zenon
by COPA-DATA

zenon Treiber Handbuch

KabaDPServer

v.8.20

© 2020 Ing. Punzenberger COPA-DATA GmbH

Alle Rechte vorbehalten.

Die Weitergabe und Vervielfältigung dieses Dokuments ist - gleich in welcher Art und Weise - nur mit schriftlicher Genehmigung der Firma COPA-DATA gestattet. Technische Daten dienen nur der Produktbeschreibung und sind keine zugesicherten Eigenschaften im Rechtssinn. Änderungen - auch in technischer Hinsicht - vorbehalten.

Inhaltsverzeichnis

1 Willkommen bei der COPA-DATA Hilfe 5

2 KabaDPServer 5

3 KabaDPServer - Datenblatt 6

4 Treiber-Historie 7

5 Voraussetzungen 8

5.1 Steuerung 8

6 Konfiguration 8

6.1 Anlegen eines Treibers 9

6.2 Einstellungen im Treiberdialog 12

6.2.1 Allgemein 13

6.2.2 KABA Server Connection 17

7 Variablen anlegen 18

7.1 Variablen im Editor anlegen 18

7.2 Adressierung 22

7.3 Treiberobjekte und Datentypen 24

7.3.1 Treiberobjekte 24

7.3.2 Zuordnung der Datentypen 25

7.3.3 Alarme und Meldungen 26

7.4 Variablen anlegen durch Import 36

7.4.1 XML Import 37

7.4.2 DBF Import/Export 38

7.4.3 Online-Import 43

7.5 Kommunikationsdetails (Treibervariablen) 44

8 Treiberspezifische Funktionen 50

9 Funktion Treiberkommandos 50

10 Fehleranalyse 56

10.1 TCP API-Fehlernummern 56

10.2 Analysetool 64

10.3 Treiberüberwachung 65

10.4 Checkliste 66

1 Willkommen bei der COPA-DATA Hilfe

ZENON VIDEO-TUTORIALS

Praktische Beispiele für die Projektierung mit zenon finden Sie in unserem YouTube-Kanal (https://www.copadata.com/tutorial_menu). Die Tutorials sind nach Themen gruppiert und geben einen ersten Einblick in die Arbeit mit den unterschiedlichen zenon Modulen. Alle Tutorials stehen in englischer Sprache zur Verfügung.

ALLGEMEINE HILFE

Falls Sie in diesem Hilfekapitel Informationen vermissen oder Wünsche für Ergänzungen haben, wenden Sie sich per E-Mail an documentation@copadata.com.

PROJEKTUNTERSTÜTZUNG

Unterstützung bei Fragen zu konkreten eigenen Projekten erhalten Sie vom Customer Service, den Sie per E-Mail an support@copadata.com erreichen.

LIZENZEN UND MODULE

Sollten Sie feststellen, dass Sie weitere Module oder Lizenzen benötigen, sind unsere Mitarbeiter unter sales@copadata.com gerne für Sie da.

2 KabaDPServer

Der Treiber dient zur Darstellung der Zustände und Alarme von Türen, deren Überwachung mit einem KABA System realisiert wird.

3 KabaDPServer - Datenblatt

Allgemein:	
Treiberdateiname	KabaDPServer.exe
Treiberbezeichnung	Kaba Datenpunktserver Treiber
Steuerungs-Typen	Kaba exos 9300
Steuerungs-Hersteller	Kaba

Treiber unterstützt:	
Protokoll	Kaba Server XML-Frames
Adressierung: Adress-basiert	Name based
Adressierung: Namens-basiert	--
Kommunikation spontan	X
Kommunikation pollend	--
Online Browsing	X
Offline Browsing	--
Echtzeitfähig	--
Blockwrite	--
Modemfähig	--
RDA numerisch	--
RDA String	--
Hysterese	--
erweiterte API	--
Unterstützung von Statusbit **WR-SUC**	--
alternative IP-Adresse	--

Voraussetzungen:	
Hardware PC	--
Software PC	--
Hardware Steuerung	--
Software Steuerung	--
Benötigt v-dll	--

Plattformen:	
Betriebssysteme	Windows 10; Windows 7; Windows 8; Windows 8.1; Windows Server 2008 R2; Windows Server 2012; Windows Server 2012 R2; Windows Server 2016

4 Treiber-Historie

Datum	Build Nummer	Änderung
31.10.14	15380	Treiberdokumentation wurde neu erstellt

TREIBERVERSIONIERUNG

Mit zenon 7.10 wurde die Versionierung der Treiber verändert. Ab dieser Version gibt es eine versionsübergreifende Build-Nummer. Das ist die Zahl an der 4. Stelle der Dateiversion.
Zum Beispiel: **7.10.0.4228** bedeutet: Der Treiber ist für Version **7.10**, Service Pack **0** und hat die Build-Nummer **4228**.

Erweiterungen oder Fehlerbehebungen werden zukünftig in einem Build eingebaut und sind dann ab der nächsthöheren Build-Nummer verfügbar.

Beispiel

Eine Treibererweiterung wurde in Build **4228** implementiert. Der Treiber, den Sie im Einsatz haben, verfügt über die Build-Nummer **8322**. Da die Build-Nummer Ihres Treibers höher ist als die Build-Nummer der Erweiterung, ist die Erweiterung enthalten. Die Versionsnummer des Treiber (die ersten drei Stellen der Dateiversion) spielen dabei keine Rolle. Die Treiber sind versionsunabsabhängig

5 Voraussetzungen

Dieses Kapitel enthält Informationen zu den Voraussetzungen, die für die Verwendung des Treibers erforderlich sind.

5.1 Steuerung

Der Treiber kommuniziert mit einem KABA Datenpunktserver. Die Zugangsdaten (IP-Adresse, Port, Benutzer und Passwort) werden vom Server Administrator bekannt gegeben.

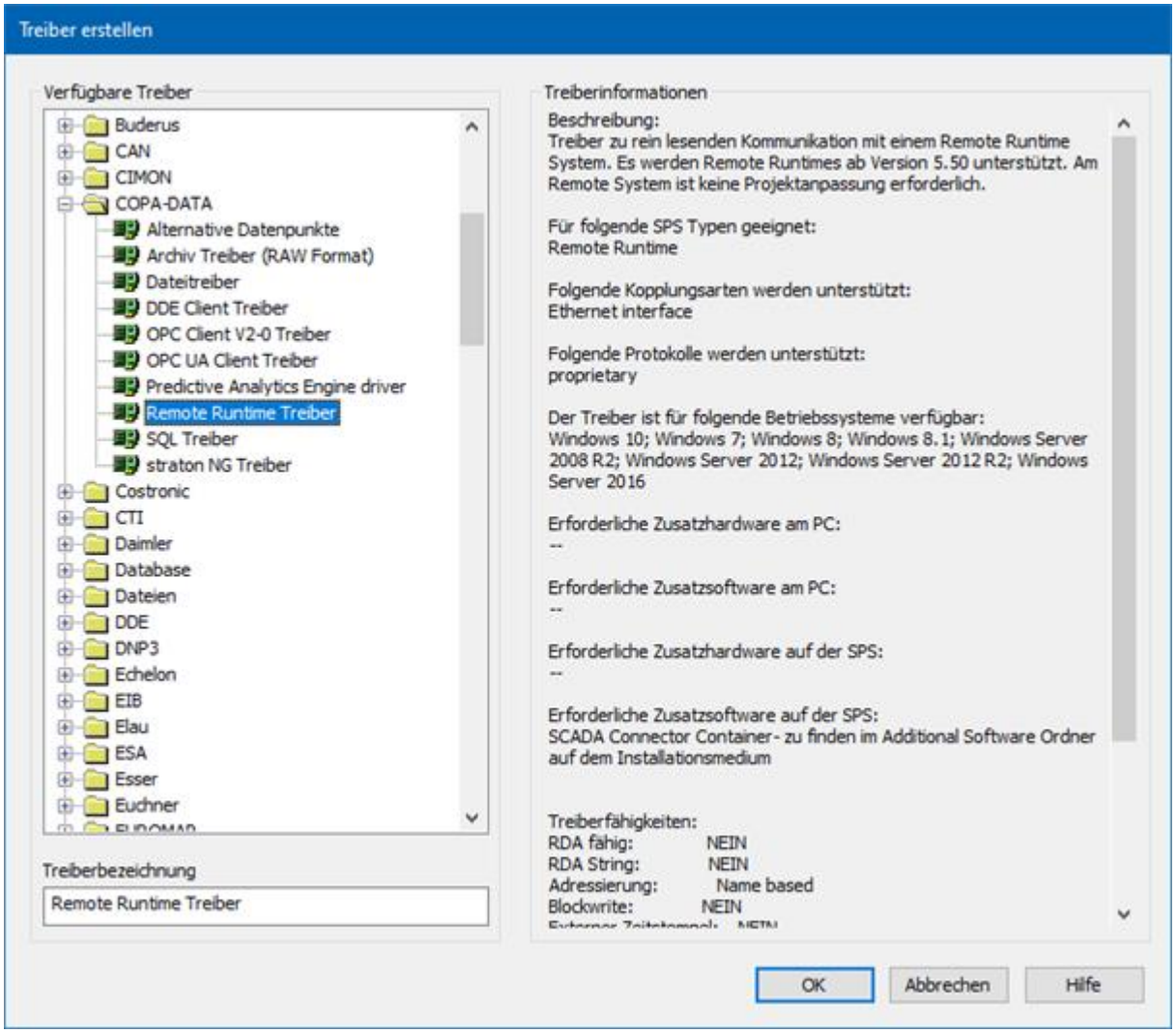
6 Konfiguration

In diesem Kapitel lesen Sie, wie Sie den Treiber im Projekt anlegen und welche Einstellungen beim Treiber möglich sind.

> **Info**
>
> Weitere Einstellungen, die Sie für Variablen in zenon vornehmen können, finden Sie im Kapitel Variablen der Online-Hilfe.

6.1 Anlegen eines Treibers

Im Dialog **Treiber erstellen** wählen Sie aus einer Liste jenen Treiber, den Sie neu anlegen wollen.



Parameter	Beschreibung
Verfügbare Treiber	Liste aller verfügbaren Treiber. Die Darstellung erfolgt in einer Baumstruktur: *[+]* erweitert die Ordnerstruktur und zeigt die darin enthaltenen Treiber. [-] reduziert die Ordnerstruktur Default: *keine Auswahl*
Treiberbezeichnung	Eindeutige **Bezeichnung** des Treibers. Default: *leer* Das Eingabefeld wird nach Auswahl eines Treibers

Parameter	Beschreibung
	aus der Liste der verfügbaren Treiber mit der vordefinierten **Bezeichnung** vorausgefüllt.
Treiberinformationen	Weiterführende Informationen über den gewählten Treiber. Default: *leer* Nach Auswahl eines Treibers werden in diesem Bereich die Informationen zum gewählten Treiber angezeigt.

DIALOG BEENDEN

Option	Beschreibung
OK	Übernimmt alle Einstellungen und öffnet den Treiberkonfigurationsdialog des ausgewählten Treibers.
Abbrechen	Verwirft alle Änderungen und schließt den Dialog.
Hilfe	Öffnet die Online-Hilfe.

> **Info**
>
> Die Inhalte dieses Dialogs sind in der Datei Treiber_[Sprachkürzel].xml gespeichert. Sie finden diese Datei im Ordner *C:\ProgramData\COPA-DATA\zenon[Versionsnummer]*.

TREIBER NEU ANLEGEN

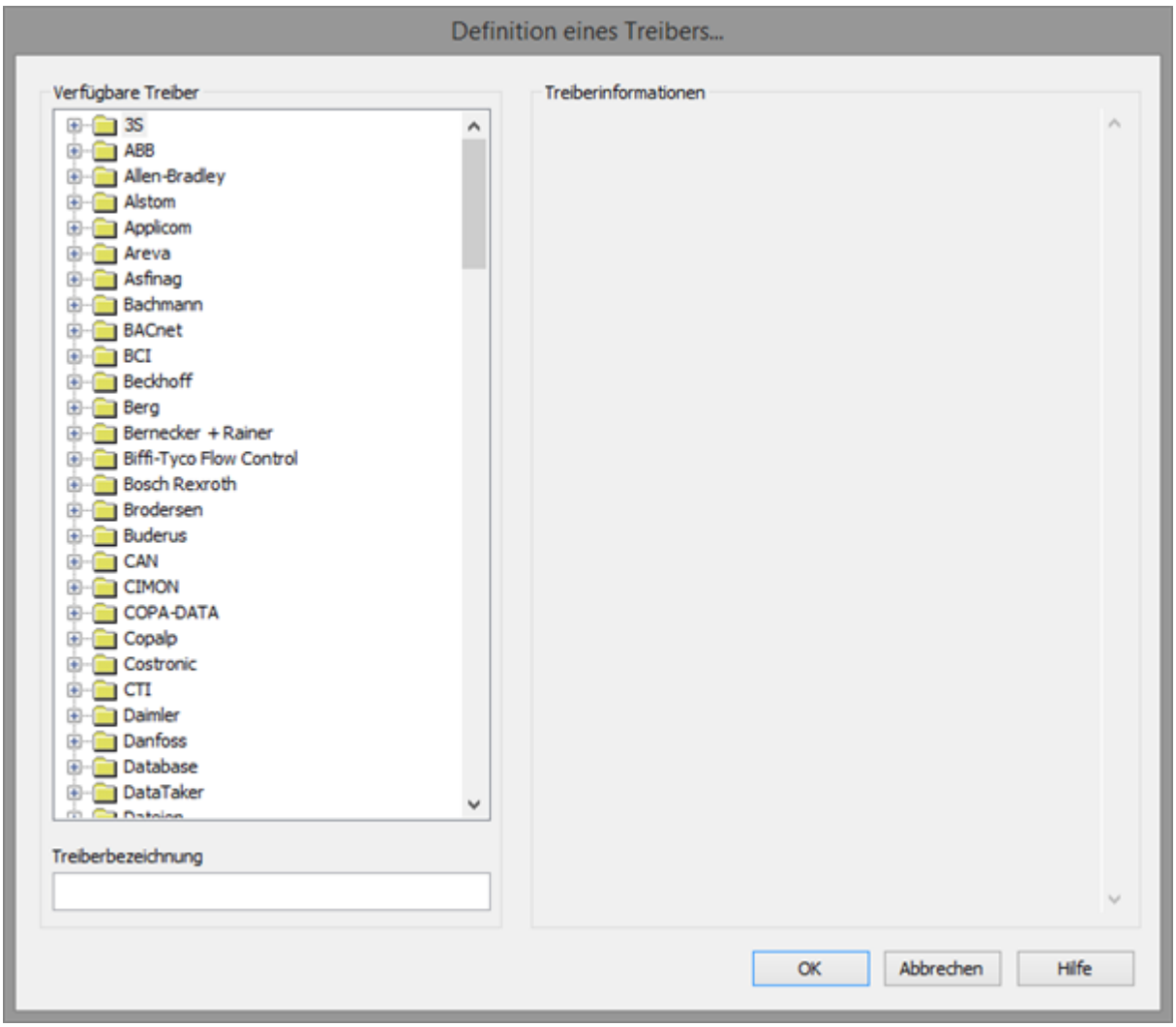
Um einen neuen Treiber anzulegen:

1. Klicken Sie mit der rechten Maustaste im Projektmanager auf **Treiber** und wählen Sie im Kontextmenü **Treiber neu** aus.

