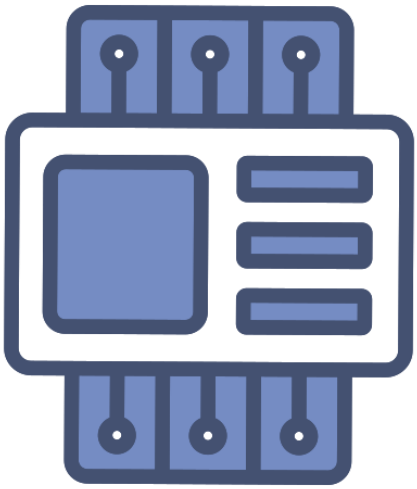
SYSTEMHANDBUCH FÜR CODESYS V3.5 UND PLM800



SABO Elektronik GmbH
Lohbachstr. 14
58239 Schwerte
Tel. 02304 / 97102 - 0
Fax 02304 / 97102 - 22
E-Mail info@sabo.de
Internet www.sabo.de

1 Inhalt

2 WICHTIGE HINWEISE

2.1 BESTIMMUNGSMÄßIGE VERWENDUNG

Die Anwendungsgebiete der SABO PLM Baureihe erstrecken sich von der Regelungs- und Steuerungstechnik über die Gebäudeautomation, bis zur industriellen Nutzung in der Automatisierung. In allen Anwendungsbereichen ist darauf zu achten, dass die maximalen Spannungen die auf den technischen Datenblatt genannten Höchstgrenzen nicht überschritten werden. Änderungen sind nur im Rahmen des Handbuchs genannten Möglichkeiten zulässig.

Insbesondere ist die Verwendung von SABO-Produkten nicht zulässig für: Überwachung von Kernreaktionen in Kernkraftwerken, Flugleitsysteme, Flugsicherung, Steuerung von Massentransportmitteln, medizinische Lebenserhaltungssysteme, Steuerung von Waffensystemen.

2.2 URHEBERSCHUTZ

Dieses Handbuch sowie alle dazugehörigen Bilder sind Eigentum der SABO Elektronik GmbH und sind urheberrechtlich geschützt. Eine Vervielfältigung, Veränderung oder Veräußerung an Dritte ist nicht gestattet, es sei denn es liegt eine schriftliche Einverständniserklärung der SABO Elektronik GmbH vor. Zuwiderhandlungen ziehen rechtliche Gegenmaßnahmen nach sich.

Die in dieser Dokumentation beschriebenen Produkte und Funktionen können jederzeit den neusten technologischen Entwicklungen angepasst werden. Die gegebenen Informationen können somit nicht als Vertragsgegenstand angesehen werden.

2.3 PERSONALQUALIFIKATION

SABO Produkte dürfen ausschließlich von Fachkräften mit einer Ausbildung in der SPS-Programmierung oder Elektrofachkräften mit einer Unterweisung in den dafür geltenden spezifischen Normen angeschlossen und gewartet werden.

Für Fehler und Schäden, die an SABO Produkten und oder Fremdprodukten entstehen, die auf die Missachtung der Handhabung zurückzuführen sind, übernimmt die SABO Elektronik GmbH keine Haftung.

2.4 SICHERHEITSHINWEISE

Die in dieser Dokumentation gemachten Sicherheitshinweise erheben keinen Anspruch auf Vollständigkeit. Bei Unklarheiten und der Möglichkeit einer potenziellen Gefährdung von Mensch und Maschine ist im Zweifelsfall der zuständige Distributor der SABO Elektronik GmbH hinzuzuziehen. Die in dieser Dokumentation gemachten Hinweise sind Vorschläge und müssen bei der Übertragung auf die jeweilige Anwendung auf deren Machbarkeit und Funktionsfähigkeit überprüft werden.

Im Allgemeinen dienen die Vorschriften nach VDE beim Umgang mit elektrischen Anlagen als Richtlinie.

3 INSTALLATION DER CODESYS V3.5 IDE, DER DEVICEDESCRIPTIONS UND BIBLIOTHEKEN FÜR SABO-STEUERUNGEN

3.1 INSTALLATION DER CODESYS IDE

Zur Installation der CODESYS IDE wird die entsprechende Installationsdatei benötigt. Sie können die Datei unter https://sabo.de/downloads herunterladen. Unter dem Menüpunkt CODESYS V3 finden Sie zwei Installationsdateien, CODESYS V3.5.16.40 (64Bit) und CODESYS V3.5.19.50 (64Bit). Siehe *Abbildung 1*.

Nutzen Sie bitte die Codesys Version, die dem Runtime auf der Steuerung entspricht, mit der Sie programmieren möchten. Die Version des Runtimes kann über einen Webbrowser und dem Webconfig Tool unter dem Punkt CODESYS V3 von der Steuerung angezeigt werden.

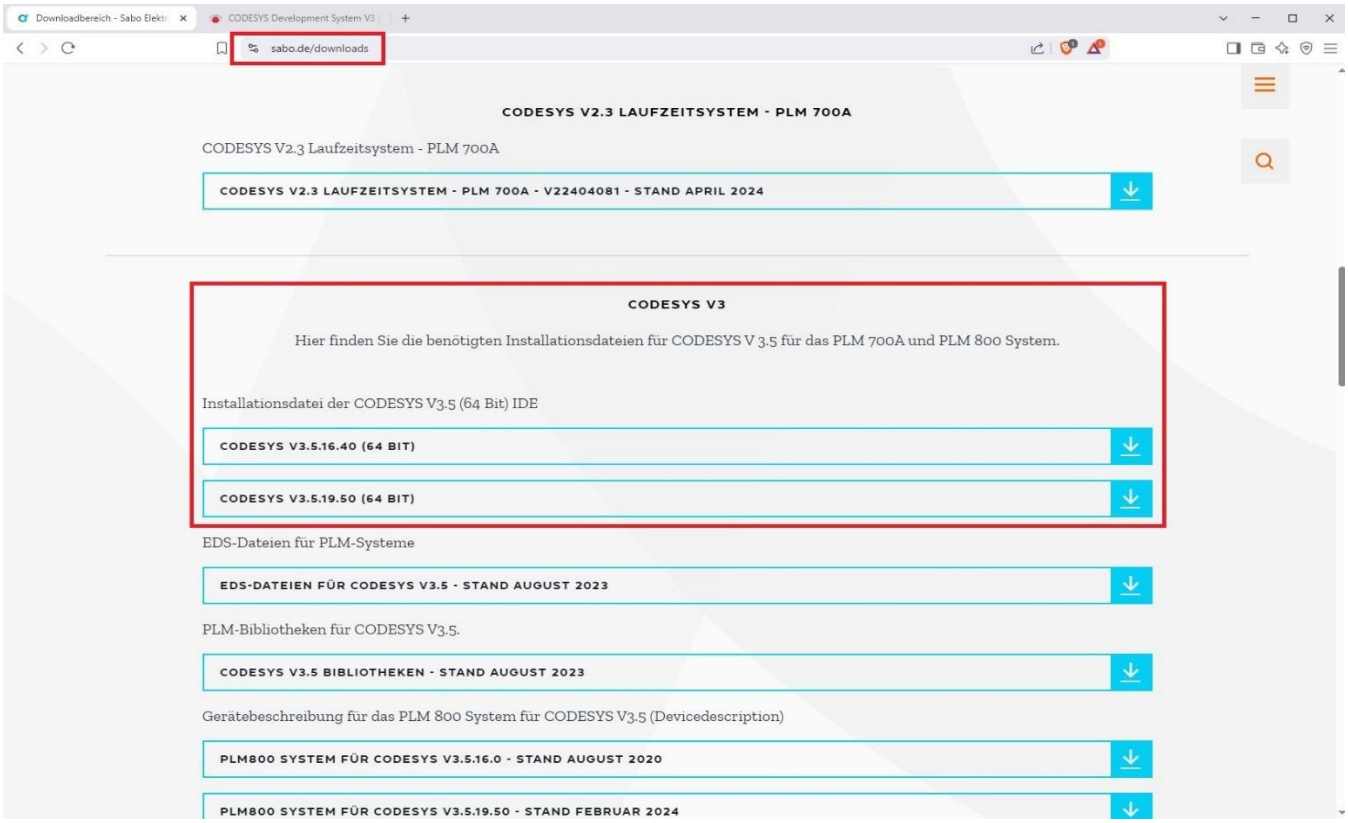


Abbildung 1: Download der CODESYS IDE von der SABO-Webseite

Alternativ können Sie die Datei in dem CODESYS-Store auf der Seite von CODESYS herunterladen https://store.codesys.com/de/ , *Abbildung 2*.

Bitte beachten Sie, dass wir momentan nur die CODESYS Versionen 3.5.16.40 und 3.5.19.50 unterstützen, **Stand 22.05.2026**. Laden Sie daher keine andere Version herunter, da es sonst zu Inkompatibilitäten mit der Gerätebeschreibung und den Bibliotheken kommen kann.

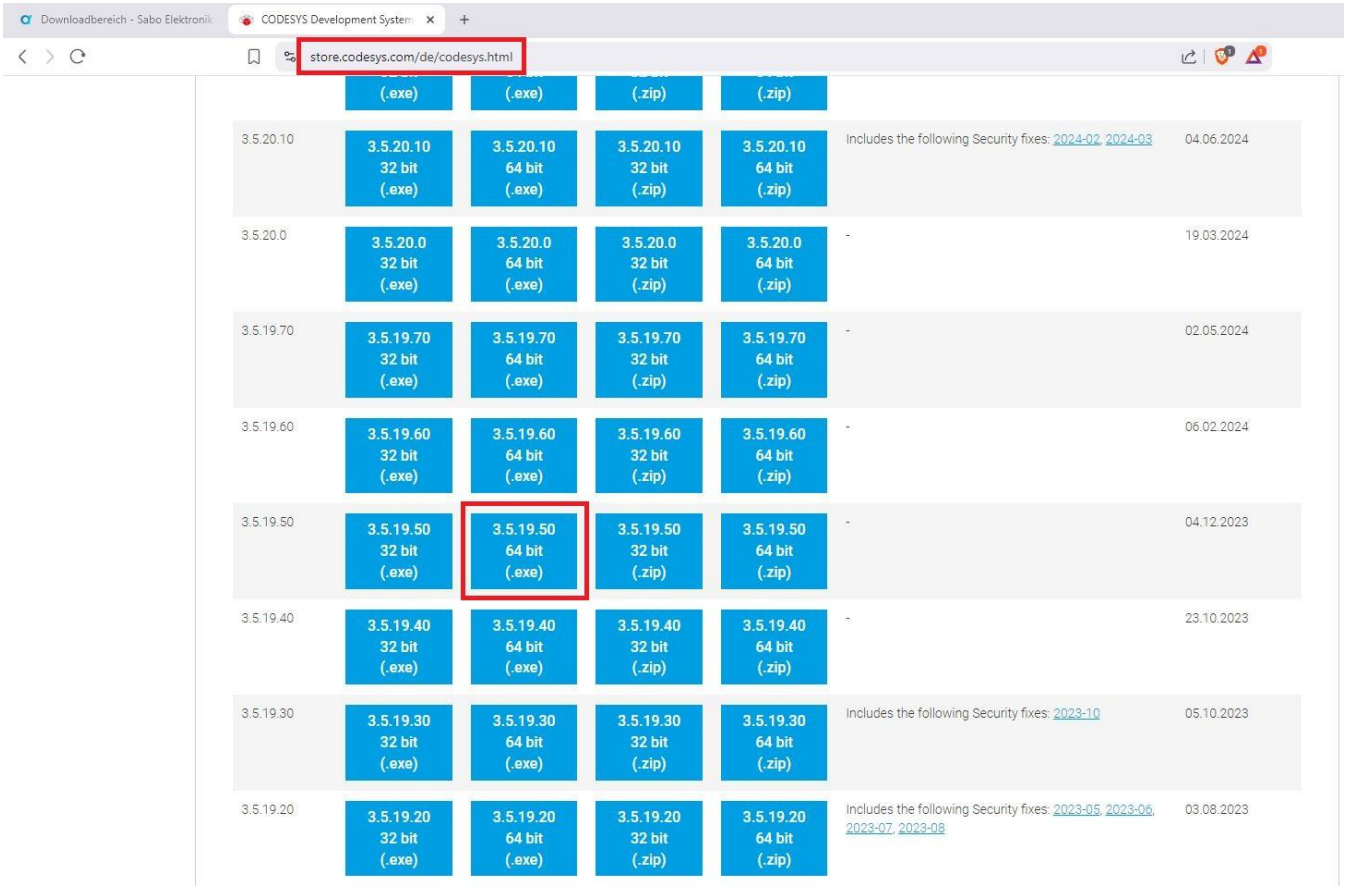


Abbildung 2: Download der CODESYS IDE aus dem CODESYS Store

Nachdem Sie die Datei heruntergeladen haben, können Sie die Installation über einen Doppelklick auf die exe-Datei ausführen. Da die Installationsdatei groß ist, kann die Installation, je nach Leistung des PC, einige Zeit in Anspruch nehmen.

Wenn möglich installieren Sie die CODESYS V3.5 Programmierumgebung in das vorgeschlagene Verzeichnis, das CODESYS Ihn anzeigt. Sollte bereits eine CODESYS V3.5 Installation auf Ihrem Rechner installiert sein, dann wird während der Installation gefragt, ob Sie bestehen Einstellungen der vorhandenen CODESYS V3.5 Programmierumgebung übernehmen oder ob Sie diesen Punkt überspringen möchten. Das bleibt Ihnen überlassen, sollten Sie bereits Einstellungen vorgenommen haben, die Ihnen die Arbeit erleichtern, dann macht es Sinn, diese in die neue Installation zu übernehmen.

Nach erfolgreicher Installation können Sie die installierte Programmieroberfläche mit einem Doppelklick auf das Icon starten.

Abbildung 3: CODESYS V3.5.19.50 Icon

3.2 INSTALLATION DER DEVICEDESCRIPTIONS

Um in CODESYS die verschiedenen SABO-Steuerungen in der Steuerungskonfiguration nutzen zu können, müssen sogenannte Devicedescriptions (Gerätebeschreibungsdatei) der jeweiligen Geräte installiert werden.

Diese Devicedescriptions können Sie auf der SABO-Webseite unter Downloads https://sabo.de/downloads herunterladen. Unter dem Downloadpunkt CODESYS V3 finden Sie die benötigten Dateien, *Abbildung 4*.

Dabei müssen Sie beachten, dass Sie die passende Version zu Ihrer CODESYS Version herunterladen. Das bedeutet, dass wenn Sie eine CODESYS Version 3.5.16.40 installiert haben, Sie auch die Devicedescription für die Version 3.5.16.x herunterladen müssen. Die Version 3.5.19.x entsprechend, wenn Ihre CODESYS Version 3.5.19.50 ist.

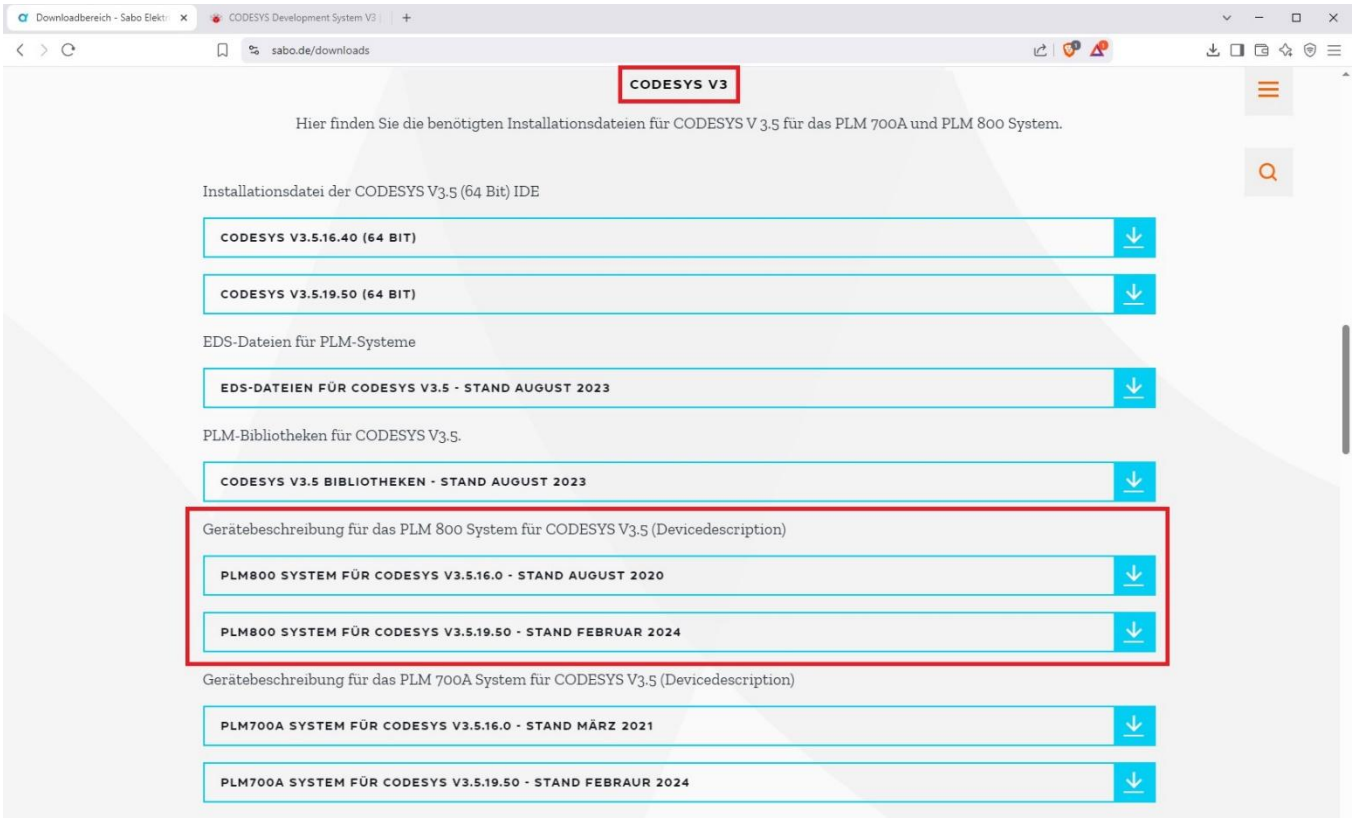


Abbildung 4: Devicedescription im Downloadbereich auf der SABO-Webseite

Nach dem Sie die Datei heruntergeladen haben, müssen Sie die Datei entpacken. Merken Sie sich das Verzeichnis, in das Sie die Dateien entpackt haben, da Sie zur Installation dieser Dateien, auf das Verzeichnis zugreifen müssen.

Starten Sie die CODESYS V3.5 IDE und warten Sie, bis das Programm geladen wurde. Sobald die Startseite der IDE angezeigt wird, wählen Sie aus der oberen Menüleiste den Punkt **Tools** an und anschließend **Geräte-Repository...** , *Abbildung 5*.

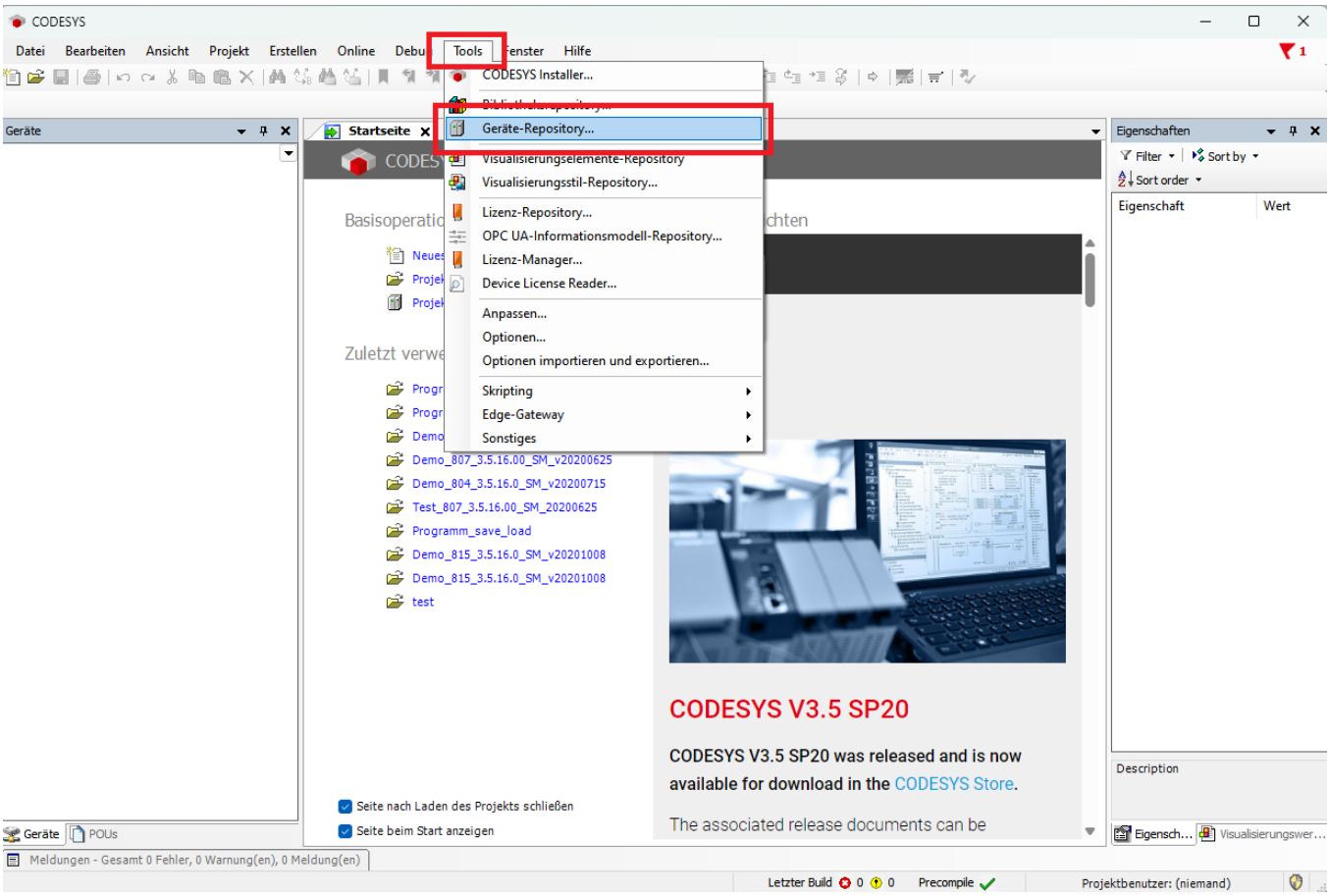


Abbildung 5: Das Geräte-Repository

In dem neu erscheinenden Fenster klicken Sie auf die Schaltfläche **Installieren...** .

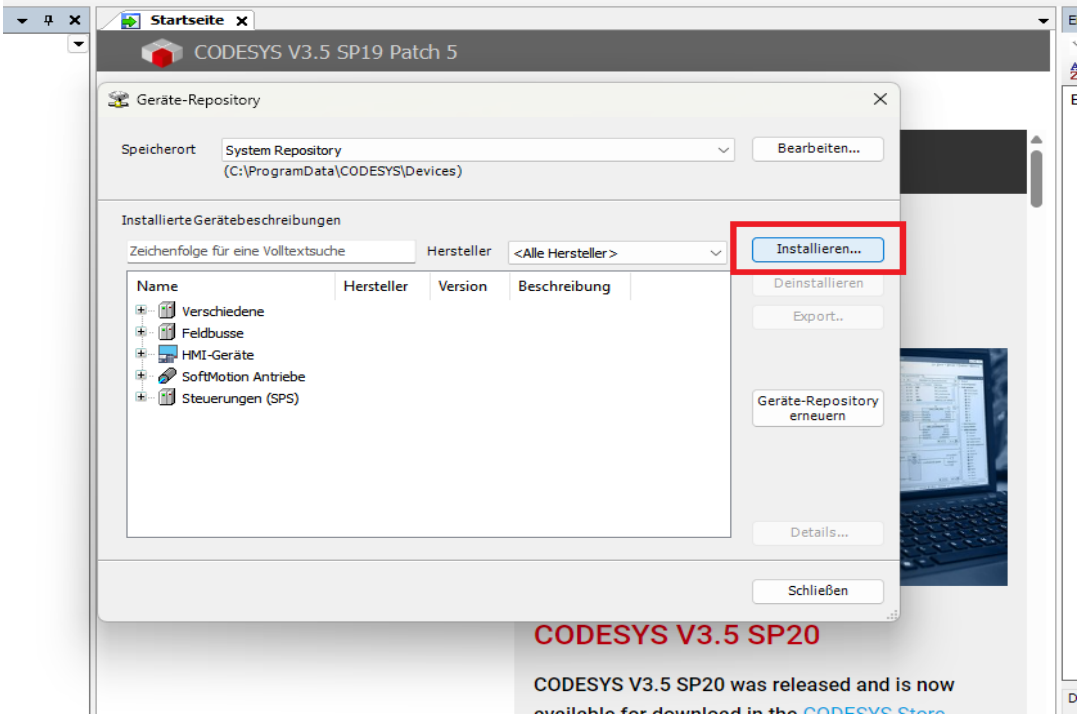


Abbildung 6: Devicedescription installieren

Navigieren Sie jetzt zu dem Verzeichnis, in das Sie die Devicedescription Dateien entpackt haben. Wählen Sie, wie in der *Abbildung 7* gezeigt, alle Dateien mit der Endung **.devdesc.xml** aus und klicken Sie anschließend auf die Schaltfläche **Öffnen**.

