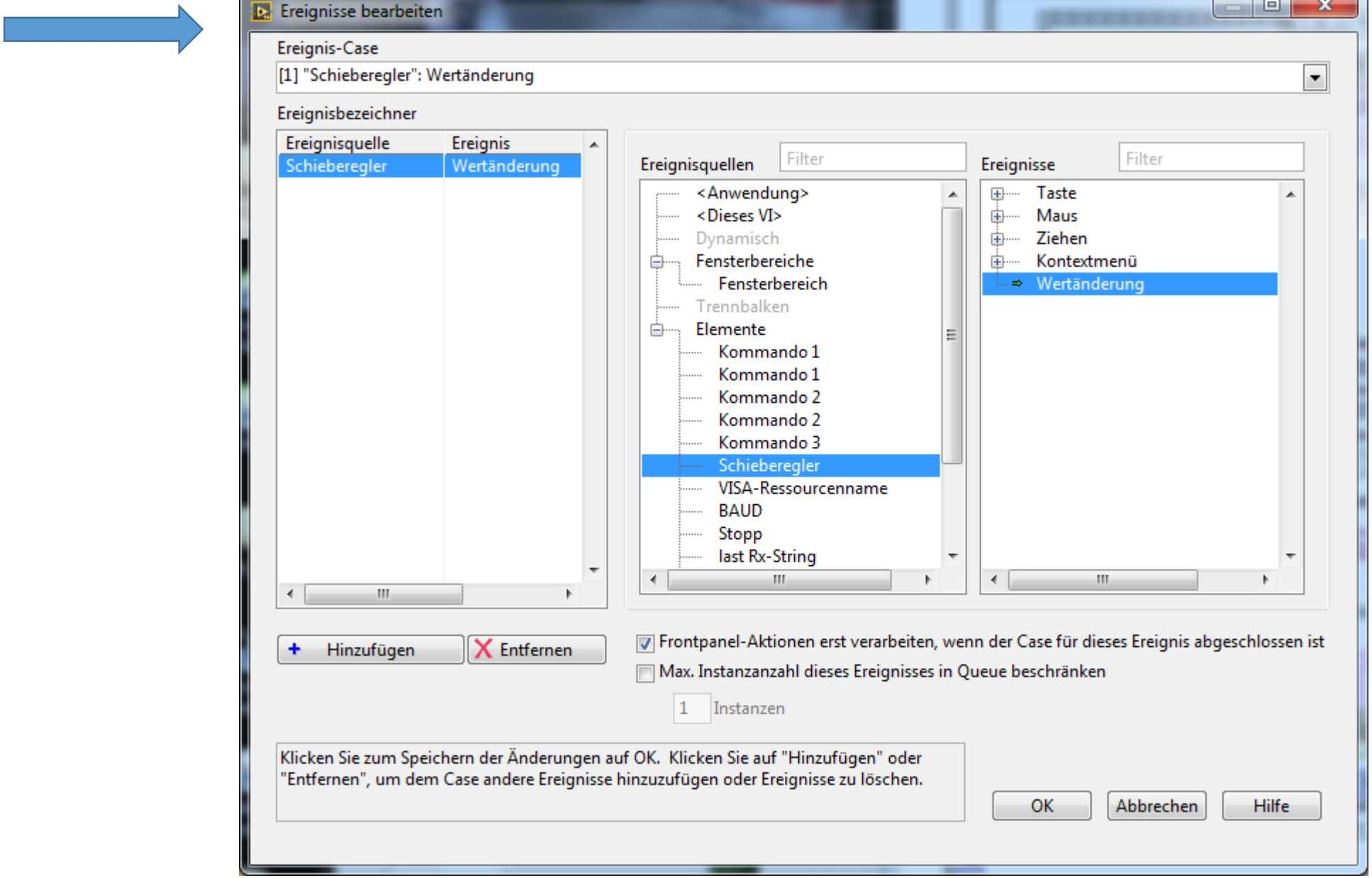
Keine Verbindungen zwischen den Loops



Sonst stottert das VI im Ablauf. (Signalfluss)

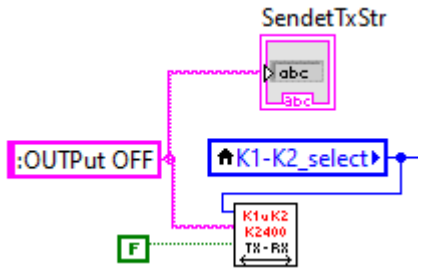
RC→ Sequenz
→ „Ereignisse dieses Cases bearbeiten“ klicken



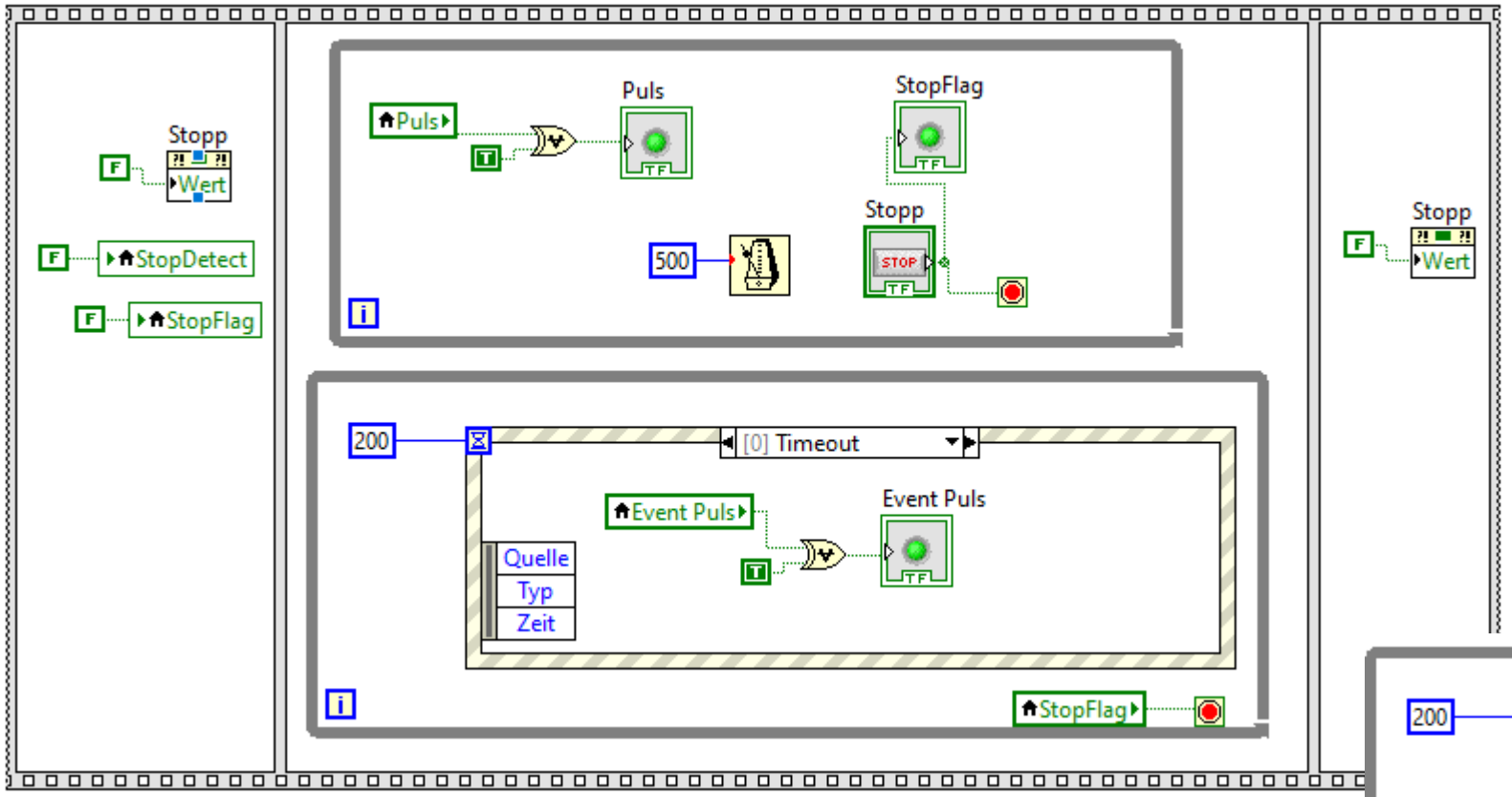

Ereignisse bearbeiten
Ereignis-Case
[1] "Schieberegler": Wertänderung
Ereignisbezeichner
Ereignisquelle
Ereignis
Schieberegler
Wertänderung
Ereignisquellen
Filter
<Anwendung>
<Dieses VI>
Dynamisch
Fensterbereiche
Fensterbereich
Trennbalken
Elemente
Kommando 1
Kommando 1
Kommando 2
Kommando 2
Kommando 3
Schieberegler
VISA-Ressourcenname
BAUD
Stopp
last Rx-String
Ereignisse
Filter
Taste
Maus
Ziehen
Kontextmenü
Wertänderung
+ Hinzufügen
Entfernen
Frontpanel-Aktionen erst verarbeiten, wenn der Case für dieses Ereignis abgeschlossen ist
Max. Instanzanzahl dieses Ereignisses in Queue beschränken
1
Instanzen
Klicken Sie zum Speichern der Änderungen auf OK. Klicken Sie auf "Hinzufügen" oder "Entfernen", um dem Case andere Ereignisse hinzuzufügen oder Ereignisse zu löschen.
OK
Abbrechen
Hilfe

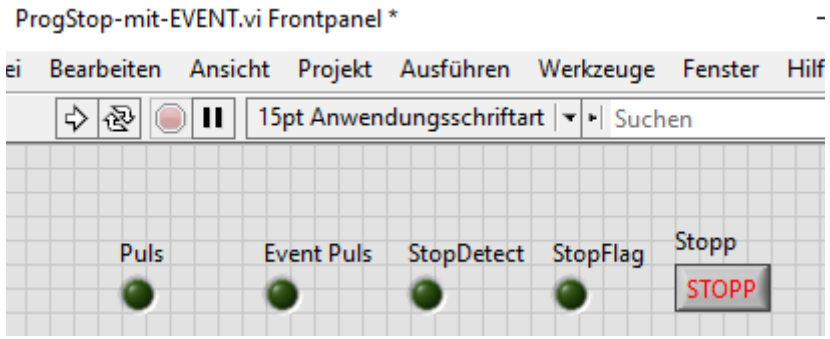
Konzepte zur EVENT Struktur = Quasi Parallel (interrupt) Verarbeitung

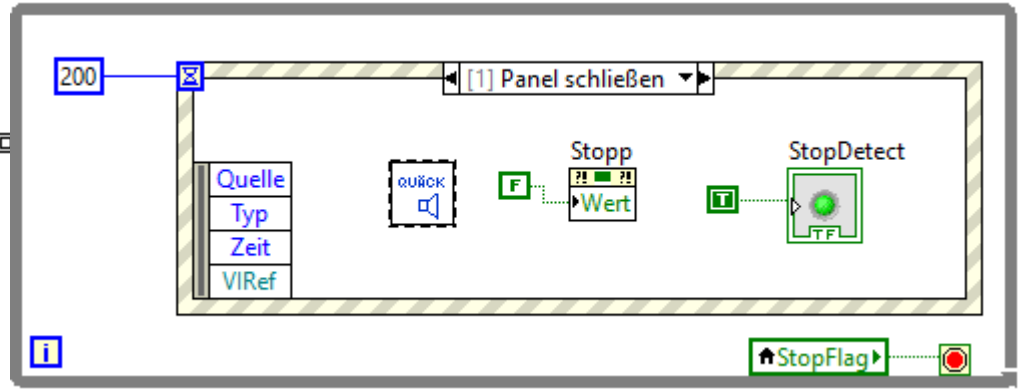
Da nun zwei While Loops gleichzeitig laufen , hat man es quasi mit einer parallelen Verarbeitung zu tun.<
Das bedeutet KOLLISSIONGEFAHR ! .
Früher habe ich mit Synchronisationskonzepten wie SEMAPHOREN gearbeitet.
Es ist einfacher, alle Kollisiongefärdeten VIs nur EINMAL Realisieren. So kann auch nur diese EINE aufgerufen werden. Bestes Beispiel. Die serielle COM_TxRX Sende und Empfangsroutine. Ständig gebraucht.



Das F oder T im Eingang steuert, ob ein Empfang gebraucht wird oder nicht.
Sonst ist diese Routine universell und nur einmal im Speicher.

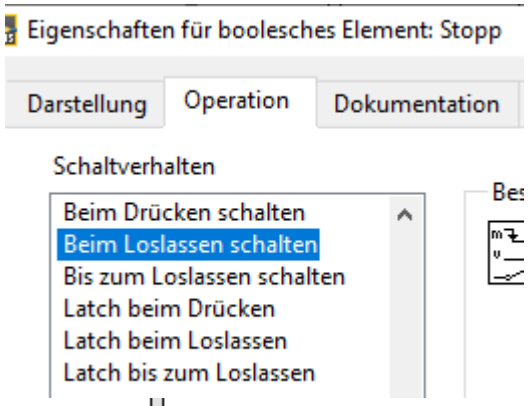






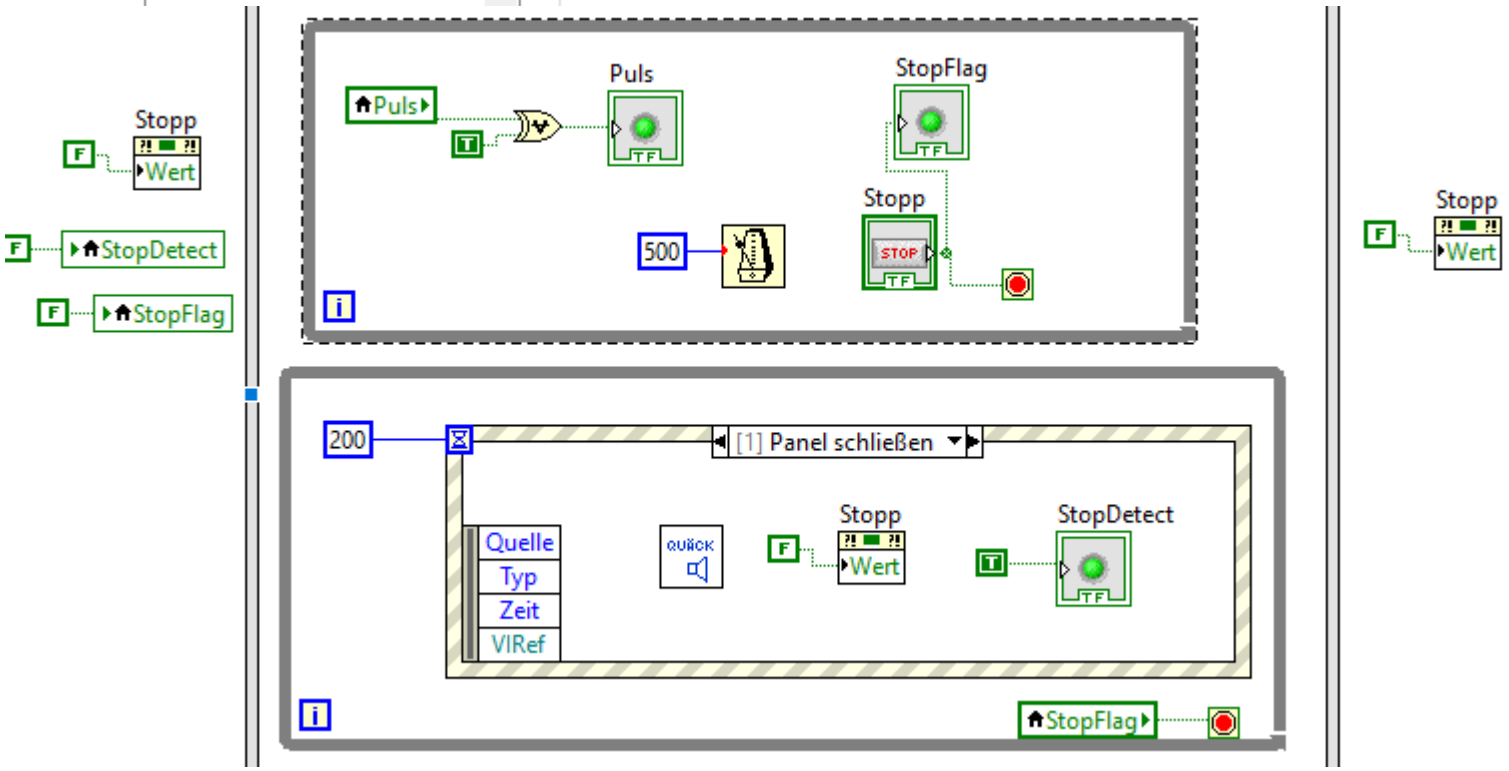
Hier ein Beispiel für die Nutzung der Event Struktur zur Erkennung „Panel Schließen“.
Gemixt mit der normalen Stop Struktur
So können geziehlt alle Ports geschlossen werden, alle Files bendet werden usw