   Optional: Wählen Sie die Schaltfläche **Treiber neu** aus der Symbolleiste der Detailansicht der **Variablen**.

   Der Dialog **Treiber erstellen** wird geöffnet.

2. Der Dialog bietet eine Auflistung aller verfügbaren Treiber an.



3. Wählen Sie den gewünschten Treiber und benennen Sie diesen im Eingabefeld **Treiberbezeichnung**.
   Dieses Eingabefeld entspricht der Eigenschaft **Bezeichnung**. Per Default wird der Name des ausgewählten Treibers in diesem Eingabefeld automatisch eingefügt.
   Für die **Treiberbezeichnung** gilt:

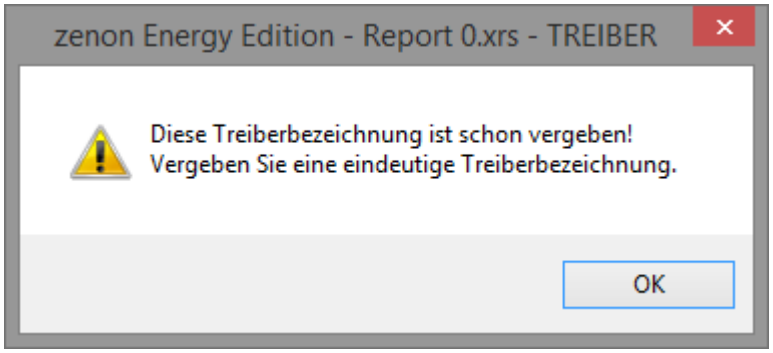
   - Die **Treiberbezeichnung** muss eindeutig sein.
     Wird ein Treiber mehrmals im Projekt verwendet, so muss jeweils eine neue Bezeichnung vergeben werden.
     Dies wird durch Klick auf die Schaltfläche **OK** evaluiert. Ist die Treiber im Projekt bereits vorhanden wird dies mit einem Warndialog angezeigt.
   - Die **Treiberbezeichnung** ist Bestandteil des Dateinamens.
     Daher darf Sie nur Zeichen enthalten, die vom Betriebssystem unterstützt werden. Nicht gültige Zeichen werden durch einen Unterstrich (_) ersetzt.
   - **Achtung:** Die Bezeichnung kann später nicht mehr geändert werden.

4. Bestätigen Sie den Dialog mit Klick auf die Schaltfläche **OK**.
   Der Konfigurationsdialog des ausgewählten Treibers wird geöffnet.

Hinweis: Treibernamen sind nicht sprachumschaltbar. Sie werden später immer in der Sprache angezeigt, in der sie angelegt wurden, unabhängig von der Sprache des Editors. Das gilt auch für Treiberobjekttypen.

DIALOG TREIBERBEZEICHNUNG BEREITS VORHANDEN

Ist ein Treiber bereits im Projekt vorhanden wird dies in einem Dialog angezeigt. Mit Klick auf die Schaltfläche **OK** wird der Warndialog geschlossen. Der Treiber kann korrekt benannt werden.



ZENON PROJEKT

Bei neu angelegten Projekten werden die folgenden Treiber automatisch angelegt:

- **Intern**
- **MathDr32**
- **SysDrv**

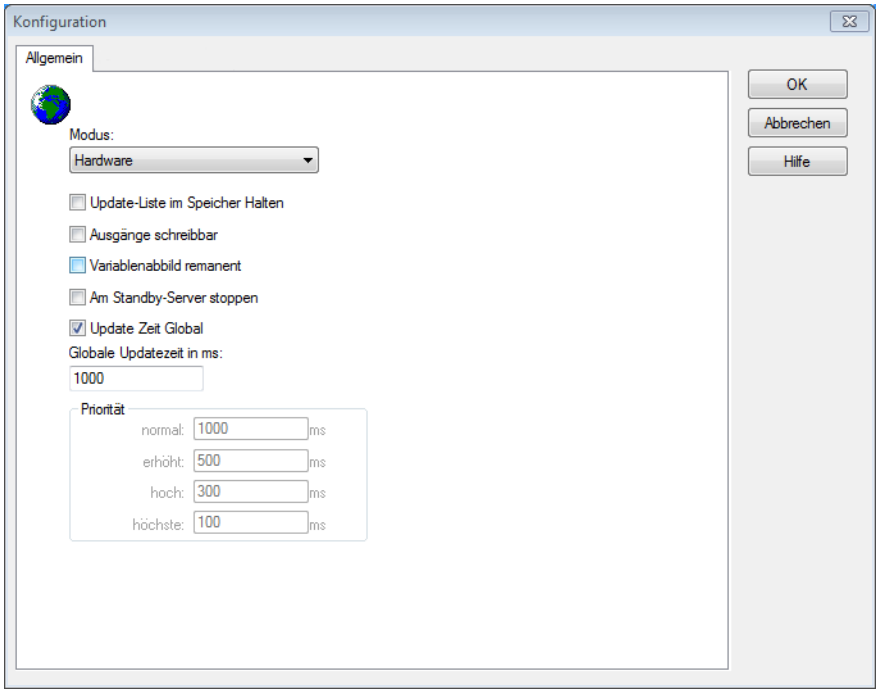
> **Info**
>
> In einem zenon Projekt müssen nur die benötigten Treiber vorhanden sein. Treiber können bei Bedarf zu einem späteren Zeitpunkt hinzugefügt werden.

6.2 Einstellungen im Treiberdialog

Folgende Einstellungen können Sie beim Treiber vornehmen:

6.2.1 Allgemein

Beim Anlegen eines Treibers wird der Konfigurationsdialog geöffnet. Um den Dialog später zum Bearbeiten zu öffnen, führen Sie einen Doppelklick auf den Treiber in der Liste aus oder klicken Sie auf die Eigenschaft **Konfiguration**.



Option	Beschreibung
Modus	Ermöglicht ein Umschalten zwischen Hardware und Simulationsmodus ▸ *Hardware*: Die Verbindung zur Steuerung wird hergestellt. ▸ Simulation - statisch: Es wird keine Kommunikation zur Steuerung aufgebaut, die Werte werden vom Treiber simuliert. In diesem Modus bleiben die Werte konstant oder die Variablen behalten die über zenon Logic gesetzen Werte. Jede Variable hat seinen eigenen Speicherbereich. Zum Beispiel zwei Variablen vom Typ Merker mit Offset 79, können zur Runtime unterschiedliche Werte haben und beeinflussen sich gegenseitig nicht. Ausnahme: Der Simulatortreiber. ▸ *Simulation - zählend*: Es wird keine Kommunikation zur Steuerung aufgebaut, die Werte werden vom Treiber simuliert.

Option	Beschreibung
	In diesem Modus zählt der Treiber die Werte innerhalb ihres Wertebereichs automatisch hoch. ▸ *Simulation - programmiert*: Es wird keine Kommunikation zur Steuerung aufgebaut, die Werte werden von einem frei programmierbaren Simulationsprojekt berechnet. Das Simulationsprojekt wird mit der zenon Logic Workbench erstellt und läuft in einer in den Treiber integrierten zenon Logic Runtime ab. Details siehe Kapitel Treibersimulation.
Update-Liste im Speicher Halten	Einmal angeforderte Variablen werden weiterhin von der Steuerung angefordert, auch wenn diese aktuell nicht mehr benötigt werden. Dies hat den Vorteil, dass z B. mehrmalige Bildumschaltungen nach dem erstmaligen Aufschalten beschleunigt werden, da die Variablen nicht neu angefordert werden müssen. Der Nachteil ist eine erhöhte Belastung der Kommunikation zur Steuerung.
Ausgänge schreibbar	▸ *Aktiv*: Ausgänge können beschrieben werden. ▸ *Inaktiv*: Das Beschreiben der Ausgänge wird unterbunden. **Hinweis**: Steht nicht für jeden Treiber zur Verfügungen.
Variablenabbild remanent	Diese Option speichert und restauriert den aktuellen Wert, den Zeitstempel und die Status eines Datenpunkts. Grundvoraussetzung: Die Variable muss einen gültigen Wert und Zeitstempel besitzen. Das Variablenabbild wird im Modus Hardware gespeichert, wenn einer dieser Status aktiv ist: ▸ Benutzerstatus *M1* (*0*) bis *M8* (*7*) ▸ *REVISION(9)* ▸ *AUS(20)* ▸ *ERSATZWERT(27)* Das Variablenabbild wird immer gespeichert wenn: ▸ die Variable vom Objekttyp

Option	Beschreibung
	Kommunikationsdetails ist ▸ der Treiber im Simulationsmodus läuft. (nicht programmierte Simulation) Folgende Status werden beim Start der Runtime nicht restauriert: ▸ *SELECT(8)* ▸ *WR-ACK(40)* ▸ *WR-SUC(41)* Der Modus **Simulation - programmiert** beim Treiberstart ist kein Kriterium, um das remanente Variablenabbild zu restaurieren.
Am Standby Server stoppen	Einstellung für Redundanz bei Treibern, die nur eine Kommunikationsverbindung erlauben. Dazu wird der Treiber am Standby Server gestoppt und erst beim Hochstufen wieder gestartet. **Achtung:** Ist diese Option aktiv, ist die lückenlose Archivierung nicht mehr gewährleistet. ▸ *Aktiv*: Versetzt den Treiber am nicht-prozessführenden Server automatisch in einen Stopp-ähnlichen Zustand. Im Unterschied zum Stoppen über Treiberkommando erhält die Variable nicht den Status **abgeschaltet**, sondern einen leeren Wert. Damit wird verhindert, dass beim Hochstufen zum Server nicht relevante Werte in AML , CEL und Archiv erzeugt werden. Default: *inaktiv* **Hinweis:** Nicht verfügbar, wenn CE Terminal als Datenserver dient. Weitere Informationen dazu erhalten Sie im Handbuch zenon Operator im Kapitel CE Terminal als Datenserver.
Update Zeit Global	Einstellung für globale Update-Zeiten in Millisekunden: ▸ *Aktiv*: Die eingestellte **Globale Update Zeit** wird für alle Variablen im Projekt verwendet. Die bei den Variablen eingestellte Priorität wird nicht

Option	Beschreibung
	verwendet. ▸ *Inaktiv*: Die eingestellten Prioritäten werden für die einzelnen Variablen verwendet. **Ausnahmen:** Spontane Treiber ignorieren diese Option. Sie nutzen in der Regel die kürzest mögliche Update Zeit. Details siehe Abschnitt **Update Zeit spontane Treiber**.
Priorität	Hier werden die Pollingzeiten der einzelnen Prioritätsklassen eingestellt. Alle Variablen mit der entsprechenden Priorität werden in der eingestellten Zeit gepollt. Die Zuordnung der Variablen erfolgt separat bei jeder Variablen über die Einstellungen in den Variableneigenschaften. Mit den Prioritätsklassen kann die Kommunikation der einzelnen Variablen auf die Wichtigkeit oder benötigte Aktualität abgestuft werden. Daraus ergibt sich eine verbesserte Verteilung der Kommunikationslast. **Achtung:** Prioritätsklassen werden nicht von jedem Treiber unterstützt, z.B. von spontan kommunizierenden zenon Treibern.

DIALOG BEENDEN

Option	Beschreibung
OK	Übernimmt alle Änderungen in allen Registerkarten und schließt den Dialog.
Abbrechen	Verwirft alle Änderungen in allen Registerkarten und schließt den Dialog.
Hilfe	Öffnet die Online-Hilfe.

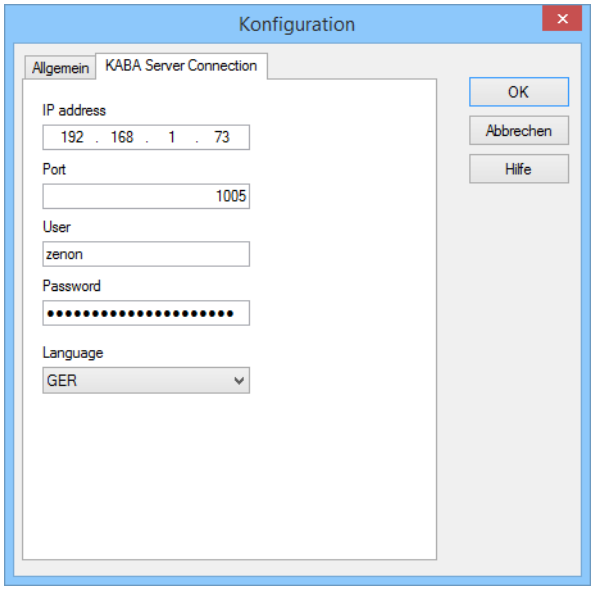
UPDATE ZEIT SPONTANE TREIBER

Bei spontanen Treibern wird beim **Sollwert Setzen**, **Advisen** von Variablen und bei **Requests** sofort ein Lesezyklus ausgelöst - unabhängig von der eingestellten Update Zeit. Damit wird sicher gestellt, dass der Wert nach dem Schreiben in der Visualisierung sofort zur Verfügung steht. In der Regel beträgt die Updatezeit 100 ms.

Spontane Treiber sind **ArchDrv**, **BiffiDCM**, **BrTcp32**, **DNP3**, **Esser32**, **FipDrv32**, **FpcDrv32**, **IEC850**, **IEC870**, **IEC870_103**, **Otis**, **RTK9000**, **S7DCOS**, **SAIA_Slave**, **STRATON32** und **Trend32**.

6.2.2 KABA Server Connection

In dieser Registerkarte konfigurieren Sie die Verbindung zum KABA Server.



Parameter	Beschreibung
IP address	IP-Adresse des Servers.
Port	Port, über den kommuniziert wird. Default: *1005*
User	Anmeldename.
Password	Passwort.
Language	Sprache, in der Texte für Stringvariablen angezeigt werden. Auswahl aus Dropdownliste: ▸ *GER*: Deutsch ▸ *FRA*: Französisch ▸ *ENG*: Englisch ▸ *ITA*: Italienisch

DIALOG BEENDEN

Option	Beschreibung
OK	Übernimmt Einstellungen und schließt den Dialog.
Abbrechen	Verwirft alle Änderungen und schließt den Dialog.
Hilfe	Öffnet die Online-Hilfe.