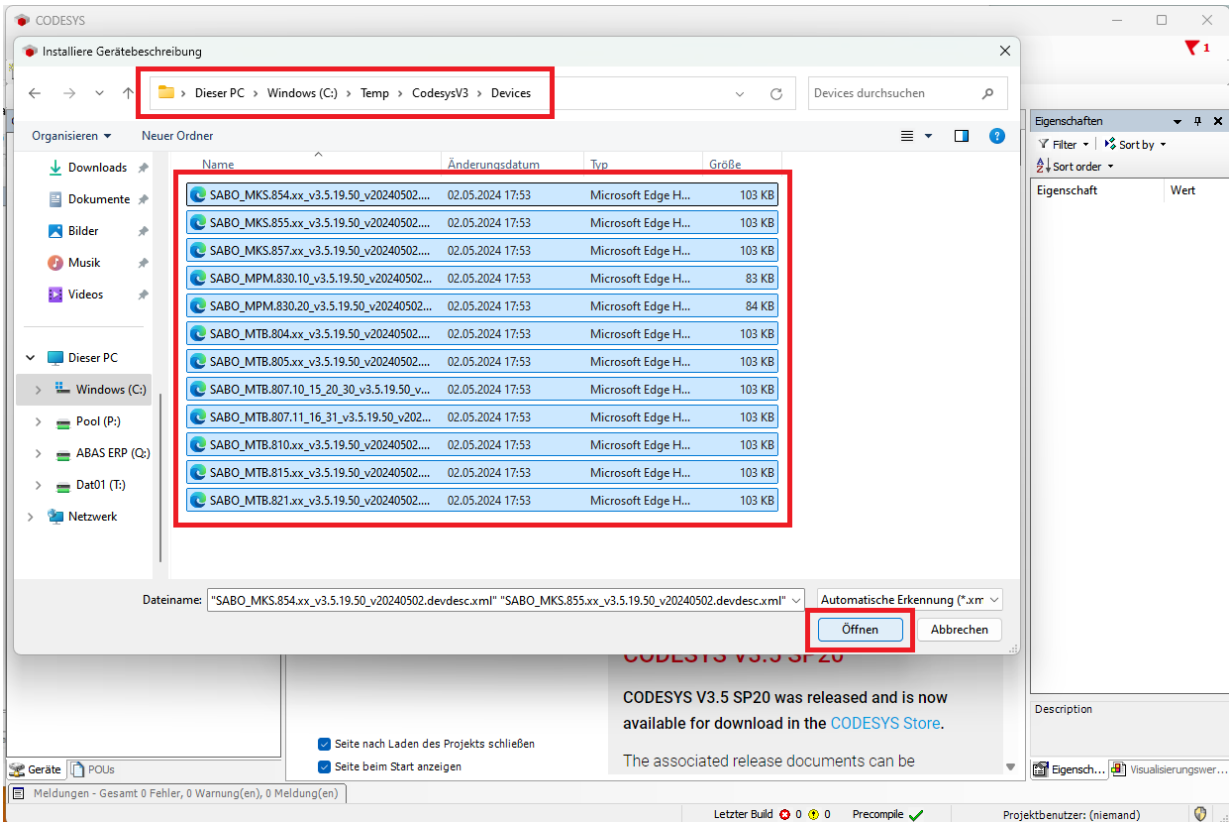


Abbildung 7: Installation der ausgewählten Devicedescription Dateien

Nach dem Klick auf **Öffnen** werden die von Ihnen ausgewählten Devicedescriptions in das Geräte-Repository installiert. Wenn der Vorgang abgeschlossen wurde, wird Ihnen in einem neuen Fenster angezeigt, welche Dateien dem Geräte-Repository hinzugefügt wurden.

In unserem Beispiel wurden die Devicedescriptions von den SABO Steuerungen für die Codesys Version V3.5.19.50 installiert. Siehe *Abbildung 8*.

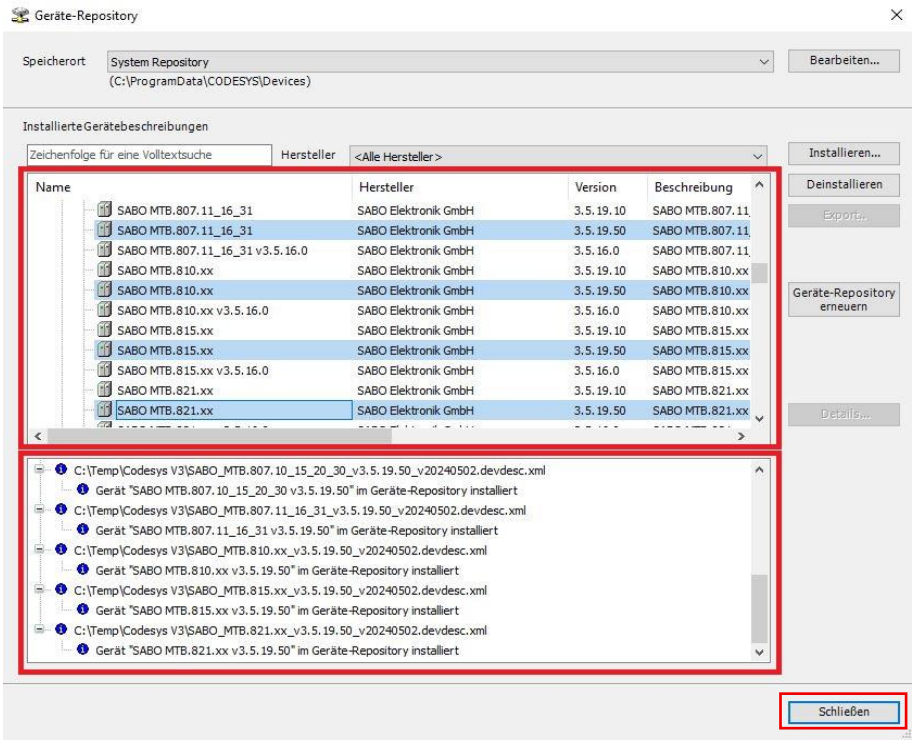


Abbildung 8: Anzeige der kürzlich installierten Devicedescriptions mit der Versionsnummer V3.5.19.50

Über die Schaltfläche **Schließen** können Sie das Fenster verlassen. Somit ist der Vorgang zur Installation der Devicedescription abgeschlossen.

3.3 INSTALLATION VON EDS-DATEIEN

Die Installation der EDS-Dateien (Electronical Data Sheet) erfolgt genauso wie die Installation der Devicedescription über das Geräte-Repository. Die EDS-Dateien zu unseren CAN-Modulen können Sie über den Downloadbereich https://sabo.de/downloads auf unserer Webseite herunterladen. Siehe *Abbildung 9*.

Abbildung 9: EDS-Dateien Download von der SABO-Webseite

Nachdem Sie das zip-File von der SABO-Webseite heruntergeladen haben, muss es noch entpackt werden. Merken Sie sich das Verzeichnis, in das Sie die EDS-Dateien entpackt haben, denn es wird später noch wichtig bei der Installation der Dateien.

Starten Sie über einen Doppelklick die CODESYS IDE, falls Sie diese noch nicht gestartet haben, und wählen aus dem oberen Menüband den Menüpunkt **Tools** und klicken dann auf **Geräte-Repository...** . Siehe *Abbildung 10*.

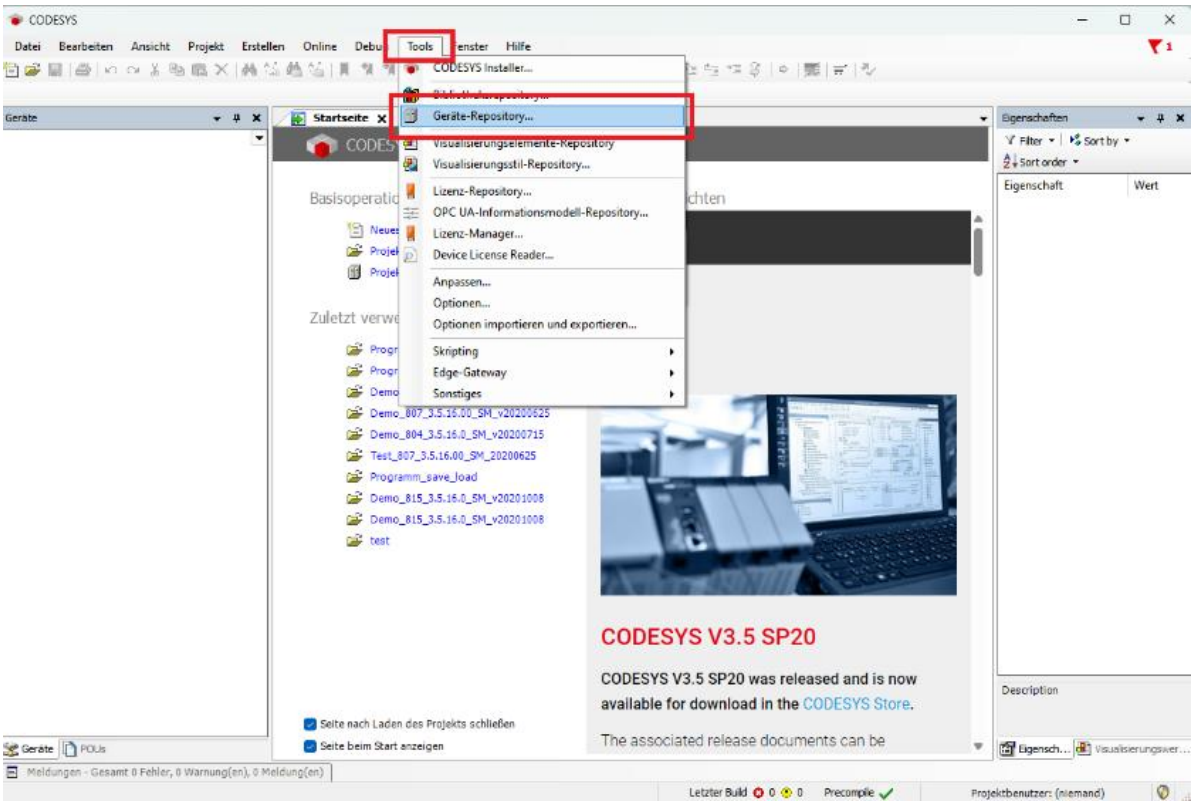


Abbildung 10: Geräte-Repository

In dem neu erscheinenden Fenster klicken Sie auf die Schaltfläche **Installieren...** .

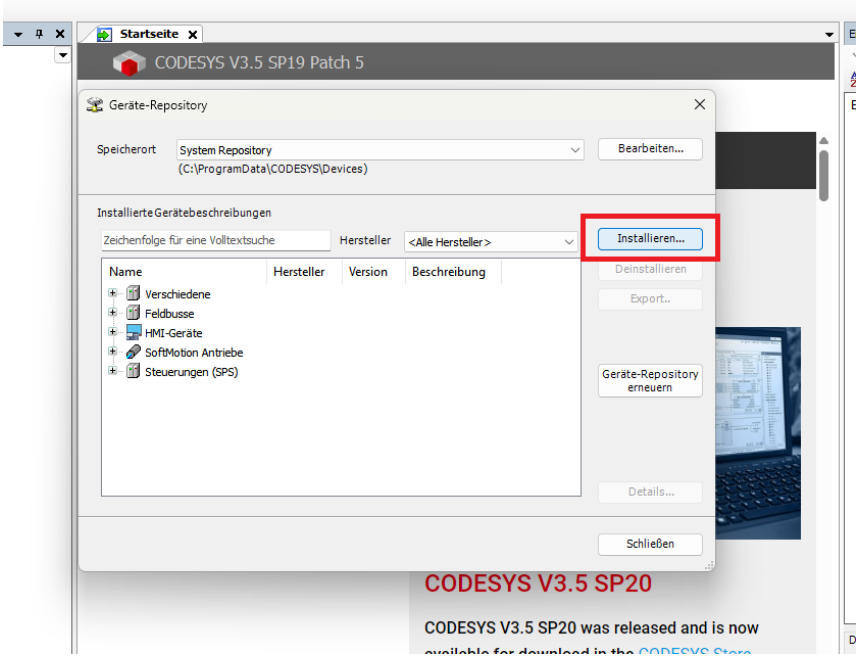


Abbildung 11: EDS-Dateien installieren

Öffnen Sie das Verzeichnis, in dem die EDS-Dateien entpackt wurden. Wählen Sie anschließend entweder alle Dateien aus, um sämtliche EDS-Dateien zu installieren, oder beschränken Sie die Auswahl auf die Dateien, die für Ihre Anwendung bzw. Steuerung benötigt werden. Die Dateiendung der EDS-Dateien ist **.eds**. *Abbildung 12*.

Mit der Tastenkombination **strg + a** können Sie alle Dateien aus dem Verzeichnis auf einmal markieren.

Sobald Sie die gewünschten Dateien ausgewählt haben und anschließend auf die Schaltfläche **Öffnen** klicken, werden diese in das Geräte-Repository installiert.

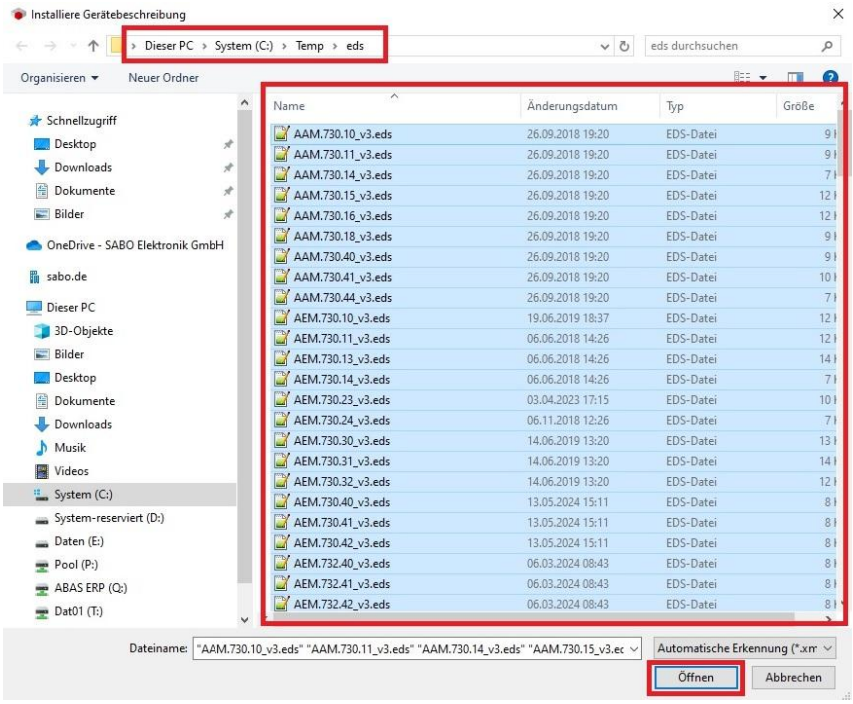


Abbildung 12: Installation der EDS-Dateien

Nach der Installation der EDS-Dateien wird Ihnen ein Fenster angezeigt, in dem Sie sehen können, welche Dateien Ihrem Geräte-Repository hinzugefügt wurden. *Abbildung 13*.

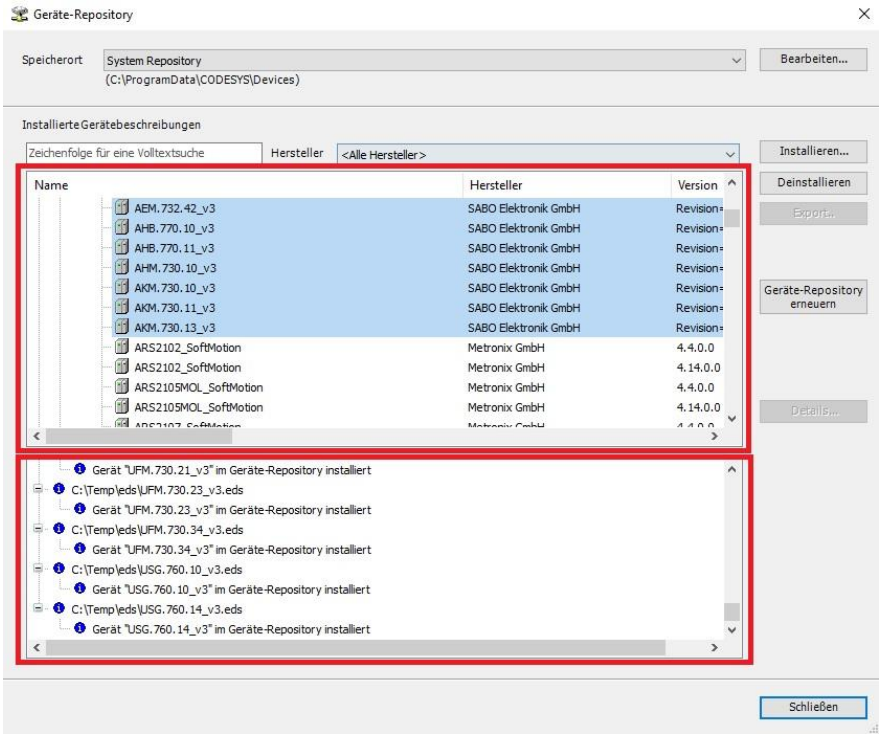


Abbildung 13: Übersicht über die hinzugefügten EDS-Dateien in das Geräte-Repository

Mit der Schaltfläche **Schließen** wird das Fenster geschlossen und die Installation der EDS-Dateien ist damit abgeschlossen. Sie können jetzt die hinzugefügten CAN-Module in Ihrem Projekt nutzen.

3.4 INSTALLATION VON BIBLIOTHEKEN

Die Installation der Bibliotheken erfolgt in der CODESYS IDE über das Bibliotheks-Repository. Bibliotheken, die installiert werden können, haben die Dateiendung **library** oder **compiledlibrary**. Alle benötigen SABO-Bibliotheken können Sie von der Webseite http://sabo.de/downloads unter dem Menüpunkt CODESYS V3 herunterladen. Siehe *Abbildung 14*.

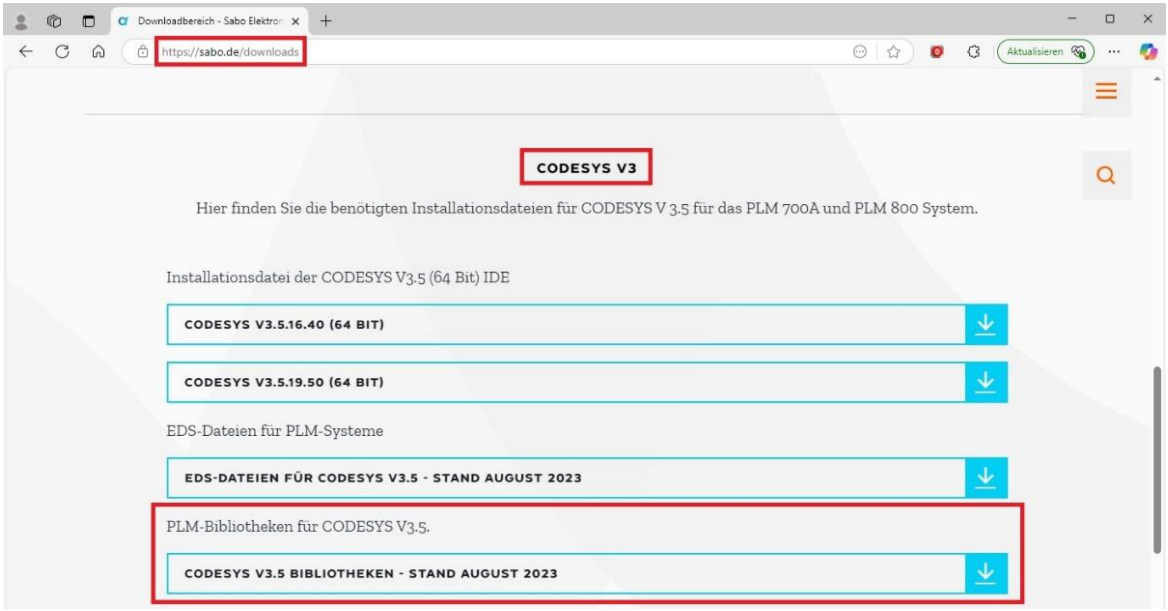


Abbildung 14: SABO-Bibliotheken auf der SABO-Webseite

Die Bibliotheken sind als zip-Datei gepackt und müssen von Ihnen entpackt werden. Bitte merken Sie sich das Verzeichnis, in das Sie die Bibliotheken entpackt haben, da Sie später darauf zugreifen müssen.
Starten Sie die CODESYS IDE und warten Sie bis das CODESYS geladen wurde und Sie die Startseite sehen. Gehen Sie anschließend in dem oberen Menüband auf den Menüpunkt **Tools** und klicken dann auf **Bibliotheksrepository...** . *Abbildung 15*.

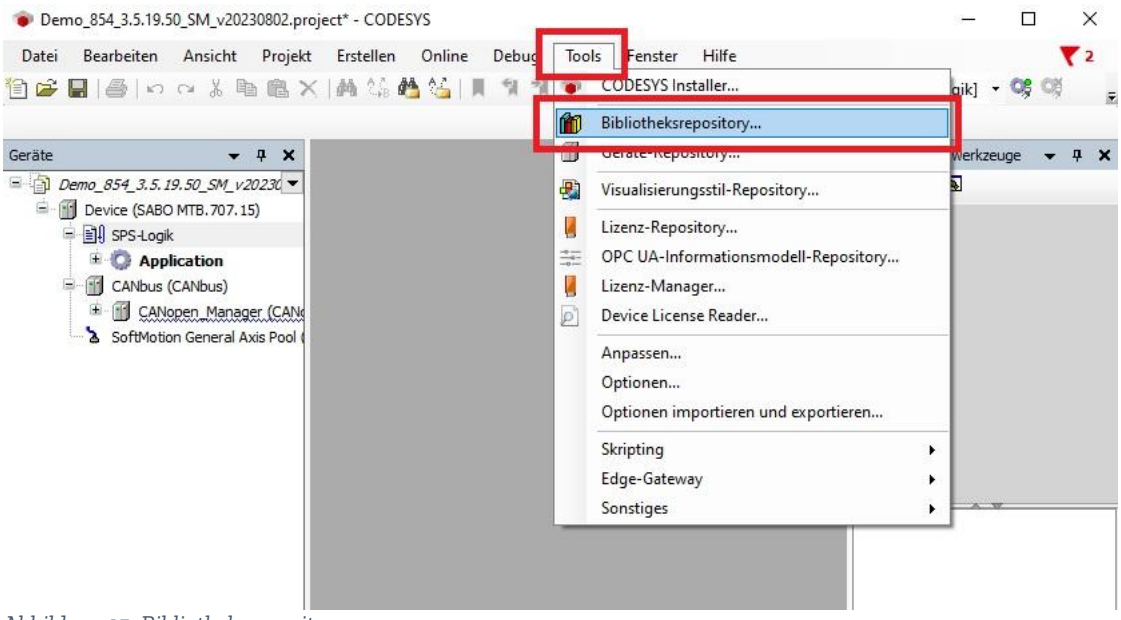


Abbildung 15: Bibliotheksrepository

In dem neu erscheinenden Fenster klicken Sie auf **Installieren...** .

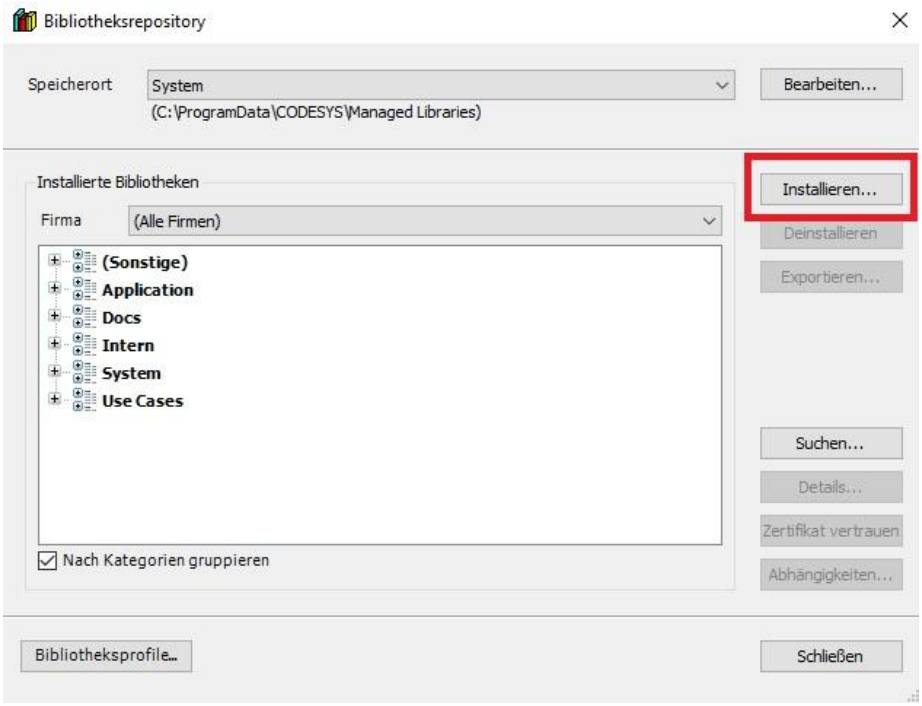


Abbildung 16: Bibliotheksdateien installieren

Navigieren Sie jetzt in das Verzeichnis, in das Sie die Bibliotheken entpackt haben. In unserem Beispiel ist *c:\temp\codesys V3\plm-compiled-libraries*. Markieren Sie in dem Verzeichnis alle Bibliotheksdateien, die Sie installieren möchten.
In unserem Fall sind das alle SABO-Bibliotheken, die vorne immer die Bezeichnung **Plm** haben. Die Dateiendung von zulässigen Bibliotheksdateien ist **.library** oder .**compiledlibrary**. Nachdem Sie alle Dateien ausgewählt haben, klicken Sie auf **Öffnen**. *Abbildung 17*.

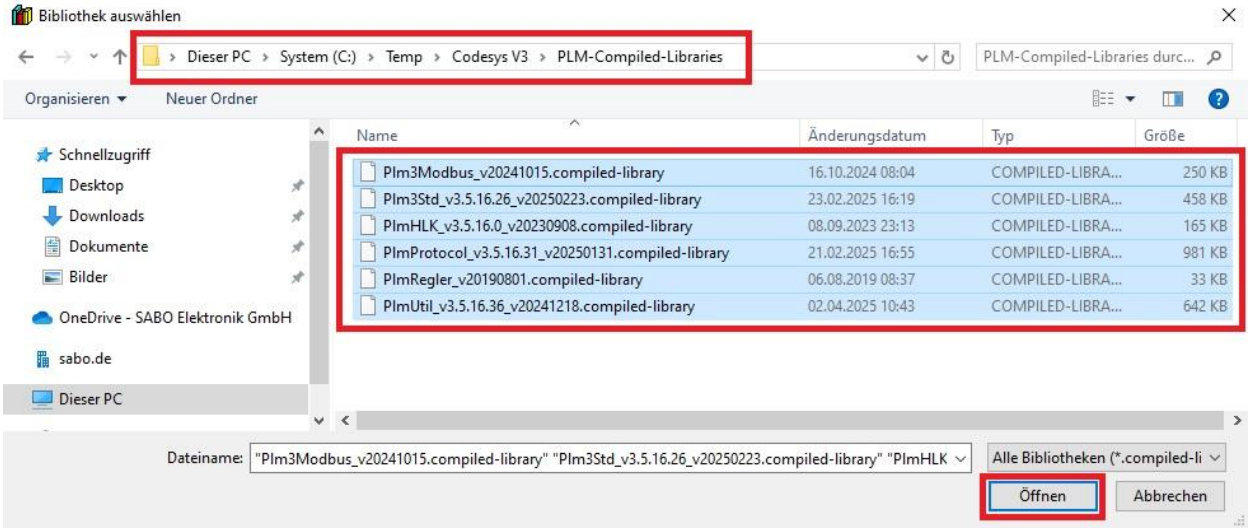


Abbildung 17: Installation der Bibliotheksdateien

Nachdem die von Ihnen ausgewählten Bibliotheksdateien dem Bibliotheksrepository hinzugefügt wurden, wird Ihnen in einem neuen Fenster eine Übersicht angezeigt, welche Bibliotheken neu installiert wurden. Dieses Fenster zeigt Ihnen eventuell schon vorher installierte Bibliotheken an. So können von einzelnen Bibliotheken mehrere unterschiedliche Versionen installiert sein. *Abbildung 18*.