Wichtig



In Event Struktur erforderliche Anpassung

Die Standard Einstellung für Tasten MÜSSEN umgestellt werden.
Keine Latsch Funktion!
1. Stelle auf bei Loslassen schalten
2. Speichere dies Information in einem Flag, dass sowohl den Main wie den Event While Loop unterbricht
3. Erzeuge „WERT“ Eigenschaft (nicht signalisierend),
Stelle diese auf Schreiben. Setze diese vor und nach Benutzung auf FALSE

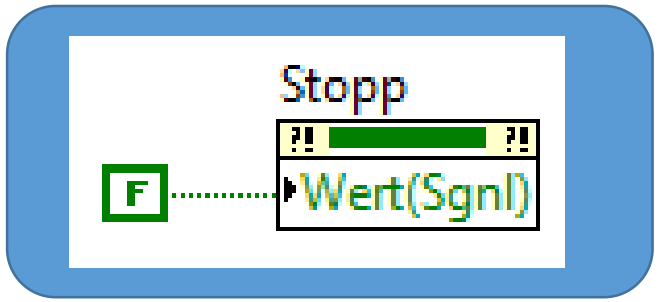


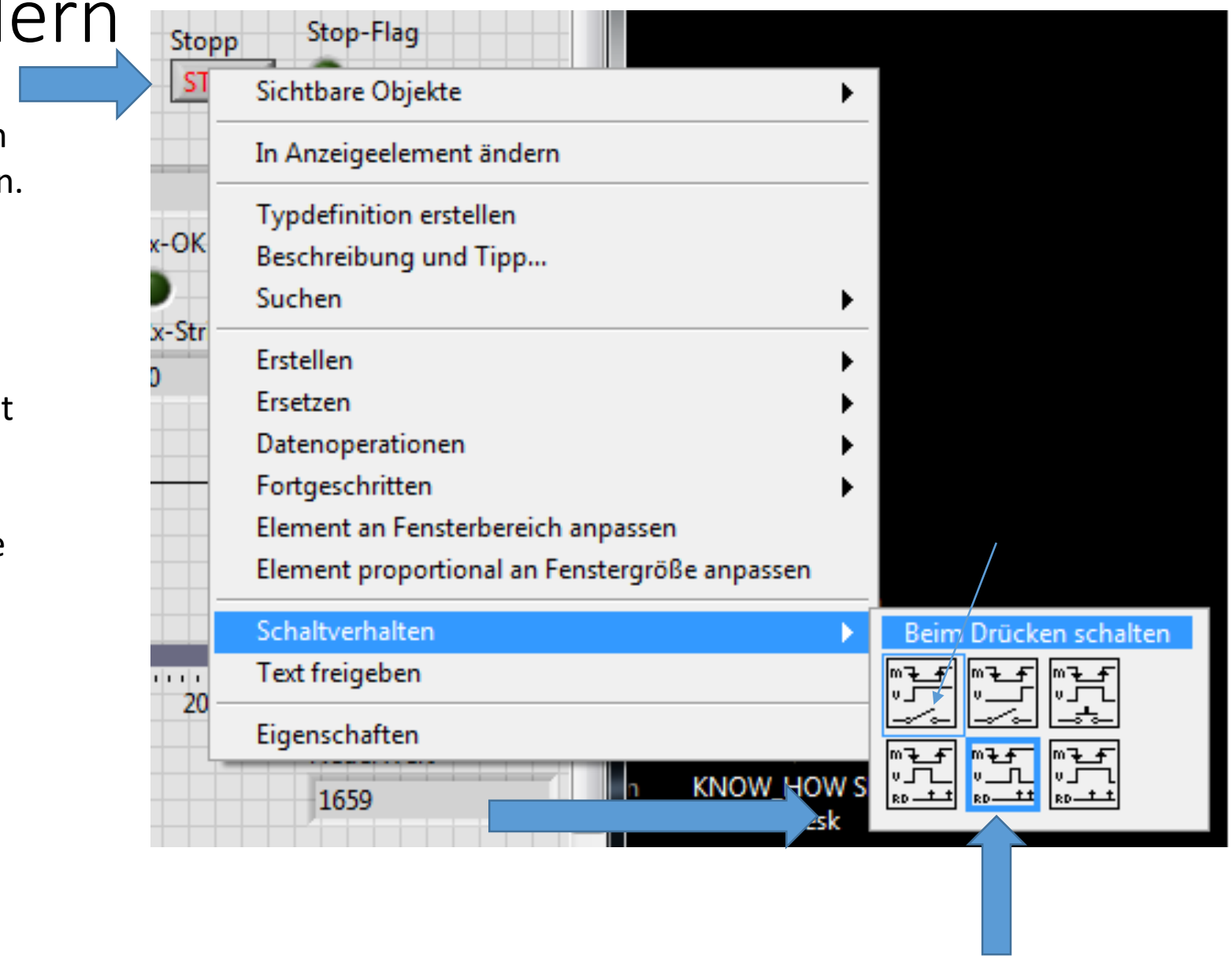
Schaltverhalten ändern

In Eventstukturen ist das Schaltverhalten von Button etwas kniffliger zu handhaben.

Es ist besser es so zumachen:
Ausgewertet soll das Drücken, nicht das loslassen.
Dazu kann das Schaltverhalten umgestellt werden.

Es ist jedoch anschließend notwendig die Schalter Eigenschaft manuell auf FALSE zurückzustellen. Da ist zugegeben etwas kompliziert, aber hilf- und lehrreich.



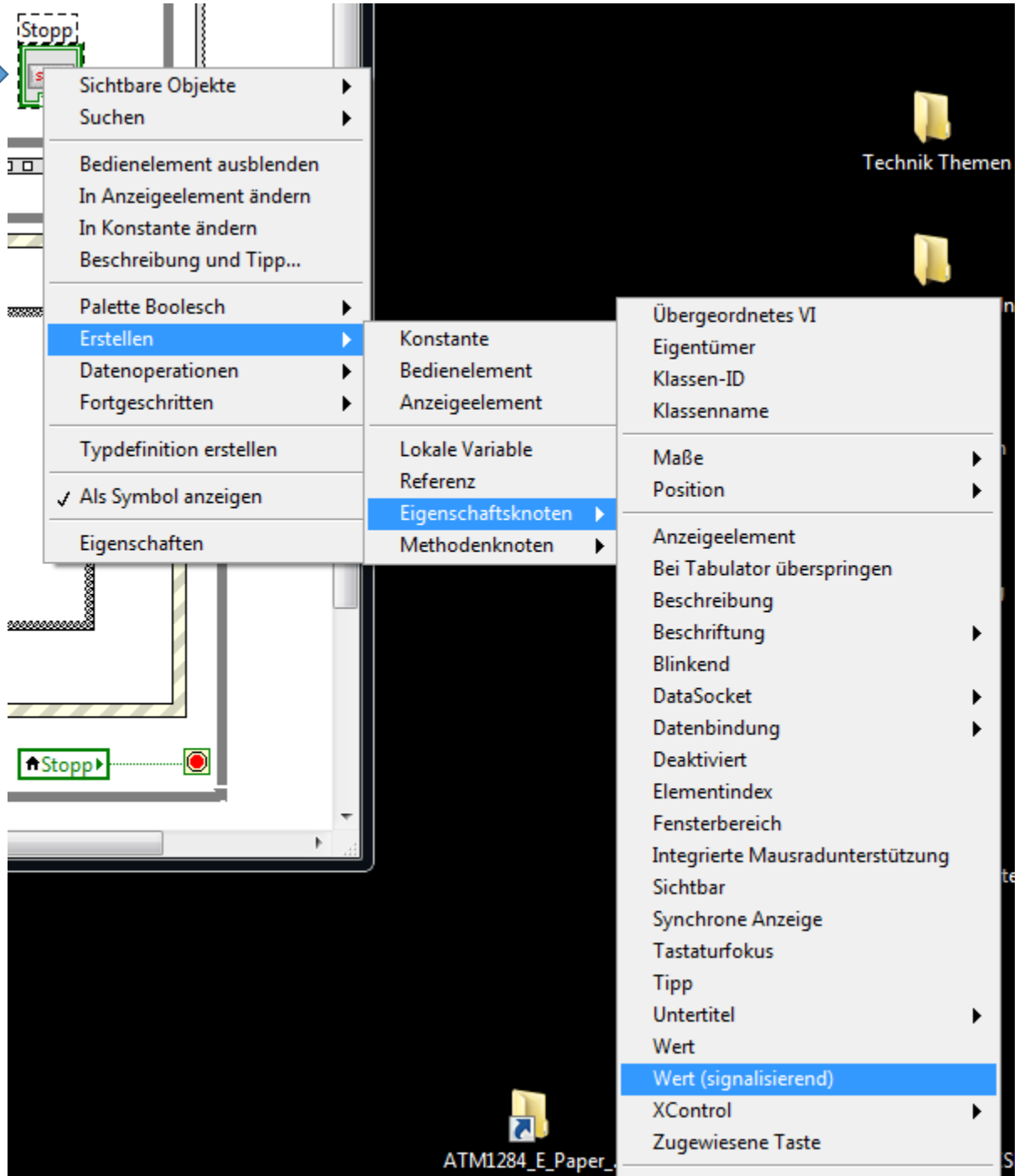


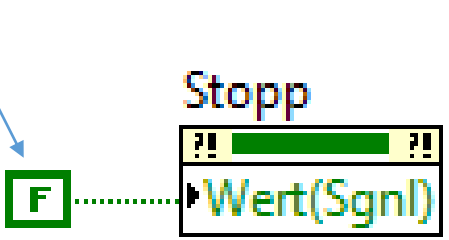
Jeder Schalter kann im Eigenschafts-knoten ‚**fernbedient**‘ werden.

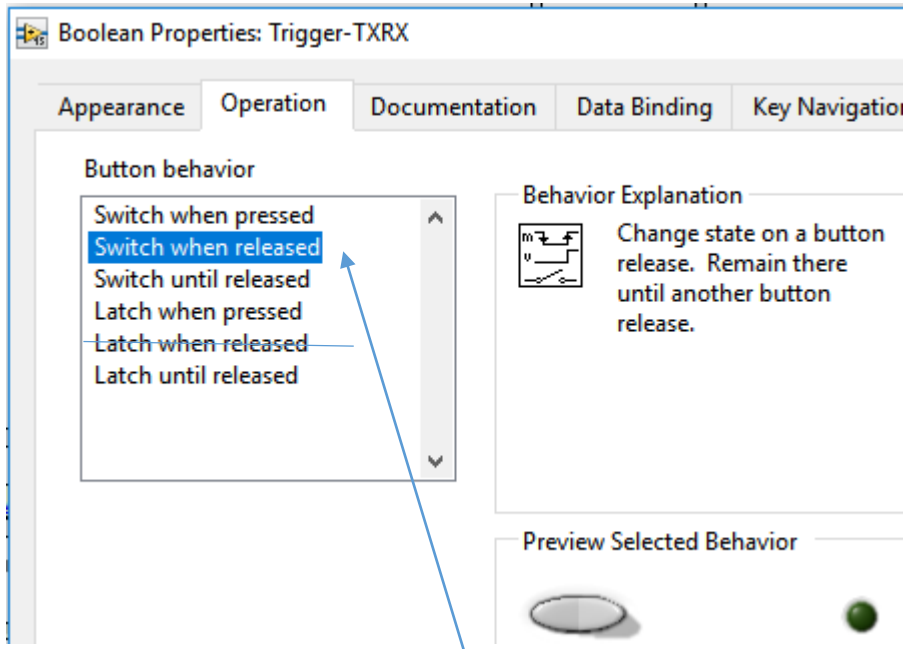
RC Marathon
Erstellen
→Eigenschafts-Knoten
→ Wert
→ (signalisierend)

→ FALSE auf Wert zuweisen

Dies meist am Ende der Eventausführung



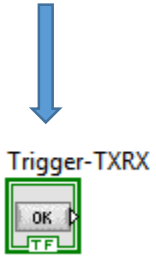


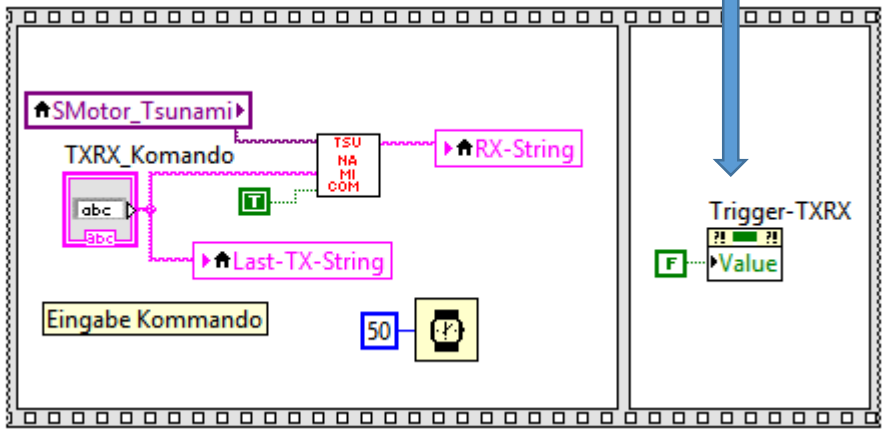


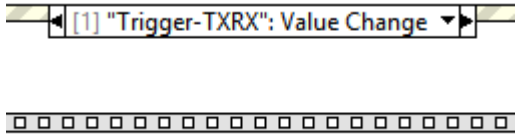
Event: **TASTER** innerhalb einer Eventstruktur
Der Taster ist

Eigenschaftsknoten vom Button erzeugen:
RC → Property node → „Value“ → In Schreiben ändern, **False** setzen

Button Property, Latch when pressed in **Switch when released** ändern







Event erzeugen: Signalisierend

Bevor ich die Synchronisationsmöglichen wie Semaphoren erwähne:
Konzeptänderung:

Die einfachste Möglichkeit Kollisionen von VI Zugriffen aus dem Weg zu gehen, #
Ist es nur EINE Einzige VI für Hardwarezugriffe etc. zu haben, die alle Anwendungen abdeckt.