7 Variablen anlegen

So werden Variablen im zenon Editor angelegt:

7.1 Variablen im Editor anlegen

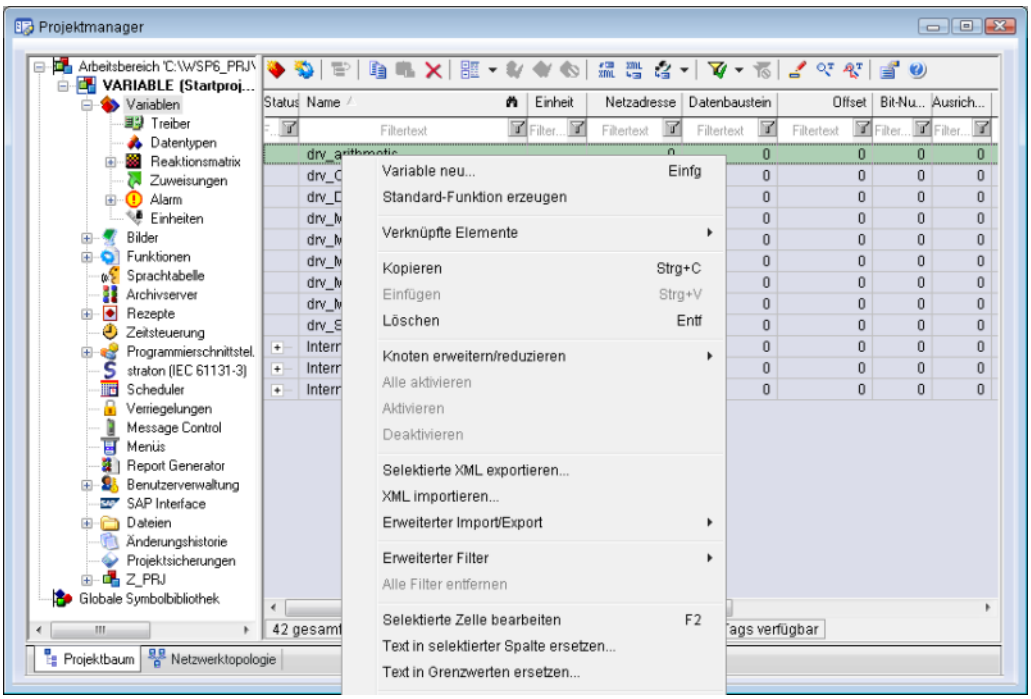
Variablen können angelegt werden:

- als einfache Variable
- in Arrays
- als Struktur-Variablen

DIALOG VARIABLE

Um eine neue Variable zu erstellen, gleich welchen Typs:

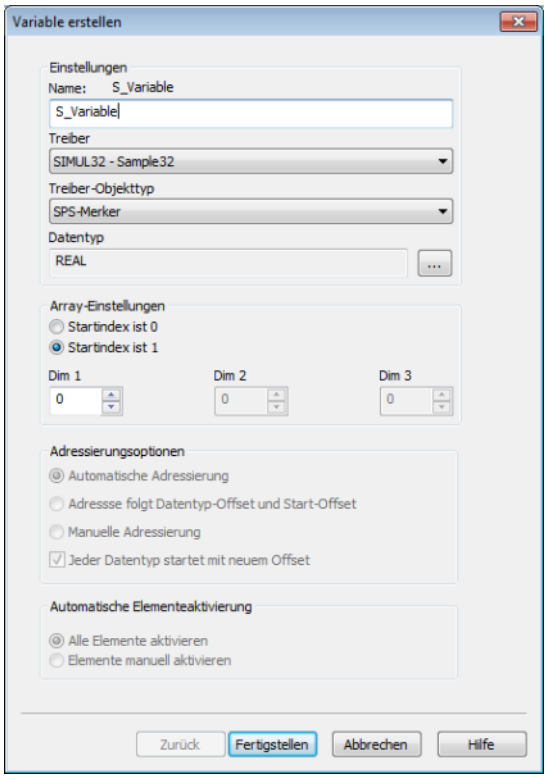
1. Wählen Sie im Knoten **Variablen** im Kontextmenü den Befehl **Variable neu**.



Der Dialog zur Konfiguration der Variable wird geöffnet.

2. Konfigurieren Sie die Variable.
3. Welche Einstellungen möglich sind, hängt ab vom Typ der Variablen.

DIALOG VARIABLE ERSTELLEN



Eigenschaft	Beschreibung
Name	Eindeutiger Name der Variablen. Ist eine Variable mit gleichem Namen im Projekt bereits vorhanden, kann keine weitere Variable mit diesem Namen angelegt werden. Maximale Länge: 128 Zeichen **Achtung:** Die Zeichen **#** und **@** sind für Variablennamen nicht erlaubt. Bei Verwendung nicht zugelassener Zeichen kann die Variablenerstellung nicht abgeschlossen werden, die Schaltfläche **Fertigstellen** bleibt inaktiv. **Hinweis:** Manche Treiber erlauben die Adressierung auch über die Eigenschaft **Symbolische Adresse**.
Treiber	Wählen Sie aus der Dropdownliste den gewünschten Treiber. **Hinweis:** Sollte im Projekt noch kein Treiber angelegt sein, wird automatisch der Treiber für interne Variable (**Intern.exe**) geladen.
Treiberobjekttyp	Wählen Sie aus der Dropdownliste den passenden Treiberobjekttyp aus.
Datentyp	Wählen Sie den gewünschten Datentyp. Klick auf die Schaltfläche

Eigenschaft	Beschreibung
	... öffnet den Auswahl-Dialog.
Array-Einstellungen	Erweiterte Einstellungen für Array-Variablen. Details dazu lesen Sie im Abschnitt Arrays.
Adressierungsoptionen	Erweiterte Einstellungen für Arrays und Struktur-Variablen. Details dazu lesen Sie im jeweiligen Abschnitt.
Automatische Elementeaktivierung	Erweiterte Einstellungen für Arrays und Struktur-Variablen. Details dazu lesen Sie im jeweiligen Abschnitt.

SYMBOLISCHE ADRESSE

Die Eigenschaft **Symbolische Adresse** kann für die Adressierung alternativ zu **Name** oder **Kennung** der Variablen verwendet werden. Die Auswahl erfolgt im Treiberdialog, die Konfiguration in der Variableneigenschaft. Beim Import von Variablen unterstützter Treiber wird die Eigenschaft automatisch eingetragen.

Maximale Länge: *1024* Zeichen.

Folgende Treiber unterstützen die **Symbolische Adresse**:

- **3S_V3**
- **AzureDrv**
- **BACnetNG**
- **IEC850**
- **KabaDPServer**
- **OPCUA32**
- **Phoenix32**
- **POZYTON**
- **RemoteRT**
- **S7TIA**
- **SEL**
- **SnmpNg32**
- **PA_Drv**
- **EUROMAP63**

ABLEITUNG VOM DATENTYP

Messbereich, **Signalbereich** und **Sollwert Setzen** werden immer:

- vom Datentyp abgeleitet
- beim Ändern des Datentyps automatisch angepasst

Hinweis Signalbereich: Bei einem Wechsel auf einen Datentyp, der den eingestellten **Signalbereich** nicht unterstützt, wird der **Signalbereich** automatisch angepasst. Zum Beispiel wird bei einem Wechsel von **INT** auf **SINT** der **Signalbereich** auf *127* geändert. Die Anpassung erfolgt auch dann, wenn der **Signalbereich** nicht vom Datentyp abgeleitet wurde. In diesem Fall muss der **Messbereich** manuell angepasst werden.

7.2 Adressierung

Gruppe/Eigenschaft	Beschreibung
Allgemein	Gruppe mit allgemeinen Eigenschaften.
Name	Frei vergebbarer Name. **Achtung:** Je zenon Projekt muss der Name eindeutig sein.
Kennung	Frei vergebbare Kennung. Z. B. für **Betriebsmittelkennung**, Kommentar usw.
Adressierung	Eigenschaften für die Adresse der Variablen.
Netzadresse	Wird für diesen Treiber nicht verwendet.
Datenbaustein	Wird für diesen Treiber nicht verwendet.
Offset	Wird für diesen Treiber nicht verwendet.
Ausrichtung	Wird für diesen Treiber nicht verwendet.
Bitnummer	Wird für diesen Treiber nicht verwendet.

Gruppe/Eigenschaft	Beschreibung
Symbolische Adresse	Die Eigenschaft **Symbolische Adresse** kann für die Adressierung alternativ zu **Name** od Variablen verwendet werden. Die Auswahl erfolgt im Treiberdialog, die Konfiguration in d Variableneigenschaft. Beim Import von Variablen unterstützter Treiber wird die Eigenscha eingetragen. Maximale Länge: *1024* Zeichen. Die reale Adresse der Variable für den Treiber besteht aus: ▸ Präfix der KABA Adresse ▸ Attribut in eckigen Klammen Beispiel: *I01000201[AlarmID]* Möglichen Attribute: ▸ *State*: Status der Tür ▸ *AlarmID*: Alarm-ID ▸ *AlarmText*: Alarm-Text **Tipp:** Importieren Sie die Variablen vom Server in den Editor, um Tippfehler zu vermeide
Stringlänge	Länge der Alarmtexte. Default: *256 Zeichen*
Treiber Anbindung	Eigenschaften für die Treiberanbindung.
Treiberobjekttyp	Folgende Objekttypen stehen zur Verfügung: ▸ *Status der Tür* ▸ *Alarm ID der Tür* ▸ *Alarm Text der Tür*
Datentyp	Datentypen sind den Objekttypen fix zugeordnet: ▸ *Status der Tür*: *INTEGER* ▸ *Alarm ID der Tür*: *UINT* ▸ *Alarm Text der Tür*: *WSTRING*
Treiber Anbindung/Priorität	Wird für diesen Treiber nicht verwendet. Der Treiber unterstützt keine zyklisch pollende K Prioritätsklassen.

7.3 Treiberobjekte und Datentypen

Treiberobjekte sind in der Steuerung verfügbare Bereiche wie z. B. Merker, Datenbausteine usw. Hier lesen Sie, welche Treiberobjekte vom Treiber zur Verfügung gestellt werden und welche IEC-Datentypen dem jeweiligen Treiberobjekt zugeordnet werden können.

7.3.1 Treiberobjekte

Folgende Objekttypen stehen in diesem Treiber zur Verfügung:

Treiberobjekttyp	Kanaltyp	Lesen	Schreiben	Unterstützte Datentypen	Beschreibung
Alarm ID der Tür	*0x0041* *(65)*	X	--	*UINT*	Variable zum Darstellen der Alarm-ID einer Tür.
Alarm Text der Tür	*0x0042* *(66)*	X	--	*WSTRING*	Variable zum Darstellen des Alarm-Textes einer Tür.
Status der Tür	*0x0040* *(64)*	X	--	*INT*	Variable zum Darstellen des Status einer Tür.
Kommunikationsdetails	35	X	X	*BOOL*, *DINT*, *INT*, *REAL*, *SINT*, *STRING*, *UDINT*, *UINT*, *USINT*	Variablen für die statische Analyse der Kommunikation. Werte werden nur zwischen Treiber und Runtime übertragen, nicht zur SPS! **Hinweis**: Die Adressierung und das Verhalten ist bei den meisten zenon Treibern gleich. Weitere Informationen dazu

Treiberobjekttyp	Kanaltyp	Lesen	Schreiben	Unterstützte Datentypen	Beschreibung
					finden Sie im Kapitel Kommunikationsdetails (Treibervariablen) (auf Seite 44).

Legende:

X: wird unterstützt

--: wird nicht unterstützt

KANALTYP

Der Begriff "**Kanaltyp**" ist die interne numerische Bezeichnung des Treiberobjekttyps. Diese wird auch für den erweiterten DBF Import/Export der Variablen verwendet.

"**Kanaltyp**" wird für den erweiterten CSV Import/Export der Variablen in der Spalte "**HWObjectType**" verwendet.

7.3.2 Zuordnung der Datentypen

Alle Variablen in zenon werden von IEC-Datentypen abgeleitet. In folgender Tabelle werden zur besseren Übersicht die IEC-Datentypen den Datentypen der Steuerung gegenübergestellt.

Steuerung	zenon	Datenart
-	BOOL	8
-	USINT	9
-	SINT	10
Alarm ID der Tür	UINT	2
Status der Tür	INT	1
-	UDINT	4
-	DINT	3
-	ULINT	27

Steuerung	zenon	Datenart
-	LINT	26
-	REAL	5
-	LREAL	6
-	STRING	12
Alarm Text der Tür	WSTRING	21
-	DATE	18
-	TIME	17
-	DATE_AND_TIME	20
-	TOD (Time of Day)	19

DATENART

Der Begriff **Datenart** ist die interne numerische Bezeichnung des Datentyps. Diese wird auch für den erweiterten DBF Import/Export der Variablen verwendet.

7.3.3 Alarme und Meldungen

Über den Treiber können Alarme und Meldungen angezeigt werden.

Alarme werden über die **AlarmID** angezeigt. Bei Systemmeldungen wird unterschieden zwischen Meldungen über den aktuellen Status einer Systemkomponente (**State**) und Meldungen zu individuellen Parametrierungen von Unterstationen und Durchgängen (**Mode**).

Alarme und Meldungen werden über ganzzahlige IDs angezeigt.

7.3.3.1 Alarme

Alarme werden über die **AlarmID** angezeigt. Die **AlarmID** ist eine ganze Zahl, die spezifiziert, ob eine Variable einen Alarm aufweist und um welchen Alarm es sich handelt.
AlarmID:

- *0*: kein Alarm
- *>0*: Alarm.

Die Satzarten setzen sich zusammen aus Satzartentyp und Alarmnummer (z. B. *A100*).

Folgende Satzarten werden verwendet:

Definition	Typ	Bezeichnung	Anwählbar im Dialog Systemsteuerung, Ereignis-/Alarmverwaltung unter
Alarmlogbuch	*E*	Ereignis	Alarmlogbuch
Alarmbehandlung	*A*	Alarm	Alarmbehandlung (+Alarmlogbuch)
	C	quittierter Alarm	
	R	gelöschter Alarm	

Jede Meldung kann als Typ *A* oder *E* definiert werden. Die Nummern entsprechen denen der Alarmmeldeliste im Dialog der Alarmverwaltung.

ALARMNUMMERN

Wert (TextID)	Beschreibung + evtl. zusätzlicher Wert (= Argument)
1	Firmencode falsch.
2	Benutzermedium unbekannt oder gesperrt.
3	Fehler im Blockprüfzeichen.
4	Versionsnummer falsch.
5	Benutzermedium gesperrt.
6	Medium nicht gültig.
8	Keine Raumzonenberechtigung.
9	Wochentag/Tageszeit falsch.
10	Biometriefehler.
11	PIN-Code nicht eingegeben.
12	PIN-Code falsch.
13	Max. Anzahl falscher PIN-Codeeingaben erreicht, Benutzermedium gesperrt.
14	Stiller Alarm.
15	Raumsequenz falsch.
16	Doppelzutrittssperre.
17	Durchtritt erfolgt.