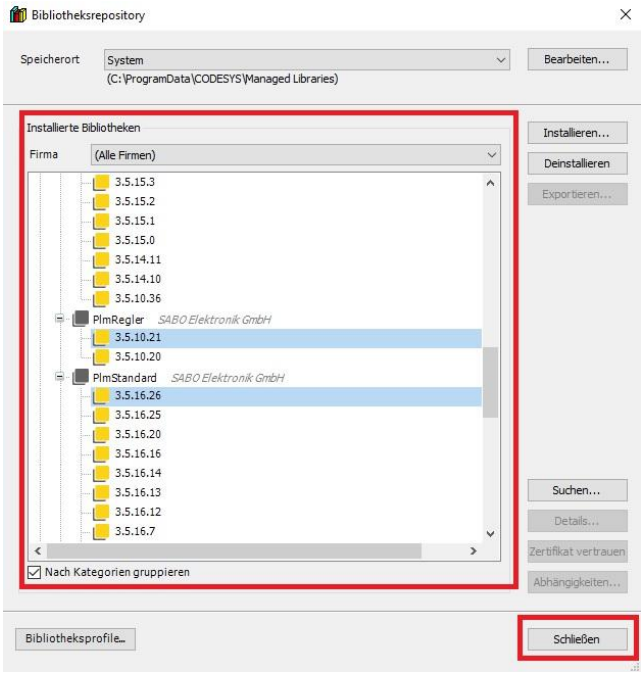


Abbildung 18: Übersicht über die installierten Bibliotheken

Über die Schaltfläche **Schließen** wird das Fenster geschlossen und damit ist die Installation der Bibliotheken abgeschlossen.

Sobald Sie alle vorherigen Schritte abgeschlossen haben, steht Ihnen eine CODESYS-Programmierumgebung zur Verfügung, mit der Sie die SABO-Steuerungen programmieren können.

4 EINRICHTEN UND KONFIGURIEREN EINES STANDARDPROJEKTS IN CODESYS V3.5

4.1 LEERES STANDARDPROJEKT ERSTELLEN

Starten Sie die CODESYS Programmierumgebung und warten Sie, bis der Startbildschirm erscheint. Sobald Sie den Startbildschirm sehen, klicken Sie im oberen Menüband auf **Datei** und wählen **Neues Projekt...** aus. Siehe *Abbildung 19*.

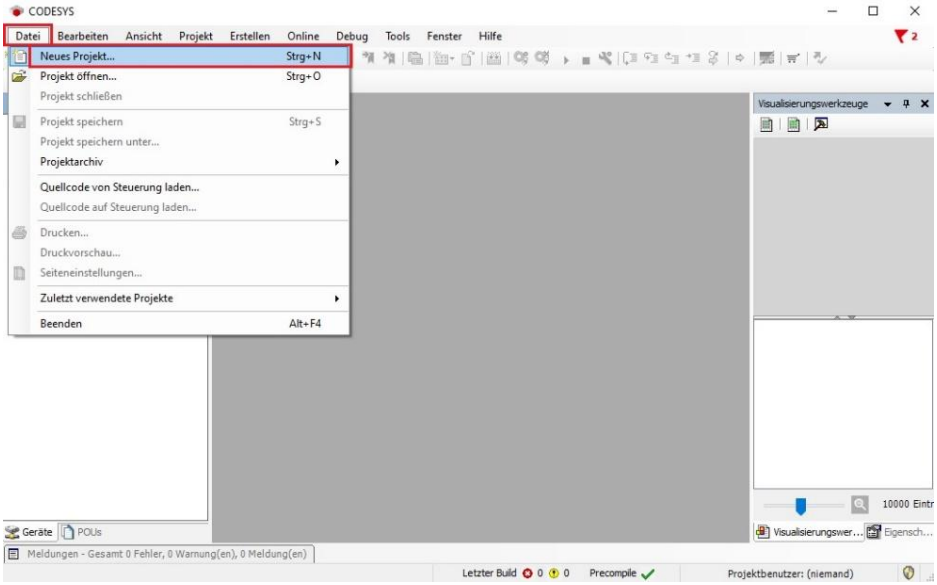


Abbildung 19: Neues Projekt erstellen

Im neuen Fenster wählen Sie bitte **Standardprojekt** aus. Geben Sie anschließend einen Namen für Ihr Projekt ein und legen Sie das Verzeichnis fest, in dem das Projekt gespeichert werden soll. In unserem Beispiel nennen wir das Projekt „*My_project*" und speichern es im Dokumente-Ordner des Benutzers. Siehe dazu auch *Abbildung 20*.

Sie können den Namen und das Verzeichnis, in dem das Projekt abgelegt wird, frei wählen. Die Dateiendung für ein CODESYS V3.5 Projekt ist **.project** und wird automatisch von CODESYS an Ihren Projektnamen angehangen. Bestätigen Sie das Fenster mit **OK.**

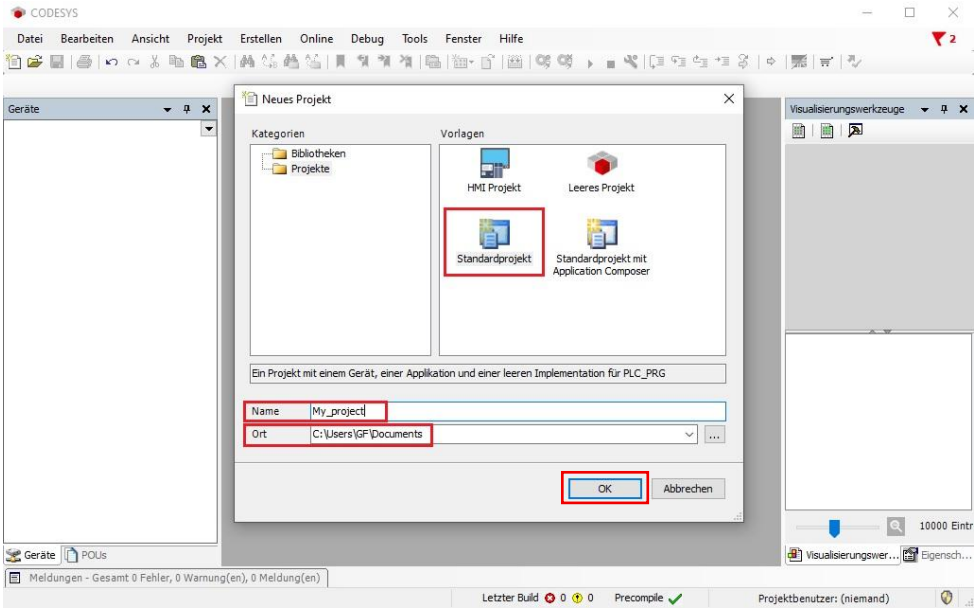


Abbildung 20: Standardprojekt erstellen

Im nächsten Schritt werden Sie gefragt, welches Gerät (Steuerung) Sie programmieren wollen und in welcher Programmiersprache der Programmbaustein PLC_PRG angelegt werden soll. Es steht Ihnen Strukturierter Text (ST), Kontaktplan (KOP), Funktionsbausteinsprache (FUP), Continuous Function Chart (CFC) und Ablaufsprache (AS) zur Verfügung.

Wir entscheiden uns in unserem Beispiel für Strukturierten Text (ST) und bei dem Gerät, was einer Steuerung entspricht, wählen wir MTB.807.10 aus, *Abbildung 21*. Mit **OK** übernehmen Sie Ihre Auswahl.

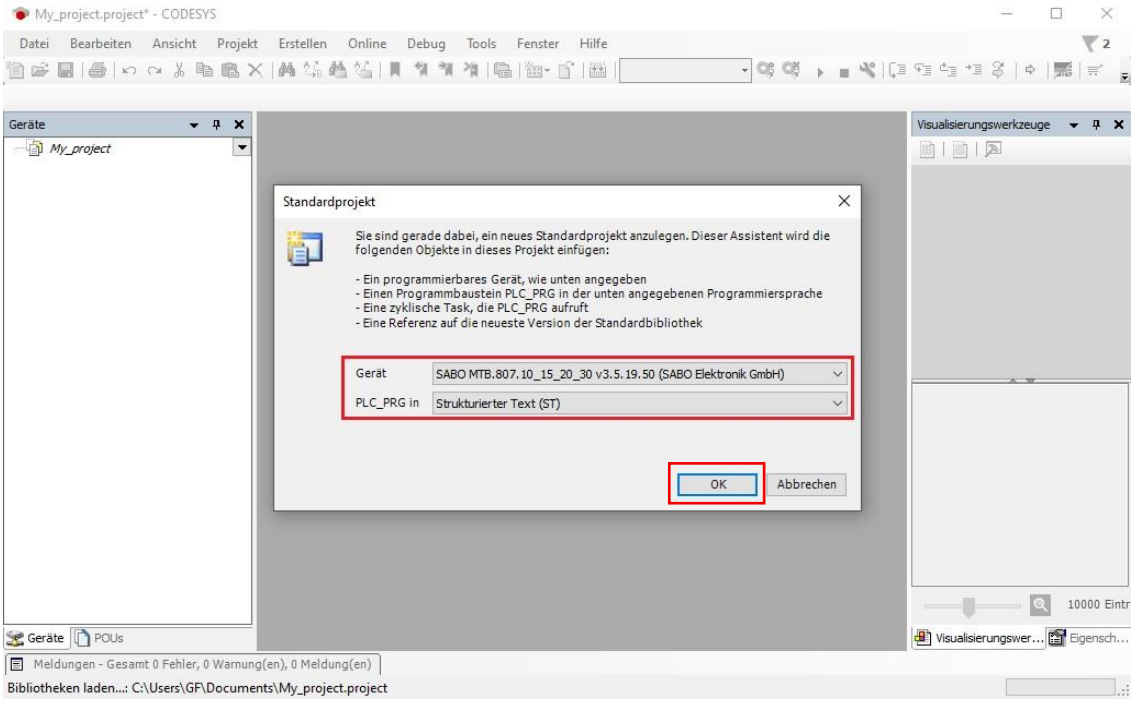


Abbildung 21: Konfiguration des Standardprojekts

Sie können sich auch für ein anderes Gerät entscheiden. Wichtig ist, dass Sie ein vorhandenes Gerät auswählen. Sollten bei Ihnen keine Geräte aus der SABO-Produktserie angezeigt werden, dann liegt das wahrscheinlich daran, dass Sie die Devicedescription (Gerätebeschreibung) für die SABO-Produktserie nicht installiert haben. Siehe *Abschnitt* 3.2 .

Wenn Sie alle Schritte wie beschrieben ausgeführt haben, dann sollte die Programmierumgebung genauso aussehen, wie in der *Abbildung* 22. Falls Sie sich für eine andere Steuerung entschieden haben, wird Ihnen entsprechend diese dort angezeigt.

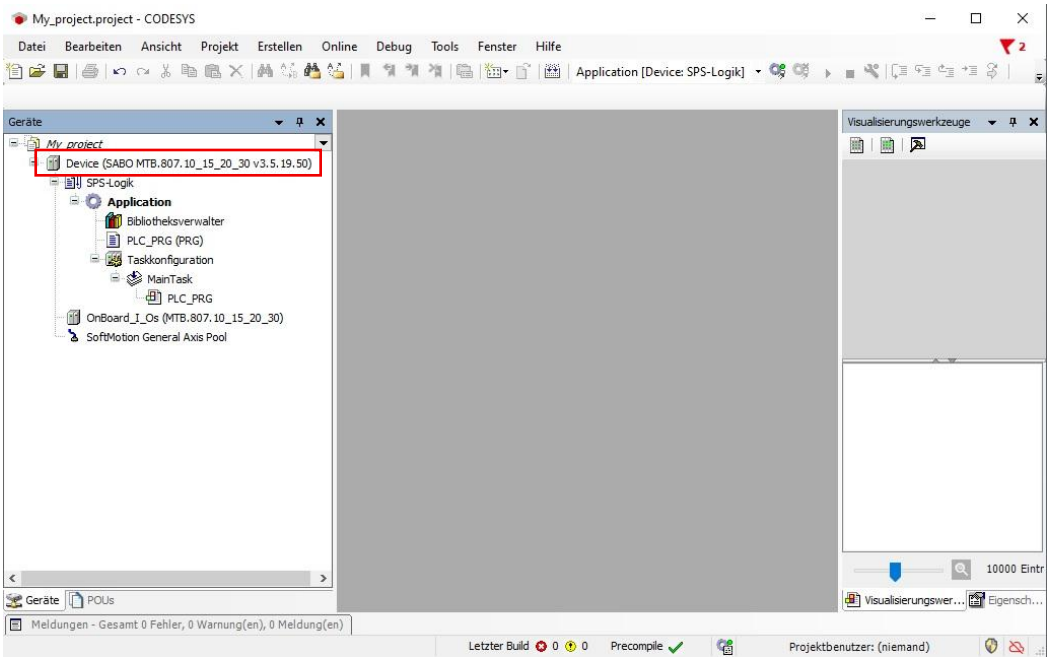


Abbildung 22: Standardprojekt

Mit diesen Schritten haben Sie ein Standardprojekt erstellt, auf dem Sie weiter aufbauen können. In den nächsten Kapiteln wird erklärt, wie man CAN-Module anhängt und konfiguriert, Visualisierungen hinzufügt, den Visualisierungsmanager einrichtet und wichtige Einstellungen am Projekt vornimmt.

4.1.1 STEUERUNGSTYP ÄNDERN

Um Missverständnisse zu vermeiden, sollten einige Begriffe, wie sie in CODESYS verwendet werden, im Vorfeld erläutert werden. Wenn in CODESYS der Begriff **Gerät** (im Englischen **Device**) verwendet wird, ist damit die Steuerung gemeint. Zur Vereinfachung verwenden wir nach Möglichkeit in diesem Handbuch den Begriff **Steuerung**. Trotzdem kann es vorkommen, dass die Begriffe Device oder Gerät benutzt werden müssen, da sie so in der CODESYS Programmierumgebung angezeigt werden.

Sie können jederzeit die Steuerung in Ihrem Projekt austauschen, falls Sie sich vertan haben oder eine andere Steuerung nutzen möchten. Dazu führen Sie in der Programmierumgebung auf der linken Seite einen Rechtsklick auf **Device** aus und klicken anschließend auf **Gerät aktualisieren**... . Siehe *Abbildung 23*.

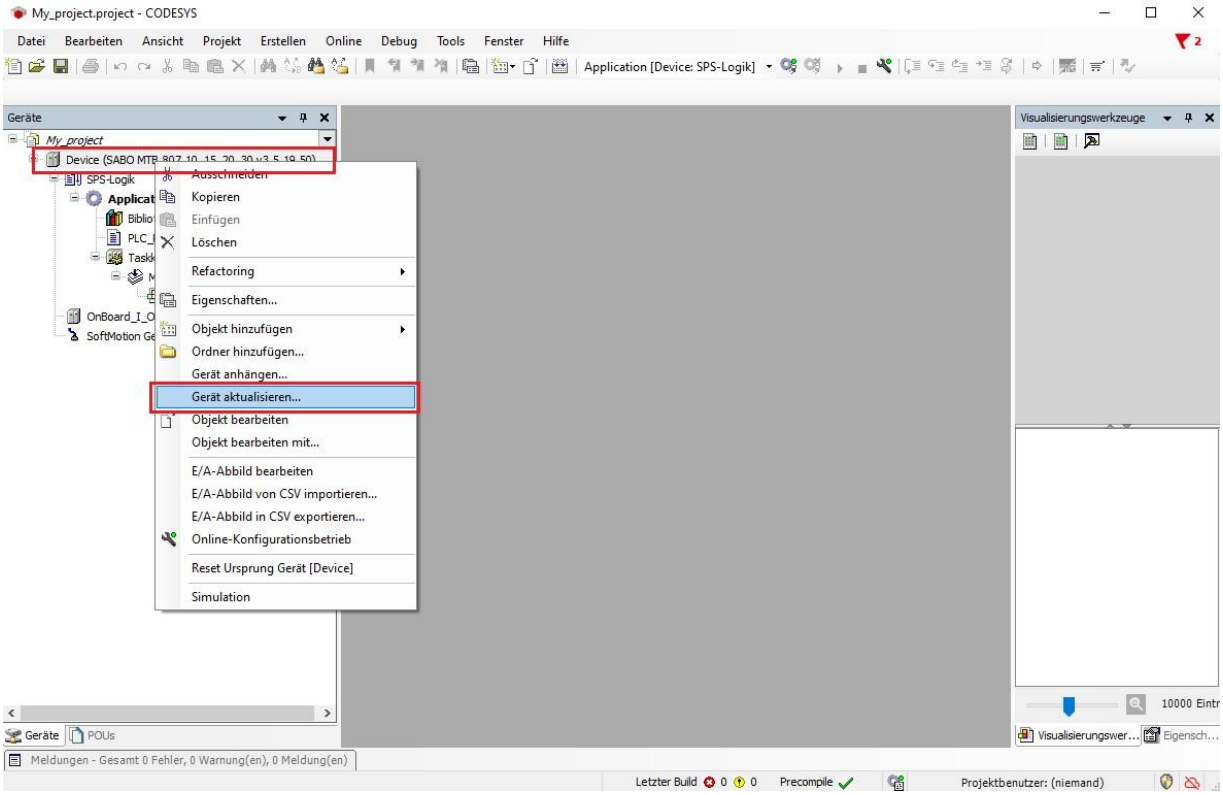


Abbildung 23: Steuerung aktualisieren

In dem neuen Fenster (*Abbildung 24*) werden Ihnen alle Gerätebeschreibungen zu Steuerungen angezeigt, die in dem Geräterepository installiert sind. In der Standardansicht werden Ihnen immer die neusten Versionen der Gerätebeschreibung angezeigt, auch wenn Sie ältere Versionen installiert haben.

Um alle installierten Gerätebeschreibungen sichtbar zu machen, müssen Sie in dem Fenster die zwei Häkchen für **Alle Versionen anzeigen (nur für Experten)** und **veraltete Versionen anzeigen** setzten. Erst dann werden Ihnen in der Übersicht alle installierten Versionen der Gerätebeschreibung angezeigt.

Wählen Sie jetzt aus den angezeigten Steuerungen die aus, die Sie nutzen möchten und klicken Sie anschließend auf **Gerät aktualisieren**, damit die von Ihnen ausgewählte Steuerung in das Projekt übernommen wird. Sobald das abgeschlossen ist, können Sie mit **Schließen** das Fenster verlassen.

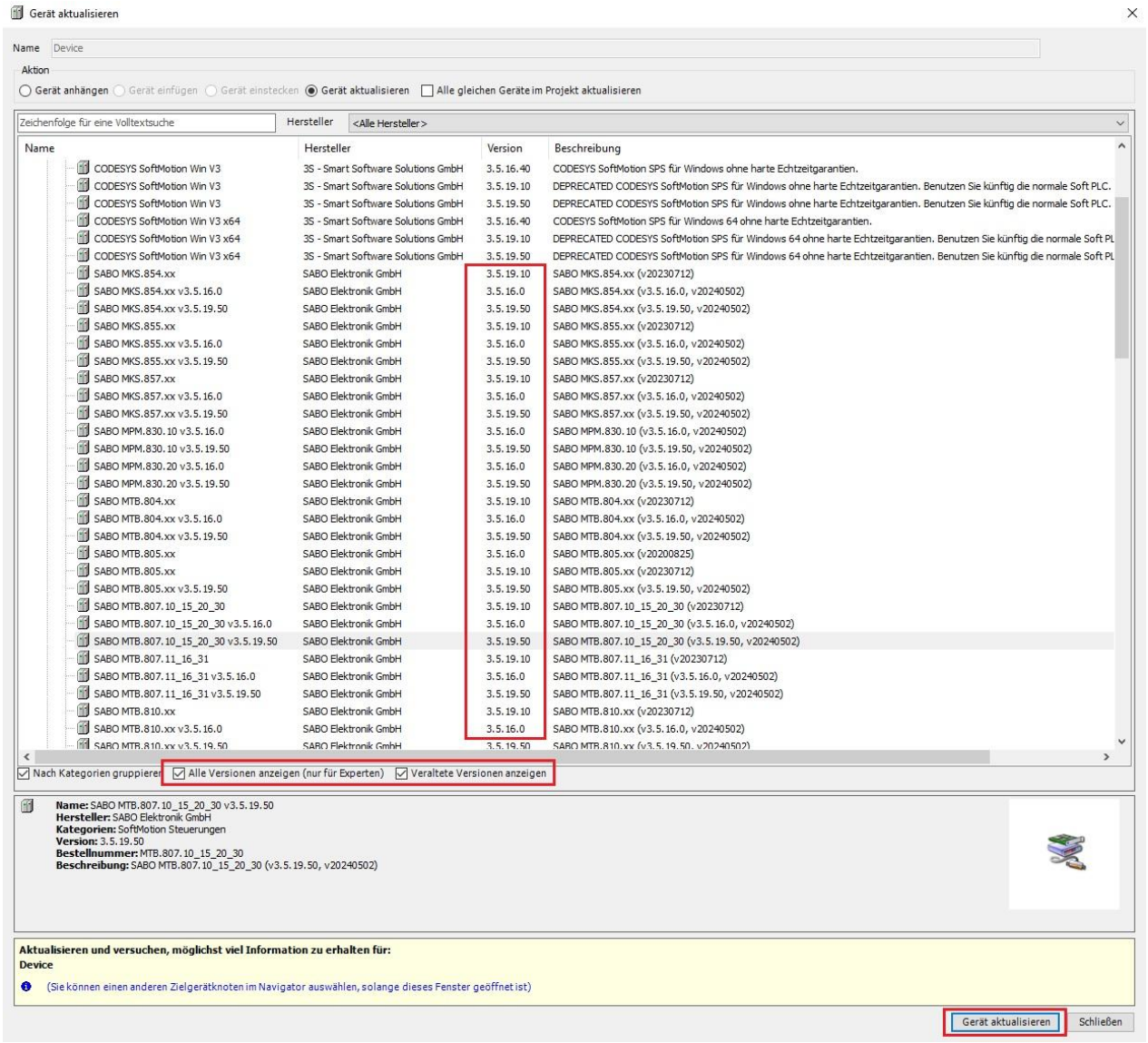


Abbildung 24:Ansicht der installierten Gerätebeschreibungen

In der Programmierumgebung sollten Sie hinter Device die von Ihnen neu ausgewählte Steuerung angezeigt bekommen.

In unserem Beispiel haben wir die Steuerung von einer *SABO MTB.807.10_15_20_30 v3.5.19.50* gegen eine *SABO MTB.810.xx v3.5.19.50* ausgetauscht. Siehe *Abbildungen 25 und 26*.

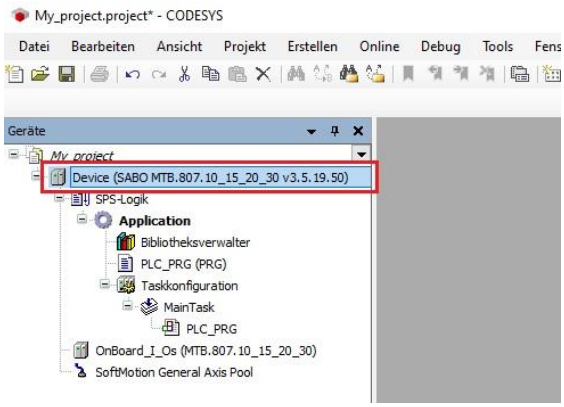


Abbildung 25: Vorher

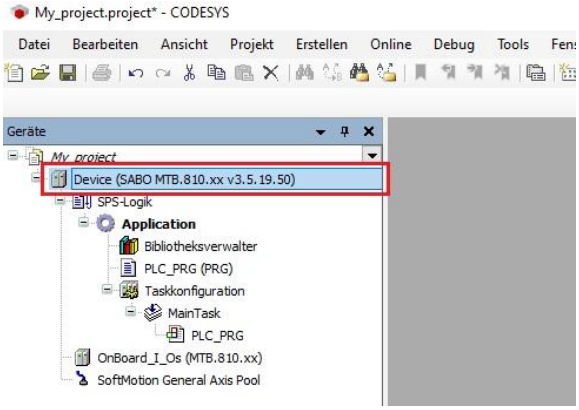


Abbildung 26: Nachher

4.2 CAN-BUS UND CAN-MODULE EINRICHTEN

In unserem Beispiel möchten wir an die Steuerung einen CAN-Master einfügen, an dem wiederrum das CAN-Modul EWB.800.10 angehängt wird.

Dazu führen Sie in der Programmierumgebung auf der linken Seite einen Rechtsklick auf **Device** aus und klicken dann auf **Gerät anhängen**... .

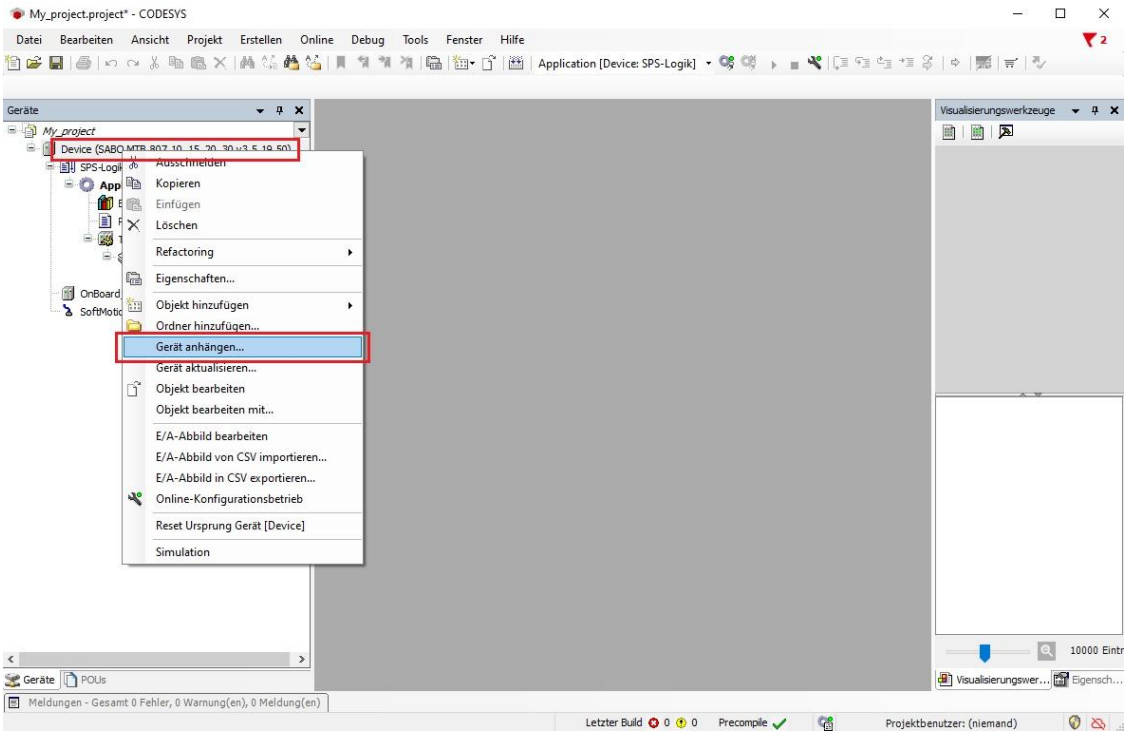


Abbildung 27: Gerät anhängen

In dem neuen Fenster wählen Sie **Feldbusse** aus und den Unterpunkt **CANbus**. Klicken Sie anschließend auf **Gerät anhängen**, aber nicht auf **Schließen**!