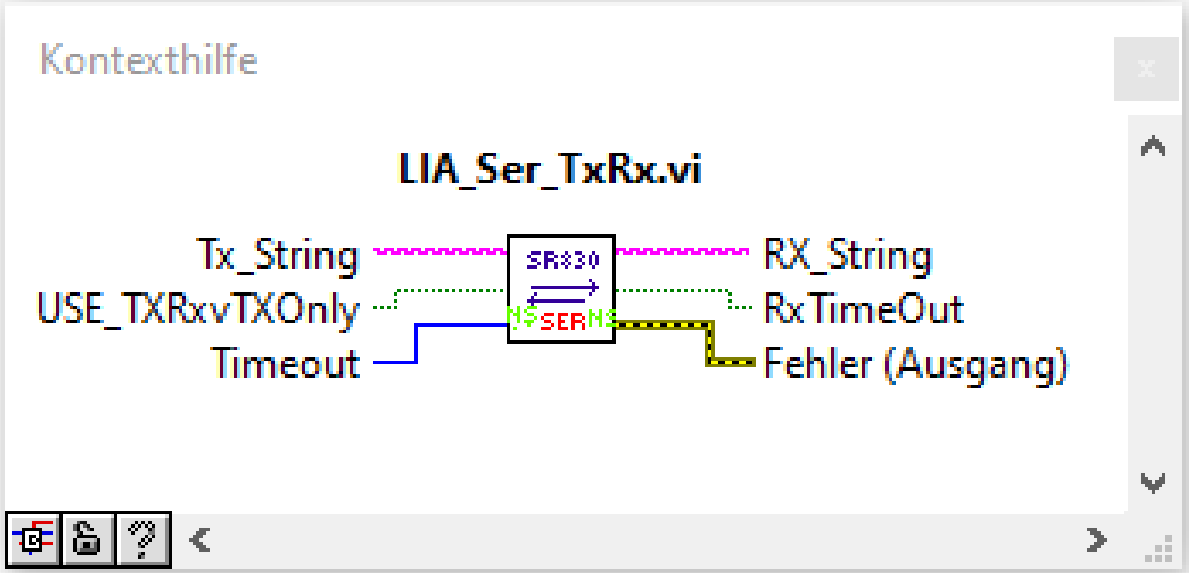
Bsp.

Also ein und dieselbe VI für das „NUR Senden“ und „Senden UND Empfangen“ für die serielle Schnittstelle (COM)

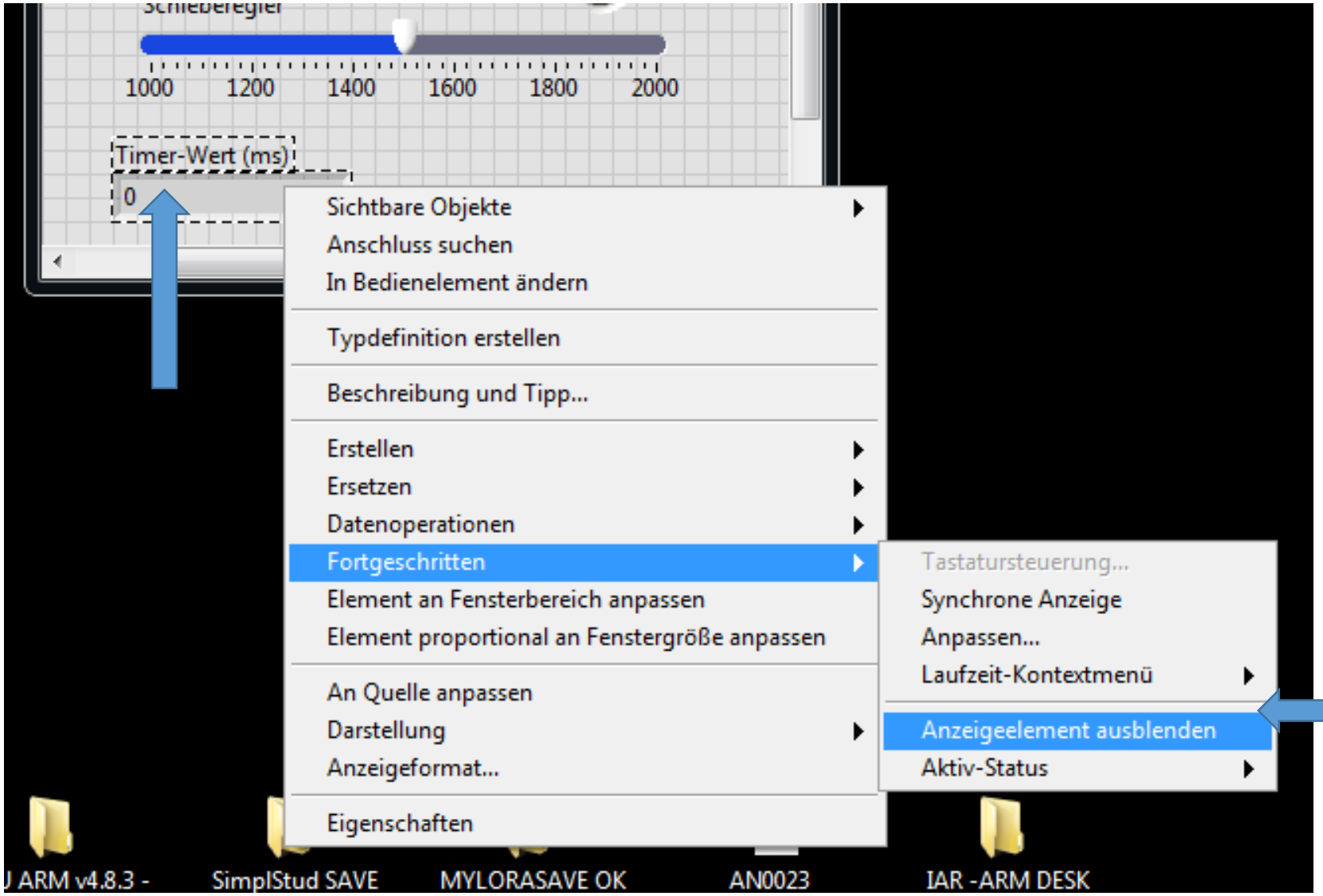
Gesteuert wird das hier z.B. mit einem:

FALSE für Tx Only

TRUE für TxRx



Elemente Ausblenden



Manchmal braucht man Variablen als Zwischenspeicher oder zur logischen Verarbeitung etc. Diese stören jedoch die Oberfläche. Man kann diese einfach ausblenden. Diese Kosmetik ist am Ende der Entwicklung sinnvoll.

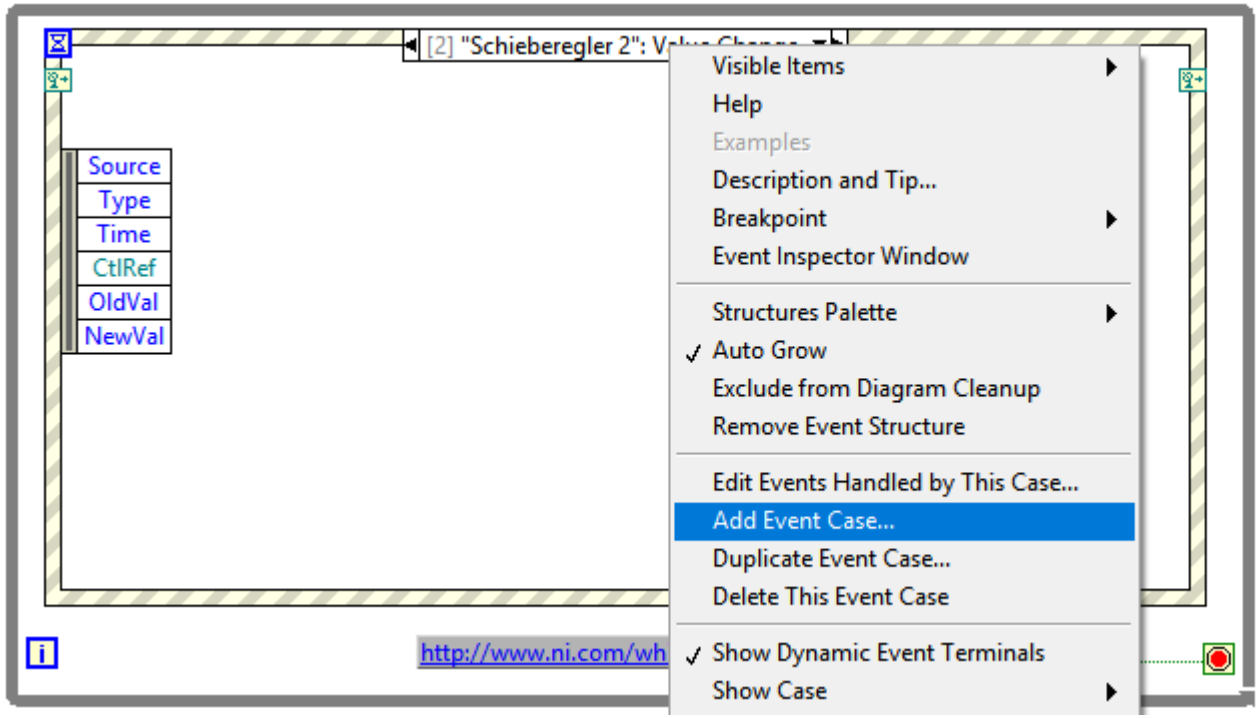
So kann man z.B. das für die Laufzeit unwichtige Semaphor Frontelement unsichtbar machen

Event Element Hinzufügen

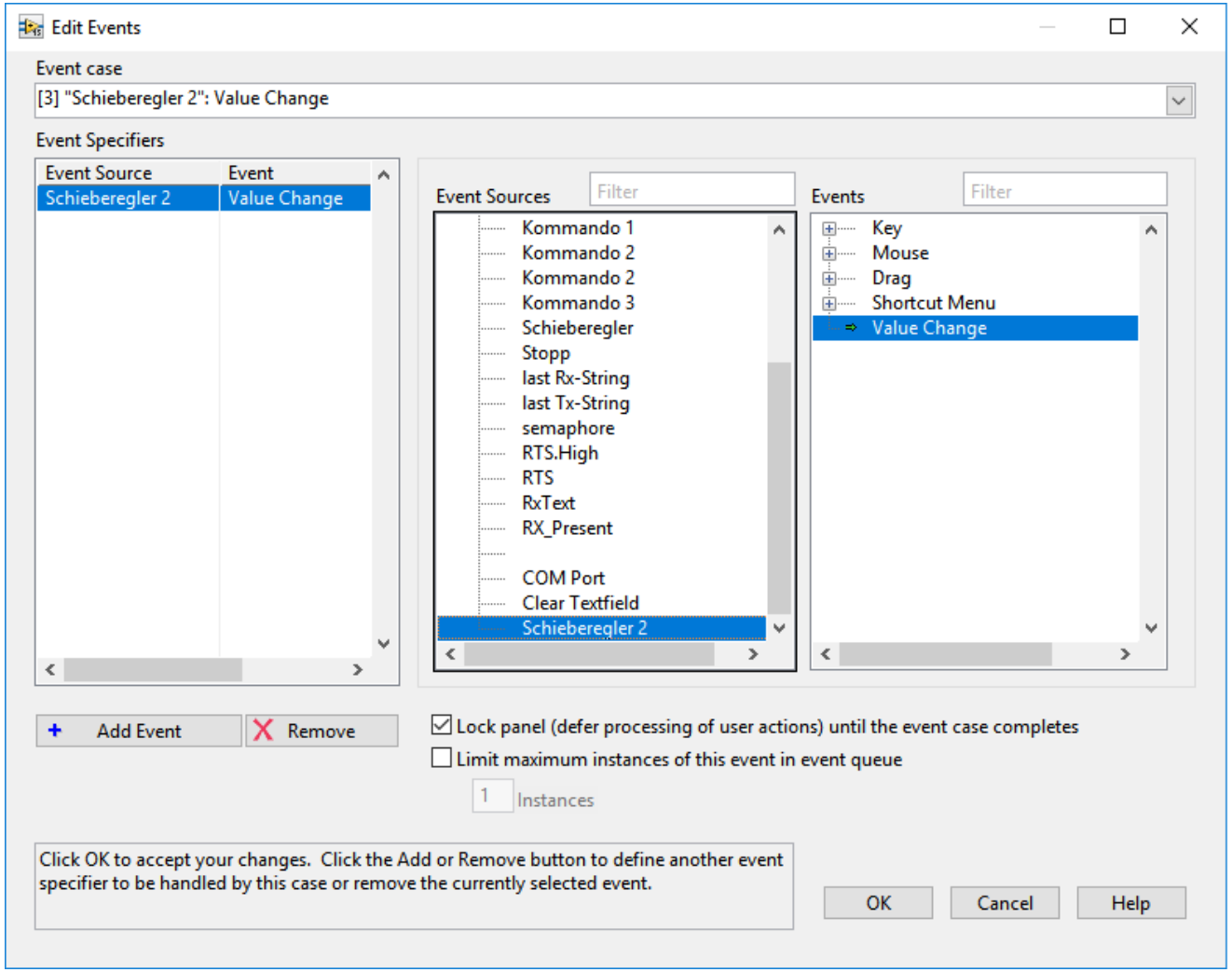
http://www.ni.com/white-paper/3331/en/

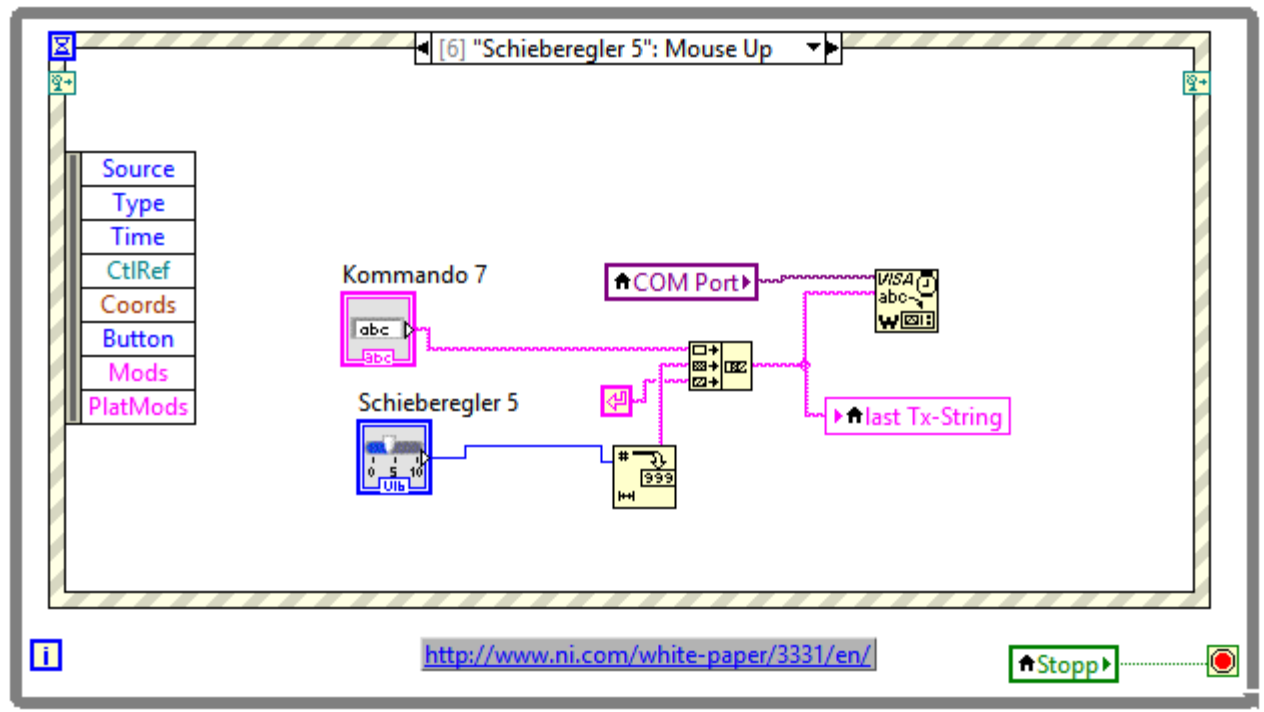
1.) Platziere neues Steuer Element… z.B: Schieberegler