Wert (TextID)	Beschreibung + evtl. zusätzlicher Wert (= Argument)
18	Kein Durchtritt erfolgt.
19	Biometrieverifikation falsch.
20	Max. Personenkontrolle von Raum verletzt.
22	Min. Personenkontrolle von Raum verletzt.
23	Sabotagealarm.
24	Ende Sabotagealarm.
25	Zutritt berechtigt.
26	Spezialfunktion.
27	Standard-Benutzermedium.
28	Zutritt gestattet mit Spezial-Benutzermedium.
29	Anzahl geladener Personen.
30	Firmwareversion.
31	Letzter Parameterdownload Satzanzahl.
34	Beginn des Downloads
35	Ende des Downloads.
36	Ungenügende Sicherheitsstufe.
37	Voralarm: max. Personenkontrolle von Raum verletzt.
38	Voralarm: min. Personenkontrolle von Raum verletzt.
39	Leserkonfiguration.
40	AMC wurde neu gestartet.
43	Lade Datenbank in den AMC.
50	Tür zu lange offen.
51	Unberechtigte Türöffnung.
52	Tür wieder zu.
53	Sabotagealarm.
54	Ende Sabotagealarm.

Wert (TextID)	Beschreibung + evtl. zusätzlicher Wert (= Argument)
60 - 69	(Freifeld)
70	Buchungs-Speicher voll.
71	Buchungs-Speicher leer.
72	Parametrierung Anfang.
73	Parametrierung Ende.
74	Diagnose Anfang.
75	Diagnose Ende.
76	Satzinhalt falsch.
77	Kommando zur Zeit nicht ausführbar.
78	Kommando nicht ausführbar.
79	Unbekannter Datensatz.
81	Betriebsmodus der Zutrittszentrale/des Terminals geändert.
83	Riegelstellung falsch.
90	Subterminal antwortet nicht.
91	Subterminal antwortet wieder
92	Zutrittszentrale/Zutrittsmanager/Terminal antwortet nicht.
93	Zutrittszentrale/Zutrittsmanager/Terminal antwortet wieder.
94	Kommunikationszentrale antwortet nicht.
95	Kommunikationszentrale antwortet wieder.
96	Kommunikationszentrale ist neu gestartet worden.
97	Port antwortet nicht.
98	Port antwortet wieder.
99	Doppelt benützte GID/DID.
100	Kommunikation zu Port konnte nicht initialisiert werden.
101	Zutrittszentrale/Zutrittsmanager/Terminal antwortet.
102	Port antwortet.

Wert (TextID)	Beschreibung + evtl. zusätzlicher Wert (= Argument)
103	Verbindung zum Port wird aufgebaut.
104	Kommunikationszentrale wird neu gestartet.
105	Kommunikationszentrale wird geladen.
106	Verbindung zu Filiale ist hergestellt.
107	Verbindung zu Filiale ist abgebaut.
108	Subterminal antwortet.
109	Sabotagealarm.
110	Ende Sabotagealarm.
111	Kurzschluss des Kontakts.
112	Unterbruch des Kontakts.
115	Depot zu lange offen.
116	CardLink: Validierung fehlgeschlagen.
117	CardLink: Batterie niedrig.
118	CardLink: Ereignispuffer voll.
120	Eingebucht auf Stellfläche.
121	Ausgebucht aus Stellfläche.
122	Keinen Parkplatz/Stellfläche gefunden.
123	Parkplatz/Stellfläche belegt/voll.
124	Bereits ein Auto geparkt.
125	Parkplatz/Stellfläche nicht belegt.
126	Kann Leser im Parkbereich nicht zuordnen.
127	Zutritt berechtigt - Betriebsart Durchgang: freigeschaltet.
128	Zutritt berechtigt - Betriebsart Durchgang: gesperrt.
143	Zutritt berechtigt und Passagiermodus aktiviert.
144	Passagiermodus deaktiviert.
145	Passagiermodus gesperrt.

Wert (TextID)	Beschreibung + evtl. zusätzlicher Wert (= Argument)
146	Benutzermedium durch Funktion **Personalauslosung** gesperrt.
166	Riegelstellung wieder korrekt.
170	Zutrittsmanager wurde neu gestartet.
171	Beim Zutrittsmanager sind inkonsistente Daten möglich.
172	Es sind weniger Erweiterungsmodule **Kaba exos OC8** angeschlossen, als in **Kabaexos** konfiguriert sind.
198	Kurzschluss.
199	Unterbruch.
200	Aus.
201	Aus.
600	Ein.
601	Ein.
9101	Fabrikationsschlüssel wurde erfolgreich durch Applikationsschlüssel ersetzt.
9102	Mehrfaches Umwandeln des Fabrikationsschlüssels in einen Applikationsschlüssel.
9103	CardLink: Validierungsstempel-Reihenfolge verletzt.
9105	Umwandeln des Fabrikationsschlüssels in einen Applikationsschlüssel bei unbekanntem Medium.
9970	Ausschalten.
9971	Einschalten.
9980	Überwachen.
9981	Für einen Zutritt öffnen.
9982	Sperren.
9983	Statisch öffnen (nicht überwacht).
9996	Service-Modus ein.
9997	Service-Modus aus.
9998	Alarm quittiert.

Wert (TextID)	Beschreibung + evtl. zusätzlicher Wert (= Argument)
9999	Alarmquittierung bestätigt.

7.3.3.2 Meldungen

Bei Systemmeldungen wird unterschieden zwischen:

- Zustand (**State**): Systemseitige Meldung über den aktuellen Status der Systemkomponente.
- Betriebsart (**Mode**): Meldung zu individuellen Parametrierungen von Unterstationen und Durchgängen. Die Betriebsart eines Durchgangs wird nur gemeldet, wenn er geschlossen ist.

MELDUNGEN

STATE

Optional.

Eine ganze Zahl, die den Variablenstatus repräsentiert. Der Wert *-1* bedeutet, dass der Status nicht bestimmt werden konnte.

Betrifft	Beschreibung	Wert
Front Server	Unbekannt.	*-1*
	Antwortet nicht.	*0*
	Normalbetrieb.	*1*
	Download läuft.	*2*
	Startet auf.	*3*
Port	Unbekannt.	*-1*
	Nicht verbunden.	*0*
	Verbunden.	*1*
	Verbindung wird aufgebaut.	*2*
Unterstation	Unbekannt.	*-1*
	Nicht verbunden.	*0*
	Verbunden.	*1*

Betrifft	Beschreibung	Wert
	Verbindung wird aufgebaut.	*2*
Subdevice Türmanager/Leser	Unbekannt.	*-1*
	Nicht verbunden.	*0*
	Verbunden.	*1*
Durchgang	Unbekannt.	*-1*
	offen.	*0*
	geschlossen.	*1*
	verriegelt.	*2*
Binäre Ausgänge	Unbekannt.	*-1*
	Aus.	*0*
	Ein.	*1*
Binäre Eingänge	Unbekannt.	*-1*
	Aus.	*0*
	Ein.	*1*
	Unterbruch.	*2*
	Kurzschluss.	*3*

STATETEXT

Optional.

Eine Zeichenkette, die den Status beschreibt.

MODE

Optional.

Eine ganze Zahl, die den Variablenmodus repräsentiert. Der Wert *-1* bedeutet, dass der Modus nicht bestimmt werden konnte

Betrifft	Beschreibung	Wert
Unterstation (Zutrittszentrale/-Manager)	Unbekannt	*-1*

Betrifft	Beschreibung	Wert
	Zentrale Entscheidung.	*0*
	Lokale Entscheidung.	*1*
Durchgang	Unbekannt.	*-1*
	Zutrittsprofilprüfung.	*0*
	Einmalig freigeschaltet.	*1*
	Gesperrt.	*2*
	Dauerhaft freigeschaltet.	*3*
Drehkreuz	Unbekannt.	*-1*
	Zutrittsprofilprüfung.	*0*
	Einmalig freigeschaltet.	*1*
	Gesperrt.	*2*
	Dauerhaft freigeschaltet.	*3*
	Einmalig freigeschaltet außen.	*4*
	Einmalig freigeschaltet innen.	*5*
Lift	Unbekannt.	*-1*
	Zutrittsprofilprüfung.	*0*
	Gesperrt.	*2*
	Dauerhaft freigeschaltet.	*3*
Depot	Unbekannt.	*-1*
	Zutrittsprofilprüfung.	*0*
	Gesperrt.	*2*

PRIORITY

Optional.

Eine ganze Zahl, die die Alarmpriorität spezifiziert.

ALARMTEXT

Optional.

Eine Zeichenkette, die den Alarm beschreibt.

CHECKLIST

Optional.

Eine Zeichenkette, die beschreibt, was beim Auftreten des vorliegenden Alarms zu tun ist.

DATE

Optional.

Eine Zeichenkette, die das Datum angibt, an dem die Änderung aufgetreten ist.

Syntax: **YYYYMMDD**

TIME

Optional.

Eine Zeichenkette, die Uhrzeit angibt, an dem die Änderung aufgetreten ist.

Syntax: **hhmmss**

DOORTYPE

Optional.

Wird nur übergeben, wenn **Type** den Wert *DR* hat.

Wert	Beschreibung
0	Normaler Durchgang
1	Drehsperre

NUR FÜR ABONNENTEN

COMMAND

Optional.

Eine ganze Zahl, die eine Variablen-Befehlsnummer spezifiziert. Durch Angabe dieses Arguments kann ein Befehl an die Variable geschickt werden.

Command	Wert Type	Beschreibung
-1	*DR*	Normalbetrieb.
0	*DR*	Überwachen.
1	*DR*	Für einen Zutritt öffnen.
2	*DR*	Sperren.
3	*DR*	Statisch öffnen.
4	*DR*	Für einen Zutritt außen öffnen.
5	*DR*	Für einen Zutritt innen öffnen.
0	*PR*	Verbindung abbauen (bei Remote Anbindung).
1	*PR*	Verbindung aufbauen (bei Remote Anbindung).
0	*BO*	Ausschalten.
-1	*BO*	Einschalten.

Info

Eingänge und Ausgänge, die in einer Tür *DR* verwendet werden, können nur mit den Kommandos des Typs *DR* angesteuert werden.

Das heißt: Ein Türöffner kann nur mit dem Kommando **1** (**Für einen Zutritt öffnen**) angesteuert werden. Der Zustand eines Rahmenkontakts kann nur über den Status Durchgang *offen* oder *geschlossen* herausgefunden werden.

7.4 Variablen anlegen durch Import

Variablen können auch mittels Variablenimport angelegt werden. Für jeden Treiber stehen XML- und DBF-Import zur Verfügung.

Info

Details zu Import und Export von Variablen finden Sie im Handbuch Import-Export im Abschnitt Variablen.

7.4.1 XML Import

Beim XML- Import von Variablen oder Datentypen werden diese erst einem Treiber zugeordnet und dann analysiert. Vor dem Import entscheidet der Benutzer, ob und wie das jeweilige Element (Variable oder Datentyp) importiert werden soll:

- *Importieren*:
  Das Element wird neu importiert.
- *Überschreiben*:
  Das Element wird importiert und überschreibt ein bereits vorhandenes Element.
- *Nicht importieren*:
  Das Element wird nicht importiert.

Hinweis: Beim Import werden die Aktionen und deren Dauer in einem Fortschrittsbalken angezeigt. In der folgenden Dokumentation wird der Import von Variablen beschrieben. Datentypen werden analog dazu importiert.

VORAUSSETZUNGEN

Beim Import gelten folgende Bedingungen:

- **Abwärtskompatibilität**

  Beim XML Import/Export ist keine Abwärtskompatibilität gegeben. Daten aus älteren zenon Versionen können übernommen werden. Die Übergabe von Daten aus neueren Versionen an ältere wird nicht unterstützt.

- **Konsistenz**

  Die zu importierende XML-Datei muss konsistent sein. Beim Import der Datei erfolgt keine Plausibilitätsprüfung. Weisen die importierten Daten Fehler auf, kann es zu unerwünschten Effekten im Projekt kommen.

  Dies muss vor allem auch beachtet werden, wenn in einer XML-Datei nicht alle Eigenschaften vorhanden sind und diese dann durch Default-Werte ersetzt werden. Z. B.: Eine binäre Variable hat einen Grenzwert von *300*.

- **Struktur-Datentypen**

  Struktur-Datentypen müssen über die gleiche Anzahl von Strukturelementen verfügen. Beispiel: Ein Strukturdatentyp im Projekt hat 3 Strukturelemente. Ein gleichnamiger Datentyp in der XML-Datei hat 4 Strukturelemente. Dann wird keine der auf diesem Datentyp basierenden Variablen der Datei in das Projekt importiert.

> **Tipp**
>
> Weitere Informationen zum XML-Import finden Sie im Handbuch **Import - Export**, im Kapitel **XML-Import**.

7.4.2 DBF Import/Export

Daten können nach dBase exportiert und aus dBase importiert werden.

> **Info**
>
> Import und Export über CSV oder dBase unterstützt keine treiberspezifischen Variableneinstellungen wie z. B. Formeln. Nutzen Sie dafür den Export/Import über XML.

IMPORT DBF-DATEI

Um den Import zu starten:

1. Führen Sie einen Rechtsklick auf die Variablenliste aus.
2. Wählen Sie in der Dropdownliste von **Erweiterter Export/Import** ... den Befehl **dBase importieren**.
3. Folgen Sie den Anweisungen des Importassistenten.

Das Format der Datei ist im Kapitel Dateiaufbau beschrieben.

> **Info**
>
> Beachten Sie:
>
> - Treiberobjekttyp und Datentyp müssen in der DBF-Datei an den Zieltreiber angepasst werden, damit Variablen importiert werden.
> - dBase unterstützt beim Import keine Strukturen oder Arrays (komplexe Variablen).

EXPORT DBF-DATEI

Um den Export zu starten:

1. Führen Sie einen Rechtsklick auf die Variablenliste aus.
2. Wählen Sie im Dropdownliste von **Erweiterter Export/Import** ... den Befehl **dBase exportieren...** .

3. Folgen Sie den Anweisungen des Exportassistenten.

> **Achtung**
>
> DBF-Dateien:
>
> - müssen in der Benennung dem 8.3 DOS Format für Dateinamen entsprechen (8 alphanumerische Zeichen für Name, 3 Zeichen Erweiterung, keine Leerzeichen)
> - dürfen im Pfadnamen keinen Punkt (.) enthalten.
> Z. B. ist der Pfad *C:\users\Max.Mustermann\test.dbf* ungültig.
> Gültig wäre: *C:\users\MaxMustermann\test.dbf*
> - müssen nahe am Stammverzeichnis (Root) abgelegt werden, um die eventuelle Beschränkungen für Dateinamenlänge inklusive Pfad zu erfüllen: maximal 255 Zeichen

Das Format der Datei ist im Kapitel Dateiaufbau beschrieben.