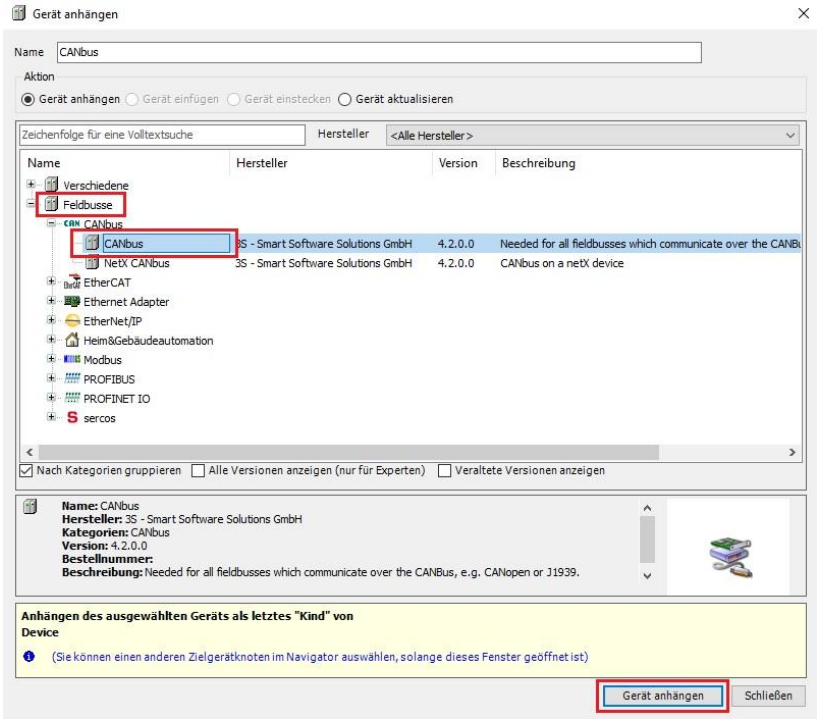


Abbildung 28: CANbus einfügen

Im nächsten Schritt muss an den CANbus ein CANopen Manager angehangen werden. Dazu klicken Sie auf der linken Seite in Ihrer Programmierumgebung auf **CANbus (CANbus)**. Das noch offene Fenster für die Geräte bietet Ihnen automatisch die möglichen Optionen an, die Sie an den CANbus anhängen können. Hier wählen Sie den **CANopen_Manager** und klicken auf **Gerät anhängen**. Klicken Sie noch nicht auf **Schließen**! *Abbildung 29*.

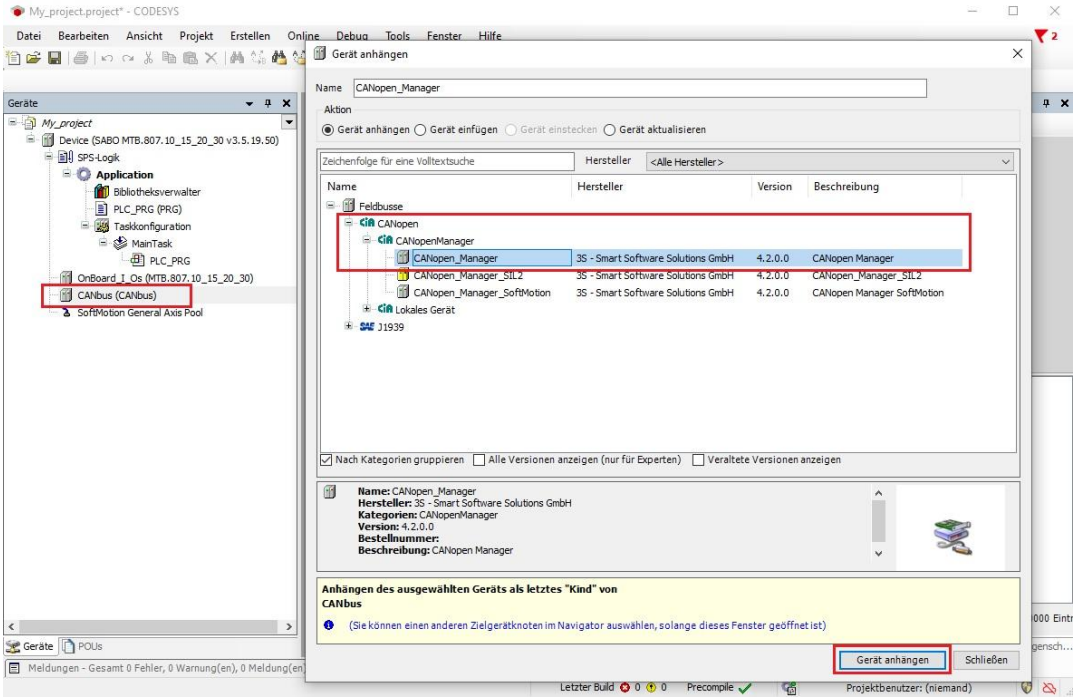


Abbildung 29: CANopen Manager einfügen

An den **CANbus_Manager** können ein oder mehrere CAN-Module angehangen werden. Für dieses Beispiel wird eine EWB.800.10 angehangen. Hierfür klickt man auf der linken Seite in der Programmierumgebung auf den **CANopen_Manager (CANopen_Manager)**, so dass in dem Fenster für die Geräteauswahl jetzt alle CAN-Module, die in dem Geräterepository installiert sind, angezeigt werden. Die EWB.800.10 wird aus der Liste ausgewählt und über die Schaltfläche **Gerät anhängen** dem CANopen_Manager hinzugefügt.

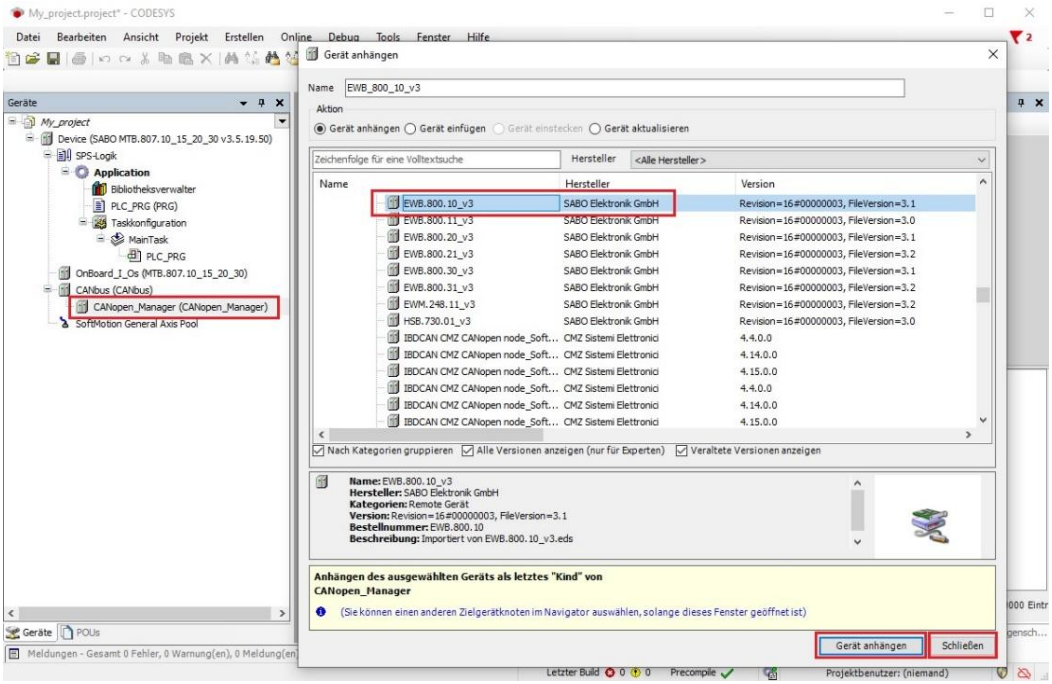


Abbildung 30: Einfügen der EWB.800.10 als CAN-Modul

Nach diesem Schritt können Sie das Fenster mit dem Knopf **Schließen** verlassen.
Das Projekt umfasst nun einen CANbus, der als CAN-Master arbeitet. Daran ist ein CANopen-Manager angeschlossen sowie ein CAN-Modul angebunden – in unserem Beispiel handelt es sich dabei um die EWB.800.10.

Es können auch mehr Module nach dem gleichen Verfahren angehangen werden, zum Anschauungszweck reicht aber eines aus. *Abbildung* 31 zeigt den Aufbau in der Programmierumgebung.

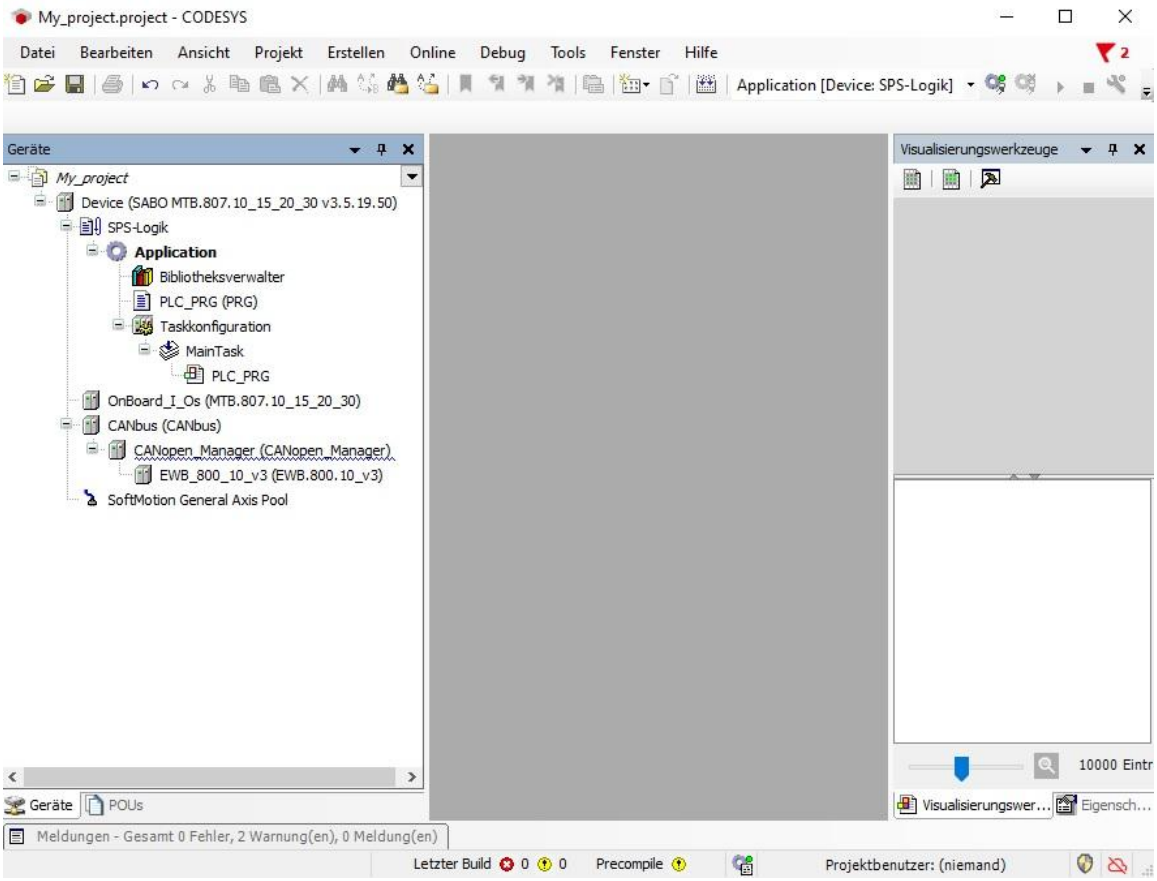


Abbildung 31: CANbus Aufbau

Wie der CANbus und das Modul konfiguriert werden müssen, wird im nächsten Abschnitt erklärt.

4.2.1 KONFIGURATION CANBUS

Im nächsten Schritt muss die Baudrate, die CAN Node-ID und die Art der CANbus-Überwachung konfiguriert werden. Details zur Einstellung der Baudrate und der CAN Node-ID an den CAN-Modulen entnehmen Sie bitte dem entsprechenden Datenblatt des jeweiligen Moduls.
Zu jedem Modul wird auf der Webseite www.sabo.de das entsprechende Datenblatt zum Download angeboten.

In unserem Beispiel ist die Baudrate der EWB.800.10 auf 125 kBaud eingestellt, und die CAN Node-ID lautet 2. Über einen Doppelklick auf **CANbus (CANbus)** wird die Konfigurationsseite für die Baudrate vom CANbus Master aufgerufen. Über ein Dropdownmenü im Reiter **Allgemein** kann die passende Baudrate ausgewählt werden. *Abbildung* 32.

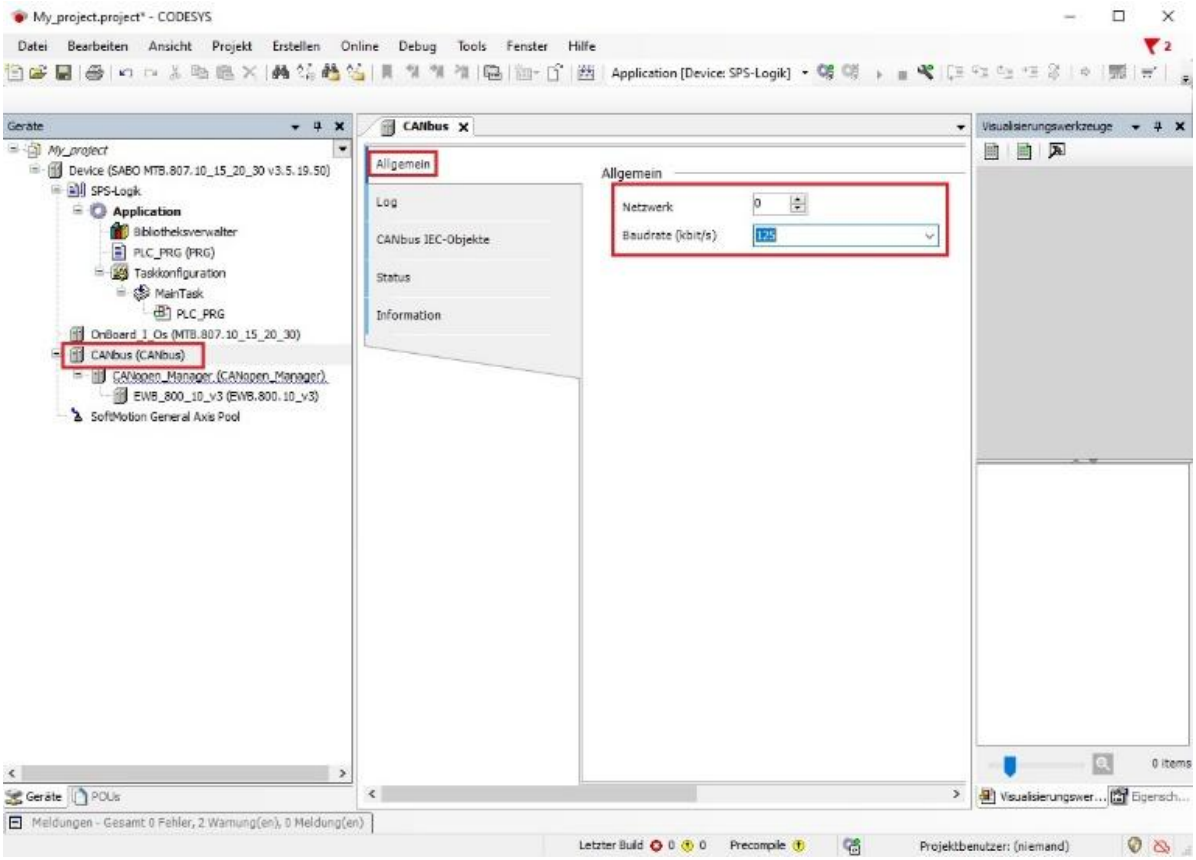


Abbildung 32: Baudrate des CANbus Master einstellen

Auf der EWB.800.10 ist eine Baudrate von 125 kBaud eingestellt, so dass die Baudrate für den CANbus Master ebenfalls auf 125 kBaud gestellt werden muss.

Mit einem Doppelklick auf **CANopen_Manager (CANopen_Manager)** wird die Konfigurationsseite für den CANopen Manager aufgerufen. In dem Reiter **Allgemein** muss das Häkchen für **Heartbeat-Erzeugung aktivieren** entfernt werden, da wir für die Überwachung das Nodeguarding nutzen wollen. *Abbildung 33*.

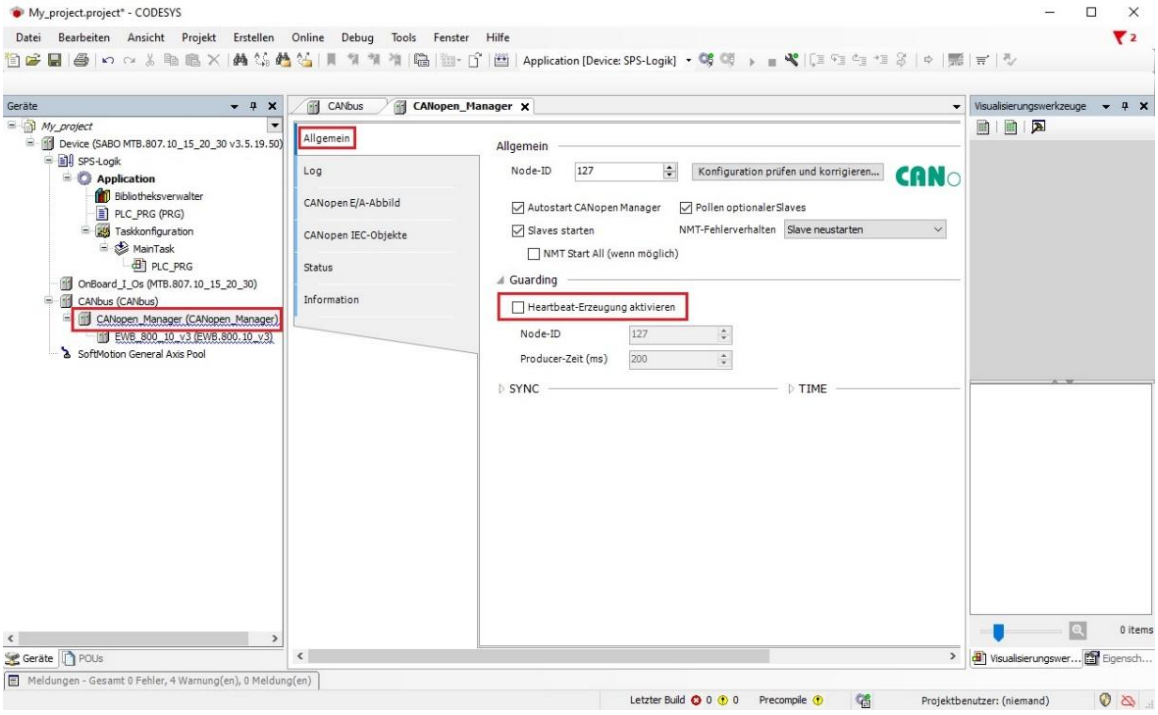


Abbildung 33: Heartbeat-Erzeugung deaktivieren

Als nächstes muss das CAN-Modul konfiguriert werden. Mittels eines Doppelklicks auf das CAN-Modul, in diesem Fall ein Doppelklick auf die **EWB_800_10_v3(EWB.800.10_v3)**, wird die Konfigurationsseite für das Modul aufgerufen.

Über den Reiter **Allgemein** können alle benötigten Einstellungen vorgenommen werden. Die am Modul eingestellte CAN Node-ID wird in dem Feld **Node-ID** (im Beispiel 2) eingetragen. Setzten Sie das Häkchen bei **Experteneinstellungen**, damit Ihnen alle möglichen Konfigurationsparameter angezeigt werden.

Entfernen Sie das Häkchen bei **Heartbeat-Erzeugung aktivieren** und setzen Sie ein Häkchen bei **Nodeguarding aktivieren**. Somit wird für das CAN-Modul die Heartbeat Überwachung deaktiviert und stattdessen das Nodeguarding aktiviert. Erfahrungen haben gezeigt, dass das Nodeguarding zuverlässiger arbeitet als die Heartbeat Überwachung. Siehe *Abbildung 34*.

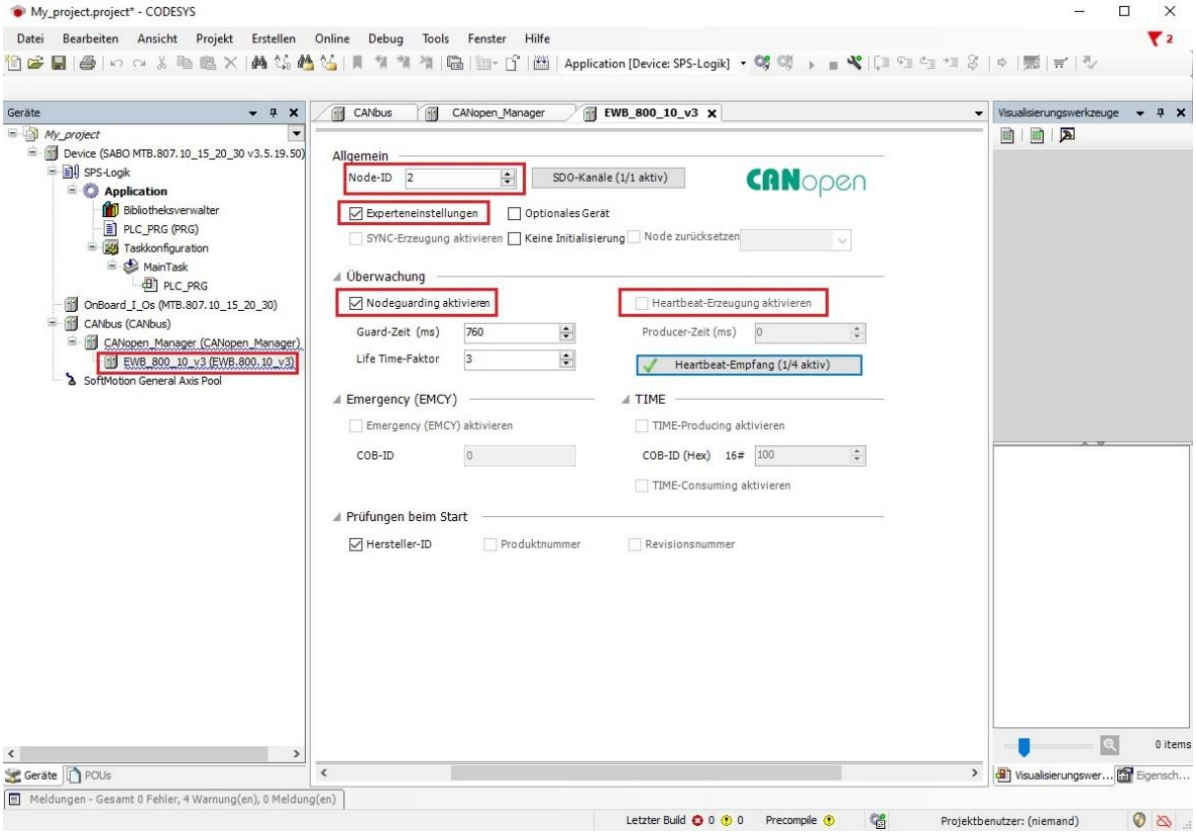


Abbildung 34: Einstellungen an einem CAN-Modul

Zuletzt muss noch der Heartbeat-Empfang deaktiviert werden. Dazu muss auf die Schaltfläche **Heartbeat-Empfang (x/x aktiv)** geklickt werden. In dem neuen Fenster entfernen Sie alle Häkchen in der Spalte Aktivieren. Anschließend kann das Fenster über **OK** geschlossen werden. Wie in *Abbildung 35* gezeigt.

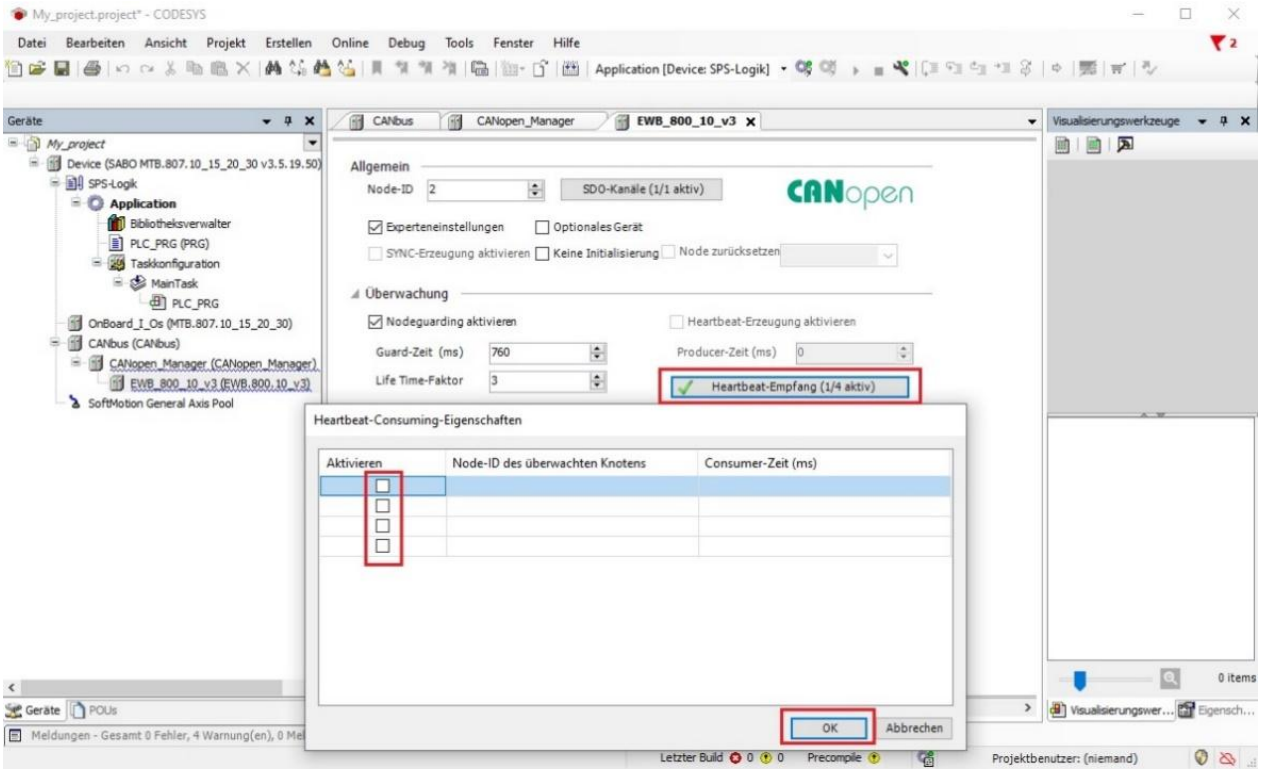


Abbildung 35: Heartbeat Empfang deaktivieren

Damit ist die Konfiguration vom CANbus und dem CAN-Modul abgeschlossen.