2.) Add Event Case



3. Selektiere Steuerelement und wähle einen Event Case → Value Changed



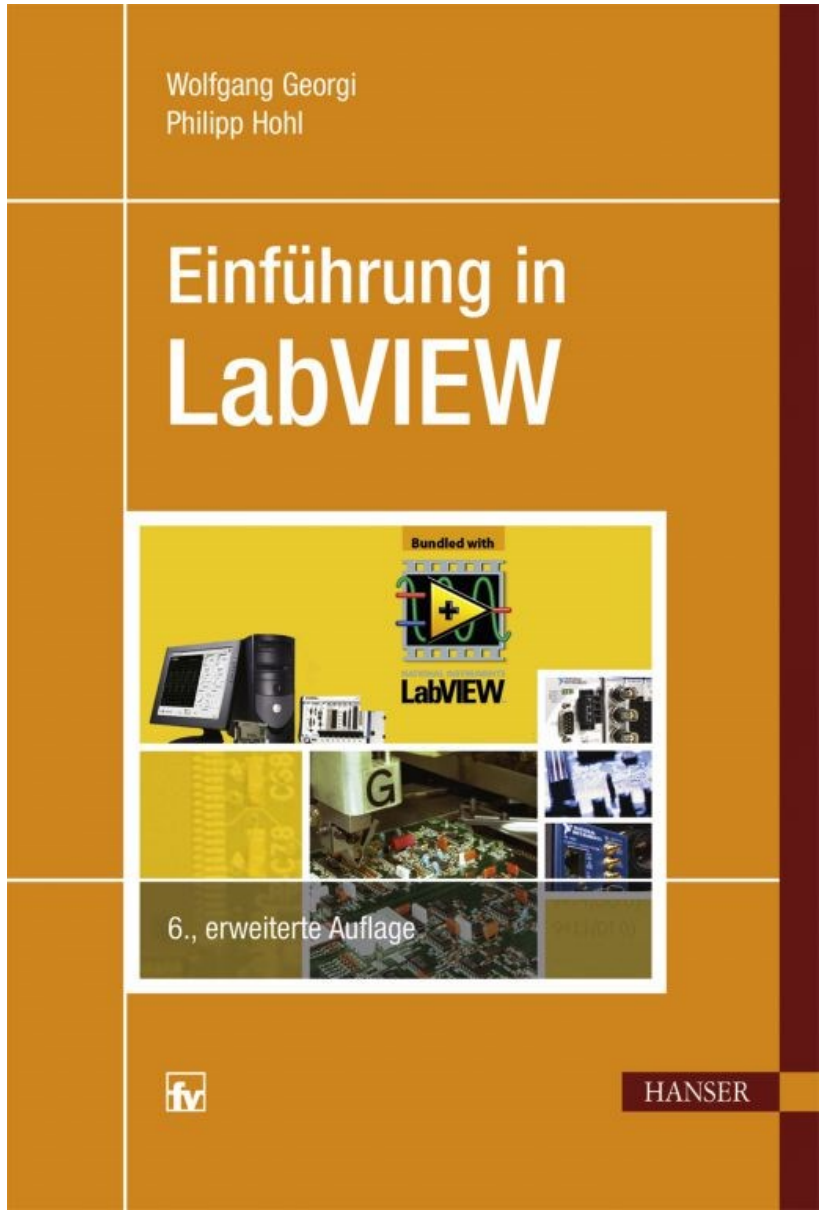

[6] "Schieberegler 5": Mouse Up
Source
Type
Time
CtlRef
Coords
Button
Mods
PlatMods
Kommando 7
COM Port
VISA
abc
Schieberegler 5
last Tx-String
http://www.ni.com/white-paper/3331/en/
Stopp

Schreiben - Lesen zu Platte

Sonderzeichen in Dateien
EOF = End of File (Systemabhängig) → Abc<EOF> kann eine „-1“ sein
LF= EOL = End Of Line 10=0x0A =
CR = 13 = 0x0D
Returntaste erzeugt Linefeed = LF = → 0A 0D
In LV wird konvertiert, und 0D Weggeschnitten, sofern man im Kontextmenü der Dateifunktion „V“ vor EOL konvertieren gesetzt hat
ABC <LF> DEF → 414243 **0A** 44 45 46
ABD
DEF

Quellen

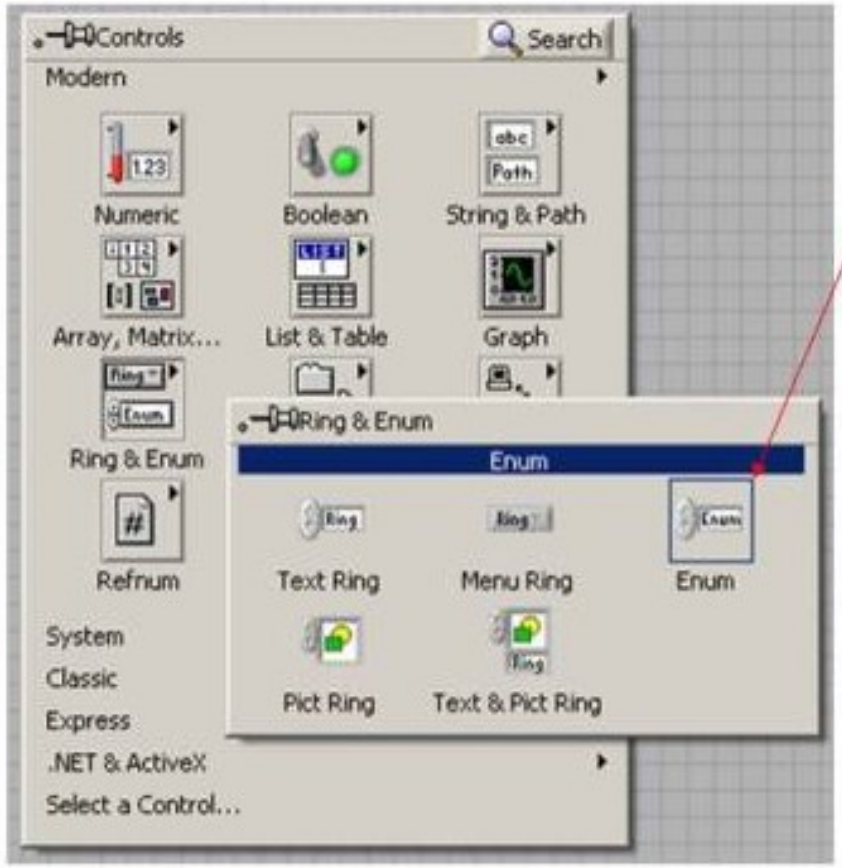
Buchempfehlung und Quelle für Beispiele
Steht z.B. in der Lehrstuhl Handapparat, Lst Lupton
8443 ST 250 L03 G352
http://www.geho-labview.de/



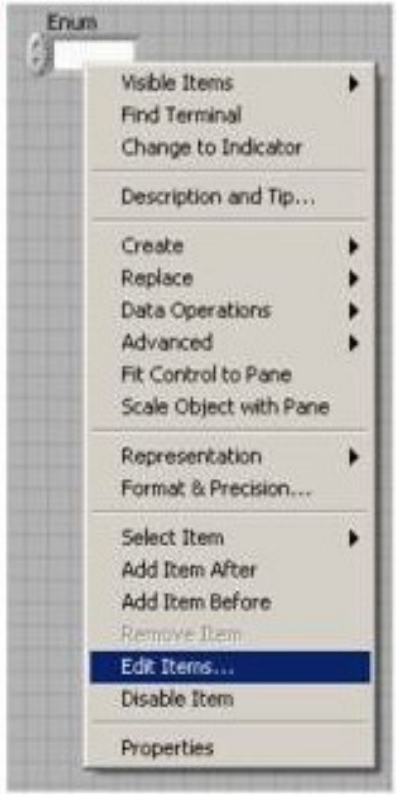
State Machine -- Zustandsautomat

Buch S:357

- Eine Zustandsmaschine ermöglicht die Ausführung von Zuständen, abhängig von Ereignissen
  - Die einfache Zustandsmaschine (State-Machine) VI **Ü13** führt im jeweiligen Zustand die jeweils gewählte Operation aus.
  - Die Eingangsgrößen sind für jeden Zustand gleich.



- Im Frontpanel das Eingabefeld Enum wählen.

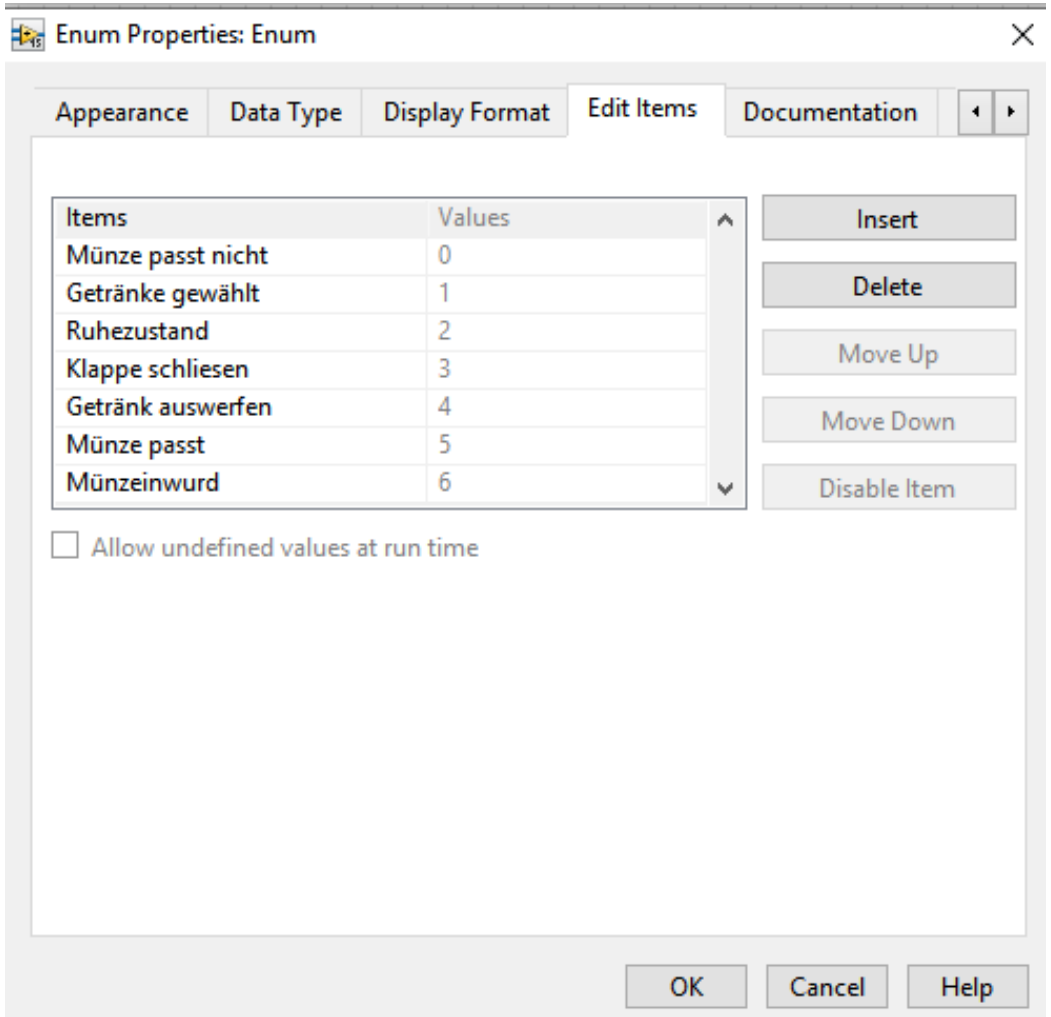


- Rechtsklick auf Enum -> Edit Items ...
  - ➢ Zustände eintragen

https://www.fu-net.de/lv/lv01/

ENUM → RC auf Objekt → Edit Items

Aus Strings werden Nummern.
Damit lässt sich schön lesbar und leicht arbeiten.



• Hier gezeigte Reihenfolge bei Erstellung einhalten!