> **Info**
>
> dBase unterstützt beim Export keine Strukturen oder Arrays (komplexe Variablen).

DATEIAUFBAU DER DBASE EXPORTDATEI

Für den Variablenimport und -export muss die dBaseIV-Datei folgende Struktur und Inhalte besitzen.

> **Achtung**
>
> dBase unterstützt keine Strukturen oder Arrays (komplexe Variablen).
>
> DBF-Dateien müssen:
>
> - in der Benennung dem 8.3 DOS Format für Dateinamen entsprechen (8 alphanumerische Zeichen für Name, 3 Zeichen Erweiterung, keine Leerzeichen)
> - nahe am Stammverzeichnis (Root) abgelegt werden

STRUKTUR

Bezeichnung	Typ	Feldgröße	Bemerkung
KANALNAME	Char	128	Variablenname. Länge kann über den Eintrag **MAX_LAENGE** in der **project.ini** eingeschränkt werden.

Bezeichnung	Typ	Feldgröße	Bemerkung
KANAL_R	C	128	Ursprünglicher Name einer Variablen, der durch den Eintrag unter VARIABLENNAME ersetzt werden soll (Feld/Spalte muss manuell angelegt werden). Länge kann über den Eintrag **MAX_LAENGE** in der **project.ini** eingeschränkt werden.
KANAL_D	Log	1	Variable wird bei Eintrag *1* gelöscht (Feld/Spalte muss manuell angelegt werden).
TAGNR	C	128	Kennung. Länge kann über den Eintrag **MAX_LAENGE** in der **project.ini** eingeschränkt werden.
EINHEIT	C	11	Technische Maßeinheit
DATENART	C	3	Datentyp (z. B. Bit, Byte, Wort, ...) entspricht dem Datentyp.
KANALTYP	C	3	Speicherbereich in der SPS (z. B. Merkerbereich, Datenbereich, ...) entspricht Treiberobjekttyp.
HWKANAL	Num	3	Netzadresse
BAUSTEIN	N	3	Datenbaustein-Adresse (nur bei Variablen aus den Datenbereich der SPS)
ADRESSE	N	5	Offset
BITADR	N	2	Für Bit-Variablen: Bitadresse Für Byte-Variablen: 0=niederwertig, 8=höherwertig Für String-Variablen: Stringlänge (max. 63 Zeichen)
ARRAYSIZE	N	16	Anzahl der Variablen im Array für Index-Variablen ACHTUNG: Nur die erste Variable steht voll zur Verfügung. Alle folgenden sind nur über VBA oder den Rezeptgruppen Manager zugänglich
LES_SCHR	L	1	Lese-Schreib-Berechtigung 0: Sollwert setzen ist nicht erlaubt 1: Sollwert setzen ist erlaubt
MIT_ZEIT	L	1	Zeitstempelung in zenon (nur wenn vom Treiber unterstützt)

Bezeichnung	Typ	Feldgröße	Bemerkung
OBJEKT	N	2	Treiberspezifische ID-Nummer des Primitivobjekts setzt sich zusammen aus TREIBER-OBJEKTTYP und DATENTYP
SIGMIN	Floa t	16	Rohwertsignal minimal (Signalauflösung)
SIGMAX	F	16	Rohwertsignal maximal (Signalauflösung)
ANZMIN	F	16	technischer Wert minimal (Messbereich)
ANZMAX	F	16	technischer Wert maximal (Messbereich)
ANZKOMMA	N	1	Anzahl der Nachkommastellen für die Darstellung der Werte (Messbereich)
UPDATERATE	F	19	Updaterate für Mathematikvariablen (in sec, eine Dezimalstelle möglich) bei allen anderen Variablen nicht verwendet
MEMTIEFE	N	7	Nur aus Kompatibilitätsgründen vorhanden
HDRATE	F	19	HD-Updaterate für hist. Werte (in sec, eine Dezimalstelle möglich)
HDTIEFE	N	7	HD-Eintragtiefe für hist. Werte (Anzahl)
NACHSORT	L	1	HD-Werte als nachsortierte Werte
DRRATE	F	19	Aktualisierung an die Ausgabe (für zenon DDE-Server, in sec, eine Kommastelle möglich)
HYST_PLUS	F	16	Positive Hysterese; ausgehend vom Messbereich
HYST_MINUS	F	16	Negative Hyterese; ausgehend vom Messbereich
PRIOR	N	16	Priorität der Variable
REAMATRIZE	C	32	Name der zugeordnete Reaktionsmatrix
ERSATZWERT	F	16	Ersatzwert; ausgehend vom Messbereich
SOLLMIN	F	16	Sollwertgrenze Minimum; ausgehend vom Messbereich
SOLLMAX	F	16	Sollwertgrenze Maximum; ausgehend vom Messbereich
VOMSTANDBY	L	1	Variable vom Standby Server anfordern; der Wert der Variable wird im redundanten Netzwerkbetrieb nicht vom Server sondern vom Standby Server angefordert

Bezeichnung	Typ	Feldgröße	Bemerkung
RESOURCE	C	128	Betriebsmittelkennung. Freier String für Export und Anzeige in Listen. Länge kann über den Eintrag MAX_LAENGE in der **project.ini** eingeschränkt werden.
ADJWVBA	L	1	Nichtlineare Wertanpassung: *0*: Nichtlineare Wertanpassung wird verwendet *1*: Nichtlineare Wertanpassung wird nicht verwendet
ADJZENON	C	128	Verknüpftes VBA-Makro zum Lesen der Variablenwerte für die nichtlineare Wertanpassung.
ADJWVBA	C	128	Verknüpftes VBA-Makro zum Schreiben der Variablenwerte für die nichtlineare Wertanpassung.
ZWREMA	N	16	Verknüpfte Zählwert-Rema.
MAXGRAD	N	16	Maximaler Gradient für die Zählwert-Rema.

Achtung

Beim Import müssen Treiberobjekttyp und Datentyp in der DBF-Datei an den Zieltreiber angepasst werden, damit Variablen importiert werden.

GRENZWERTDEFINITION

Grenzwertdefinition für Grenzwert *1* bis *4*, oder Zustand *1* bis *4*:

Bezeichnung	Typ	Feldgröße	Bemerkung
AKTIV1	L	1	Grenzwert aktiv (pro Grenzwert vorhanden)
GRENZWERT1	F	20	technischer Wert oder ID-Nummer der verknüpften Variable für einen dynamischen Grenzwert (siehe VARIABLEx) (wenn unter VARIABLEx *1* steht und hier *-1*, wird die bestehende Variablenzuordnung nicht überschrieben)
SCHWWERT1	F	16	Schwellwert für den Grenzwert
HYSTERESE1	F	14	wird nicht verwendet
BLINKEN1	L	1	Blinkattribut setzen

Bezeichnung	Typ	Feldgröße	Bemerkung
BTB1	L	1	Protokollierung in CEL
ALARM1	L	1	Alarm
DRUCKEN1	L	1	Druckerausgabe (bei CEL oder Alarm)
QUITTIER1	L	1	quittierpflichtig
LOESCHE1	L	1	löschpflichtig
VARIABLE1	L	1	dyn. Grenzwertverknüpfung der Grenzwert wird nicht durch einen absoluten Wert (siehe Feld GRENZWERTx) festgelegt.
FUNC1	L	1	Funktionsverknüpfung
ASK_FUNC1	L	1	Ausführung über die Alarmmeldeliste
FUNC_NR1	N	10	ID-Nummer der verknüpften Funktion (steht hier -1, so wird die bestehende Funktion beim Import nicht überschrieben)
A_GRUPPE1	N	10	Alarm/Ereignis-Gruppe
A_KLASSE1	N	10	Alarm/Ereignis-Klasse
MIN_MAX1	C	3	Minimum, Maximum
FARBE1	N	10	Farbe als Windowskodierung
GRENZTXT1	C	66	Grenzwerttext
A_DELAY1	N	10	Zeitverzögerung
INVISIBLE1	L	1	Unsichtbar

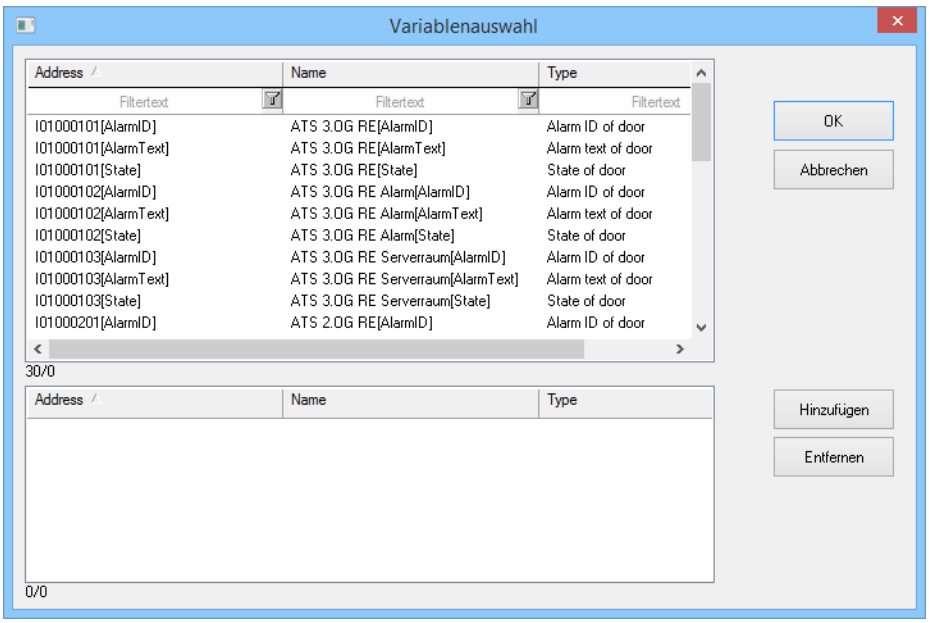
Bezeichnungen in der Spalte Bemerkung beziehen sich auf die in den Dialogboxen zur Definition von Variablen verwendeten Begriffe. Bei Unklarheiten, siehe Kapitel Variablendefinition.

7.4.3 Online-Import

Um Variablen online vom Server zu importieren:

1. Wählen Sie den KABA Treiber aus.
2. Wählen Sie in der Symbolleiste oder im Kontextmenü **Variablen vom Treiber importieren**.

3. Der Dialog zur Variablenauswahl wird geöffnet:



4. Wählen Sie die gewünschten Variablen (Mehrfachauswahl ist möglich).
5. Fügen Sie die Variablen durch Klick auf die Schaltfläche **Hinzufügen** zur Liste der zu importierenden Variablen hinzu.
6. Mit Klick auf **Entfernen** können Sie Variablen auch wieder abwählen.
7. Starten Sie den Import durch Klick auf die Schaltfläche **OK**.

Die ausgewählten Variablen werden beim Import automatisch im zenon Projekt erstellt und dem KABA Treiber zugeordnet.

7.5 Kommunikationsdetails (Treibervariablen)

Das Treiberkit implementiert eine Reihe von Treibervariablen, welche in dem Treiberobjekttyp *Kommunikationsdetails* zusammengefasst sind. Diese sind unterteilt in:

- Information
- Konfiguration
- Statistik und
- Fehlermeldungen

Die Definitionen der im Treiberkit implementierten Variablen sind in der Importdatei **DRVVAR.DBF** verfügbar und können von dort importiert werden.
Pfad zur Datei: *%ProgramData%\COPA-DATA\zenon<Versionsnummer>\PredefinedVariables*

Hinweis: Variablennamen müssen in zenon einzigartig sein. Soll nach einem Import der Variablen vom Treiberobjekttyp *Kommunikationsdetails* aus **DRVVAR.DBF** ein erneuter Import durchgeführt werden, müssen die zuvor importierten Variablen umbenannt werden.

Info

Nicht jeder Treiber unterstützt alle Treibervariablen des Treiberobjekttyps *Kommunikationsdetails*.

Zum Beispiel:

- Variablen für Modem-Informationen werden nur von modemfähigen Treibern unterstützt.
- Treibervariablen für den Polling-Zyklus stehen nur für rein pollende Treiber zur Verfügung.
- Verbindungsbezogene Informationen wie **ErrorMSG** werden nur von Treibern unterstützt, die zu einem Zeitpunkt nur eine Verbindung bearbeiten.

INFORMATION

Name aus Import	Typ	Offset	Erklärung
MainVersion	*UINT*	0	Haupt-Versionsnummer des Treibers.
SubVersion	*UINT*	1	Sub-Versionsnummer des Treibers.
BuildVersion	*UINT*	29	Build-Versionsnummer des Treibers.
RTMajor	*UINT*	49	zenon Hauptversionsnummer
RTMinor	*UINT*	50	zenon Sub-Versionsnummer
RTSp	*UINT*	51	zenon Service Pack-Nummer
RTBuild	*UINT*	52	zenon Buildnummer
LineStateIdle	*BOOL*	24.0	TRUE, wenn die Modemleitung belegt ist.
LineStateOffering	*BOOL*	24.1	TRUE, wenn ein Anruf rein kommt.
LineStateAccepted	*BOOL*	24.2	Der Anruf wird angenommen.
LineStateDialtone	*BOOL*	24.3	Rufton wurde erkannt.
LineStateDialing	*BOOL*	24.4	Wahl aktiv.
LineStateRingBack	*BOOL*	24.5	Während Verbindungsaufbau.
LineStateBusy	*BOOL*	24.6	Zielstation besetzt.
LineStateSpecialInfo	*BOOL*	24.7	Spezielle Statusinformation empfangen.

Name aus Import	Typ	Offset	Erklärung
LineStateConnected	*BOOL*	24.8	Verbindung hergestellt.
LineStateProceeding	*BOOL*	24.9	Wahl ausgeführt.
LineStateOnHold	*BOOL*	24.10	Verbindung in Halten.
LineStateConferenced	*BOOL*	24.11	Verbindung im Konferenzmodus.
LineStateOnHoldPendConf	*BOOL*	24.12	Verbindung in Halten für Konferenz.
LineStateOnHoldPendTransfer	*BOOL*	24.13	Verbindung in Halten für Transfer.
LineStateDisconnected	*BOOL*	24.14	Verbindung beendet.
LineStateUnknow	*BOOL*	24.15	Verbindungszustand nicht bekannt.
ModemStatus	*UDINT*	24	Aktueller Modemstatus.
TreiberStop	*BOOL*	28	Treiber gestoppt Bei *Treiberstop*, hat die Variable den Wert *TRUE* und ein **OFF**-Bit. Nach dem Treiberstart, hat die Variable den Wert *FALSE* und kein **OFF**-Bit.
SimulRTState	*UDINT*	60	Informiert über Status der Runtime bei Treibersimulation.
ConnectionStates	*STRING*	61	Interner Verbindungsstatus des Treibers zur SPS. Verbindungszustände: ▸ *0:* Verbindung OK ▸ *1:* Verbindung gestört ▸ *2:* Verbindung simuliert Formatierung: **<Netzadresse>:<Verbindungszustand>;...;...;** Eine Verbindung ist erst nach dem ersten Anmelden einer Variablen bekannt. Damit eine Verbindung im String enthalten ist, muss einmal eine Variable dieser Verbindung angemeldet worden sein.