4.3 VISUALISIERUNGSMANAGER KONFIGURIEREN

Um Visualisierungsseiten auf dem Display der Steuerung oder in der Webvisualisierung darzustellen, wird ein Visualisierungsmanager benötigt. Die einfachste Methode, einen solchen Manager in das Projekt zu integrieren, besteht darin, eine Visualisierungsseite zum Projekt hinzuzufügen.

Dadurch erstellt die Programmierumgebung automatisch einen Visualisierungsmanager für die Steuerungsvisualisierung (Target-Visu) sowie für die Webvisualisierung (Webvisu).

Dazu klicken Sie mit einem Rechtsklick auf **Applikation**, dann auf **Objekt hinzufügen**... und wählen hier den Punkt **Visualisierung**... aus. *Abbildung 36*.

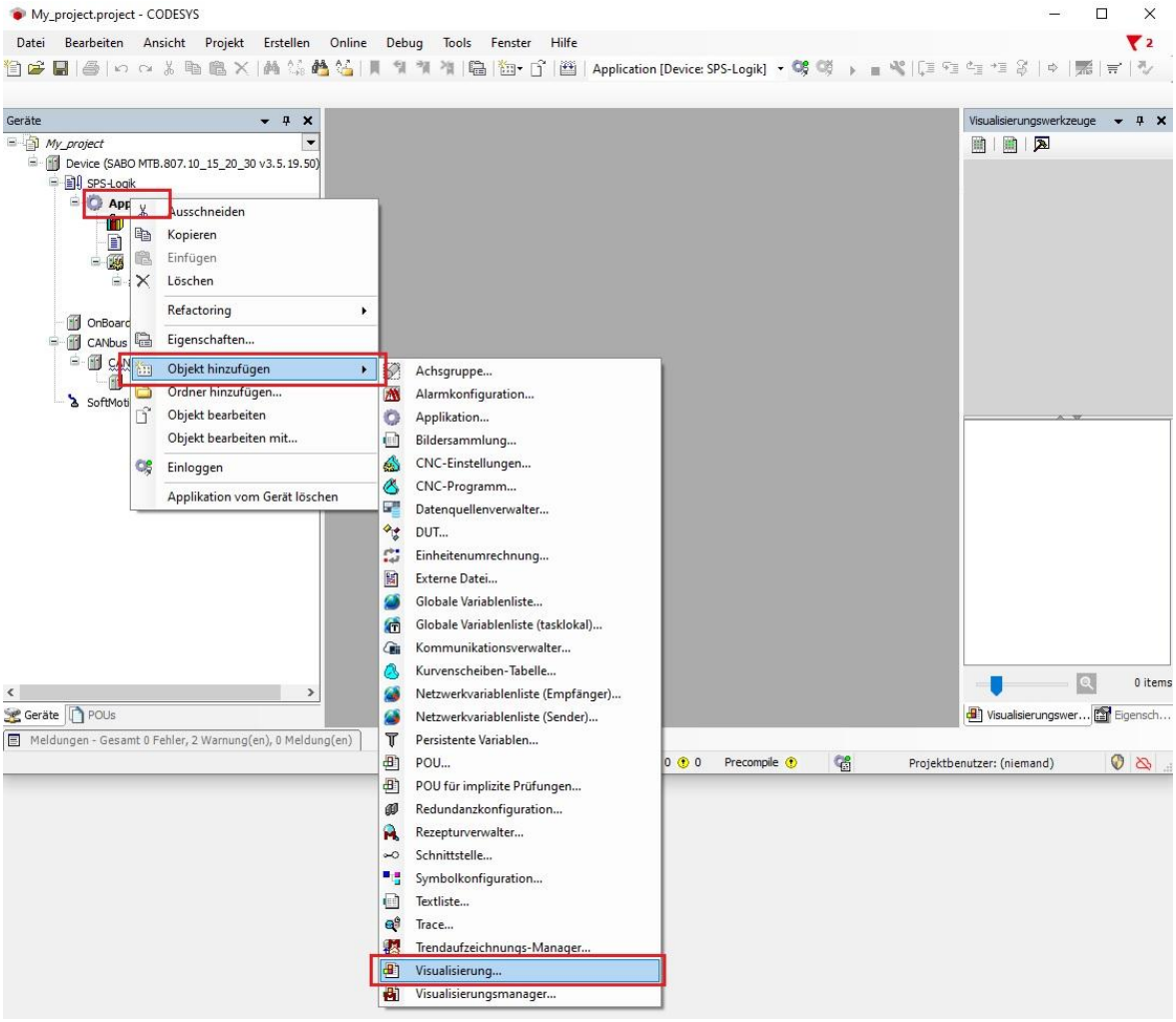


Abbildung 36: Eine neue Visualisierungsseite erstellen

In dem neu erscheinenden Fenster geben Sie den Namen für die Visualisierungsseite ein. In unserem Beispiel vergeben wir den Namen *Startseite*. Schließen Sie das Fenster anschließend über die Schaltfläche **Hinzufügen**.

Abbildung 37: Namen für die neue Visualisierungsseite vergeben

Nach dem Sie auf **Hinzufügen** geklickt haben, dauert es einen Augenblick, bis die Programmierumgebung den Visualisierungsmanager mit den benötigten Bibliotheken erstellt und hinzugefügt hat. Sobald der Vorgang abgeschlossen wurde, sollte die Programmierumgebung wie in *Abbildung 38* aussehen.

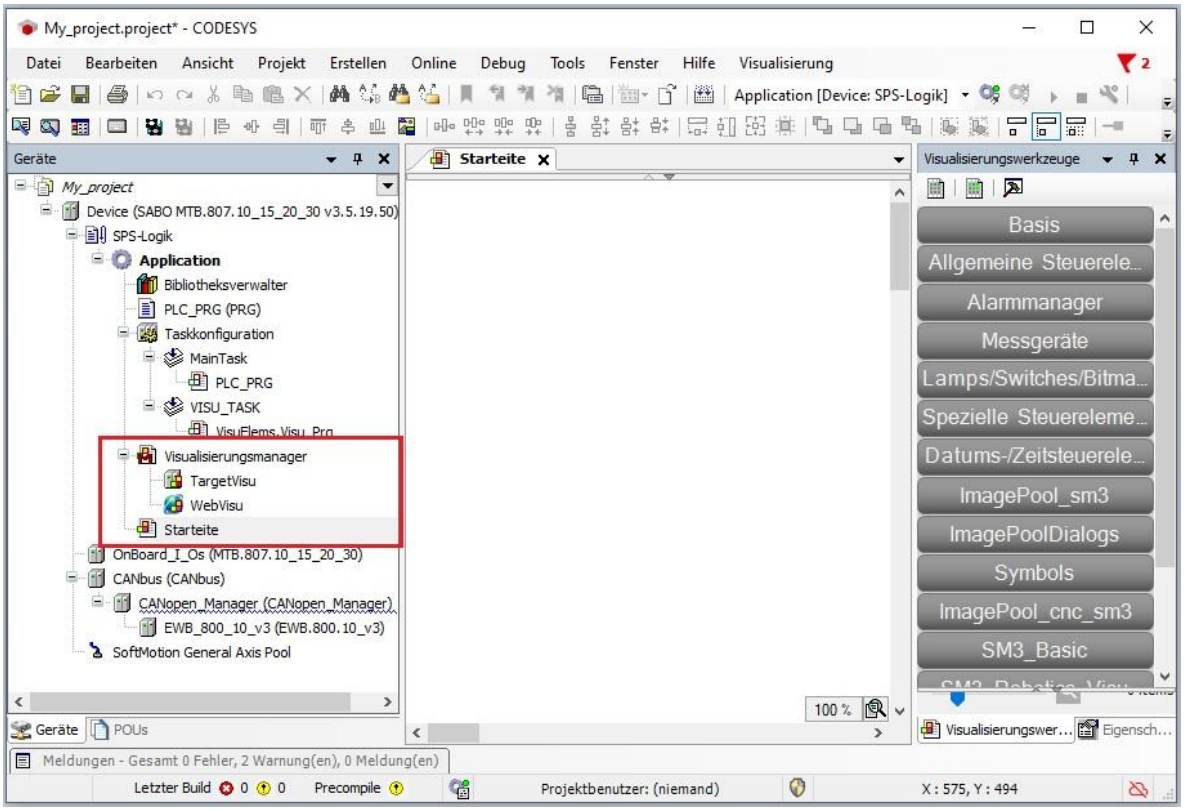


Abbildung 38: Visualisierungsmanager mit den Unterpunkten TargetVisu und WebVisu

Für zusätzliche Einstellungen öffnen Sie in der Programmierumgebung auf der linken Seite per Doppelklick den Eintrag **TargetVisu**. Ändern Sie hier die Aktualisierungsrate für die Visualisierung in der Steuerung. Tragen Sie in das Feld eine 40 ein, das bedeutet, dass die Visualisierung in der Steuerung alle 40ms aktualisiert wird.

Kleinere Zeiten als 40ms sind nicht möglich. Nach dem Sie den Wert geändert haben klicken Sie anschließend auf **Task VISU_TASK an Zykluszeit 40ms anpassen**. Damit wird der Task für die Visualisierung ebenfalls auf 40ms gestellt.

Zuletzt ändern Sie noch die Standardtexteingabe auf **Touchscreen**, wenn Sie eine Steuerung mit Display verwenden. Siehe *Abbildung 39*.

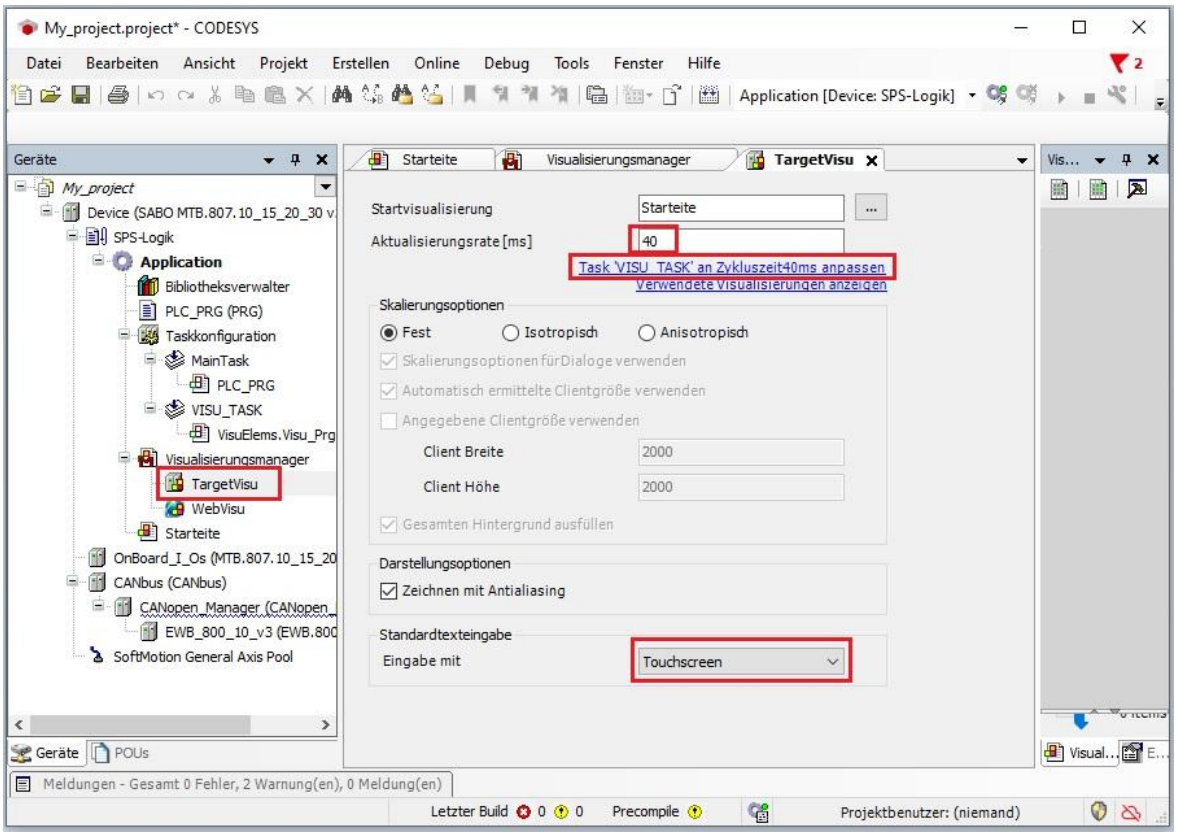


Abbildung 39: Einstellungen in der TargetVisu

Im nächsten Schritt wird die editierbare Größe der Visualisierung für das gesamte Projekt festgelegt. Hierfür klicken Sie bei den **Skalierungoptionen** auf **Isotropisch** oder **Anisotropisch** und setzen das Häkchen bei **Angegebene Clientgröße verwenden**.

Jetzt kann in den Feldern **Client Breite** und **Client Höhe** die Breite und Höhe des verbauten Displays von der Steuerung angegeben werden. Die Breite und Höhe entnehmen Sie der *Tabelle 1*.

Steuerung	Breite in Pixel	Höhe in Pixel
MTB.804.10/20	480	272
MTB.805.10/20	800	480
MTB.807.10/20	800	480
MTB.810.20	1280	800
MTB.815.20	1920	1080
MTB.821.20	1920	1080

Tabelle 1: Displayauflösungen der PLM800 Masterterminals

In unserem Beispiel sind das 800 Pixel bei Client Breite und 480 Pixel bei Client Höhe, da wir eine MTB.807.10 verwenden.

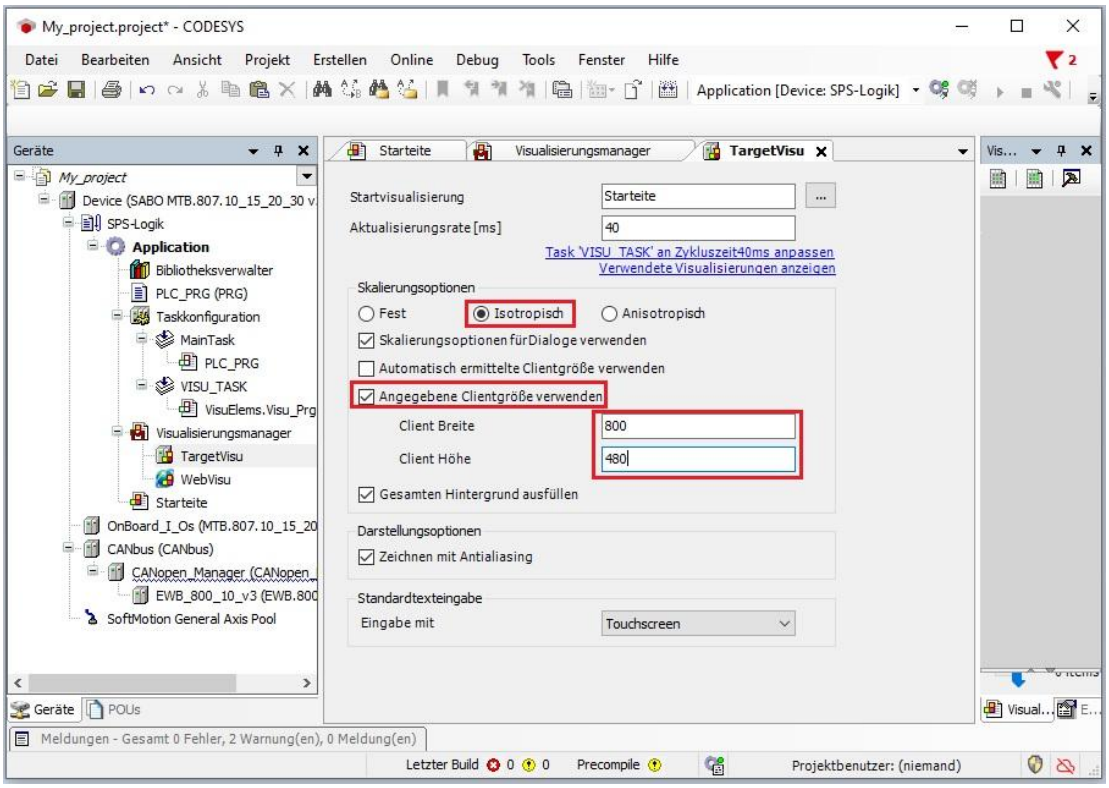


Abbildung 40: Höhe und Breite des Displays konfigurieren

Nach Eingabe der Client Breite und Höhe klicken Sie wieder auf die **Skalierungsoption Fest**.

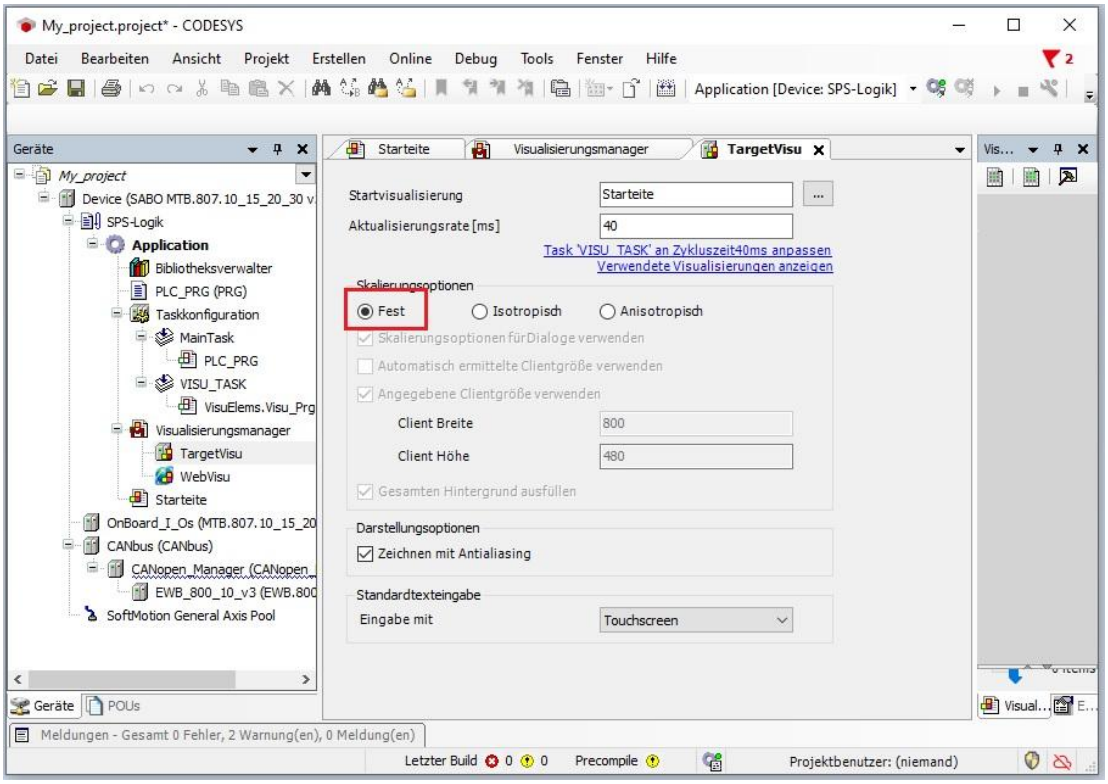


Abbildung 41: Skalierungsoptionen

Damit wir die von Ihnen angegebene Größe von dem Display der Steuerung für den editierbaren Bereich in der Visualisierung übernommen. Das vereinfacht das Zeichnen der Visualisierungsseiten. *Abbildung 42* zeigt die veränderte Zeichenfläche, die jetzt dem Display der Steuerung entspricht.

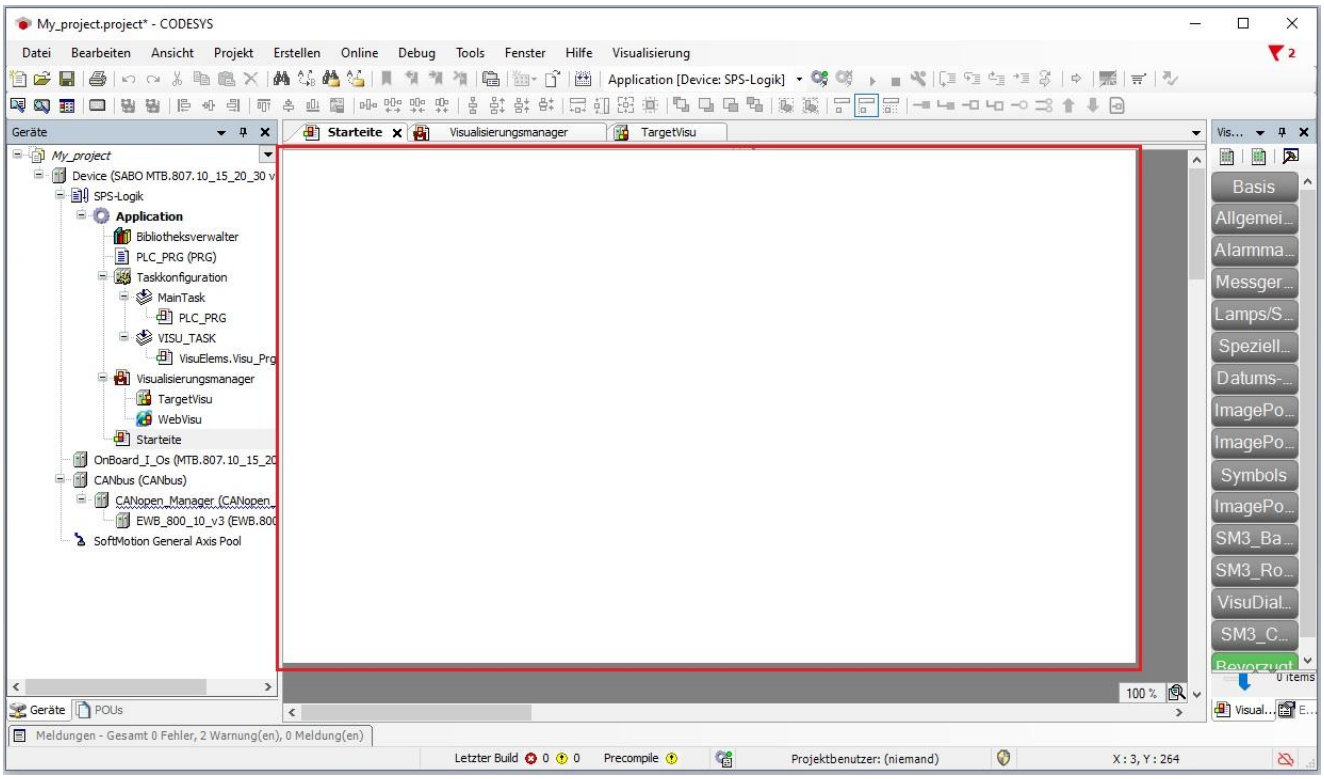


Abbildung 42: Angepasste Zeichenfläche

Sie können nun weitere Visualisierungen einfügen, die alle die gleiche editierbare Fläche haben werden.

4.4 ALLGEMEINE EINSTELLUNGEN IN CODESYS 3.5 VORNEHMEN

In diesem Abschnitt werden weitere sinnvolle Einstellungen in der CODESYS Programmierumgebung besprochen.

Die Variable **CurrentVisu** kann im Programmcode nur benutzt werden, wenn diese vorher im Visualisierungsmanager aktiviert wurde. Wird das Häkchen bei **CurrentVisu-Variable verwenden** gesetzt, dann können Sie über Programmcode diese Variable auslesen oder beschreiben. Somit haben Sie die Möglichkeit festzustellen, auf welcher Visualisierungsseite Sie sich befinden oder Sie können die Visualisierungsseite mittels Programmcode wechseln.

Wenn Sie das Häkchen aktiviert haben, dann ist die Webvisualisierung mit der Targetvisualisierung (Steuerung) verbunden. Das bedeutet, dass wenn Sie einen Seitenwechsel auf der Steuerung erzeugen, der Seitenwechsel gleichzeitig auf der Webvisualisierung geschieht. Und umgekehrt, ändern Sie in der Webvisualisierung eine Seite, dann ändert sich auch diese auf der Steuerung. Nur wenn das Häkchen nicht gesetzt ist, arbeiten die Webvisualisierung und Targetvisualisierung unabhängig voneinander.

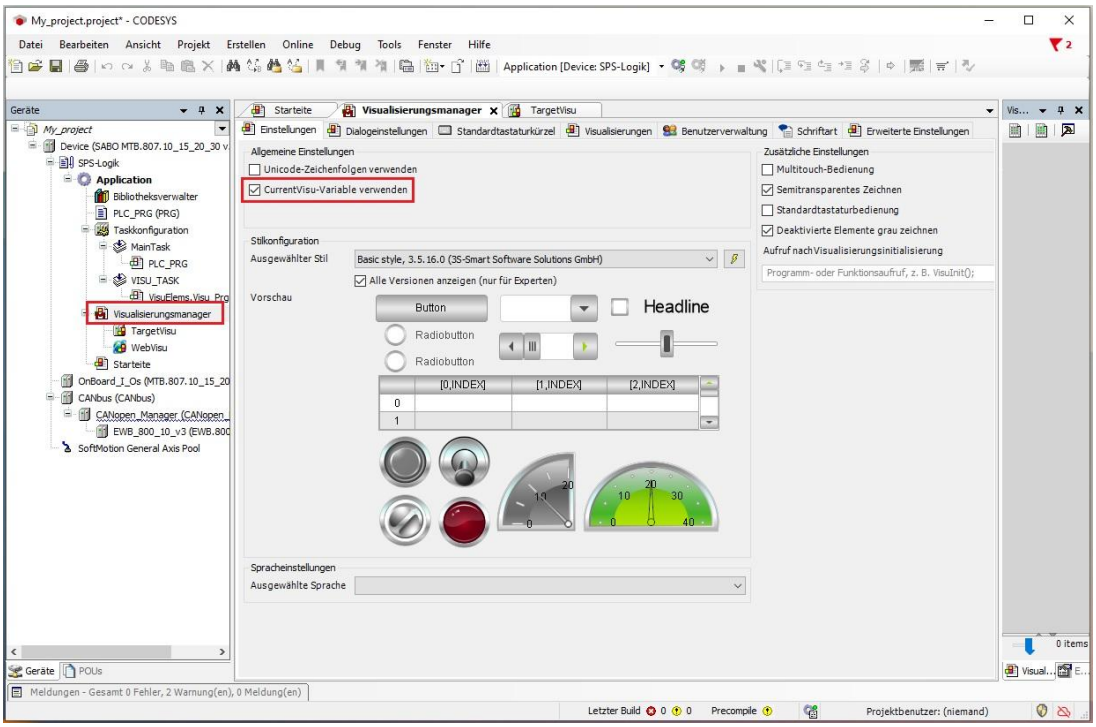


Abbildung 43: CurrentVisu-Variable konfigurieren

Wenn Sie in Ihrem Projekt in der Visualisierung Mehrsprachigkeit nutzen, dann ist eventuell das Häkchen bei **Unicode-Zeichenfolge verwenden** von Relevanz. Unicode ist ein Zeichencodierungssystem, in dem Textdaten auf Computern gespeichert und ausgetauscht werden. Jedem Zeichen der wichtigsten Schriftsysteme der Welt wird dabei ein eindeutiger Wert zugewiesen. Nur wenn Sie diese Option über das Häkchen aktiviert haben, können Sie z.B. Chinesische, Japanische, Koreanische, Kyrillische und andere Schriftzeichen in der Visualisierung anzeigen.
Ist das Häkchen nicht gesetzt und Sie nutzten andere als Lateinische Schriftzeichen, dann wird Ihnen in der Visualisierung ein falsches oder fehlendes Zeichen angezeigt.
Aktivieren Sie diese Option, wenn Sie in Ihrem Projekt Mehrsprachigkeit nutzen möchten und andere Zeichen als Lateinische Schriftzeichen benötigen.