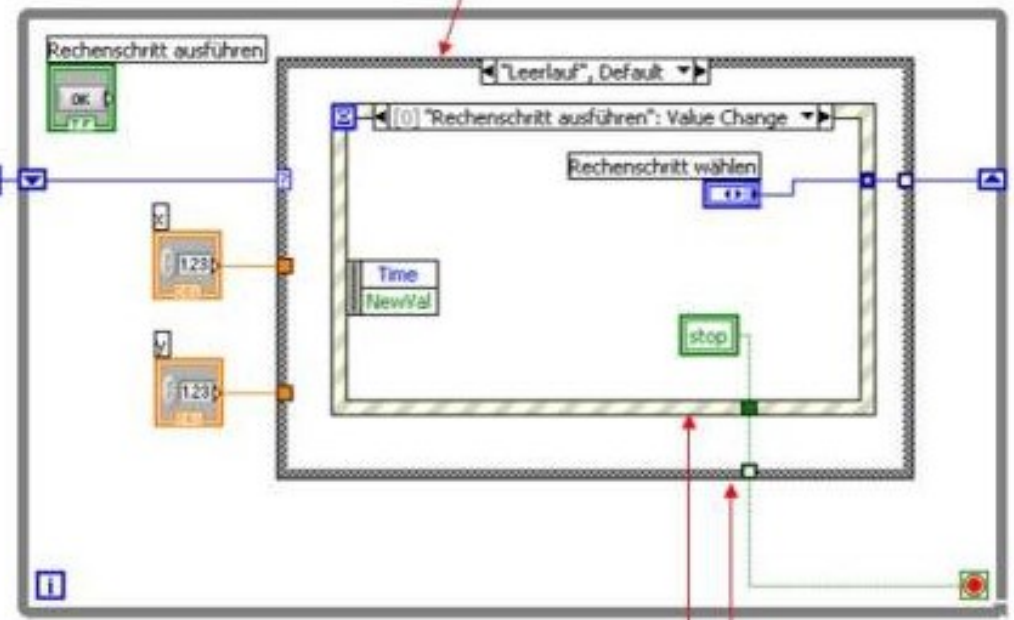
• Frontpanel

- Control **Rechenschritt ausführen** (Boolean: OK-Button)
- Control **Stop** (Boolean)
- Control **x** (Numeric)
- Control **y** (Numeric)

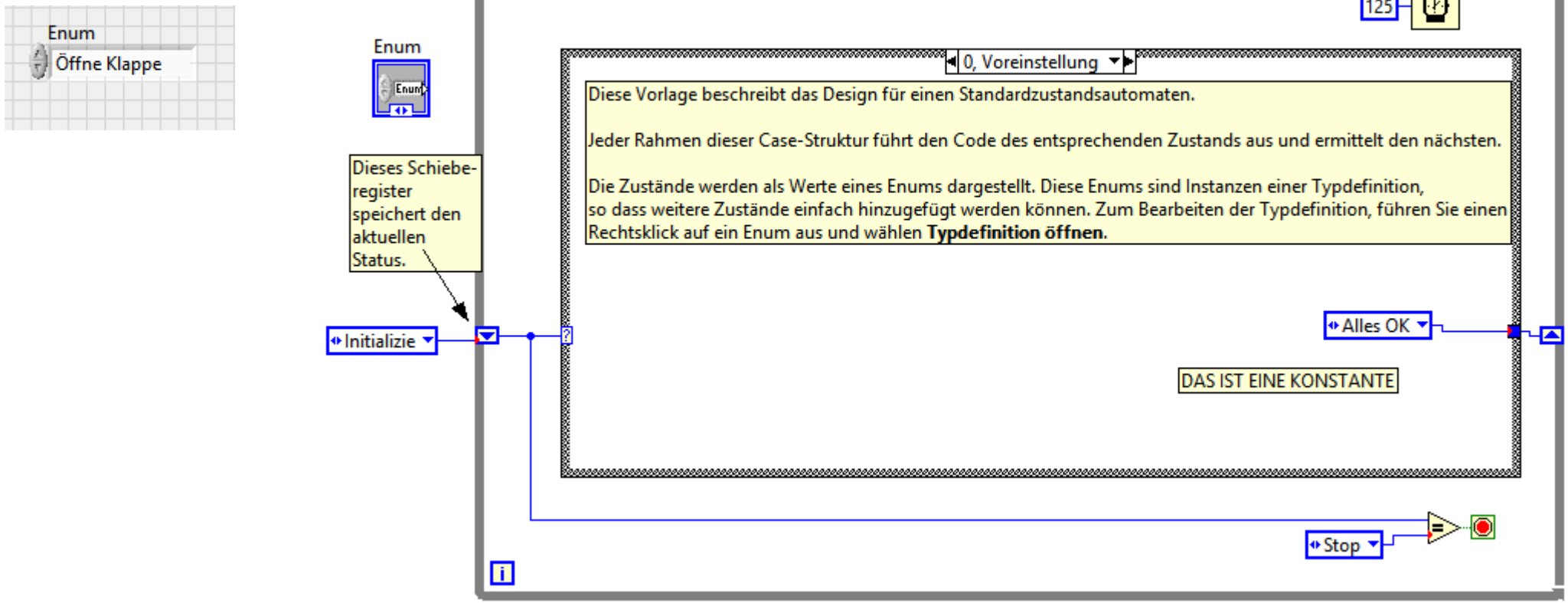
• Blockdiagramm

6. Rechts-Klick auf Shift-Register
   -> Constant erzeugen
7. Rechts-Klick auf Case-Structure
   -> Make This Case „*Leerlauf*“, *Default*
1. While-Loop
2. Case-Structure
3. Event-Structure
4. Shift-Register in While-Loop
5. Control **Enum** „Rechenschritt wählen“ in Event-Structure „Rechenschritt ausführen“ mit Shift-Register (rechter Rand) verbinden.
   - Rechts-Klick auf Event-Structure *[0] Timeout* → Edit Events Handled by This Case → *„Rechenschritt ausführen“: Value Change*
8. Rechts-Klick auf Structure-Durchgang => „Use Default if Unwired“

9. Rechts-Klick Case Structure
   -> Add Case for Every Value



Lies weiter unter: https://www.fu-net.de/lv/lv01/


Enum
Öffne Klappe
Enum
Dieses Schiebe-register speichert den aktuellen Status.
Initialize
125
0, Voreinstellung
Diese Vorlage beschreibt das Design für einen Standardzustandsautomaten.
Jeder Rahmen dieser Case-Struktur führt den Code des entsprechenden Zustands aus und ermittelt den nächsten.
Die Zustände werden als Werte eines Enums dargestellt. Diese Enums sind Instanzen einer Typdefinition, so dass weitere Zustände einfach hinzugefügt werden können. Zum Bearbeiten der Typdefinition, führen Sie einen Rechtsklick auf ein Enum aus und wählen Typdefinition öffnen.
Alles OK
DAS IST EINE KONSTANTE
Stop

Literatur:

Onlinekurs: https://www.fu-net.de/lv/lv01/

KURS WIRD FORTGEFÜHRT (eher spontan)
Nutzung mit LabView 15.

Materialien (Uni Rgbg intern NAS Server)

Übungsbeispiele

Share_AGS→V:\LabViewKurs

Share_AGS→ V:\LabViewKurs\Lab-View-Kurzlehrgang-Exampels_LV15

Share_AGS→\V:\LabView_Basis_VI‘s

Wer Beratung möchte, kann mich im Büro jederzeit dafür erreichen.

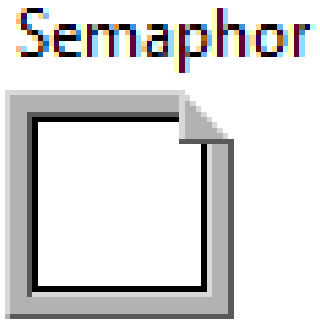
Christof Ermer

Christof.Ermer@ur.de
0941 -9432140
0179-243170

Sollten mehrere Interesse haben ist auch ein außerordentliches kleines Seminar im Seminarraum möglich.

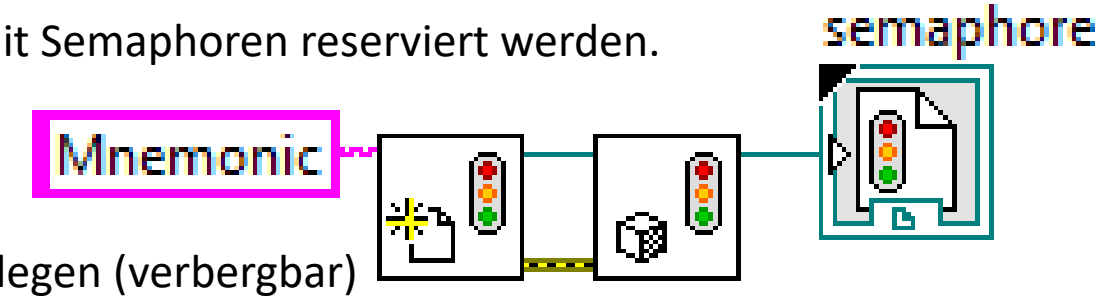
Semaphoren

..dienen in Multitasking Umgebungen dazu,
A) Tasks Ausführungen zu Organisieren,
B) Ausführungsobjekte für Zugriffe zu reservieren.



Problem:
Task 1: öffnet ein Port.
Task 2: schließt ein Port mitten in der Ausführung von Task 1.
Das führt zu schwerwiegenden Kollisionen.
Deshalb müssen **Doppelzugriff**s-gefährdete Objekte, DAten mit Semaphoren reserviert werden.

1. Schritt einen Semaphor anfordern
2. Sinnvollerweise Semaphor Status abfragen (ob OK)
3. Semaphor in eigenes Semaphoren Frontpanel Element ablegen (verbergbar)



4. Am Programmemde Semaphor wieder freigeben.

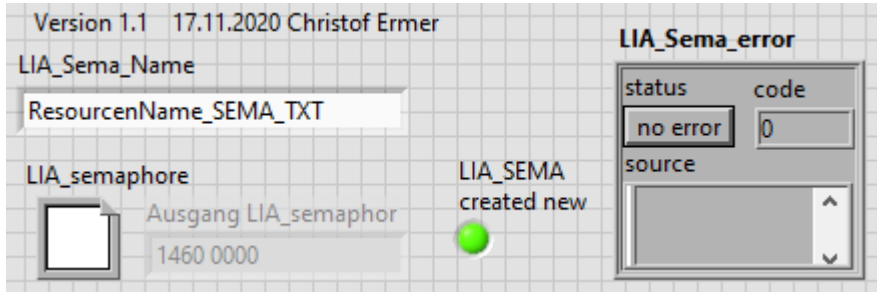
Der Semaphor kann nur verwendet werden um in Unter-VIs Reservierungen auszuführen.
Anderes Tasks warten solange mit dem Zugriff, bis die Reservierung wieder freigegeben wird,

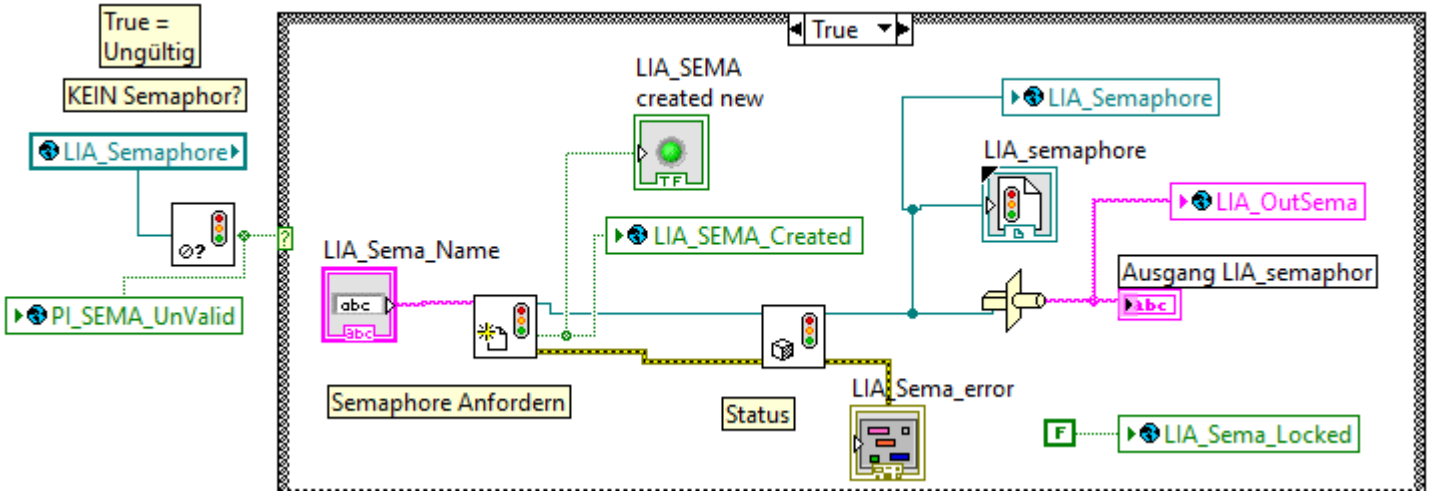
Folgendes würde ich heute nicht mehr so machen

Es diente dazu Kollisionen im Parallel Betrieb durch die Event Struktur zu vermeiden. Das kann umgangen werden wenn man „schlau“ nur eine Kollision gefährdete VI benutzt

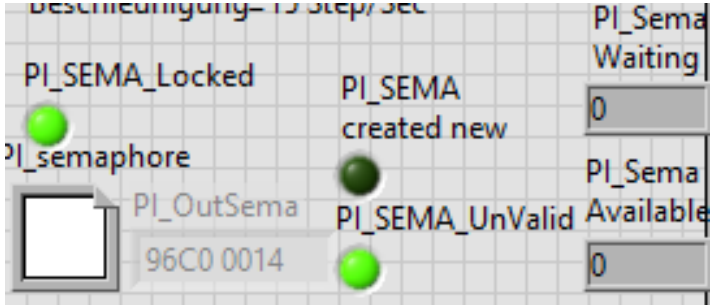
Semaphor VIs: (Semaphor = Zeuchenträger)
Help: https://zone.ni.com/reference/de-XX/help/371361R-0113/glang/semaphore_vis/
Ziel: Vor gleichzeitigem Zugriff geschützte System Resourcen.