Name aus Import	Typ	Offset	Erklärung
			Der Zustand einer Verbindung wird nur aktualisiert, wenn eine Variable der Verbindung angemeldet ist. Ansonsten wird nicht mit der entsprechenden Steuerung kommuniziert.

KONFIGURATION

Name aus Import	Typ	Offset	Erklärung
ReconnectInRead	*BOOL*	27	Wenn TRUE, dann wird beim Lesen automatisch ein Neuaufbau der Verbindung durchgeführt.
ApplyCom	*BOOL*	36	Änderungen an den Einstellungen der seriellen Schnittstelle zuweisen. Das Schreiben auf diese Variable hat unmittelbar den Aufruf der Methode SrvDrvVarApplyCom zur Folge (aktuell ohne weitere Funktion).
ApplyModem	*BOOL*	37	Änderungen an den Modemeinstellungen zuweisen. Das Schreiben auf diese Variable hat unmittelbar den Aufruf der Methode SrvDrvVarApplyModem zur Folge. Diese schließt die aktuelle Verbindung und öffnet eine neue entsprechend den Einstellungen **PhoneNumberSet** und **ModemHwAdrSet**.
PhoneNumberSet	*STRING*	38	Telefonnummer, welche verwendet werden soll.
ModemHwAdrSet	*DINT*	39	Hardwareadresse, welche zu der Telefonnummer gehört.
GlobalUpdate	*UDINT*	3	Updatezeit in Millisekunden (ms).
BGlobalUpdaten	*BOOL*	4	TRUE, wenn die Updatezeit global ist.
TreiberSimul	*BOOL*	5	TRUE, wenn der Treiber in Simulation ist.
TreiberProzab	*BOOL*	6	TRUE, wenn das Prozessabbild gehalten werden soll.
ModemActive	*BOOL*	7	TRUE, wenn das Modem bei diesem Treiber aktiv ist.

Name aus Import	Typ	Offset	Erklärung
Device	*STRING*	8	Name der seriellen Schnittstelle oder Name des Modem.
ComPort	*UINT*	9	Nummer der seriellen Schnittstelle.
Baudrate	*UDINT*	10	Baudrate der seriellen Schnittstelle.
Parity	*SINT*	11	Parität der seriellen Schnittstelle.
ByteSize	*USINT*	14	Bitanzahl pro Zeichen der seriellen Schnittstelle. Wert = *0*, wenn der Treiber keine serielle Kommunikation herstellen kann.
StopBit	*USINT*	13	Anzahl der Stoppbits der seriellen Schnittstelle.
Autoconnect	*BOOL*	16	TRUE, wenn die Modemverbindung automatisch beim Lesen/Schreiben aufgebaut werden soll.
PhoneNumber	*STRING*	17	Aktuelle Telefonnummer.
ModemHwAdr	*DINT*	21	Hardwareadresse zur aktuellen Telefonnummer.
RxIdleTime	*UINT*	18	Wenn länger als diese Zeit in Sekunden (s) erfolgreich kein Datenverkehr stattfindet, wird die Modemverbindung beendet.
WriteTimeout	*UDINT*	19	Maximale Schreibdauer bei einer Modemverbindung in Millisekunden (ms).
RingCountSet	*UDINT*	20	So oft läutet ein hereinkommender Anruf, bevor dieser angenommen wird.
ReCallIdleTime	*UINT*	53	Wartezeit zwischen Anrufen in Sekunden (s).
ConnectTimeout	*UINT*	54	Zeit in Sekunden (s) für Verbindungsaufbau.

STATISTIK

Name aus Import	Typ	Offset	Erklärung
MaxWriteTime	*UDINT*	31	Längste Zeit in Millisekunden (ms), die zum Schreiben benötigt wird.

Name aus Import	Typ	Offset	Erklärung
MinWriteTime	*UDINT*	32	Kürzeste Zeit in Millisekunden (ms), die zum Schreiben benötigt wird.
MaxBlkReadTime	*UDINT*	40	Längste Zeit in Millisekunden (ms), die zum Lesen eines Datenblocks benötigt wird.
MinBlkReadTime	*UDINT*	41	Kürzeste Zeit in Millisekunden (ms), die zum Lesen eines Datenblocks benötigt wird.
WriteErrorCount	*UDINT*	33	Anzahl der Schreibfehler.
ReadSucceedCount	*UDINT*	35	Anzahl der erfolgreichen Leseversuche.
MaxCycleTime	*UDINT*	22	Längste Zeit in Millisekunden (ms), die zum Lesen aller angeforderten Daten benötigt wurde.
MinCycleTime	*UDINT*	23	Kürzeste Zeit in Millisekunden (ms), die zum Lesen aller angeforderten Daten benötigt wurde.
WriteCount	*UDINT*	26	Anzahl der Schreibversuche.
ReadErrorCount	*UDINT*	34	Anzahl der fehlerhaften Leseversuche.
MaxUpdateTimeNormal	*UDINT*	56	Zeit seit letzter Aktualisierung der Prioritätsgruppe **Normal** in Millisekunden (ms).
MaxUpdateTimeHigher	*UDINT*	57	Zeit seit letzter Aktualisierung der Prioritätsgruppe **Höher** in Millisekunden (ms).
MaxUpdateTimeHigh	*UDINT*	58	Zeit seit letzter Aktualisierung der Prioritätsgruppe **Hoch** in Millisekunden (ms).
MaxUpdateTimeHighest	*UDINT*	59	Zeit seit letzter Aktualisierung der Prioritätsgruppe **Höchste** in Millisekunden (ms).
PokeFinish	*BOOL*	55	Geht für eine Abfrage auf *1*, wenn alle anstehenden Pokes ausgeführt wurden.

FEHLERMELDUNGEN

Name aus Import	Typ	Offset	Erklärung
ErrorTimeDW	*UDINT*	2	Zeit (in Sekunden seit 1.1.1970), wann der letzte Fehler auftrat.
ErrorTimeS	*STRING*	2	Zeit (in Sekunden seit 1.1.1970), wann der letzte Fehler als String auftrat.

Name aus Import	Typ	Offset	Erklärung
RdErrPrimObj	*UDINT*	42	Nummer des PrimObjektes, als der letzte Lesefehler verursacht wurde.
RdErrStationsName	*STRING*	43	Name der Station, als der letzte Lesefehler verursacht wurde.
RdErrBlockCount	*UINT*	44	Anzahl der zu lesenden Blöcke, als der letzte Lesefehler verursacht wurde.
RdErrHwAdresse	*DINT*	45	Hardwareadresse, als der letzte Lesefehler verursacht wurde.
RdErrDatablockNo	*UDINT*	46	Bausteinnummer, als der letzte Lesefehler verursacht wurde.
RdErrMarkerNo	*UDINT*	47	Merkernummer, als der letzte Lesefehler verursacht wurde.
RdErrSize	*UDINT*	48	Blockgröße, als der letzte Lesefehler verursacht wurde.
DrvError	*USINT*	25	Fehlermeldung als Nummer.
DrvErrorMsg	*STRING*	30	Fehlermeldung als Klartext.
ErrorFile	*STRING*	15	Name der Fehlerprotokolldatei.

8 Treiberspezifische Funktionen

Dieser Treiber unterstützt folgende Funktionen:

Darstellung der Zustände und Alarme von Türen, die von einem KABA Server überwacht werden.

9 Funktion Treiberkommandos

Die zenon Funktion **Treiberkommandos** dient dazu, Treiber über zenon zu beeinflussen.
Mit einem Treiberkommando können Sie einen Treiber:

- starten
- stoppen

- in einen bestimmten Treibermodus versetzen
- zu bestimmten Aktionen veranlassen

Hinweis: Dieses Kapitel beschreibt Standardfunktionalitäten, die für die meisten zenon Treiber gültig sind.
Nicht alle hier beschriebenen Funktionalitäten stehen für jeden Treiber zur Verfügung. Zum Beispiel enthält ein Treiber, der laut Datenblatt keine Modemverbindung unterstützt, auch keine Modem-Funktionalitäten.

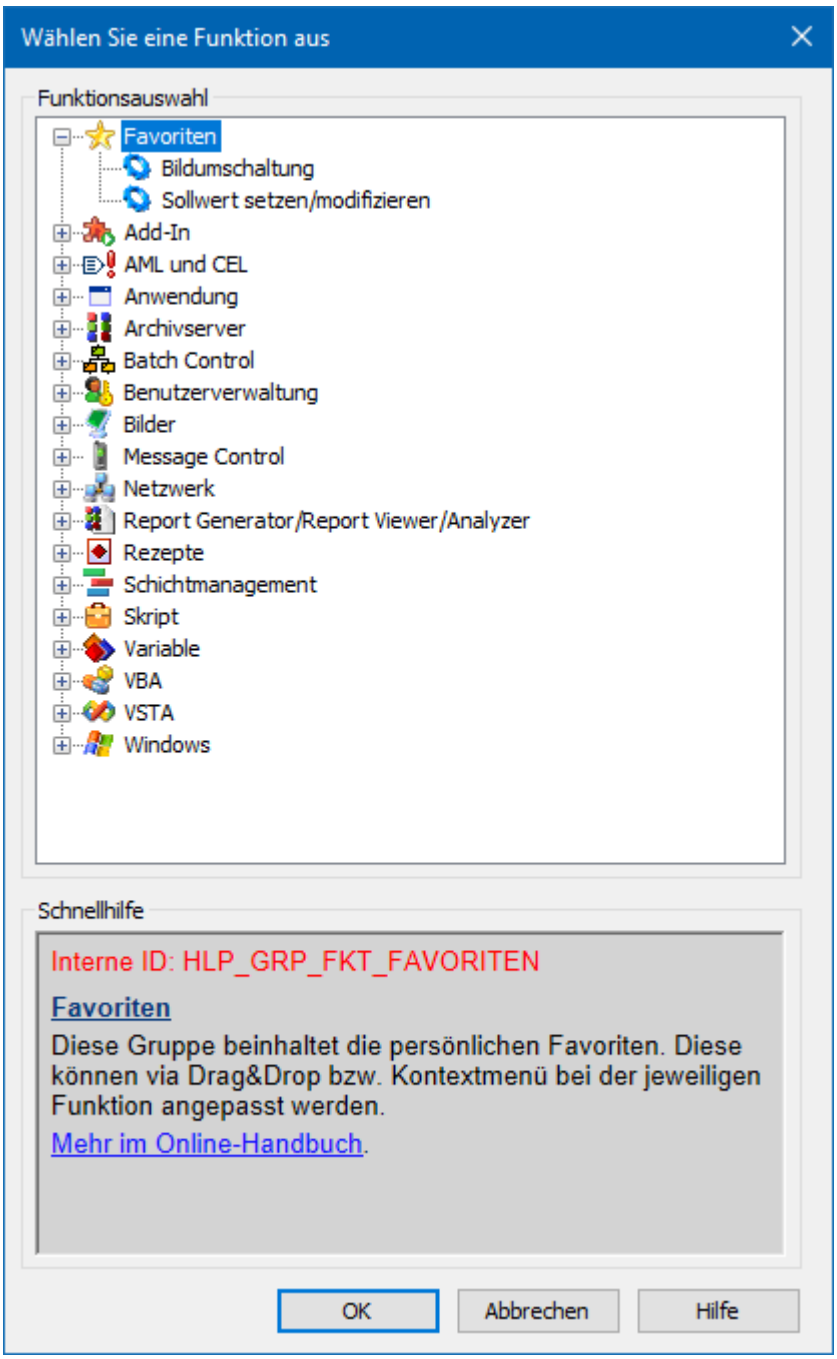
> **Achtung**
>
> Die zenon Funktion **Treiberkommandos** ist nicht ident mit den Treiberkommandos, die bei Energy-Treibern in der Runtime ausgeführt werden können!

PROJEKTIERUNG DER FUNKTION

Die Projektierung erfolgt über die Funktion **Treiberkommandos**.
Um die Funktion zu projektieren:

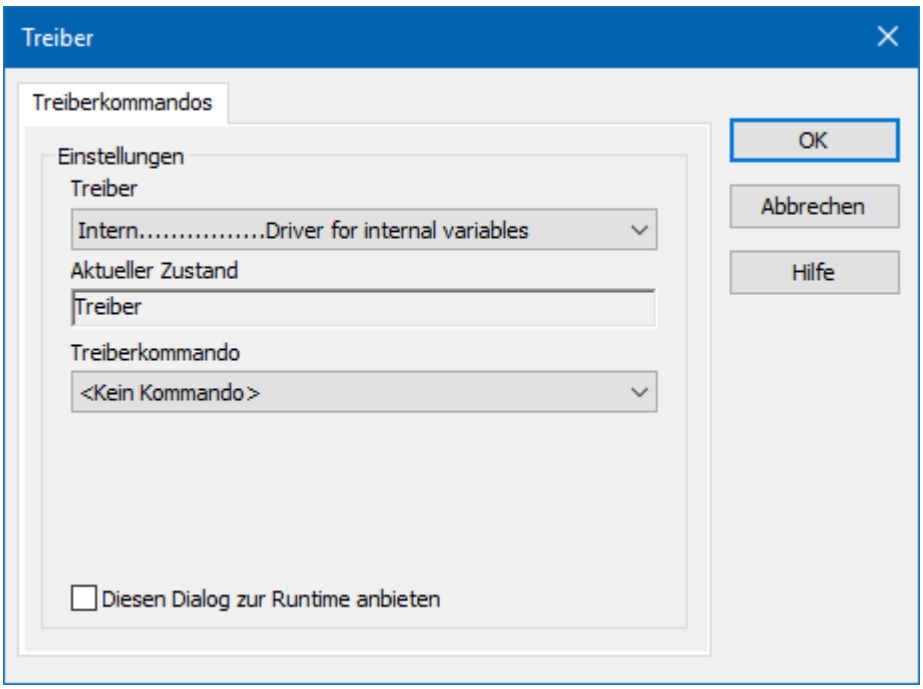
1. Legen Sie im zenon Editor eine neue Funktion an.

Der Dialog zur Auswahl einer Funktion wird geöffnet.