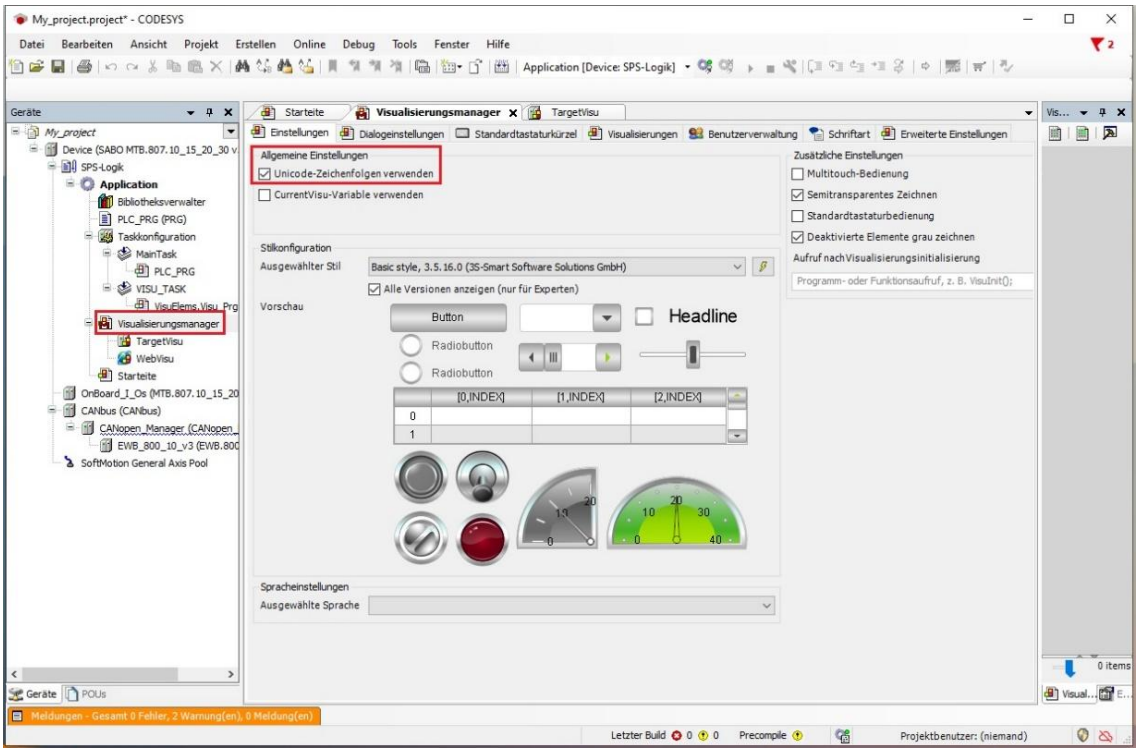


Abbildung 44: Unicode-Zeichenfolge konfigurieren

Mit den Standardeinstellungen von CODESYS V3.5 wird der Inhalt von Variablen, die Sie im Programm anlegen, aber nicht im Programmcode verwenden, nicht aktualisiert. Das gilt auch für Variablen, die Sie an Hardware I/Os mappen. Erst wenn Sie diese Variablen aktiv im Programm aufrufen, werden die Inhalte der Variablen aktualisiert.

Sie könne das Verhalten jedoch ändern, so dass der Inhalt im Buszyklus aktualisiert wird, auch wenn Sie die Variablen in Ihrem Programm nicht aufrufen. Das entspricht dem Verhalten von Variablen in CODESYS V2.3.

Wenn Sie dieses Verhalten wünschen, dann müssen Sie in der Programmierumgebung einen Doppelklick auf **Device** ausführen und aus den vorhandenen Reitern **SPS-Einstellungen** auswählen. Dort gibt es den Unterpunkt **SPS-Einstellungen** mit dem Feld **Variablen immer aktualisieren**. Stellen Sie über das Dropdownmenü von **Deaktiviert (Aktualisierung nur wenn in einem Task verwendet)** auf **Aktiviert 1 (Buszyklustask verwenden, wenn in keiner Task verwendet)** um. *Abbildung 45*.

So wird jetzt der Inhalt Ihre Variablen im Buszyklus aktualisiert, auch wenn Sie die Variablen nicht aufrufen.

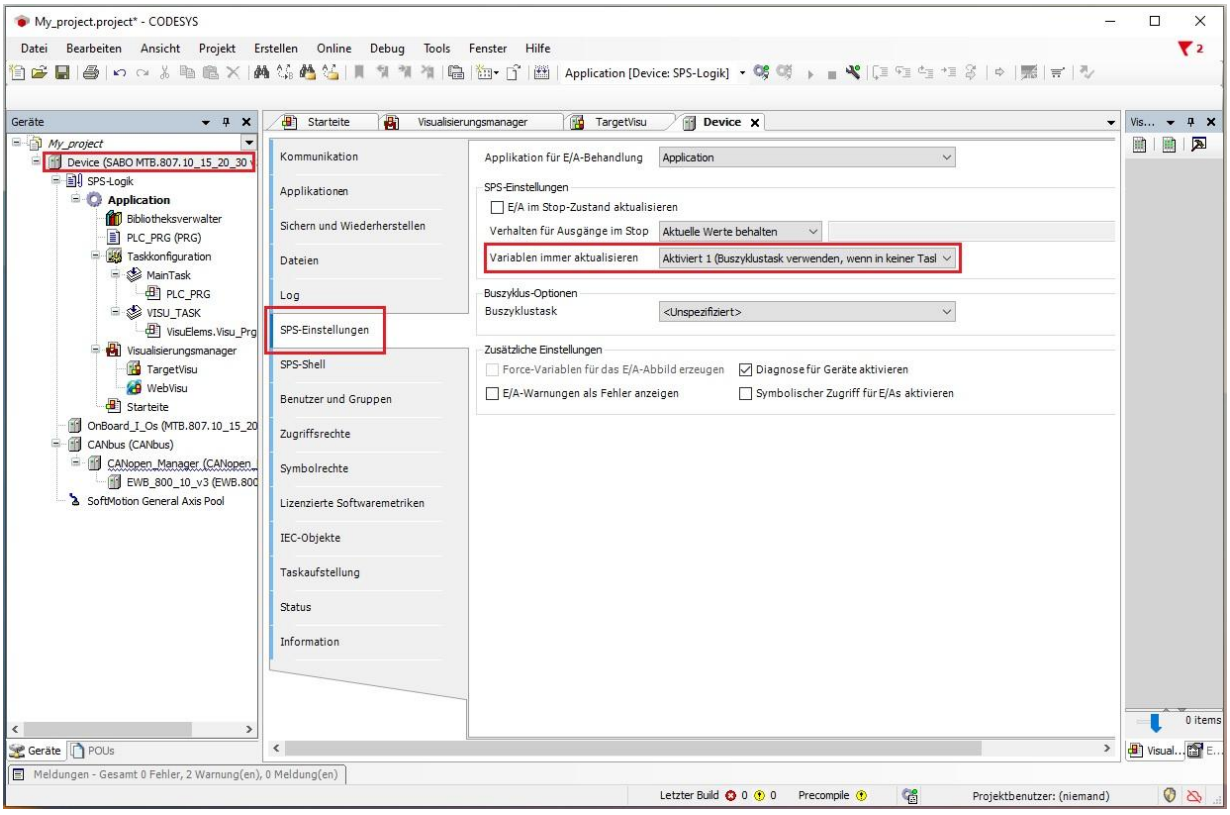


Abbildung 45: SPS-Einstellungen

Ein weiterer wichtiger Punkt ist das automatische Speichern des Projektes. CODESYS neigt dazu beim übersetzten des Projekts abzustürzen. Um zu verhindern, dass die Änderungen, die Sie vorgenommen haben, nicht verloren gehen, gibt es die Möglichkeit, das Projekt automatisch vor dem Übersetzten und alle x Minuten zu sichern. Zusätzlich kann eine Sicherungskopie des Projektes angelegt werden.

Die notwendigen Einstellungen dafür finden Sie in der obere Menüleiste unter **Tools** und den Menüpunkt **Optionen...** . *Abbildung 46*.

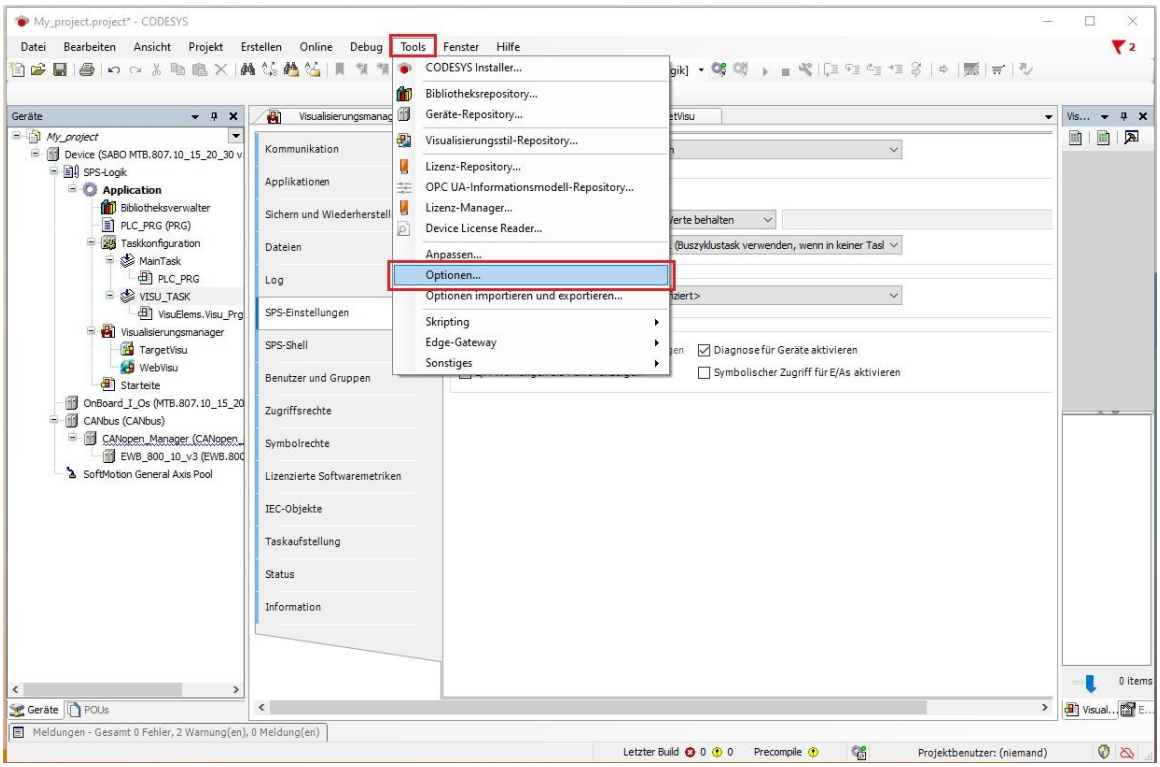


Abbildung 46: Optionen in Codesys

Wählen Sie aus der linken Spalte **Laden und speichern** aus uns setzen Sie die Häkchen, wie in der *Abbildung 47*. Die Zeit für das automatische Speichern können Sie frei wählen, wir haben uns für 5 Minuten entschieden.

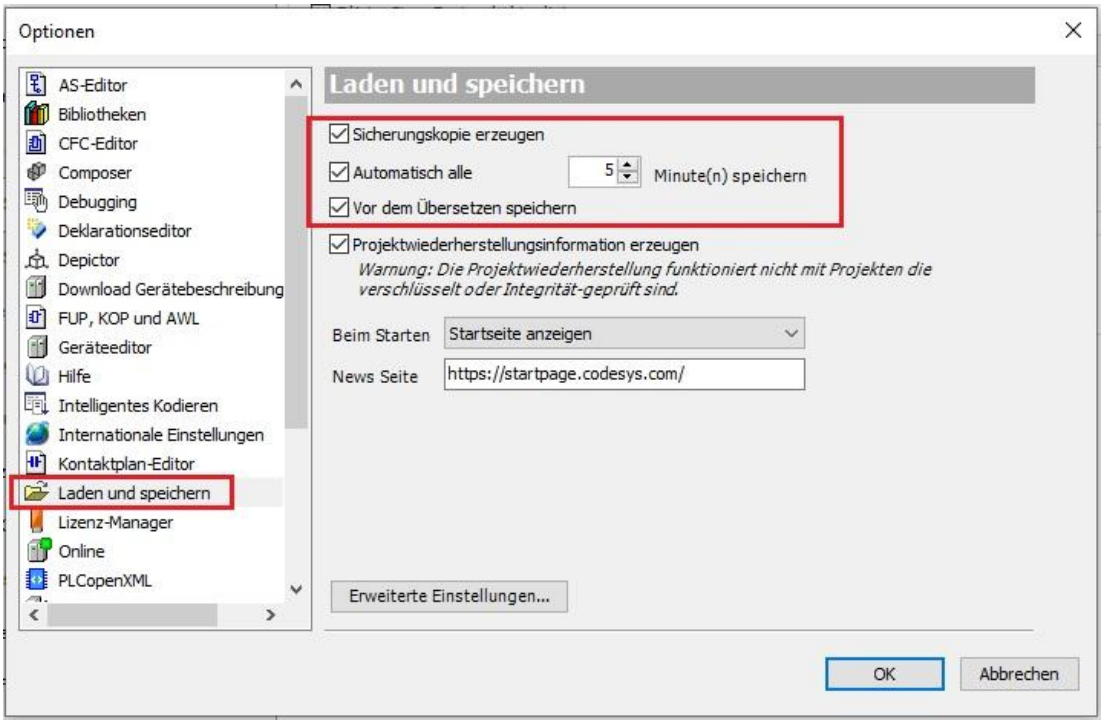


Abbildung 47: Einstellungen für Laden und Speichern

Damit sind alle wichtigen Einstellungen vorgenommen worden und Sie können mit einem neuen Projekt beginnen.

5 OPC-UA SERVER IN DER CODESYS V3.5 IDE ANLEGEN UND VARIABLEN HINZUFÜGEN

Das Protokoll OPC-UA (OPC Unified Architecture) ermöglicht einen geräteunabhängigen Datenaustausch und findet zunehmend Verbreitung in der Industrie. Die Administration erfolgt durch OPC Foundation. Weitere Informationen finden sich auf der Website der Organisation unter https://opcfoundation.org/.

Im Gegensatz zum Vorgängerprotokoll OPC beruht OPC-UA nicht auf dem proprietären COM/DCOM-Stack von Microsoft Windows, so dass die Implementierung auch auf Embedded-Systemen möglich ist.

OPC-UA wird zur Verbindung verschiedener Systemebenen eingesetzt, z.B. auf SPS-Steuerungen oder in übergeordneten Leitebenen. Die Implementierungen unterscheiden sich entsprechend im Leistungsumfang.

5.1 OPC-UA SERVER MITTELS KOMMUNIKATIONSVERWALTER IN DER CODESYS IDE ANLEGEN

Öffnen Sie die CODESYS V3.5 IDE und laden oder erstellen Sie ein CODESYS Projekt, dem Sie den OPC-UA Server hinzufügen möchten. Sobald das Projekt geladen wurde, führen Sie in der Programmierumgebung einen Rechtsklick auf **Application** aus und wählen dann **Objekt hinzufügen...** .In der angezeigten Liste klicken Sie auf **Kommunikationsverwalter**, um diesen hinzuzufügen. Siehe dazu *Abbildung 48*.

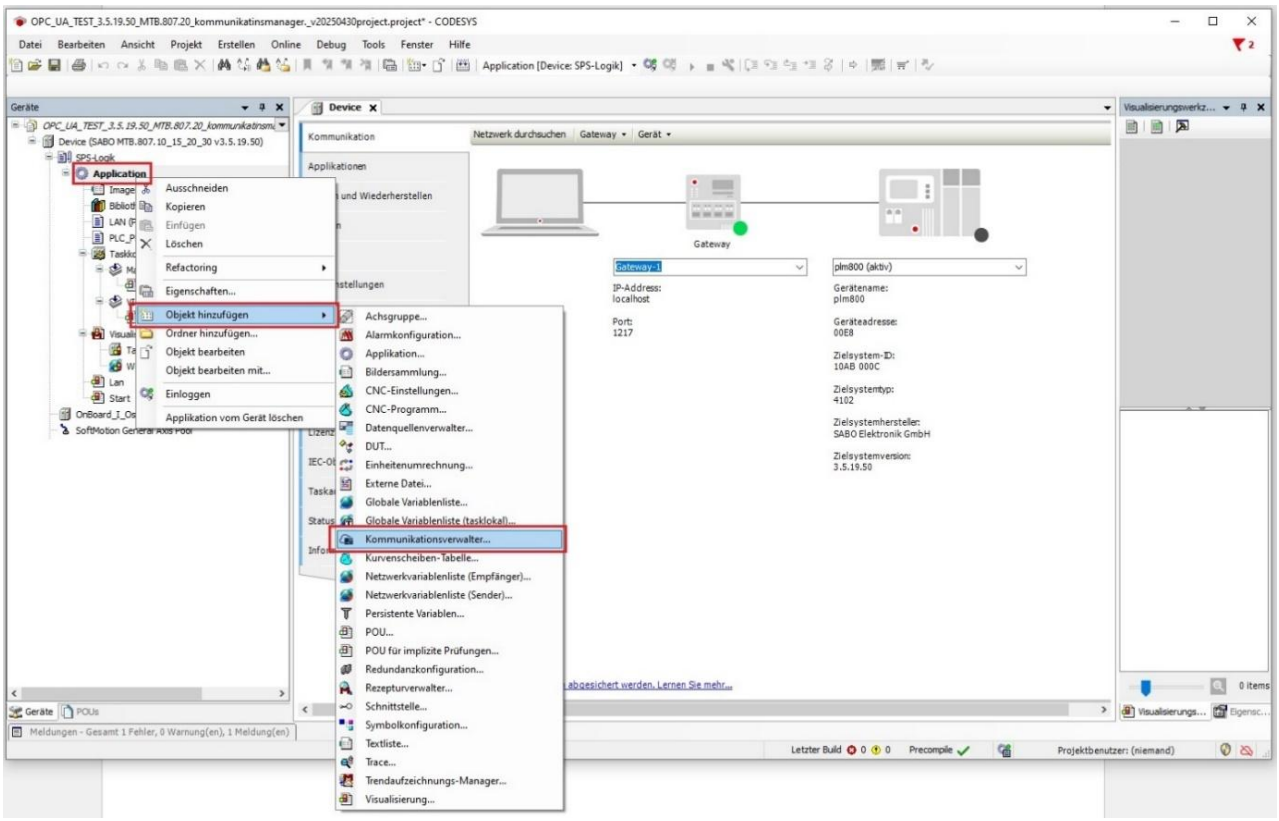


Abbildung 48: Kommunikationsverwalter hinzufügen

In dem neu angezeigten Fenster können Sie einen Namen für den Kommunikationsverwalter vergeben oder den angezeigten Namen übernehmen. Über die Schaltfläche **Hinzufügen** wird der

Kommunikationsverwalter Ihrem Projekt hinzugefügt und das Fenster wird geschlossen. *Abbildung 49.*

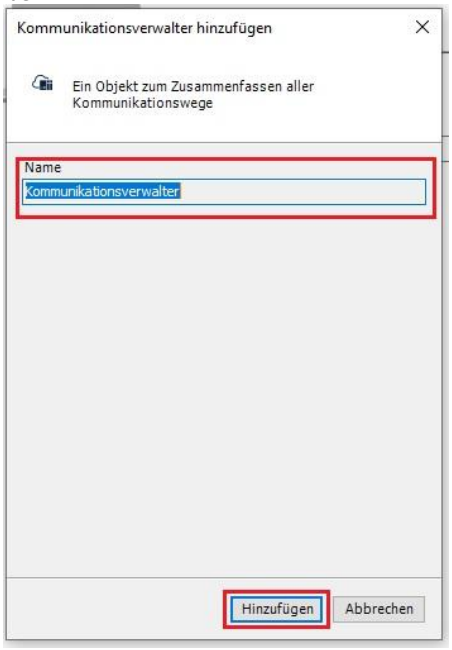


Abbildung 49: Kommunikationsverwalter benennen und hinzufügen

An den Kommunikationsverwalter muss der OPC-UA Server angehangen werden. Dazu führen Sie in der Programmierumgebung einen Rechtsklick auf den **Kommunikationsverwalter** aus, klicken auf **Objekt hinzufügen** und wählen **OPC UA Server...** . *Abbildung 50.*

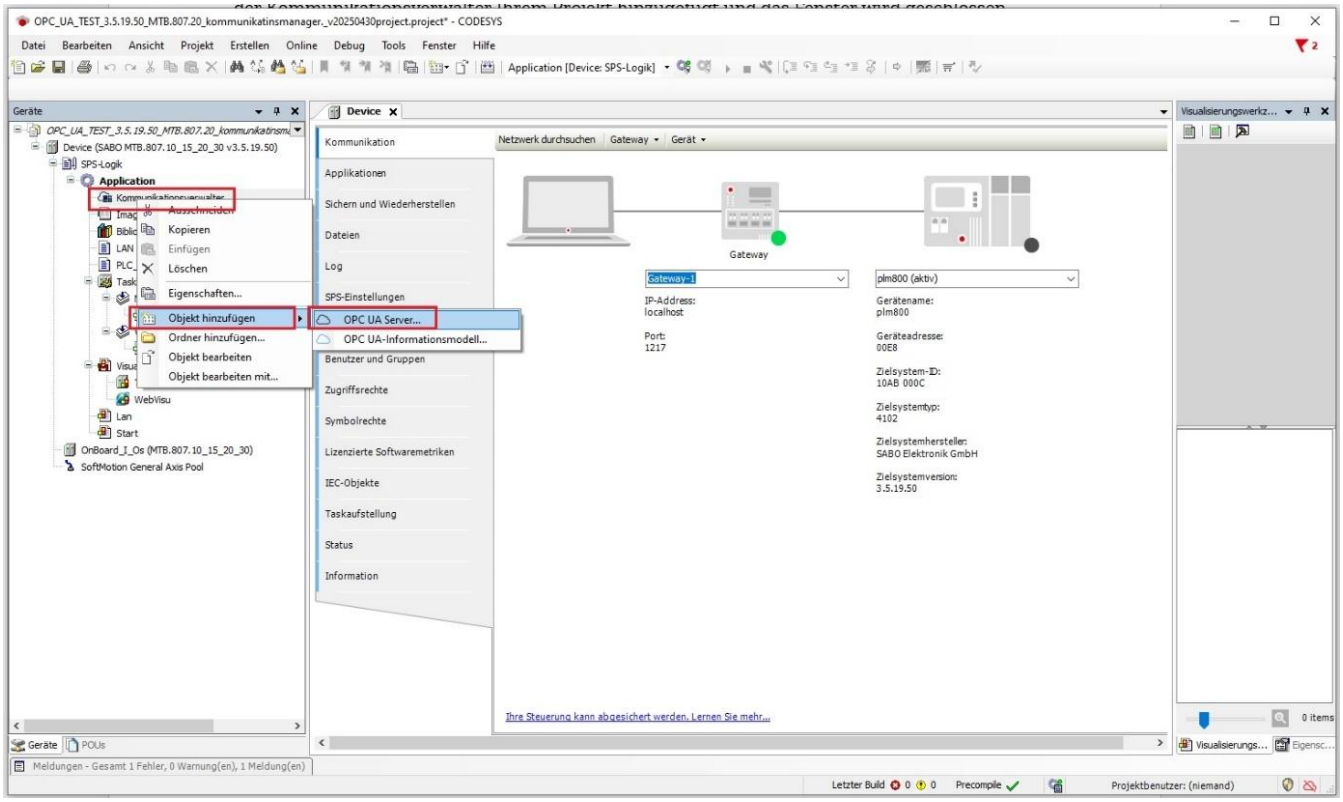


Abbildung 50: OPC-UA Server hinzufügen

Sie können den Namen für den OPC-UA Server frei vergeben oder Sie übernehmen den angezeigten Namen. Über die Schaltfläche **Hinzufügen** wird der OPC-UA Server dem Kommunikationsverwalter angehangen und das Fenster geschlossen. *Abbildung 51.*

Abbildung 51: OPC UA Server benennen und hinzufügen

Sobald der OPC-UA Server eingefügt wurde, wird automatisch zusätzlich an den OPC-UA Server eine **Symbolgruppe** angehangen. *Abbildung* 52.

Im nächsten Schritt müssen die Variablen dem OPC-UA Server übergeben und konfiguriert werden. Siehe Kapitel Variablen an den OPC UA Server übergeben und konfigurieren.

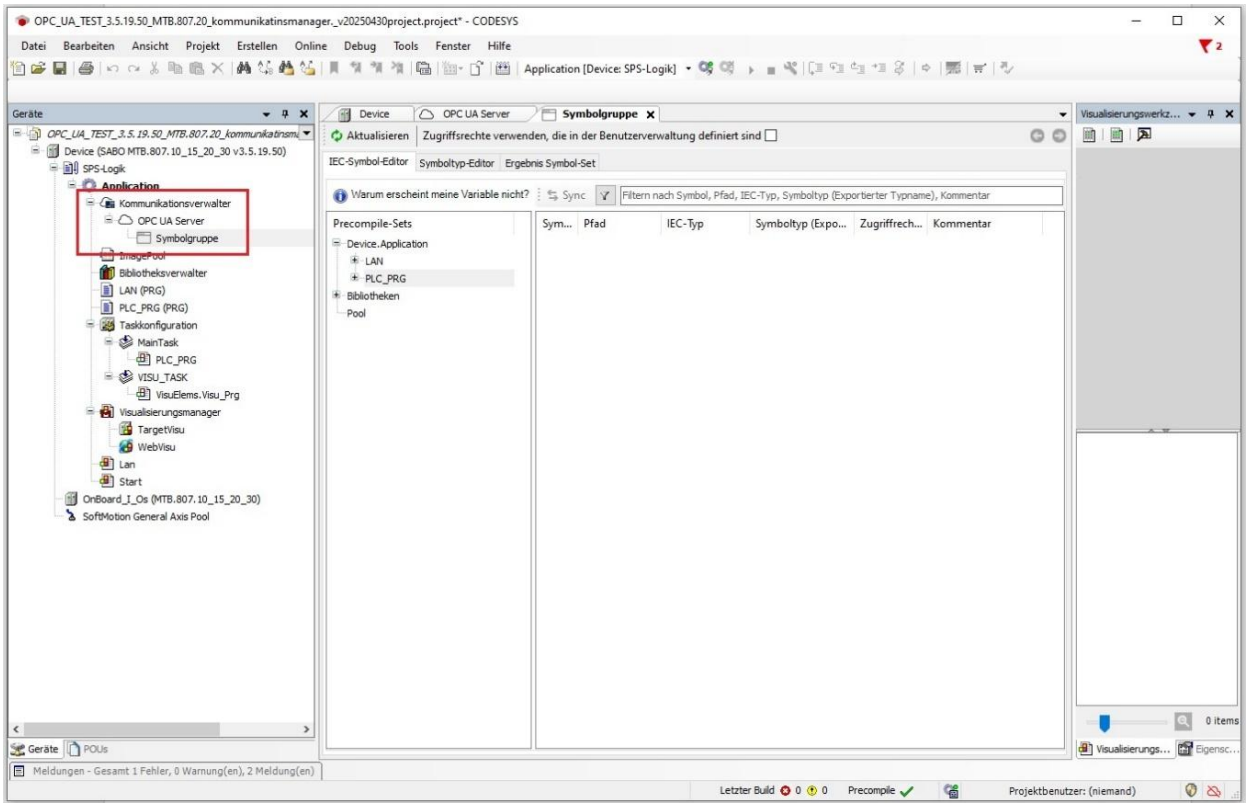


Abbildung 52: Eingebundener OPC UA Server

5.2 VARIABLEN AN DEN OPC UA SERVER ÜBERGEBEN UND KONFIGURIEREN

Ein OPC-UA Server stellt Variablen aus dem Projekt zur Verfügung, so dass ein OPC-UA Client darauf zugreifen kann. Welche Variablen zur Verfügung stehen sollen und wie man darauf zugreifen kann, schreibend, lesend oder beides, können Sie festlegen.