Erzeuge Semaphoren Resource

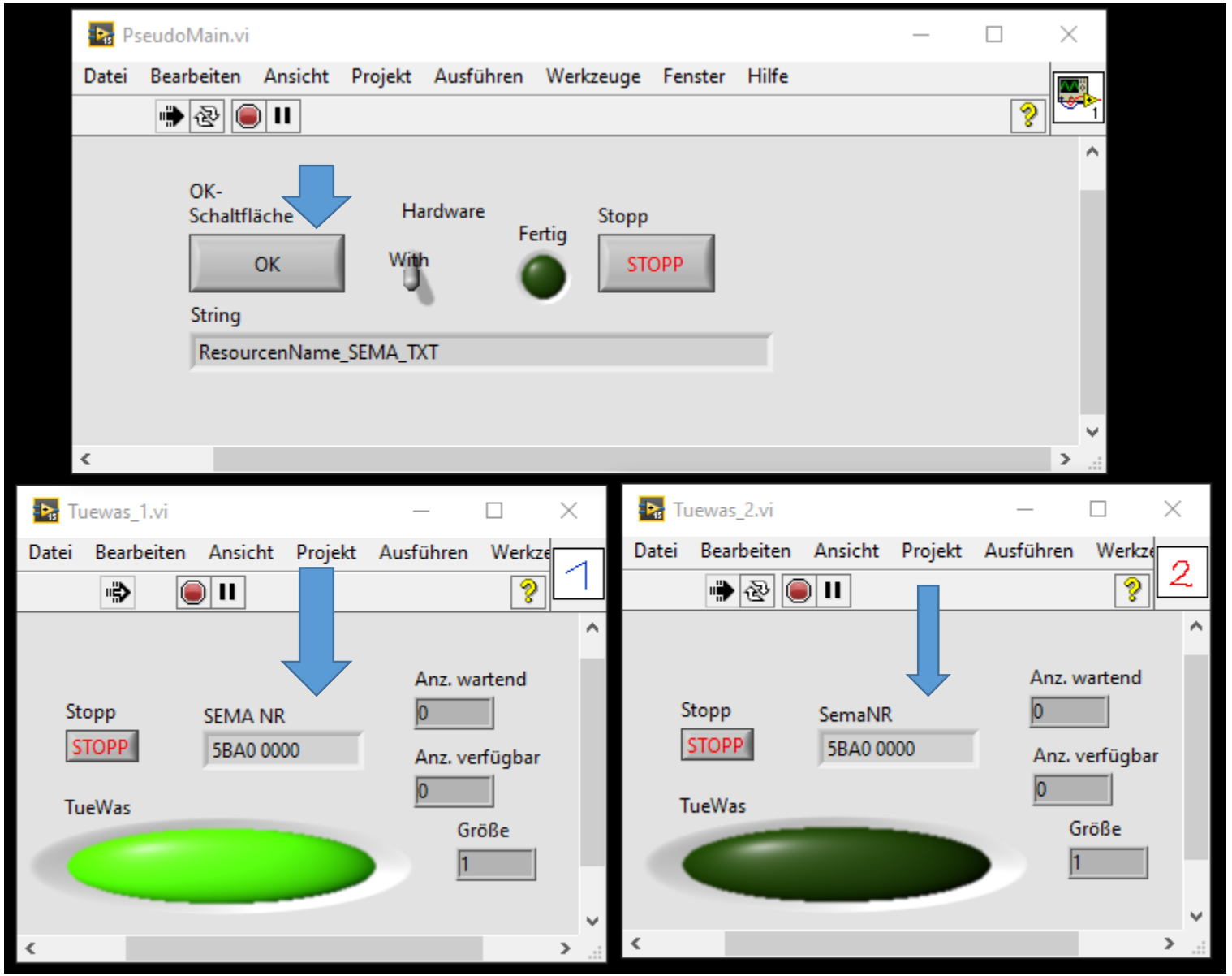




Übergeordnetes Programm muss laufen
Auszug ausGlobal.vi



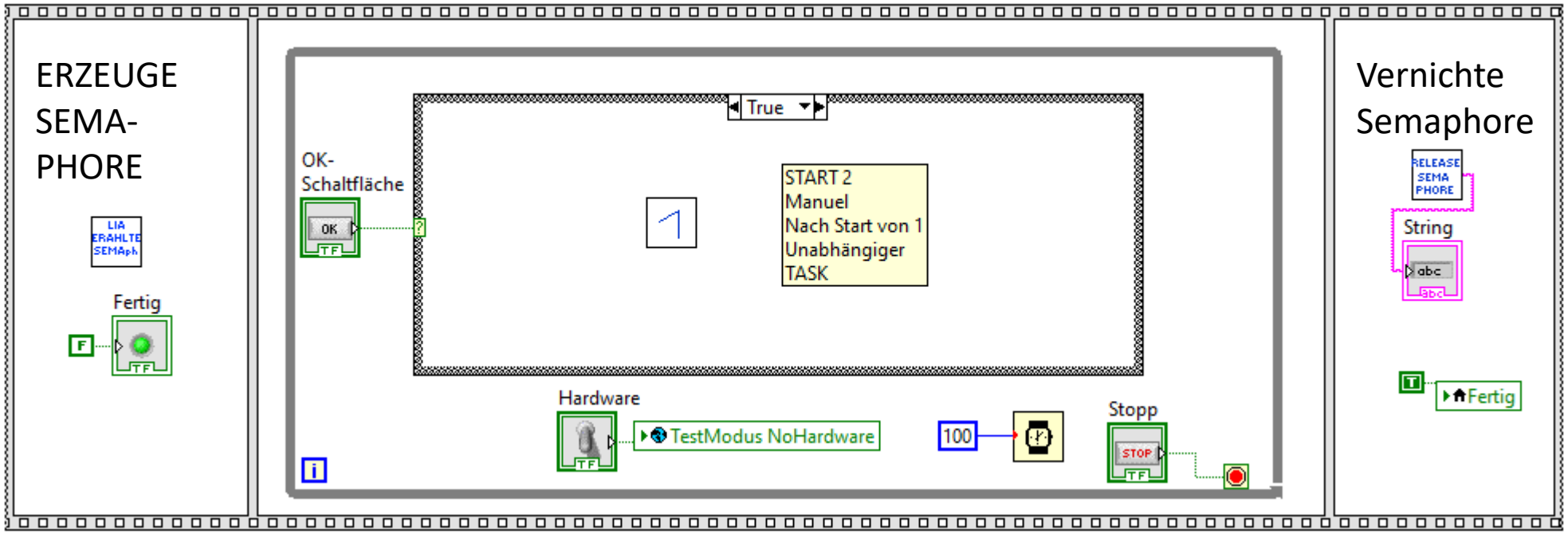
Hauptprogramm
Verwaltet
Globale VI mit
Datenspeicher
des Semaphores



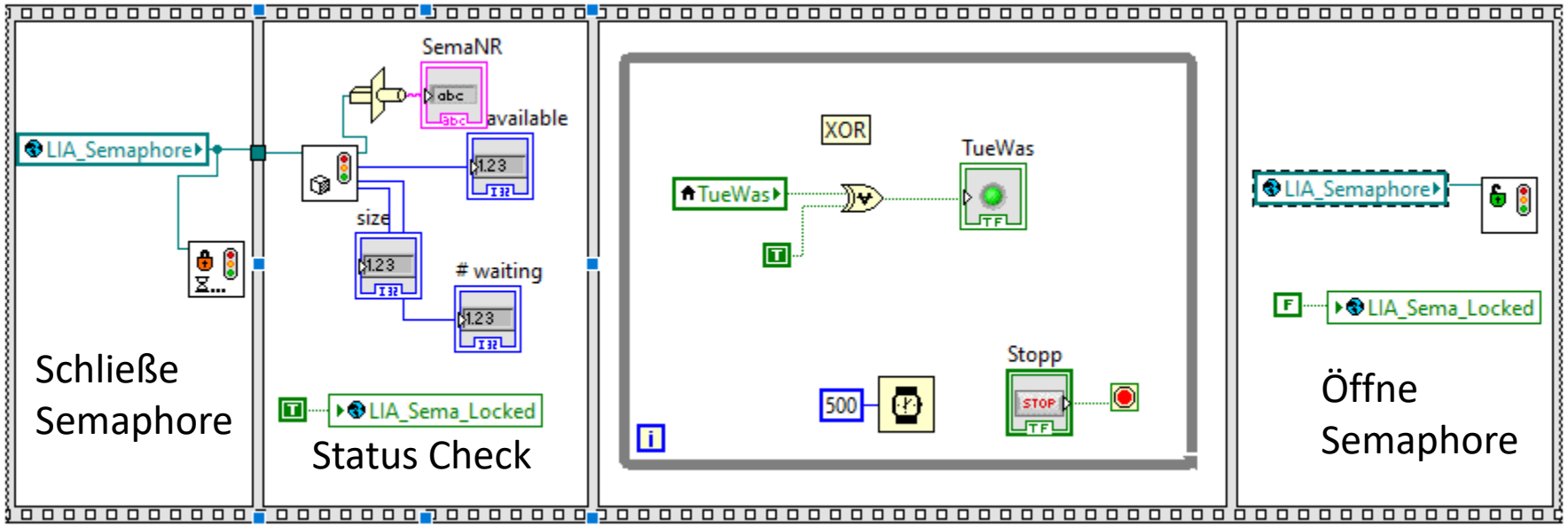
Prog 1 wird im
Hauptprogramm
gestartet

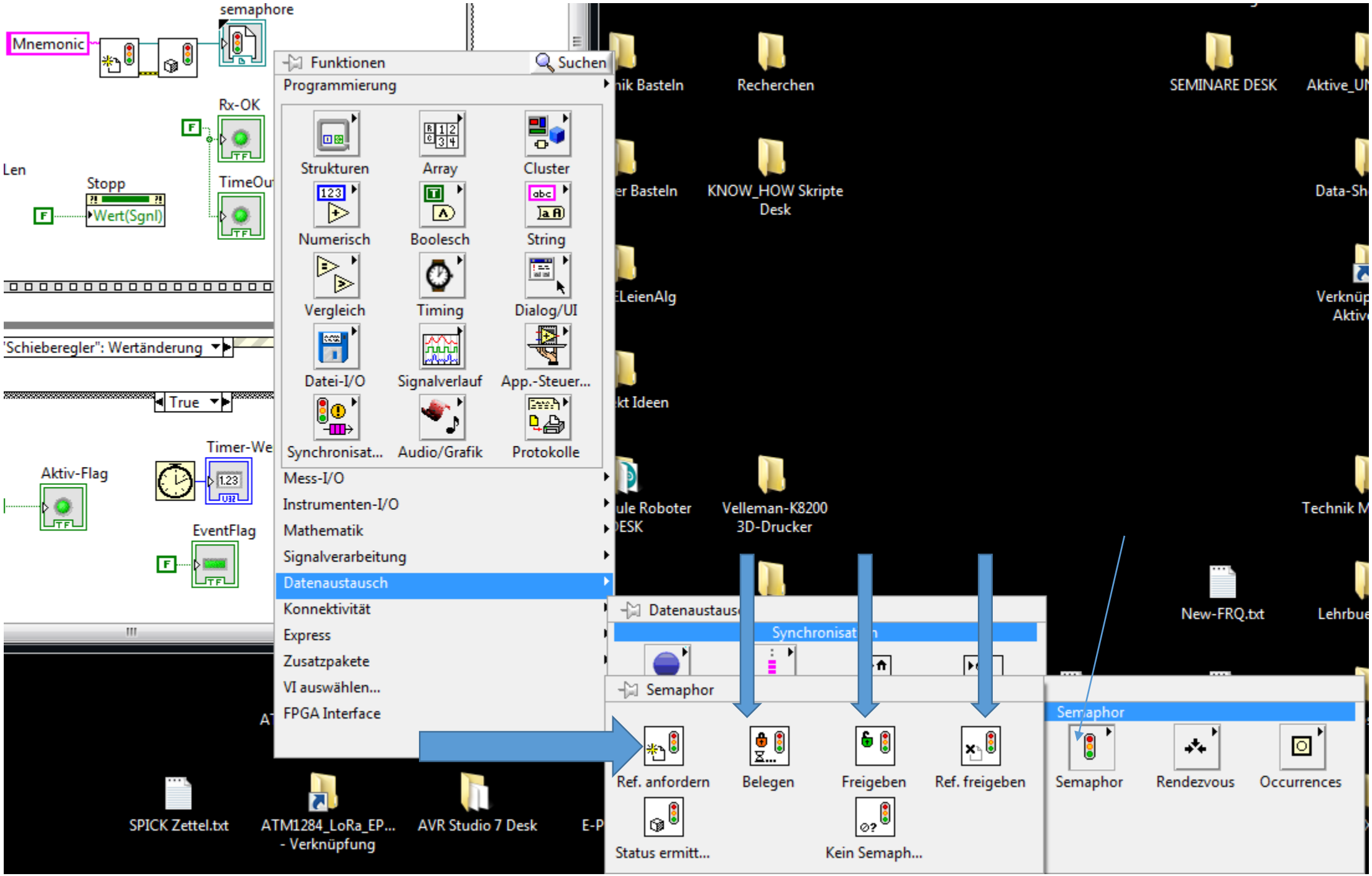
Prog 2 ist völlig
unabhängig, aber
wartet auf Ende
von Porg 1,
da dies die selbe
globale VI mit
dem SEMA
Adressspeicher
ausliest

MAIN

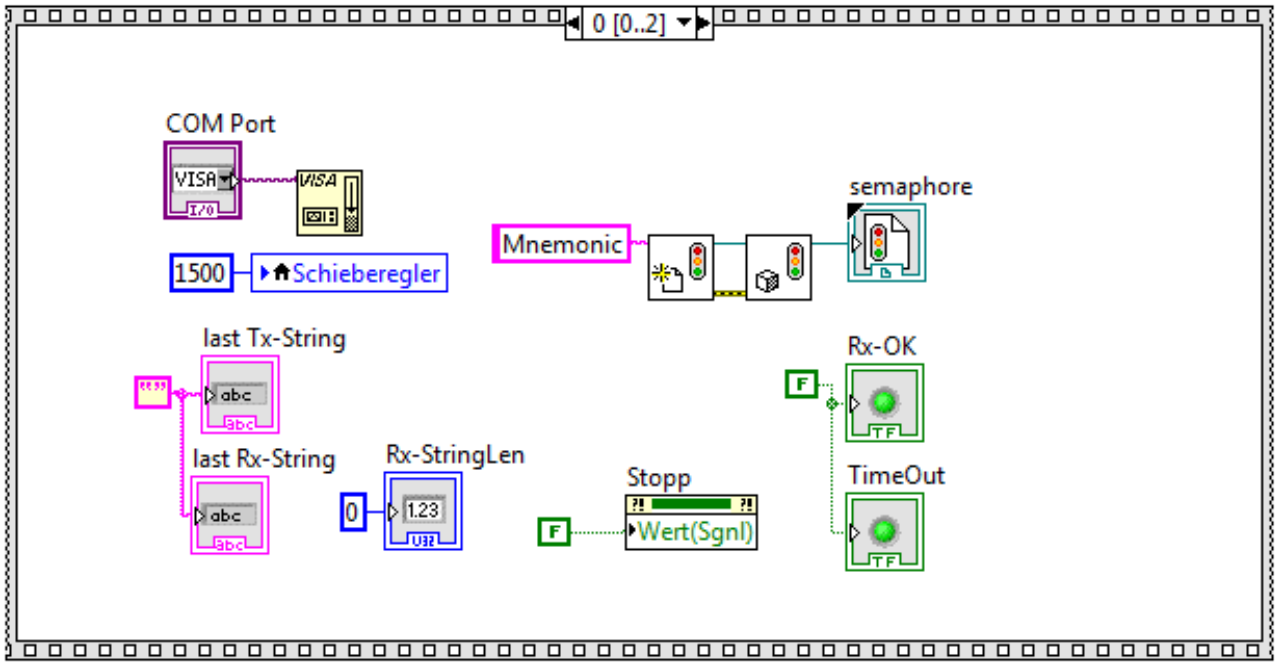

ERZEUGE SEMA-PHORE
LIA ERHALTE SEMAph
Fertig
OK-Schaltfläche
OK
TF
True
START 2
Manuel
Nach Start von 1
Unabhängiger
TASK
Hardware
TestModus NoHardware
100
Stopp
STOP
Vernichte Semaphore
RELEASE SEMA PHORE
String
abc
Fertig

SUB VI


LIA_Semaphore
SemaNR
abc
available
size
waiting
1.23
I32
Schließe Semaphore
LIA_Sema_Locked
Status Check
XOR
TueWas
500
Stopp
STOP
TF
Öffne Semaphore
LIA_Sema_Locked

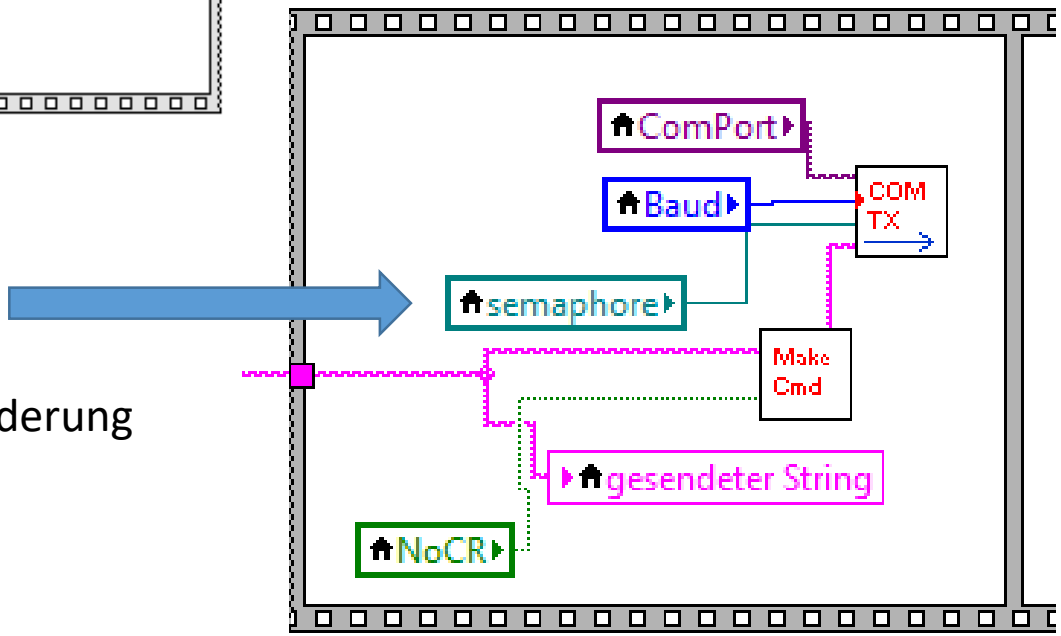

semaphore
Mnemonic
Rx-OK
Len
Stopp
Wert(Sgnl)
TimeOut
"Schieberegler": Wertänderung
True
Timer-We
Aktiv-Flag
EventFlag
Funktionen
Suchen
Programmierung
Strukturen
Array
Cluster
Numerisch
Boolesch
String
Vergleich
Timing
Dialog/UI
Datei-I/O
Signalverlauf
App.-Steuer...
Synchronisat...
Audio/Grafik
Protokolle
Mess-I/O
Instrumenten-I/O
Mathematik
Signalverarbeitung
Datenaustausch
Konnektivität
Express
Zusatzpakete
VI auswählen...
FPGA Interface
Semaphor
Ref. anfordern
Belegen
Freigeben
Ref. freigeben
Status ermitt...
Kein Semaph...
Semaphor
Rendezvous
Occurrences
Recherchen
SEMINARE DESK
KNOW_HOW Skripte Desk
Velleman-K8200 3D-Drucker
New-FRQ.txt
SPICK Zettel.txt
ATM1284_LoRa_EP... - Verknüpfung
AVR Studio 7 Desk

Start



Ende

Ausführung im VI.