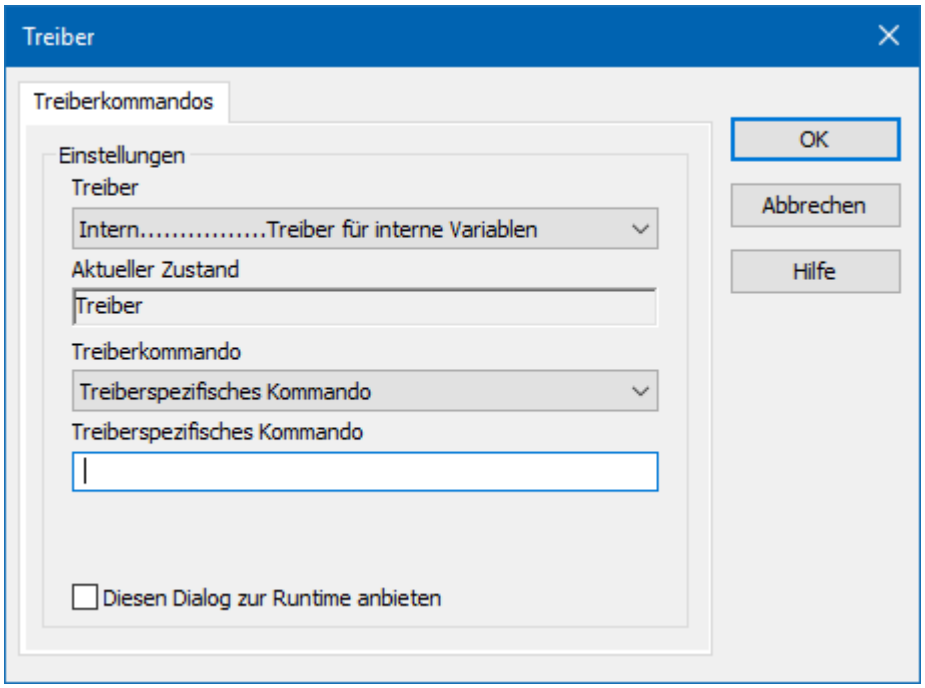
2. Navigieren Sie zum Knoten **Variable.**
3. Wählen Sie den Eintrag **Treiberkommandos**.

Der Dialog zur Konfiguration wird geöffnet.



4. Wählen Sie den gewünschten Treiber und das benötigte Kommando aus.
5. Schließen Sie den Dialog mit Klick auf **OK** und stellen Sie sicher, dass die Funktion in der Runtime ausgeführt wird.
   Beachten Sie die Hinweise im Abschnitt **Funktion Treiberkommandos im Netzwerk**.

DIALOG TREIBERKOMMANDOS



Option	Beschreibung
Treiber	Auswahl des Treibers aus der Dropdownliste. Diese enthält alle im Projekt geladenen Treibern.
Aktueller Zustand	Fixer Eintrag, der vom System gesetzt wird. In aktuellen Versionen ohne Funktion.
Treiberkommando	Auswahl des gewünschten Treiberkommandos aus Dropdownliste. Details zu den konfigurierbaren Treiberkommandos siehe Abschnitt **Verfügbare Treiberkommandos**.
Treiberspezifisches Kommando	Eingabe eines für den gewählten Treiber spezifischen Kommandos. **Hinweis:** Nur verfügbar, wenn für die Option **Treiberkommando** der Eintrag *Treiberspezifisches Kommando* gewählt wurde.
Diesen Dialog zur Runtime anbieten	Konfiguration, ob die Konfiguration in der Runtime geändert werden kann: ▸ *Aktiv*: Dieser Dialog wird in der Runtime vor dem Ausführen der Funktion geöffnet. Die Konfiguration kann damit in der Runtime vor der Ausführung noch geändert werden. ▸ *Inaktiv*: Die Editor-Konfiguration wird in der Runtime beim Ausführen der Funktion angewendet. Default: *inaktiv*

DIALOG BEENDEN

Option	Beschreibung
OK	Übernimmt Einstellungen und schließt den Dialog.
Abbrechen	Verwirft alle Änderungen und schließt den Dialog.
Hilfe	Öffnet die Online-Hilfe.

VERFÜGBARE TREIBERKOMMANDOS

Diese Treiberkommandos stehen - abhängig vom gewählten Treiber - zur Verfügung:

Treiberkommando	Beschreibung
Kein Kommando	Es wird kein Kommando gesendet. Damit kann auch ein bereits bestehendes Kommando aus einer projektierten Funktion entfernt werden.
Treiber starten (Onlinemodus)	Treiber wird neu initialisiert und gestartet. **Hinweis:** Wenn der Treiber bereits gestartet wurde, muss er gestoppt werden. Erst dann kann der Treiber wieder neu initialisiert und gestartet werden.
Treiber stoppen (Offlinemodus)	Treiber wird angehalten, es werden keine neuen Daten angenommen. **Hinweis:** Ist der Treiber im Offline-Modus, erhalten alle Variablen, die für diesem Treiber angelegt wurden, den Status *Abgeschaltet* (*OFF*; Bit *20*).
Treiber in Simulationsmodus	Treiber wird in den Simulationsmodus gesetzt. Die Werte aller Variablen des Treibers werden vom Treiber simuliert. Es werden keine Werte von der angeschlossenen Hardware (z. B. SPS, Bussystem, ...) angezeigt.
Treiber in Hardwaremodus	Treiber wird in den Hardwaremodus gesetzt. Für die Variablen des Treibers werden die Werte von der angeschlossenen Hardware (z. B. SPS, Bussystem, ...) angezeigt.
Treiberspezifisches Kommando	Eingabe eines treiberspezifischen Kommandos. Öffnet Eingabefeld für die Eingabe eines Kommandos.
Treiber Sollwertsetzen aktivieren	Sollwert setzen auf Treiber ist möglich.
Treiber Sollwertsetzen deaktivieren	Sollwert setzen auf Treiber wird verhindert.
Verbindung mit Modem aufbauen	Verbindung aufbauen (für Modem-Treiber). Öffnet Eingabefelder für Hardware-Adresse und Eingabe der zu wählenden Nummer.
Verbindung mit Modem trennen	Verbindung beenden (für Modem-Treiber).
Treiber in Simulationsmodus zählend	Treiber wird in den zählenden Simulationsmodus gesetzt. Alle Werte werden mit *0* initialisiert und in der eingestellten Updatezeit jeweils um *1* bis zum Maximalwert inkrementiert und beginnen dann wieder

Treiberkommando	Beschreibung
	bei *0*.
Treiber in Simulationsmodus statisch	Es wird keine Kommunikation zur Steuerung aufgebaut. Alle Werte werden mit *0* initialisiert.
Treiber in Simulationsmodus programmiert	Die Werte werden von einem frei programmierbaren Simulationsprojekt berechnet. Das Simulationsprojekt wird mit der zenon Logic Workbench erstellt und läuft in der zenon Logic Runtime ab.

FUNKTION TREIBERKOMMANDOS IM NETZWERK

Wenn sich der Rechner, auf dem die Funktion **Treiberkommandos** ausgeführt wird, im zenon Netzwerk befindet, werden zusätzlich weitere Aktionen ausgeführt:

- Ein spezielles Netzwerkkommando wird vom Rechner zum Server des Projekts gesendet. Dieser führt dann die gewünschte Aktion auf seinem Treiber aus.
- Zusätzlich sendet der Server das gleiche Treiberkommando zum Standby des Projekts. Der Standby führt die Aktion auch auf seinem Treiber aus.

Dadurch ist gewährleistet, dass Server und Standby synchronisiert sind. Dies funktioniert nur, wenn Server und Standby jeweils eine funktionierende und unabhängige Verbindung zur Hardware haben.

10 Fehleranalyse

Sollte es zu Kommunikationsproblemen kommen, bietet dieses Kapitel Hilfe, um den Fehler zu finden.

10.1 TCP API-Fehlernummern

ERROR CODES IN THE API

The following is a list of possible error codes returned by the WSAGetLastError call, along with their ex-tended explanations. Errors are listed in alphabetical order by error macro. Some error codes defined in Winsock2.h are not returned from any function-these are not included in this topic.

Error (Code)	Meaning	Description
WSAEACCES **(10013)**	Permission denied.	An attempt was made to access a socket in a way forbidden by its access permissions. An example is using a broadcast address for sendto without broadcast permission being set using set-sockopt(SO_BROADCAST).

Error (Code)	Meaning	Description
		Another possible reason for the WSAEACCES error is that when the bind function is called (on Windows NT 4 SP4 or later), another application, service, or kernel mode driver is bound to the same address with exclusive access. Such exclusive access is a new feature of Windows NT 4 SP4 and later, and is imple-mented by using the SO_EXCLUSIVEADDRUSE option.
WSAEADDRINUSE **(10048)**	Address already in use.	Typically, only one usage of each socket address (protocol/IP address/port) is permitted. This error oc-curs if an application attempts to bind a socket to an IP address/port that has already been used for an existing socket, or a socket that was not closed properly, or one that is still in the process of closing. For server applications that need to bind multiple sockets to the same port number, consider using set-sockopt(SO_REUSEADDR). Client applications usually need not call bind at all-connect chooses an unused port automatically. When bind is called with a wildcard address (involving ADDR_ANY), a WSAEADDRINUSE error could be delayed until the specific address is committed. This could happen with a call to another function later, including connect, listen, WSAConnect, or WSAJoinLeaf.
WSAEADDRNOTAVAIL **(10049)**	Cannot assign requested address.	The requested address is not valid in its context. This normally results from an attempt to bind to an address that is not valid for the local machine. This can also result from connect, sendto, WSAConnect, WSAJoinLeaf, or WSASendTo when the remote address or port is not valid for a remote machine (for example, address or port 0).
WSAEAFNOSUPPORT **(10047)**	Address family not supported by protocol family.	An address incompatible with the requested protocol was used. All sockets are created with an associ-ated address family (that is, AF_INET for Internet Protocols) and a generic protocol type (that is, SOCK_STREAM). This error is returned if an incorrect protocol is explicitly requested in the socket call, or if an address of the wrong family is used for a socket, for example, in sendto.
WSAEALREADY	Operation already in	An operation was attempted on a nonblocking socket with an operation already in progress-that is, calling

Error (Code)	Meaning	Description
(10037)	progress.	connect a second time on a nonblocking socket that is already connecting, or canceling an asynchronous request (WSAAsyncGetXbyY) that has already been canceled or completed.
WSAECONNABORTED **(10053)**	Software caused connection abort.	An established connection was aborted by the software in your host machine, possibly due to a data transmission time-out or protocol error.
WSAECONNREFUSED **(10061)**	Connection refused.	No connection could be made because the target machine actively refused it. This usually results from trying to connect to a service that is inactive on the foreign host-that is, one with no server application running.
WSAECONNRESET **(10054)**	Connection reset by peer.	An existing connection was forcibly closed by the remote host. This normally results if the peer application on the remote host is suddenly stopped, the host is rebooted, or the remote host uses a hard close (see setsockopt for more information on the SO_LINGER option on the remote socket.) This error may also result if a connection was broken due to keep-alive activity detecting a failure while one or more operations are in progress. Operations that were in progress fail with WSAENETRESET. Subsequent operations fail with WSAECONNRESET.
WSAEDESTADDRREQ **(10039)**	Destination address required.	A required address was omitted from an operation on a socket. For example, this error is returned if sendto is called with the remote address of ADDR_ANY.
WSAEFAULT **(10014)**	Bad address.	The system detected an invalid pointer address in attempting to use a pointer argument of a call. This error occurs if an application passes an invalid pointer value, or if the length of the buffer is too small. For instance, if the length of an argument, which is a SOCKADDR structure, is smaller than the sizeof(SOCKADDR).
WSAEHOSTDOWN **(10064)**	Host is down.	A socket operation failed because the destination host is down. A socket operation encountered a dead host. Networking activity on the local host has not been initiated. These conditions are more likely to be indicated by the error WSAETIMEDOUT.

Error (Code)	Meaning	Description
WSAEHOSTUNREACH **(10065)**	No route to host.	A socket operation was attempted to an unreachable host. See WSAENETUNREACH.
WSAEINPROGRESS **(10036)**	Operation now in progress.	A blocking operation is currently executing. Windows Sockets only allows a single blocking operation-per-task or thread-to be outstanding, and if any other function call is made (whether or not it references that or any other socket) the function fails with the WSAEINPROGRESS error.
WSAEINTR **(10004)**	Interrupted function call.	A blocking operation was interrupted by a call to WSACancelBlockingCall.
WSAEINVAL **(10022)**	Invalid argument.	Some invalid argument was supplied (for example, specifying an invalid level to the setsockopt function). In some instances, it also refers to the current state of the socket-for instance, calling accept on a socket that is not listening.
WSAEISCONN **(10056)**	Socket is already connected.	A connect request was made on an already-connected socket. Some implementations also return this error if sendto is called on a connected SOCK_DGRAM socket (for SOCK_STREAM sockets, the to pa-rameter in sendto is ignored) although other implementations treat this as a legal occurrence.
WSAEMFILE **(10024)**	Too many open files.	Too many open sockets. Each implementation may have a maximum number of socket handles avail-able, either globally, per process, or per thread.
WSAEMSGSIZE **(10040)**	Message too long.	A message sent on a datagram socket was larger than the internal message buffer or some other network limit, or the buffer used to receive a datagram was smaller than the datagram itself.
WSAENETDOWN **(10050)**	Network is down.	A socket operation encountered a dead network. This could indicate a serious failure of the network sys-tem (that is, the protocol stack that the Windows Sockets DLL runs over), the network interface, or the local network itself.
WSAENETRESET **(10052)**	Network dropped	The connection has been broken due to keep-alive activity detecting a failure while the operation was in

Error (Code)	Meaning	Description
	connection on reset.	progress. It can also be returned by setsockopt if an attempt is made to set SO_KEEPALIVE on a con-nection that has already failed.
WSAENETUNREACH (10051)	Network is unreachable.	A socket operation was attempted to an unreachable network. This usually means the local software knows no route to reach the remote host.
WSAENOBUFS (10055)	No buffer space available.	An operation on a socket could not be performed because the system lacked sufficient buffer space or because a queue was full.
WSAENOPROTOOPT (10042)	Bad protocol option.	An unknown, invalid or unsupported option or level was specified in a getsockopt or setsockopt call.
WSAENOTCONN (10057)	Socket is not connected.	A request to send or receive data was disallowed because the socket is not connected and (when send-ing on a datagram socket using sendto) no address was supplied. Any other type of operation might also return this error-for example, setsockopt setting SO_KEEPALIVE if the connection has been reset.
WSAENOTSOCK (10038)	Socket operation on nonsocket.	An operation was attempted on something that is not a socket. Either the socket handle parameter did not reference a valid socket, or for select, a member of an fd_set was not valid.
WSAEOPNOTSUPP (10045)	Operation not supported.	The attempted operation is not supported for the type of object referenced. Usually this occurs when a socket descriptor to a socket that cannot support this operation is trying to accept a connection on a datagram socket.
WSAEPFNOSUPPORT (10046)	Protocol family not supported.	The protocol family has not been configured into the system or no implementation for it exists. This mes-sage has a slightly different meaning from WSAEAFNOSUPPORT. However, it is interchangeable in most cases, and all Windows Sockets functions that return one of these messages also specify WSAEAFNOSUPPORT.
WSAEPROCLIM (10067)	Too many processes.	A Windows Sockets implementation may have a limit on the number of applications that can use it simul-taneously. WSAStartup may fail with this error if the limit has been reached.