Die Symbolgruppe enthält sämtliche Variablen aus Ihrem Projekt, die der OPC-UA Server zur Verfügung stellen kann. Pointer, Array of Array, Reference, Interface und Properties sind nicht möglich.

Im IEC-Symbol-Editor werden Ihnen alle Variablen aus Ihren Programmbausteinen angezeigt. Variablen, die Sie dem OPC-UA Client zur Verfügung stellen möchten, können Sie ganz einfach per Drag and Drop in das Hauptfenster ziehen. *Abbildung* 53.

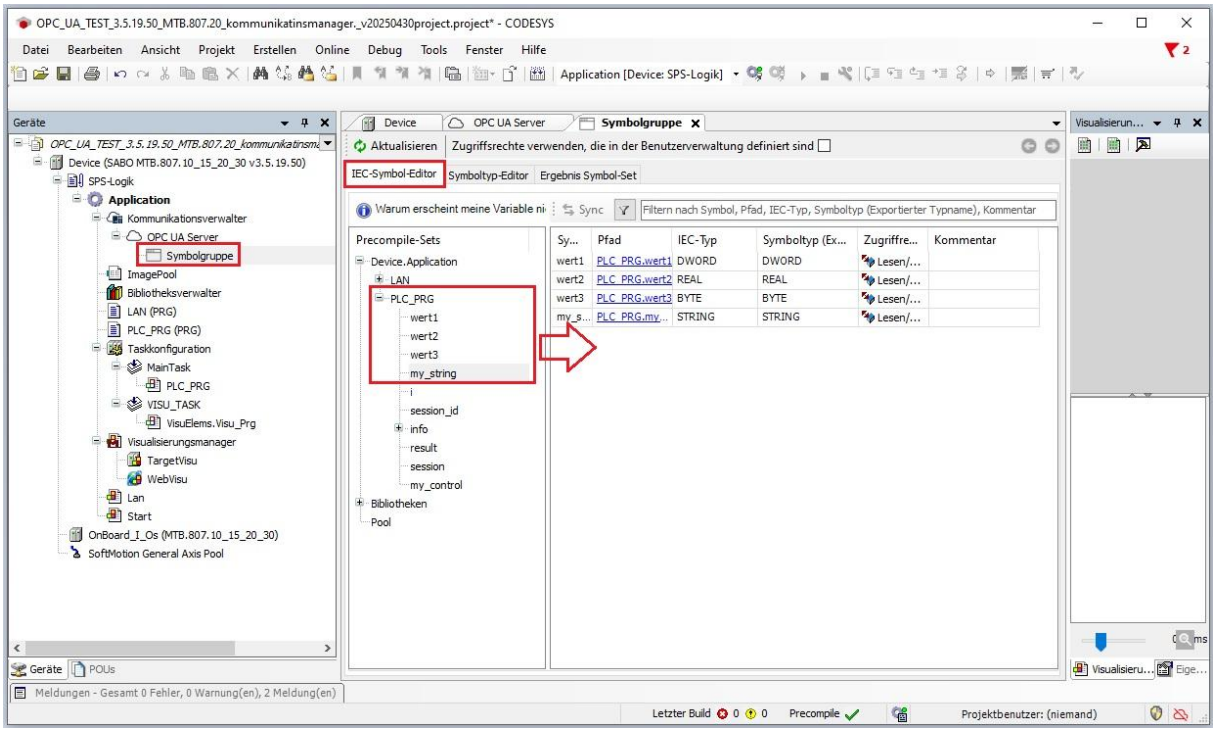


Abbildung 53: Variablen dem OPC UA Server zur Verfügung stellen

Nachdem Sie alle benötigten Variablen, die Sie dem OPC-UA Client zur Verfügung stellen wollen, eingefügt haben, müssen Sie noch festlegen, ob diese Variable nur gelesen, nur geschrieben oder ob beides möglich sein soll.

Das können Sie über ein Dropdownmenü zu jeder Variable einzeln festlegen. *Siehe Abbildung* 54.

Sie können auch ein Kommentar zu der Variable angeben, die der OPC-UA Client ebenfalls anzeigen kann.

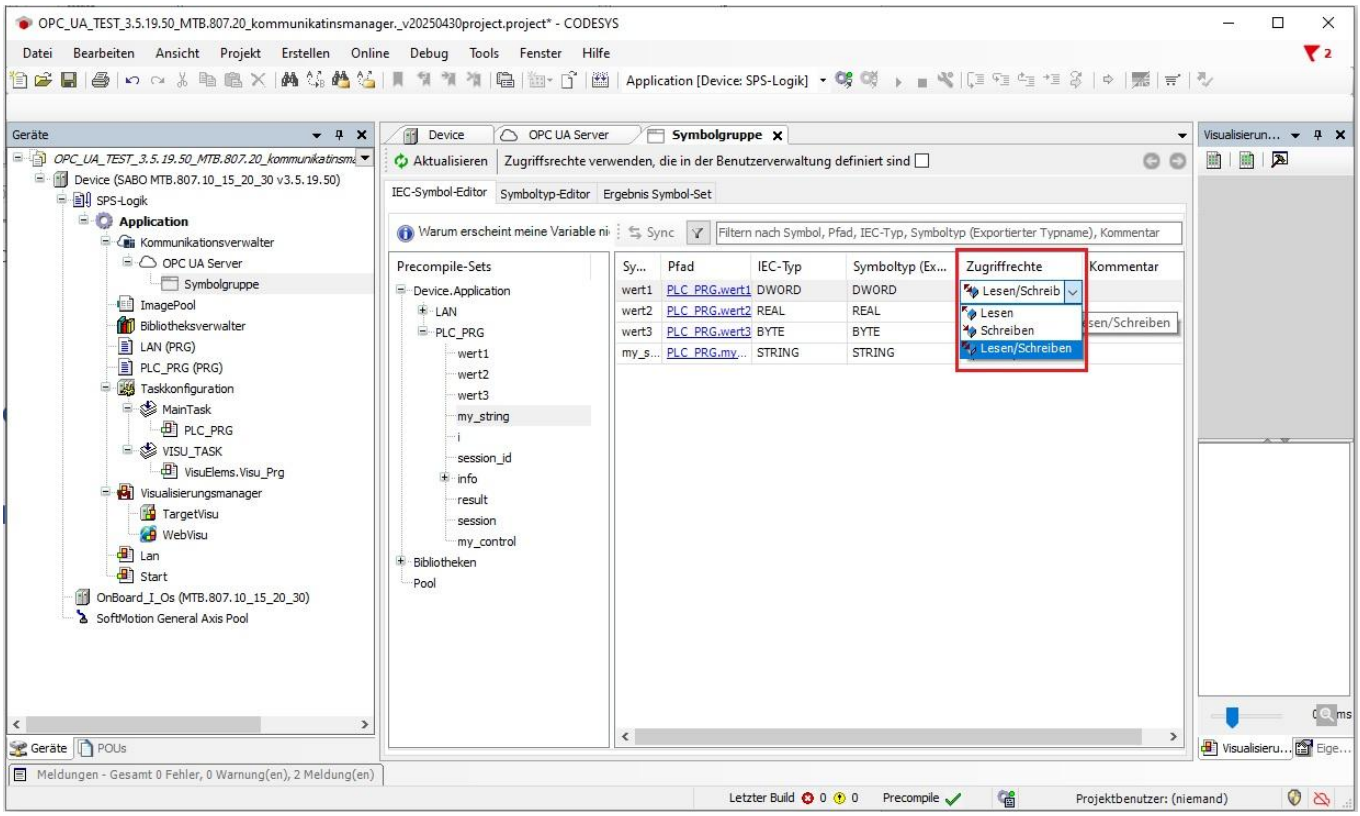


Abbildung 54: Konfiguration der Variablen

Somit ist die Konfiguration der Variablen für den OPC-UA Server abgeschlossen.

Beim nächsten Download des Projektes in die Steuerung wird die Symbolik auf die Steuerung geladen und kann von einem OPC-UA Client angezeigt werden.

5.3 ZERTIFIKATE FÜR OPC-UA UND FÜR VERSCHLÜSSELTE KOMMUNIKATION ERSTELLEN

Für die verschlüsselte Kommunikation zwischen OPC-UA Server und OPC-UA Client werden Zertifikate benötigt, die auf der Steuerung vorhanden sein müssen. Dabei handelt es sich um ein Zertifikat für OPC-UA Server und ein Zertifikat für verschlüsselte Kommunikation.

Diese Zertifikate können von der Steuerung erstellt werden, falls diese noch nicht auf der Steuerung installiert sind.

Um Zertifikate zu erstellen oder sich die vorhandenen Zertifikate auf der Steuerung anzeigen zu lassen, müssen Sie mit der Steuerung über Ethernet verbunden sein und sich auf die Steuerung eingeloggt haben.

Sobald Sie mit der Steuerung verbunden sind, können Sie in der Programmierumgebung über das **grau/gelbe Schild Symbol**, das sich unten rechts befindet, mit einem Doppelklick den **Security Screen** öffnen. *Abbildung 55*. Alternativ kann das Fenster über **Ansicht -> Security Screen** angezeigt werden.

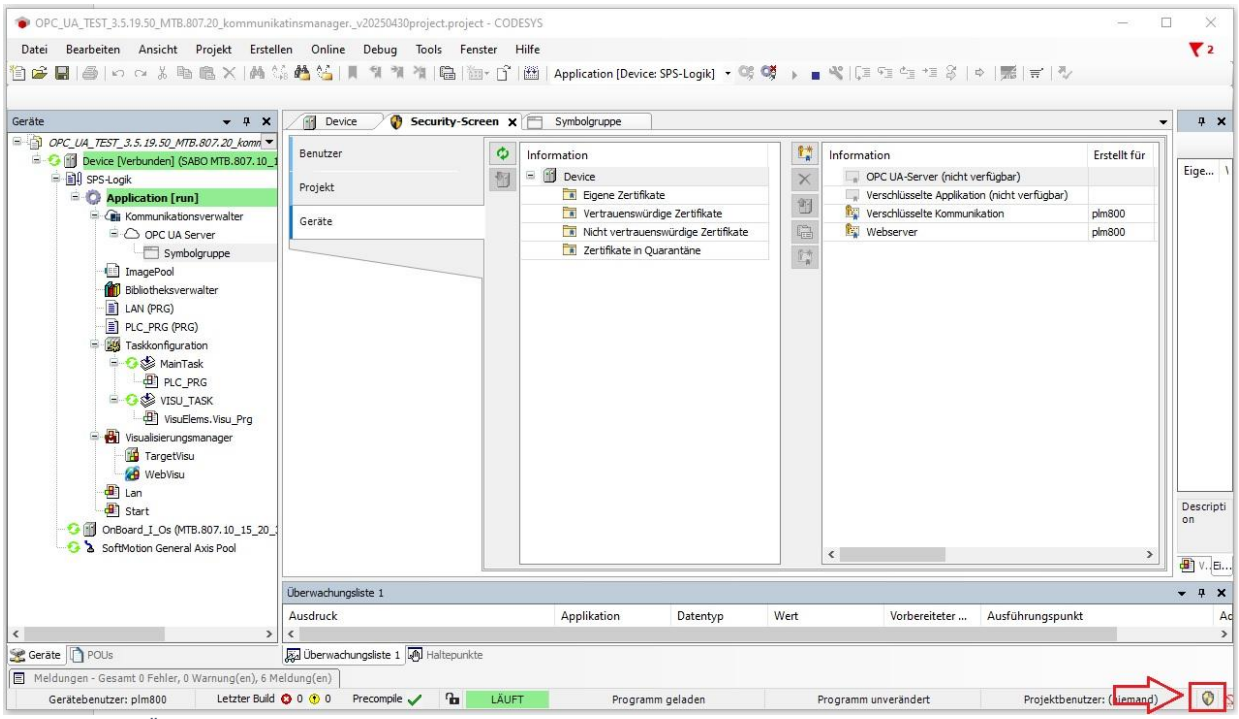


Abbildung 55: Übersicht über die installierten Zertifikate

In diesem Fenster können Sie sich die aktuellen Zertifikate auf der Steuerung anzeigen lassen oder auch neue Zertifikate erstellen.

Um ein neues OPC-UA Server Zertifikat zu erstellen, gehen Sie bitte wie folgt vor:

- Klicken Sie in dem **Security Screen** auf **Geräte**
- Klicken Sie auf **Device**
- Wählen Sie **OPC UA Server aus**
- Klicken Sie auf **Zertifikat erstellen**

Siehe *Abbildung 56*.

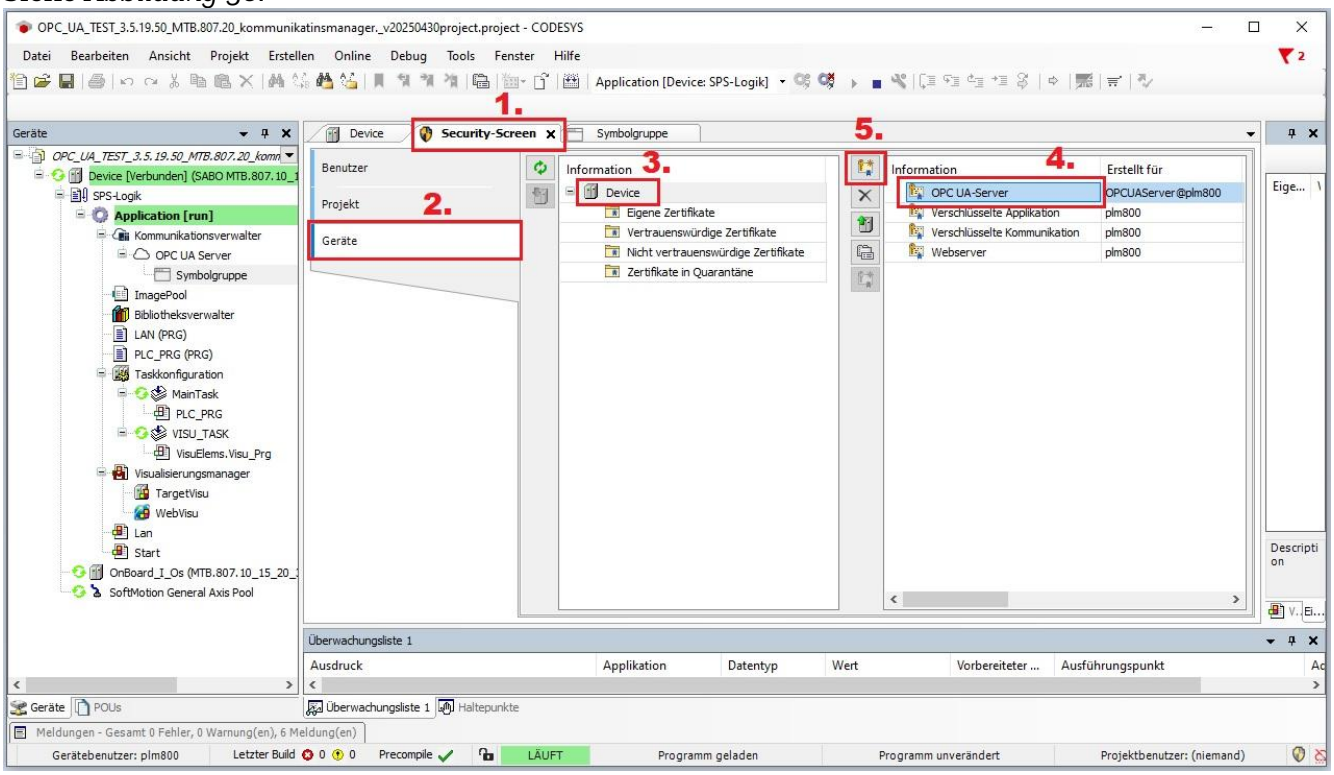


Abbildung 56: OPC UA Server Zertifikat erstellen

Wenn Sie auf **Zertifikat erstellen** geklickt haben, dann werden Sie gefragt, wie lange das Zertifikat gültig sein soll. Die Gültigkeit wird in Tagen angegeben. In unserem Beispiel haben wir das Zertifikat für 3650 Tage (10 Jahre) ausgestellt.

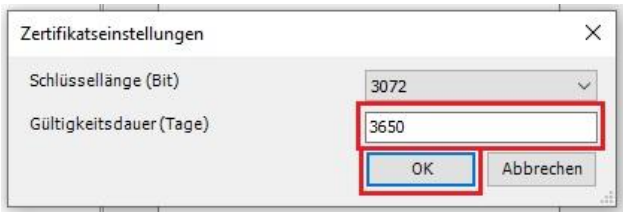


Abbildung 57: Gültigkeitsdauer des OPC UA Server Zertifikats

Das Erstellen des Zertifikats kann einige Minuten in Anspruch nehmen, da die Steuerung das Zertifikat neu berechnen muss. Sobald das Zertifikat fertig erstellt wurde, wird es Ihnen in dem **Security Screen** unter **Device** angezeigt. *Abbildung 58*.

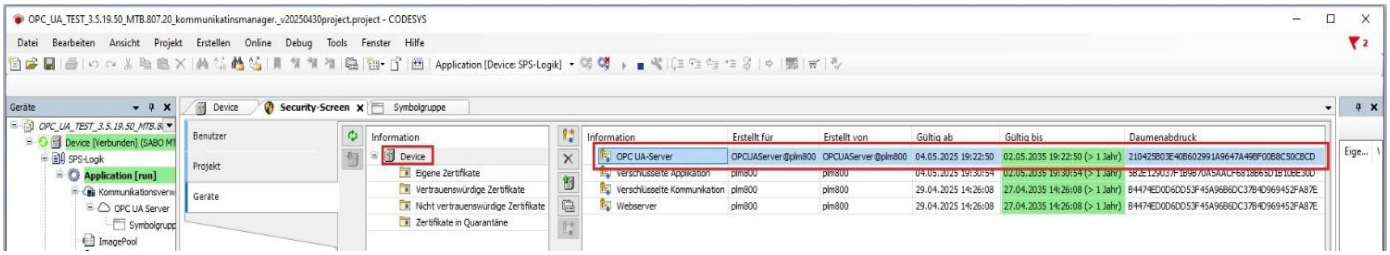


Abbildung 58: Fertig erstelltes OPC UA Server Zertifikat

Zusätzlich zu dem Zertifikat für OPC-UA Server wird das Zertifikat für verschlüsselte Kommunikation benötig. Das Erstellen dieses Zertifikats erfolgt in gleichen Schritten, wie für das OPC-UA Zertifikat.

Um ein neues Zertifikat für verschlüsselte Kommunikation zu erstellen, gehen Sie bitte wie folgt vor:

- Klicken Sie in dem **Security Screen** auf **Geräte**
- Klicken Sie auf **Device**
- Wählen Sie **Verschlüsselte Kommunikation** aus
- Klicken Sie auf **Zertifikat erstellen**

Siehe dazu *Abbildung 59*.

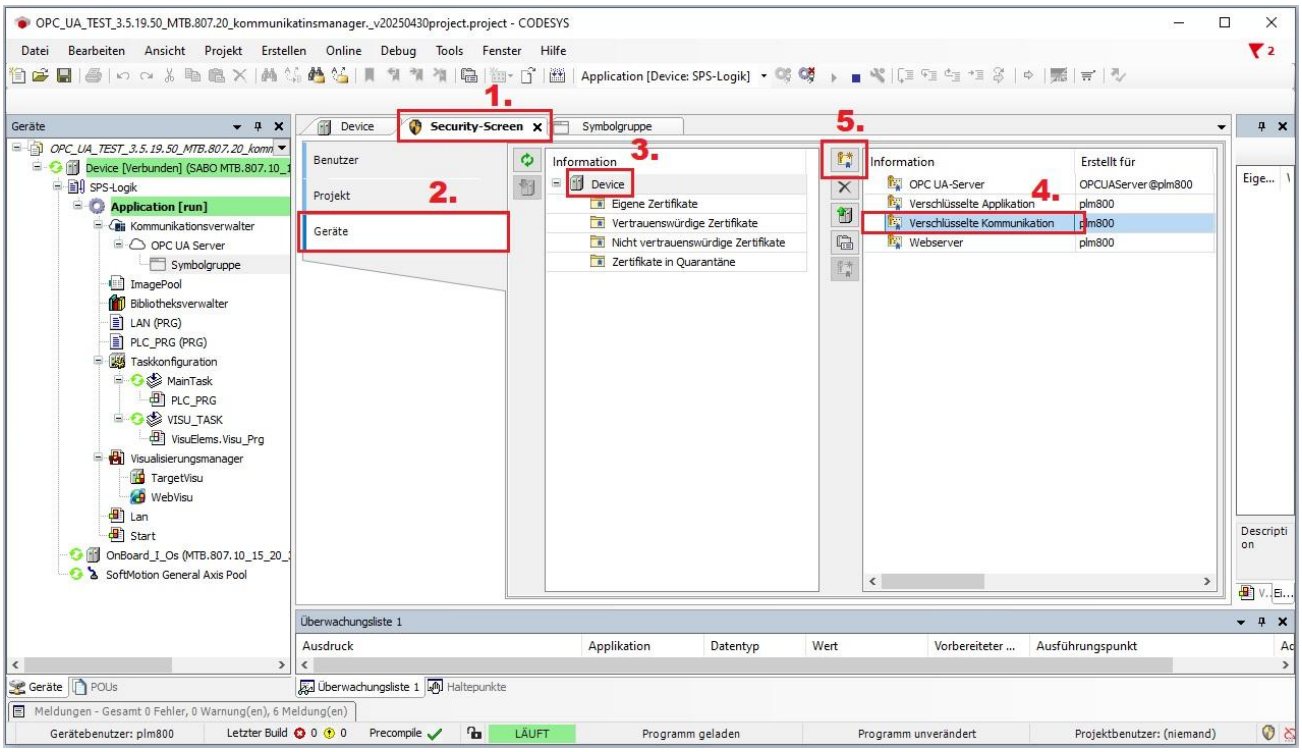


Abbildung 59: Zertifikat für verschlüsselte Kommunikation erstellen

Bei der Dauer der Gültigkeit können Sie in Tagen angeben, wie lange das Zertifikat gültig sein soll. In unserem Beispiel werden 3650 Tage (10 Jahre) angegeben.

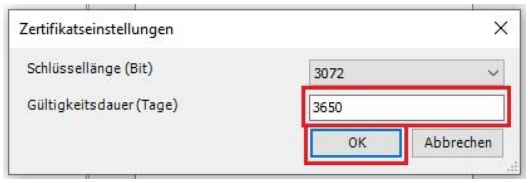


Abbildung 60: Dauer des Zertifikats festlegen

Das Erstellen des Zertifikats kann einige Zeit dauern, da das Zertifikat neu berechnet werden muss. Nachdem das Zertifikat erstellt wurde, wird es Ihnen im **Security Screen** angezeigt.

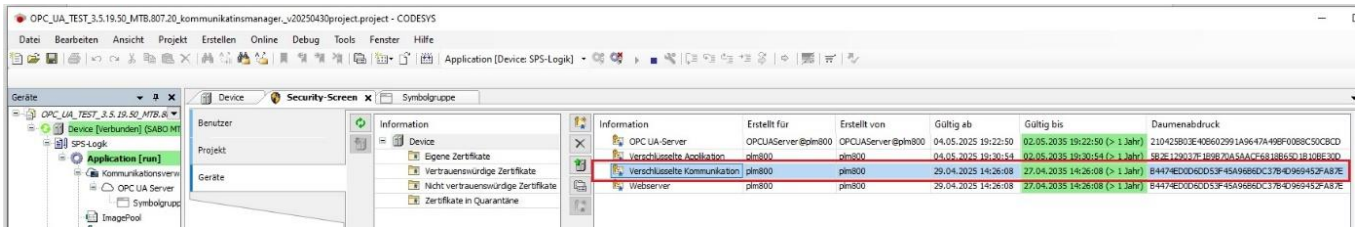


Abbildung 61: Fertig erstelltes Zertifikat für verschlüsselte Kommunikation

Damit sind die benötigten Zertifikate der Steuerung hinzugefügt worden.
Falls eine anonyme Verbindung genutzt werden soll, folgen Sie der Beschreibung in Kapitel 4.6.

Die Anmeldung mit Benutzernamen und Passwort wird im Kapitel 4.4 beschrieben.

5.4 BENUTZER DEM OPC-UA SERVER HINZUFÜGEN

Die Sicherheitseinstellungen aus den vorherigen Kapiteln legen fest, dass der Benutzer sich mit einem Benutzernamen und Passwort am OPC-UA Server anmelden muss, sobald der OPC-UA Client eine Verbindung zum OPC-UA Server herstellt.

Benutzer können über die CODESYS IDE angelegt und verwaltet werden. Über **Device** und den Reiter **Benutzer und Gruppen** können Sie neue Benutzer für den OPC-UA Server anlegen, bearbeiten oder löschen.

Um einen neuen Benutzer anlegen zu können, müssen Sie mit der Steuerung verbunden sein. Wenn Sie verbunden sind, klicken Sie auf den Reiter **Benutzer und Gruppen** und anschließend auf das **Aktualisierungssymbol**, damit die vorhandenen Benutzer angezeigt werden.

Wurden die Benutzer und Gruppen geladen, können Sie irgendwo auf die leere Fläche in dem Feld für **Benutzer** klicken oder Sie wählen einen **vorhandenen Benutzer** aus, um über die Schaltfläche **Hinzufügen** einen neuen Benutzer zum OPC-UA Server hinzuzufügen. Siehe *Abbildung 62*.