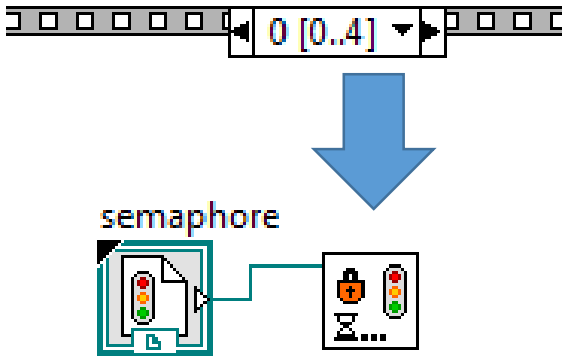


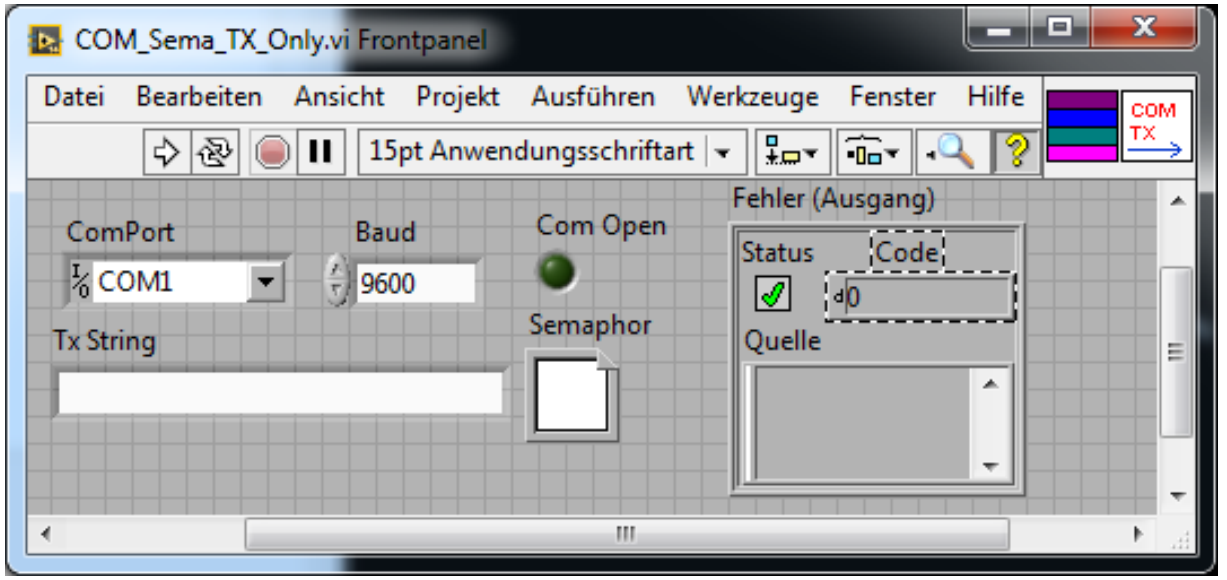
Reservierungsanforderung durch Übergabe als Parameter

Semaphor in Sub-VIs

Phase **1**: **Semaphor belegen**

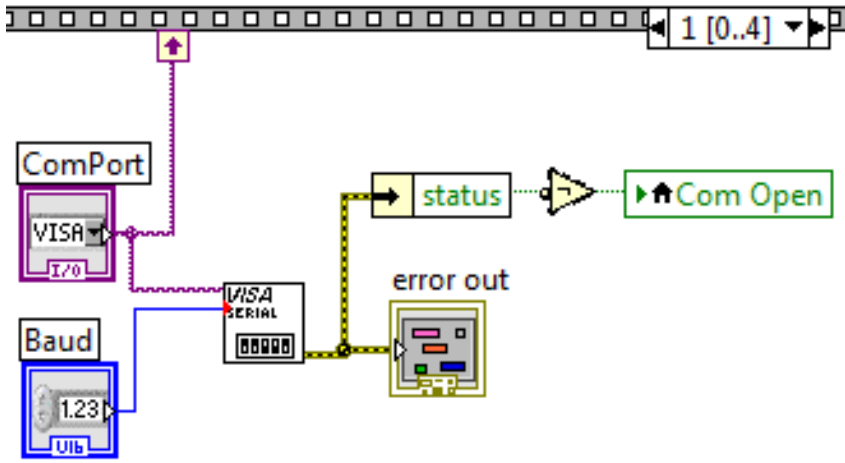
Jetzt gilt die Zugriff Resevierung



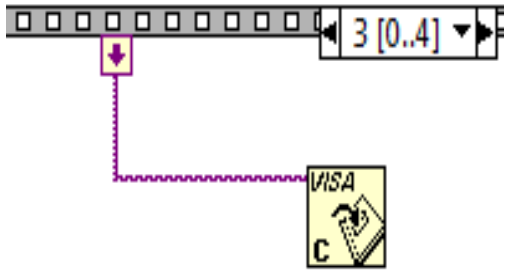


Phase **2**: Jetzt den ganzen Job erledigen, so wie gewohnt..