Error (Code)	Meaning	Description
WSAEPROTONOSUPPORT **(10043)**	Protocol not supported.	The requested protocol has not been configured into the system, or no implementation for it exists. For example, a socket call requests a SOCK_DGRAM socket, but specifies a stream protocol.
WSAEPROTOTYPE **(10041)**	Protocol wrong type for socket.	A protocol was specified in the socket function call that does not support the semantics of the socket type requested. For example, the ARPA Internet UDP protocol cannot be specified with a socket type of SOCK_STREAM.
WSAESHUTDOWN **(10058)**	Cannot send after socket shutdown.	A request to send or receive data was disallowed because the socket had already been shut down in that direction with a previous shutdown call. By calling shutdown a partial close of a socket is requested, which is a signal that sending or receiving, or both have been discontinued.
WSAESOCKTNOSUPPORT **(10044)**	Socket type not supported.	The support for the specified socket type does not exist in this address family. For example, the optional type SOCK_RAW might be selected in a socket call, and the implementation does not support SOCK_RAW sockets at all.
WSAETIMEDOUT **(10060)**	Connection timed out.	A connection attempt failed because the connected party did not properly respond after a period of time, or the established connection failed because the connected host has failed to respond.
WSATYPE_NOT_FOUND **(10109)**	Class type not found.	The specified class was not found.
WSAEWOULDBLOCK **(10035)**	Resource temporarily unavailable.	This error is returned from operations on nonblocking sockets that cannot be completed immediately, for example recv when no data is queued to be read from the socket. It is a nonfatal error, and the operation should be retried later. It is normal for WSAEWOULDBLOCK to be reported as the result from calling connect on a nonblocking SOCK_STREAM socket, since some time must elapse for the connection to be established.

Error (Code)	Meaning	Description
WSAHOST_NOT_FOUND **(11001)**	Host not found.	No such host is known. The name is not an official host name or alias, or it cannot be found in the data-base(s) being queried. This error may also be returned for protocol and service queries, and means that the specified name could not be found in the relevant database.
WSA_INVALID_HANDLE **(OS dependent)**	Specified event object handle is invalid.	An application attempts to use an event object, but the specified handle is not valid.
WSA_INVALID_PARAMETER **(OS dependent)**	One or more parameters are invalid.	An application used a Windows Sockets function which directly maps to a Win32 function. The Win32 function is indicating a problem with one or more parameters.
WSAINVALIDPROCTABLE **(OS dependent)**	Invalid procedure table from service provider.	A service provider returned a bogus procedure table to Ws2_32.dll. (Usually caused by one or more of the function pointers being null.)
WSAINVALIDPROVIDER **(OS dependent)**	Invalid service provider version number.	A service provider returned a version number other than 2.0.
WSA_IO_INCOMPLETE **(OS dependent)**	Overlapped I/O event object not in signaled state.	The application has tried to determine the status of an overlapped operation which is not yet completed. Applications that use WSAGetOverlappedResult (with the fWait flag set to FALSE) in a polling mode to determine when an overlapped operation has completed, get this error code until the operation is com-plete.
WSA_IO_PENDING **(OS dependent)**	Overlapped operations will complete later.	The application has initiated an overlapped operation that cannot be completed immediately. A comple-tion indication will be given later when the operation has been completed.
WSA_NOT_ENOUGH_MEMORY **(OS dependent)**	Insufficient memory available.	An application used a Windows Sockets function that directly maps to a Win32 function. The Win32 func-tion is indicating a lack of required memory resources.
WSANOTINITIALISED	Successful	Either the application has not called WSAStartup or

Error (Code)	Meaning	Description
(10093)	WSAStartup not yet performed.	WSAStartup failed. The application may be access-ing a socket that the current active task does not own (that is, trying to share a socket between tasks), or WSACleanup has been called too many times.
WSANO_DATA **(11004)**	Valid name, no data record of requested type.	The requested name is valid and was found in the database, but it does not have the correct associated data being resolved for. The usual example for this is a host name-to-address translation attempt (using gethostbyname or WSAAsyncGetHostByName) which uses the DNS (Domain Name Server). An MX record is returned but no A record-indicating the host itself exists, but is not directly reachable.
WSANO_RECOVERY **(11003)**	This is a nonrecoverabl e error.	This indicates some sort of nonrecoverable error occurred during a database lookup. This may be be-cause the database files (for example, BSD-compatible HOSTS, SERVICES, or PROTOCOLS files) could not be found, or a DNS request was returned by the server with a severe error.
WSAPROVIDERFAILE DINIT **(OS dependent)**	Unable to initialize a service provider.	Either a service provider's DLL could not be loaded (LoadLibrary failed) or the provider's WSPStartup/NSPStartup function failed.
WSASYSCALLFAILUR E **(OS dependent)**	System call failure.	Returned when a system call that should never fail does. For example, if a call to WaitForMultipleObjects fails or one of the registry functions fails trying to manipulate the protocol/name space catalogs.
WSASYSNOTREADY **(10091)**	Network subsystem is unavailable.	This error is returned by WSAStartup if the Windows Sockets implementation cannot function at this time because the underlying system it uses to provide network services is currently unavailable. Users should check: That the appropriate Windows Sockets DLL file is in the current path. That they are not trying to use more than one Windows Sockets implementation simultaneously. If there is more than one Winsock DLL on your system, be sure the first one in the path is appropriate for the network

Error (Code)	Meaning	Description
		subsystem currently loaded. The Windows Sockets implementation documentation to be sure all necessary components are currently installed and configured correctly.
WSATRY_AGAIN **(11002)**	Nonauthoritative host not found.	This is usually a temporary error during host name resolution and means that the local server did not receive a response from an authoritative server. A retry at some time later may be successful.
WSAVERNOTSUPPORTED **(10092)**	Winsock.dll version out of range.	The current Windows Sockets implementation does not support the Windows Sockets specification ver-sion requested by the application. Check that no old Windows Sockets DLL files are being accessed.
WSAEDISCON **(10101)**	Graceful shutdown in progress.	Returned by WSARecv and WSARecvFrom to indicate that the remote party has initiated a graceful shut-down sequence.
WSA_OPERATION_ABORTED **(OS dependent)**	Overlapped operation aborted.	An overlapped operation was canceled due to the closure of the socket, or the execution of the SIO_FLUSH command in WSAIoctl.

10.2 Analysetool

Alle zenon Module wie z. B. Editor, Runtime, Treiber, usw. schreiben Meldungen in eine gemeinsame LOG-Datei. Um sie korrekt und übersichtlich anzuzeigen, benutzen Sie das Programm Diagnosis Viewer, das mit zenon mitinstalliert wird. Sie finden es unter **Start/Alle Programme/zenon/Tools 8.20 -> Diagviewer.**

zenon Treiber protokollieren alle Fehler in LOG-Dateien. LOG-Dateien sind Textdateien mit einer speziellen Struktur. Der Standardordner für die LOG-Dateien ist der Ordner **LOG** unterhalb des Ordners **ProgramData**, zum Beispiel:

%ProgramData%\COPA-DATA\LOG.

Achtung: Mit den Standardeinstellungen zeichnet ein Treiber nur Fehlerinformationen auf. Mit dem Diagnosis Viewer kann bei den meisten Treibern die Diagnose-Ebene auf „Debug" und „Deep Debug" erweitert werden. Damit protokolliert der Treiber auch alle anderen wesentlichen Aufgaben und Ereignisse.

Im Diagnosis Viewer kann man auch:

- neu erstellte Einträge in Echtzeit mitverfolgen
- die Aufzeichnungseinstellungen anpassen
- den Ordner, in dem die LOG-Dateien gespeichert werden, ändern

Hinweise:

1. Der Diagnosis Viewer zeigt alle Einträge in UTC (Koordinierter Weltzeit) an und nicht in der lokalen Zeit.
2. Der Diagnosis Viewer zeigt in seiner Standardeinstellung nicht alle Spalten einer LOG-Datei an. Um mehr Spalten anzuzeigen, aktivieren Sie die Eigenschaft **Add all columns with entry** im Kontextmenü der Spaltentitel.
3. Bei Verwendung von reinem **Error-Logging** befindet sich eine Problembeschreibung in der Spalte **Error text**. In anderen Diagnose-Ebenen befindet sich diese Beschreibung in der Spalte **General text**.
4. Viele Treiber zeichnen bei Kommunikationsprobleme auch Fehlernummern auf, die die SPS ihnen zuweist. Diese werden in **Error text** und/oder **Error code** und/oder **Driver error parameter(1** und **2)** angezeigt. Hinweise zur Bedeutung der Fehlercodes erhalten Sie in der Treiberdokumentation und der Protokoll/SPS-Beschreibung.
5. Stellen Sie am Ende Ihrer Tests den Diagnose-Level von **Debug** oder **Deep Debug** wieder zurück. Bei **Debug** und **Deep Debug** fallen beim Protokollieren sehr viele Daten an, die auf der Festplatte gespeichert werden und die Leistung Ihres Systems beeinflussen können. Diese werden auch nach dem Schließen des Diagnosis Viewers weiter aufgezeichnet.

> **Achtung**
>
> Unter Windows CE werden aus Ressourcegründen Fehler standardmäßig nicht protokolliert.

Weitere Informationen zum Diagnosis Viewer finden Sie im Handbuch Diagnosis Viewer.

10.3 Treiberüberwachung

Die Runtime überwacht die Verfügbarkeit des Treibers via Watchdog. Ist ein Treiber nicht mehr verfügbar, wird für alle angemeldeten Variablen des Treibers zusätzlich das Statusbit *INVALID* gesetzt.

Mögliche Ursachen für Auslösen des Watchdogs:

- Der Treiberprozess läuft nicht mehr.

  Überprüfen Sie im Task Manager ob die Treiber-Exe noch läuft.
- Betriebssystem ist mit höher priorisierten Prozessen ausgelastet.

Überprüfen Sie die Konfiguration Ihres Systems auf zu wenig Arbeitsspeicher und zu geringe CPU-Leistung. In diesem Fall erfolgt das Rücksetzen des *INVALID* Statusbits durch den Treiber nur bei Wertänderung auf der Gegenstelle. Statische Werte behalten das *INVALID* Statusbit bis zum nächsten Start der Runtime oder des Treibers.

WATCHDOG KONFIGURATION

Für die Überwachung der Kommunikation zur Runtime wird die Verbindung zum Treiber in einem fix vorgegebenen Zeitraum von 60 Sekunden überprüft. Dieser Vorgang wird mehrmals wiederholt. Konnte nach 5 Versuchen (= innerhalb von 5 Minuten) keine valide Verbindung zum Treiber erkannt werden, wird das *INVALID*-Bit bei der angemeldeten (*advised*) Variablen gesetzt. Zusätzlich wird das *INVALID*-Bit auch gesetzt, wenn neue Variablen angemeldeten werden. Das *INVALID*-Bit wird nicht mehr zurückgesetzt.

Dafür werden entsprechende LOG-Einträge erstellt.

LOG-EINTRAG

Bei Auslösen des Watchdogs wird eine Fehlermeldung im LOG protokolliert:

Parameter	Beschreibung
Communication with driver:<drvExe>/<drvDesc>(id:<drvId>) timed out. No communication for <time> ms.	Keine Kommunikation mit Treiber innerhalb der angegeben Zeit. ▸ *<time>*: Zeitangabe in Millisekunden ▸ *<drvDesc>*: Treibername ▸ *<drvExe>*: Treiber EXE-Name ▸ *<drvId>*: Treiber-ID im zenon Projekt
Communication with %s timed out. Invalid-Bit will be set.	Die Kommunikation zum Treiber *%s* konnte bei 5 Versuchen nicht innerhalb von 60 Sekunden aufgebaut werden. Bei der Variable wird das *INVALID*-Bit gesetzt.
Communication with %s timed out. Timeout happened %d times	Die Kommunikation zum Treiber *%s* konnte *%d* Mal nicht innerhalb von 60 Sekunden aufgebaut werden.

10.4 Checkliste

Fragen und Hinweise zur Fehlereingrenzung:

ALLGEMEINE FEHLERSUCHE

- Analyse mit Hilfe des Diagnosis Viewers **(auf Seite 64) (die wichtigen Fehler werden protokolliert).**
- Kann der KABA Server über den Befehl *Ping* erreicht werden?

  Ping: **Kommandozeile öffnen -> ping <IP-Adresse> (z. B.: ping 192.168.0.100) -> Eingabe-Taste drücken.**

  Kommt eine Antwort mit Zeitangabe oder ein Timeout?
- Kann der KABA Server auf dem entsprechenden Port über *Telnet* erreicht werden?

  Telnet: **Kommandozeile öffnen, telnet <IP-Adresse Port-Nummer> eingeben (z. B. für Modbus: telnet 192.168.0.100 502) -> Eingabe-Taste drücken .**

  Wird der Bildschirm schwarz, konnte eine Verbindung aufgebaut werden.
- Stimmen die Zutrittsdaten (z.B. Benutzer und Passwort) überein?

  Ist die Symbolische Adresse der Variable korrekt angegeben?

MANCHE VARIABLEN MELDEN INVALID

- Ist der KABA Server erreichbar?

WERTE WERDEN NICHT ANGEZEIGT, ZAHLENWERTE BLEIBEN LEER

Treiber läuft nicht. Überprüfen Sie die:

- Installation von zenon
- Installation des Treibers
- Falsche Symbolische Adresse der "leeren" Variable

VARIABLEN WERDEN MIT BLAUEM PUNKT ANGEZEIGT

Die Kommunikation im Netzwerk ist gestört:

- Bei einem Netzwerkprojekt:
  Läuft das Netzwerkprojekt auch auf dem Server?
- Bei einem Standalone-Projekt oder Netzwerkprojekt, das auf dem Server läuft:
  Deaktivieren Sie die Eigenschaft **Nur von Standby Server anfordern** im Knoten **Treiber Anbindung**/**Adressierung**.

WERTE WERDEN FALSCH ANGEZEIGT

Überprüfen Sie die Angaben zur Berechnung im Knoten **Wertberechnung** der Variablen-Eigenschaften?

TREIBER FÄLLT SPORADISCH AUS

Analyse mit Hilfe des Diagnosis Viewers (auf Seite 64):
-> Welche Meldungen werden angezeigt?