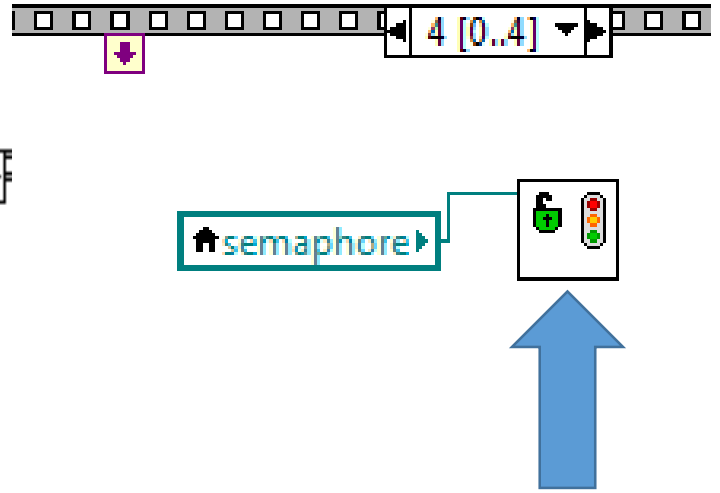
Port Open

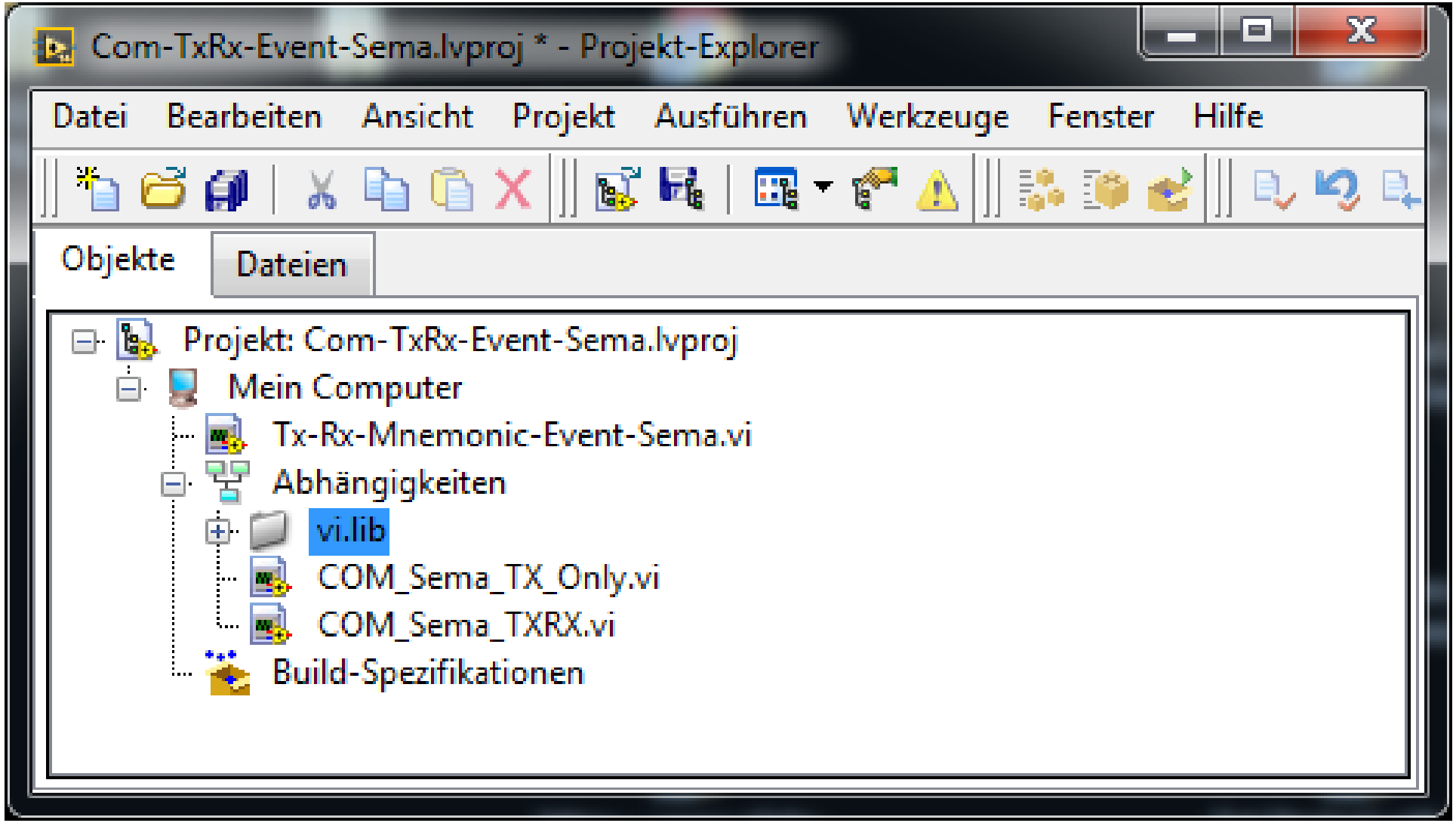


Port Close

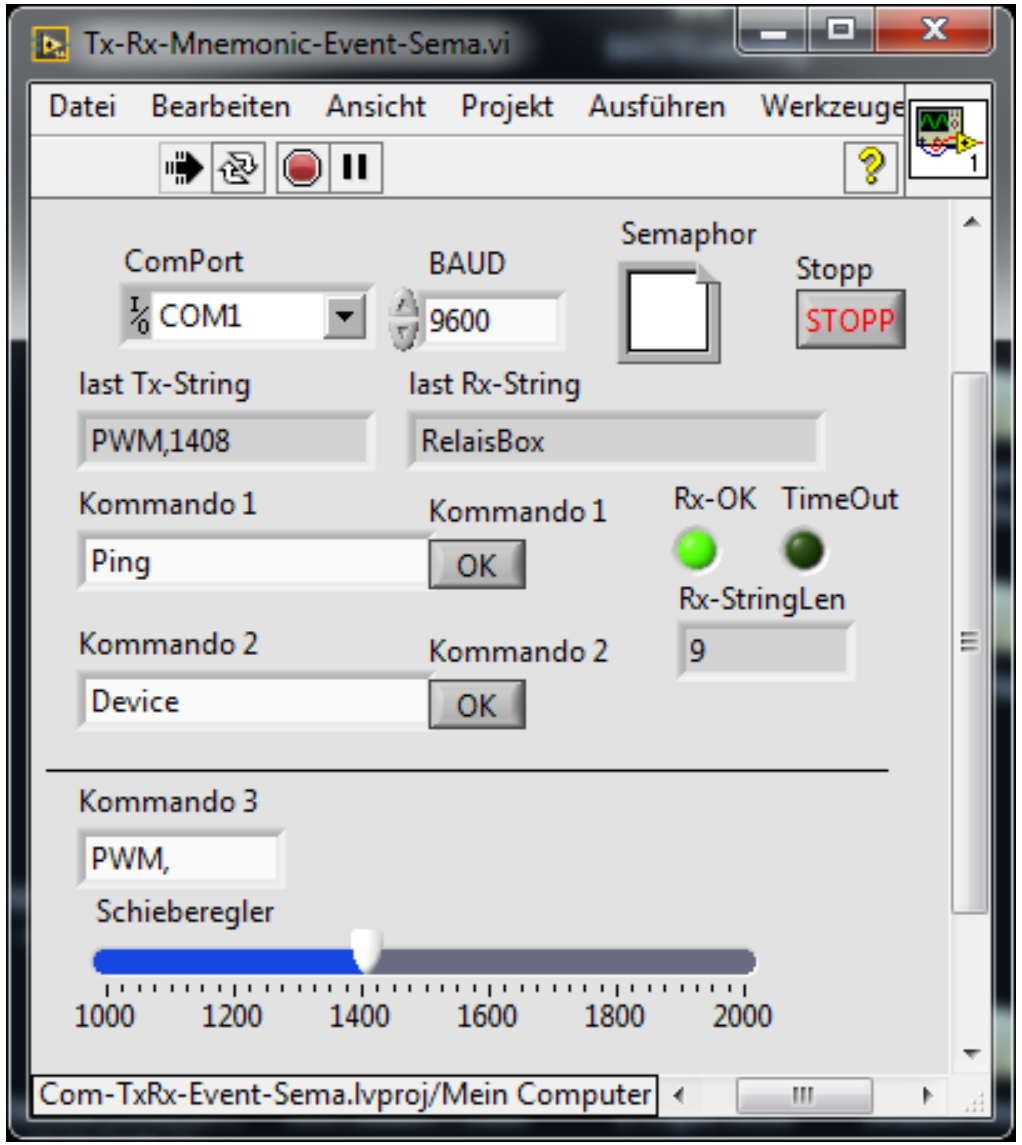


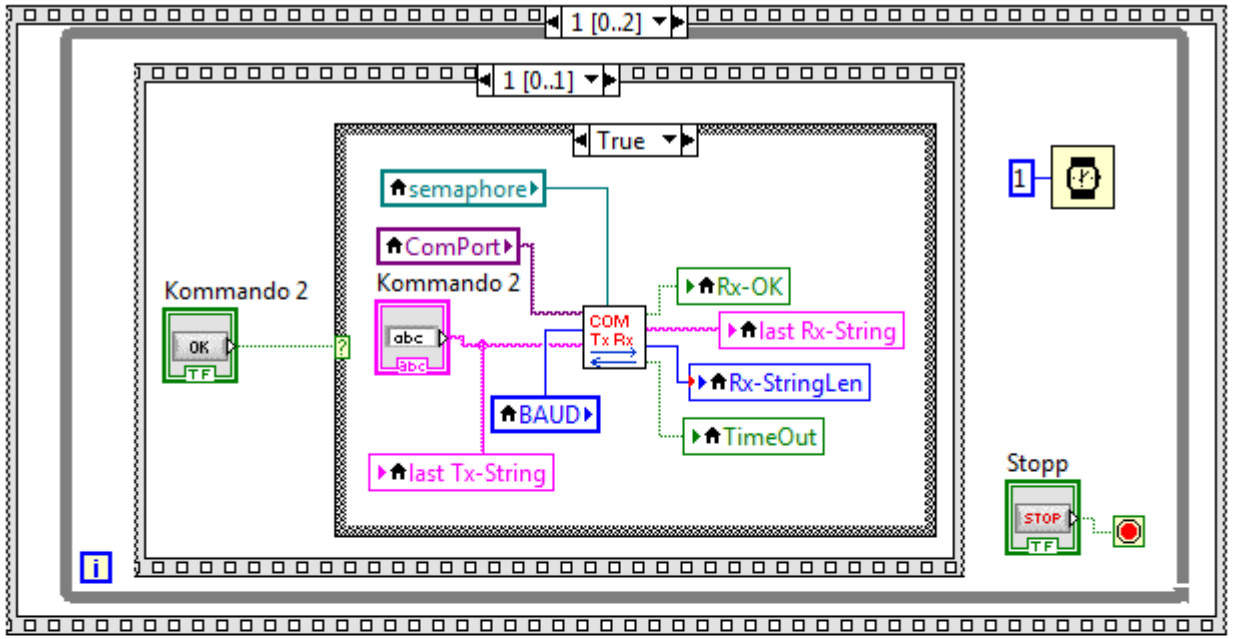
Phase **3**: **Semaphor freigeben**

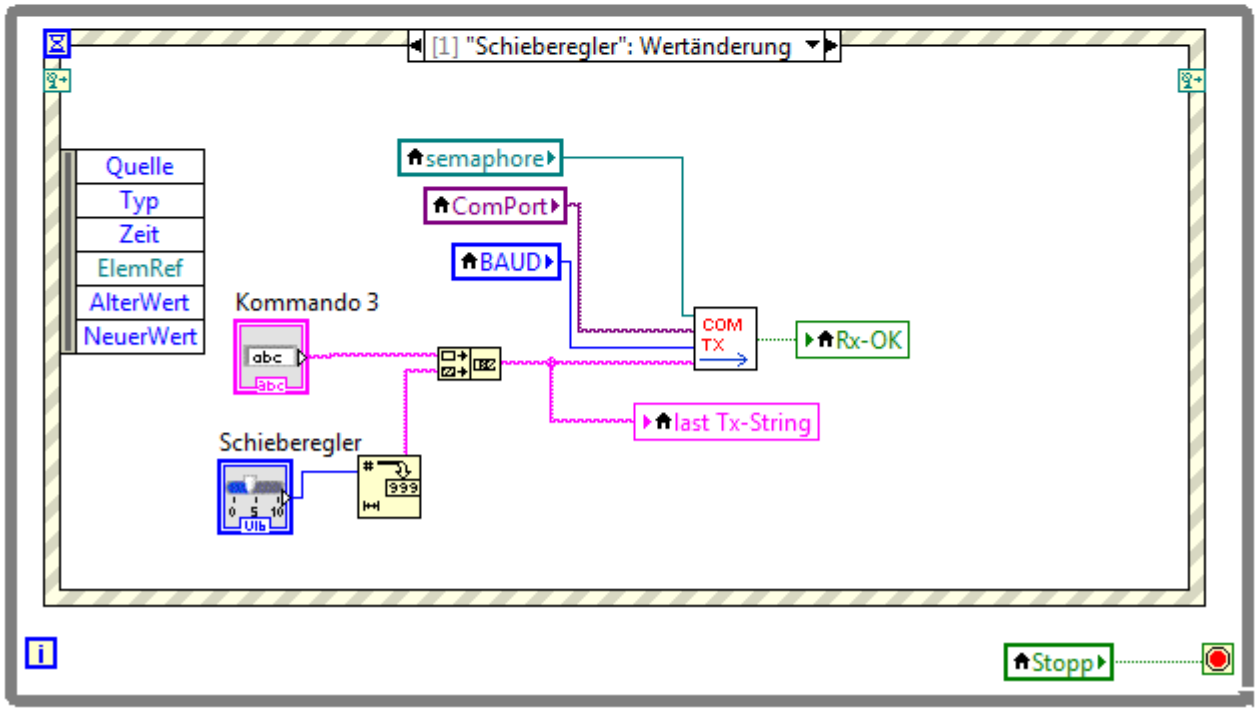



Com-TxRx-Event-Sema.lvproj * - Projekt-Explorer
Datei
Bearbeiten
Ansicht
Projekt
Ausführen
Werkzeuge
Fenster
Hilfe
Objekte
Dateien
Projekt: Com-TxRx-Event-Sema.lvproj
Mein Computer
Tx-Rx-Mnemonic-Event-Sema.vi
Abhängigkeiten
vi.lib
COM_Sema_TX_Only.vi
COM_Sema_TXRX.vi
Build-Spezifikationen

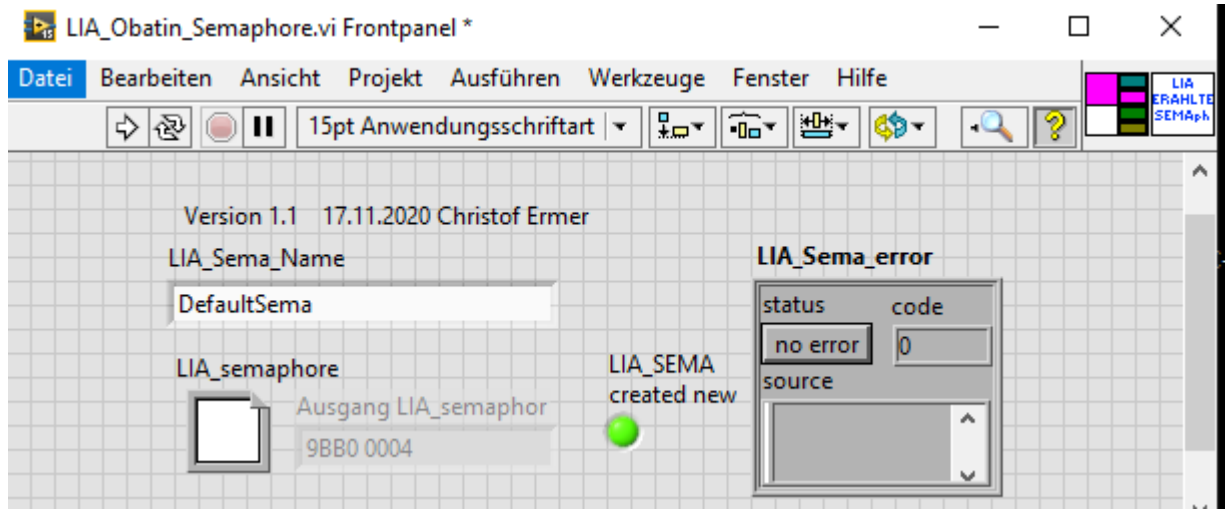
So spielt alles mit den
Semaphoren gut zusammen

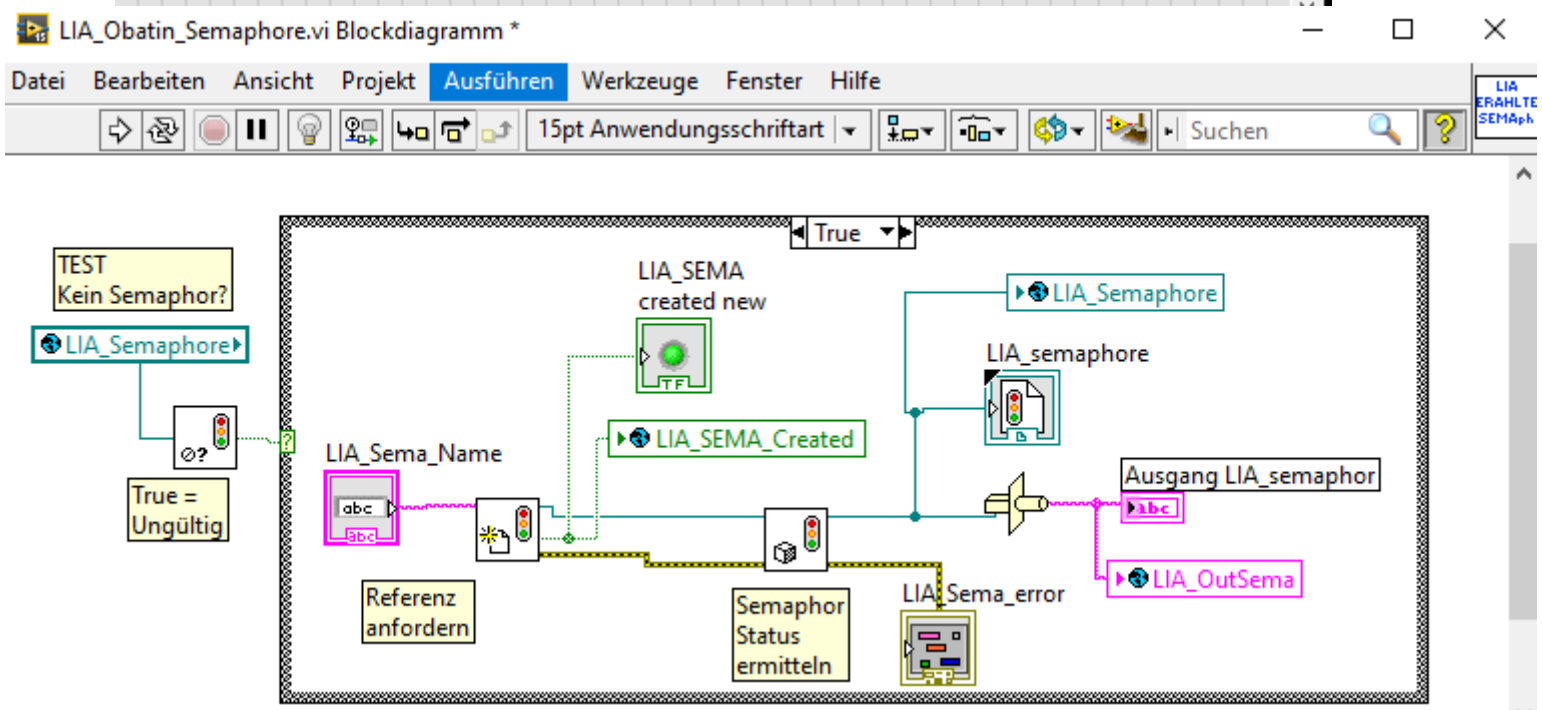






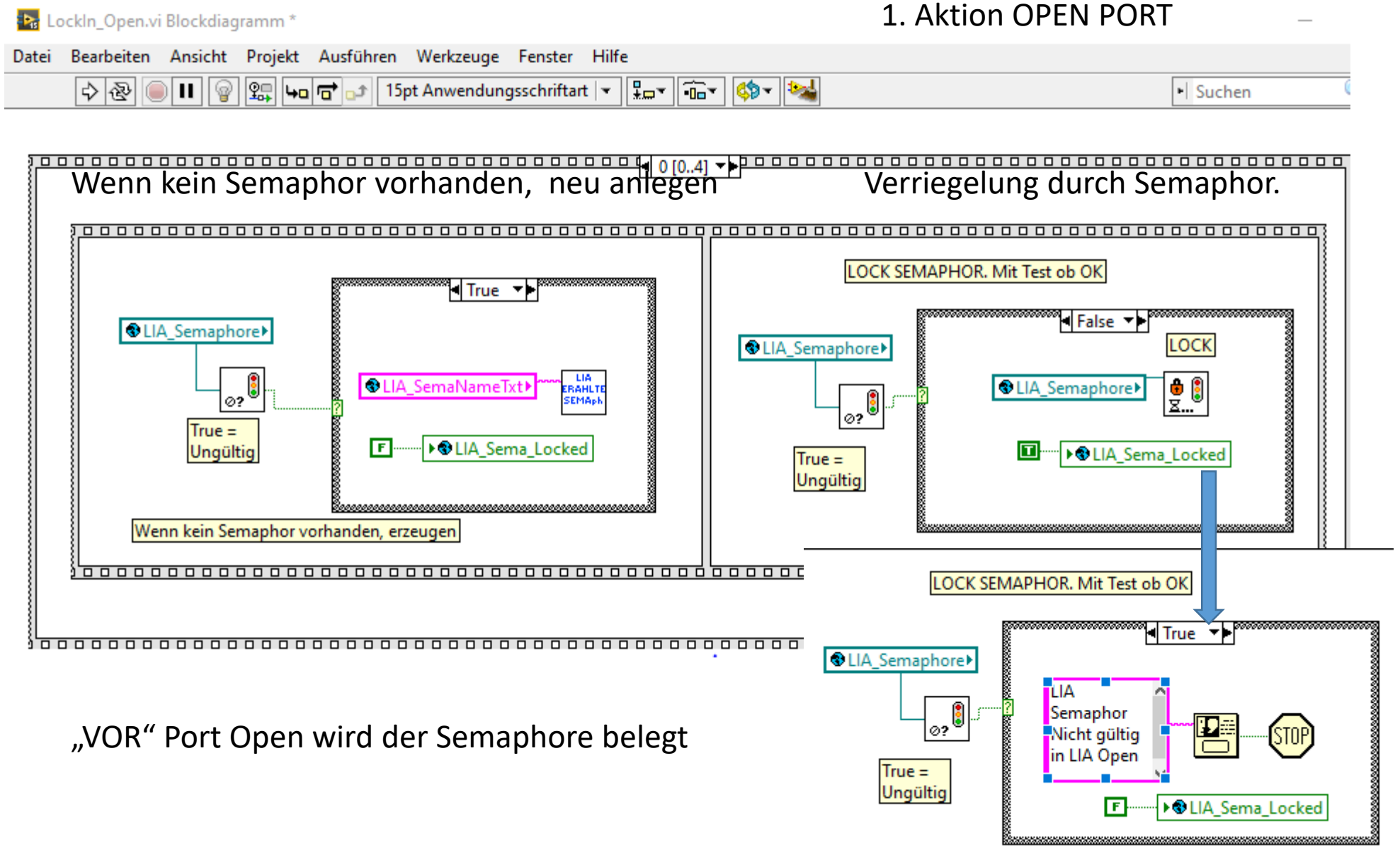
Anforderung eines Semaphores mit einem Namen für den Semaphorzweck





Anwendung des Semaphores zum Schutz mehrfacher Portaufrufe zur gleichen Zeit

1. Aktion OPEN PORT



„VOR“ Port Open wird der Semaphore belegt

„NACH“ Port Close wird er Semaphore wieder freigegeben