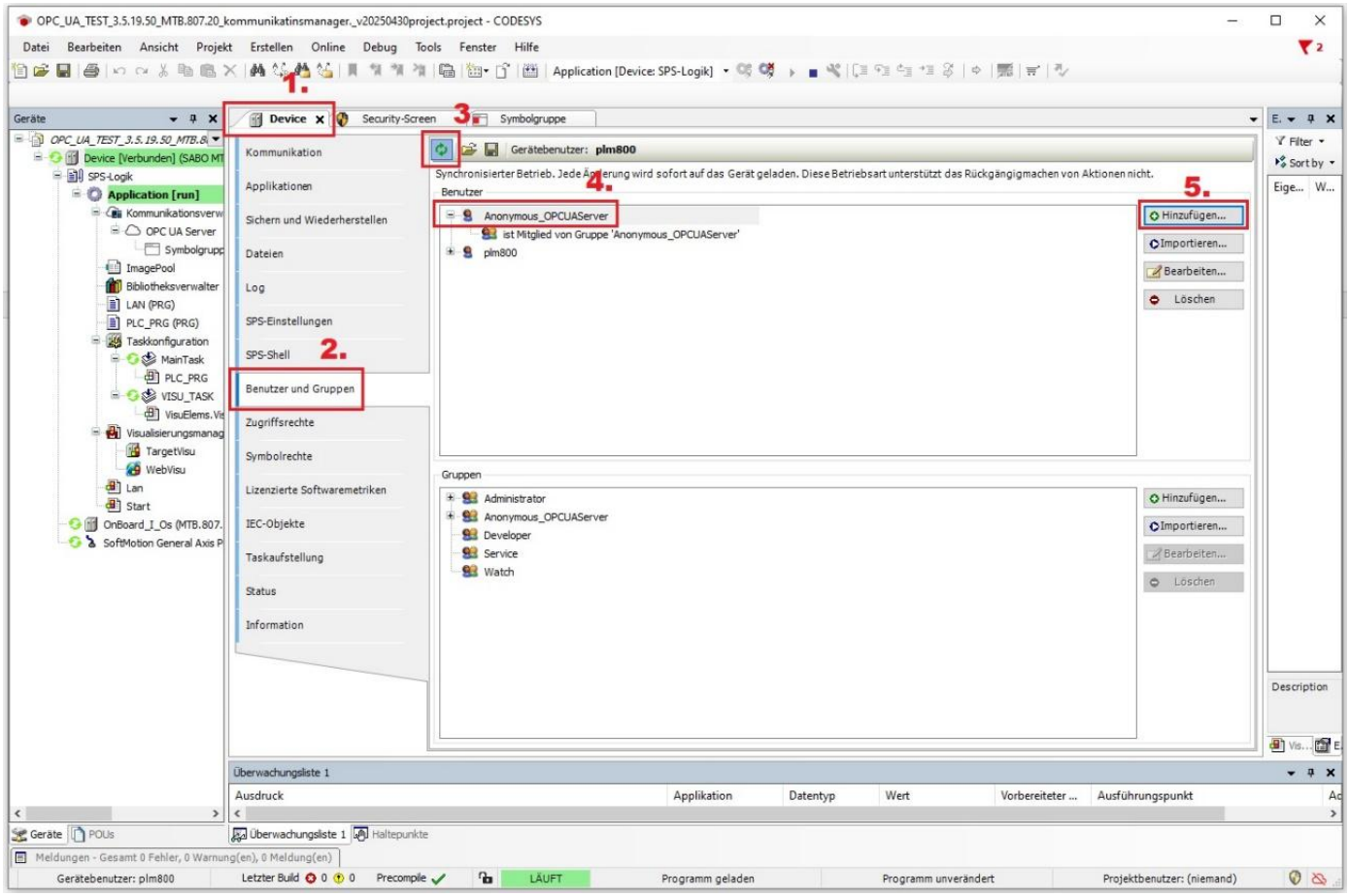


Abbildung 62: Benutzer dem OPC UA Server hinzufügen

In dem neuen Fenster kann nun ein Benutzer mit einem Benutzernamen und Passwort erstellt werden.

Geben Sie einen Namen für den Benutzer ein und fügen Sie ihn der Standardgruppe **Anonymous_OPCUAServer** hinzu.

Legen Sie anschließend ein Passwort für den Benutzer fest. *Abbildung 63*.

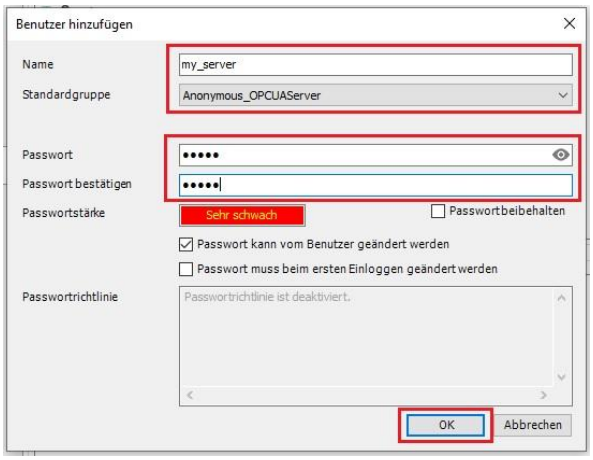


Abbildung 63: Benutzernamen und Passwort festlegen

Mit **OK** wird der Benutzer mit Passwort angelegt und das Fenster wird geschlossen. Anschließend wird der neue Benutzer in dem Feld für **Benutzer** und dem Feld **Gruppen** angezeigt.

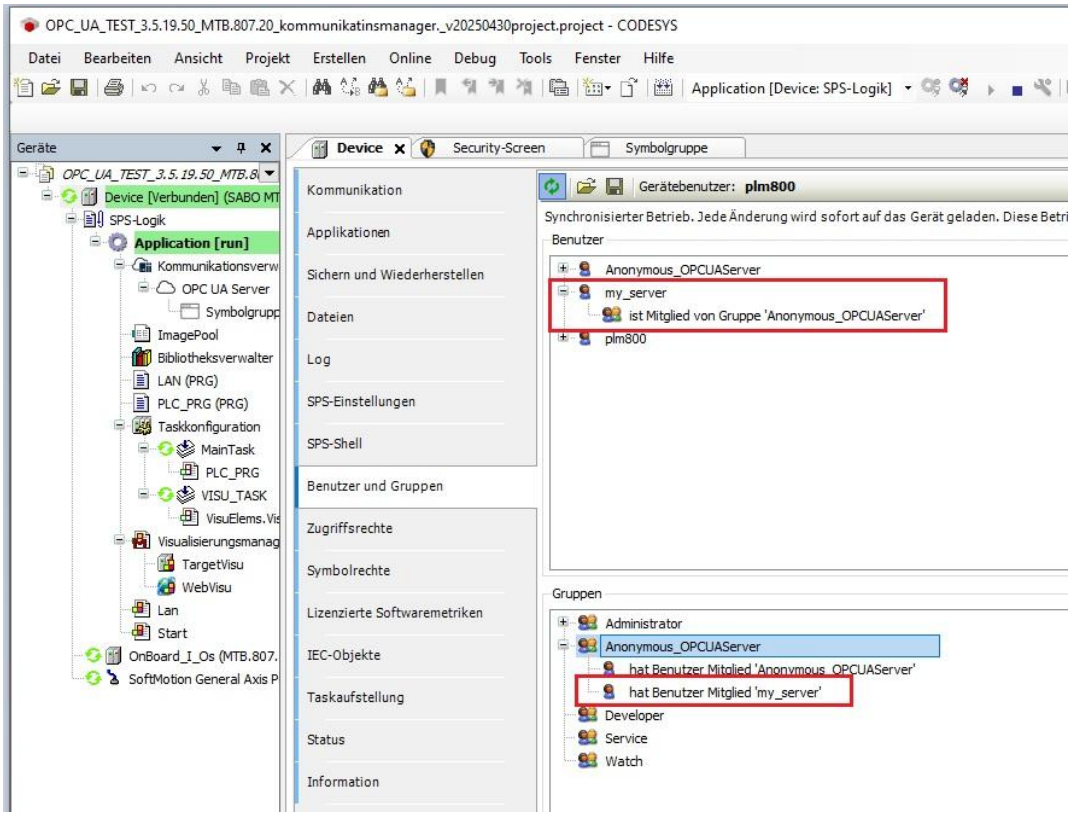


Abbildung 64: Der neu angelegt Benutzer wird in "Benutzer" und "Gruppen" angezeigt

Laden Sie abschließend das Projekt in die Steuerung, erzeugen ein Bootprojekt und starten Sie die Steuerung neu, damit die Änderungen wirksam werden.

5.5 SICHERHEITSEINSTELLUNGEN FÜR VERSCHLÜSSELUNG UND ANMELDUNG AUF DER STEUERUNG KONFIGURIEREN

Ab der CODESYS Version 3.5.17.xx wird eine verschlüsselte Kommunikation zwischen OPC-UA Server und OPC-UA Client empfohlen. Für die Umsetzung sind einige Schritte in der CODESYS IDE notwendig, da sonst keine Kommunikation zwischen Server und Client möglich ist.

Die nötigen Sicherheitseinstellungen werden in der IDE vorgenommen, dafür müssen Sie nicht auf der Steuerung eingeloggt sein.

Klicken Sie in der Programmierumgebung auf der linken Seite auf das **Device** und anschließend auf **Kommunikation**. Hier wählen Sie in dem Dropdownmenü für **Gerät** den Punkt **Sicherheitseinstellungen** aus. *Abbildung 65*.

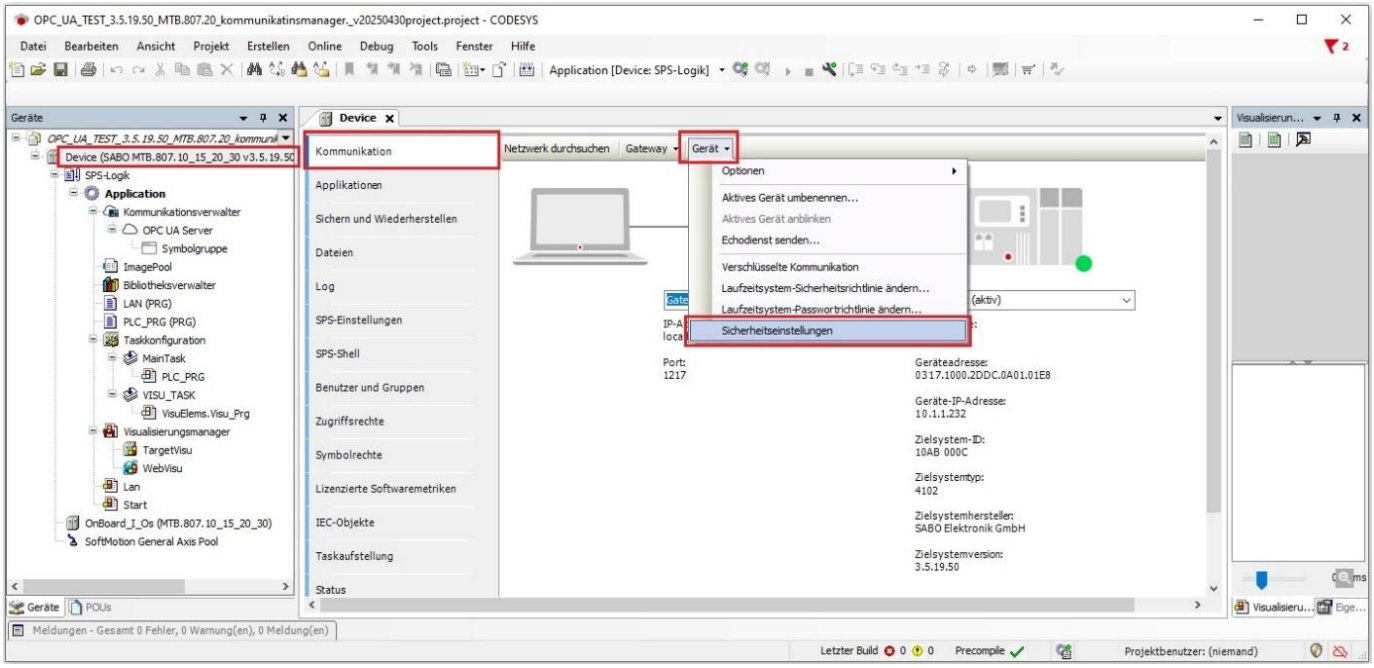


Abbildung 65: Sicherheitseinstellungen

Daraufhin öffnet sich das Fenster für Sicherheitseinstellungen, in dem Sie die Parameter für OPC-UA Server konfigurieren können. *Abbildung 66*.

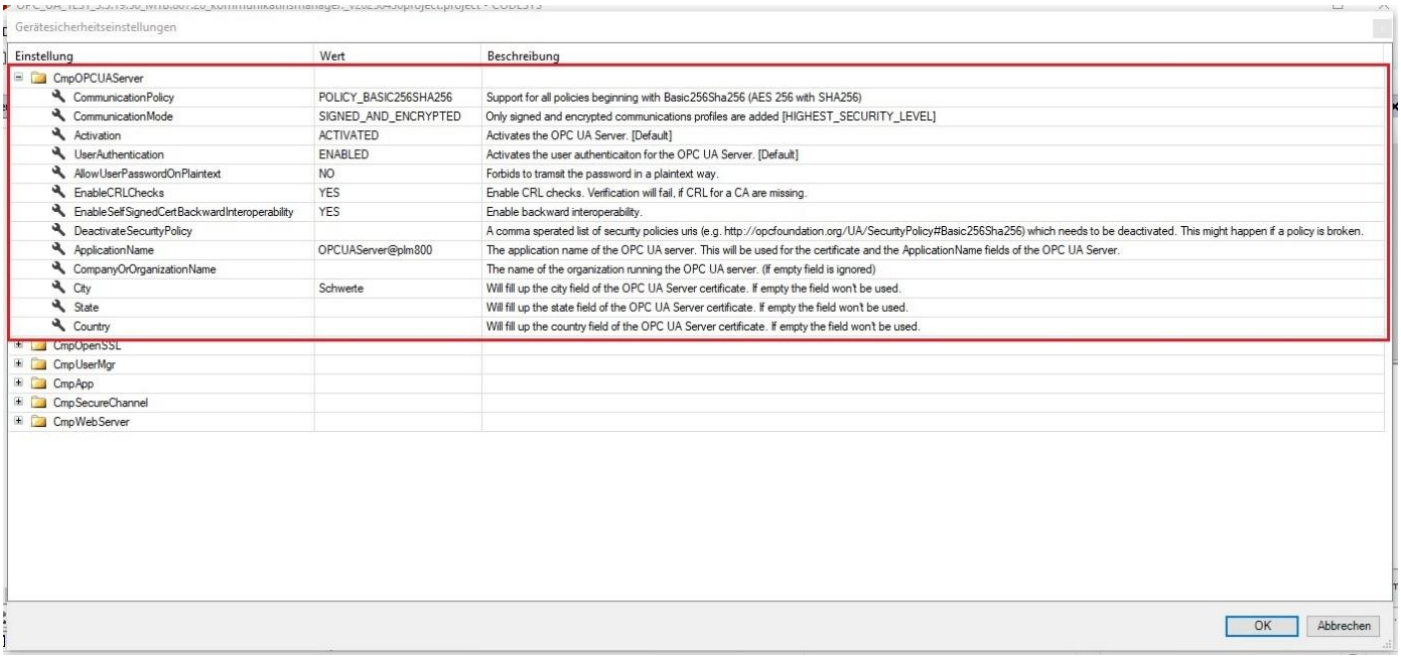


Abbildung 66: Sicherheitseinstellungen für OPC-UA Server

Die angezeigten Parameter müssen wie folgt konfiguriert werden:

CommunicationPolicy	POLICY_BASIC256SHA256
CommunicationMode	SIGNED_AND_ENCRYPTED

Activation	ACTIVATED
UserAuthentication	ENABLED
AllowUserPasswordOnPlaintext	NO
EnableCRLChecks	YES
EnableSelfSignedCertBackwardInteroperability	YES
DeactivateSecurityPolicy	
ApplicationName	OPCUAServer@plm800
CompanyOrOrganisationName	
City	Ihre Stadt
State	
Country	

Tabelle 2: Parameter der Sicherheitseinstellungen für OPC-UA Server

Mit der Schaltfläche **OK** werden die Parameter übernommen und das Fenster wird geschlossen. Damit sind die nötigen Sicherheitseinstellungen vorgenommen worden.

5.6 SICHERHEITSEINSTELLUNGEN FÜR ANONYME VERBINDUNG AUF DER STEUERUNG KONFIGURIEREN

Für eine anonyme Verbindung zwischen OPC-UA Client und OPC-UA Server müssen die Sicherheitseinstellungen auf der Steuerung angepasst werden.

Klicken Sie in der Programmierumgebung auf der linken Seite auf das **Device** und anschließend auf **Kommunikation**. Hier wählen Sie in dem Dropdownmenü für **Gerät** den Punkt **Sicherheitseinstellungen** aus. *Abbildung 67.*

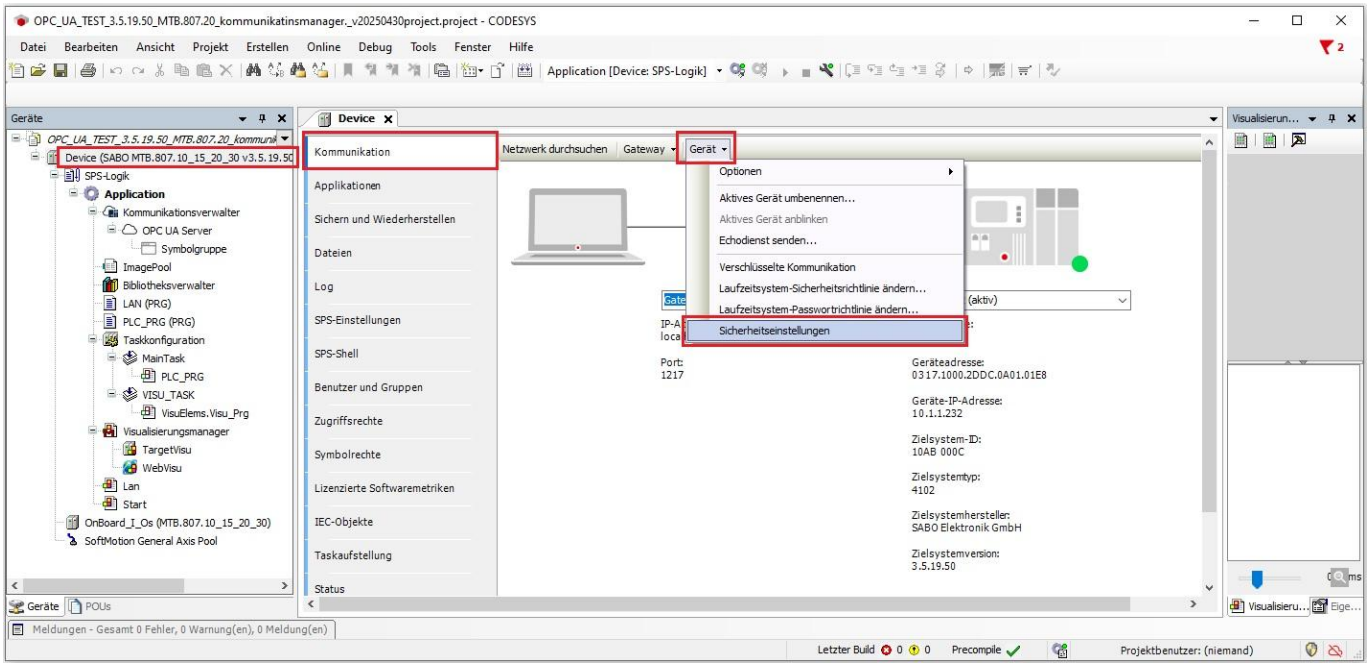


Abbildung 67: Sicherheitseinstellungen aufrufen

In dem Fenster der Gerätesicherheitseinstellungen werden in dem Ordner **CmpOPCUAServer** die Einstellungen für den OPC-UA Server verwaltet. Stellen Sie die Parameter für eine anonyme Verbindung wie in der *Abbildung 68* und *Tabelle 3* gezeigt ein.

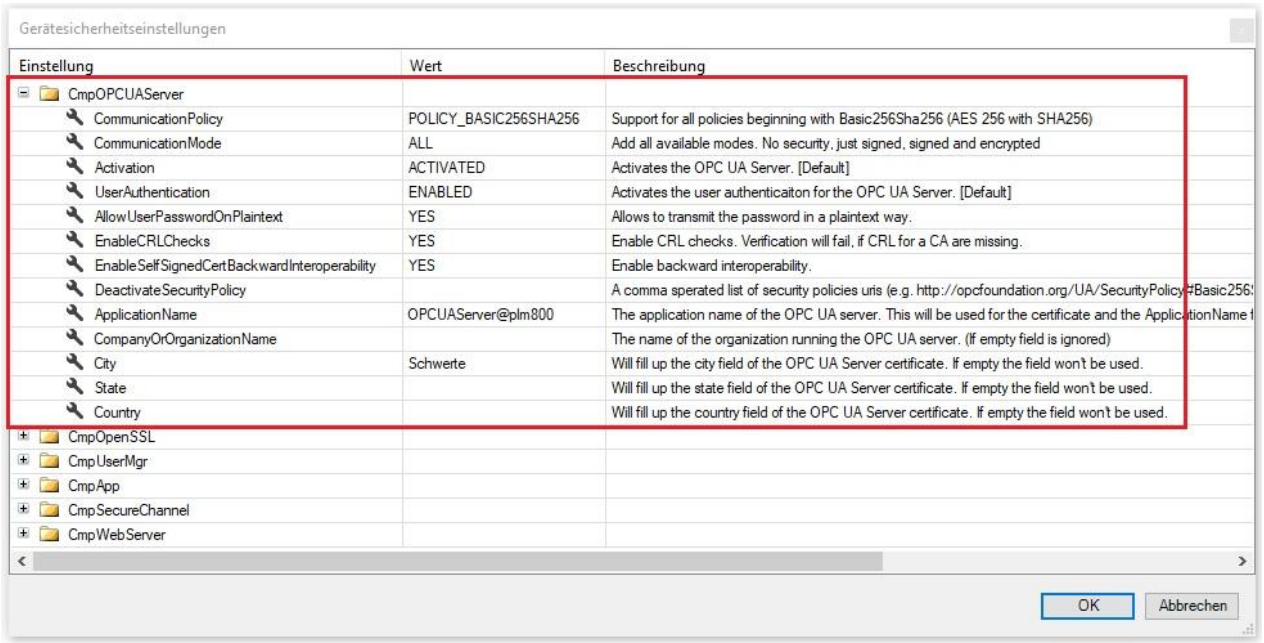


Abbildung 68: Parameter für anonyme Verbindung

Folgende Parameter für anonyme Verbindung übernehmen:

CommunicationPolicy	POLICY_BASIC256SHA256
CommunicationMode	ALL
Activation	ACTIVATED
UserAuthentication	ENABLED
AllowUserPasswordOnPlaintext	YES
EnableCRLChecks	YES
EnableSelfSignedCertBackwardInteroperability	YES
DeactivateSecurityPolicy	
ApplicationName	OPCUAServer@plm800
CompanyOrOrganisationName	
City	Ihre Stadt
State	
Country	

Tabelle 3: Parameter für anonyme Verbindung

Mit der Schaltfläche **OK** werden die Parameter übernommen und das Fenster wird geschlossen.

Das anonyme Einloggen muss zudem in der Steuerung aktiviert werden. Dafür ist eine Anpassung der Laufzeitsystem-Sicherheitsrichtlinie erforderlich.

Öffnen Sie in der Programmierumgebung das **Device** und klicken Sie dann auf den Reiter **Kommunikation**. In der Menüleiste wählen Sie **Gerät** an und klicken im Dropdown-Menü auf **Laufzeitsystem-Sicherheitsrichtlinie ändern...**, *Abbildung 69*.

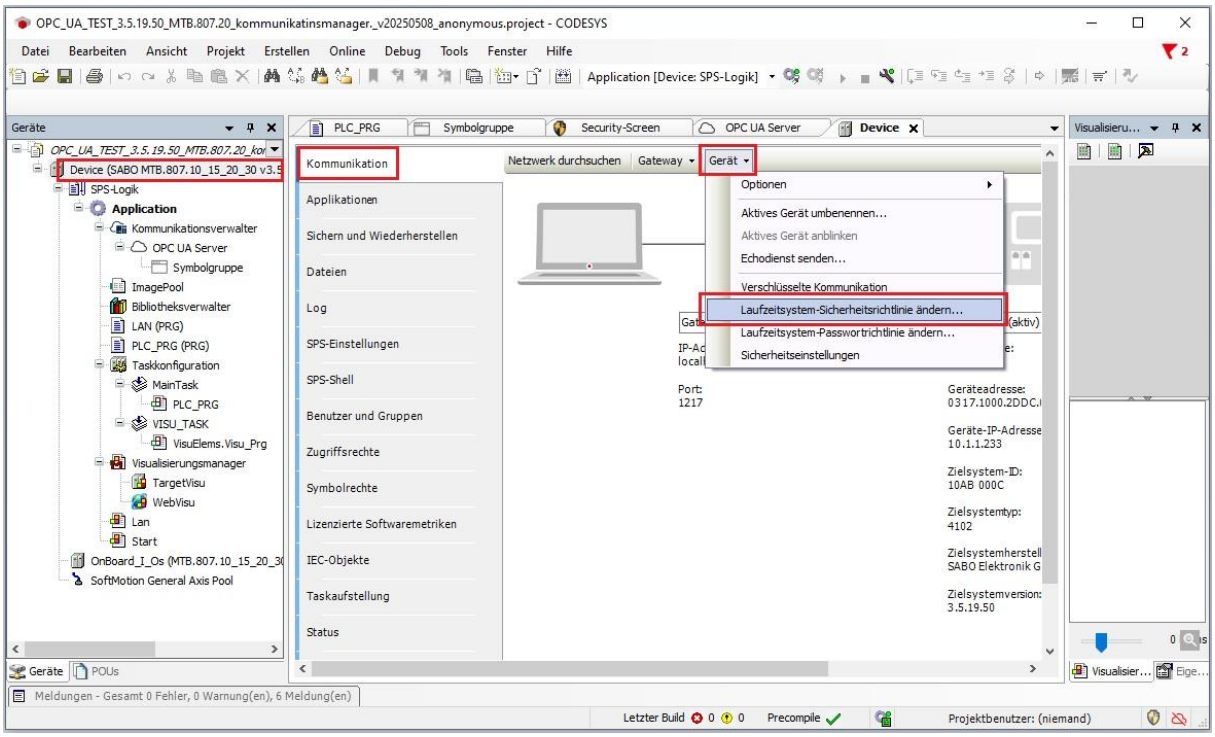


Abbildung 69: Laufzeitsystem-Sicherheitsrichtlinie ändern

In dem Fenster für die Sicherheitsrichtlinie des Laufzeitsystem muss das Häkchen bei **Anonymes Einloggen erlauben** gesetzt werden. Mit **OK** wird das Fenster geschlossen und die Einstellung übernommen.

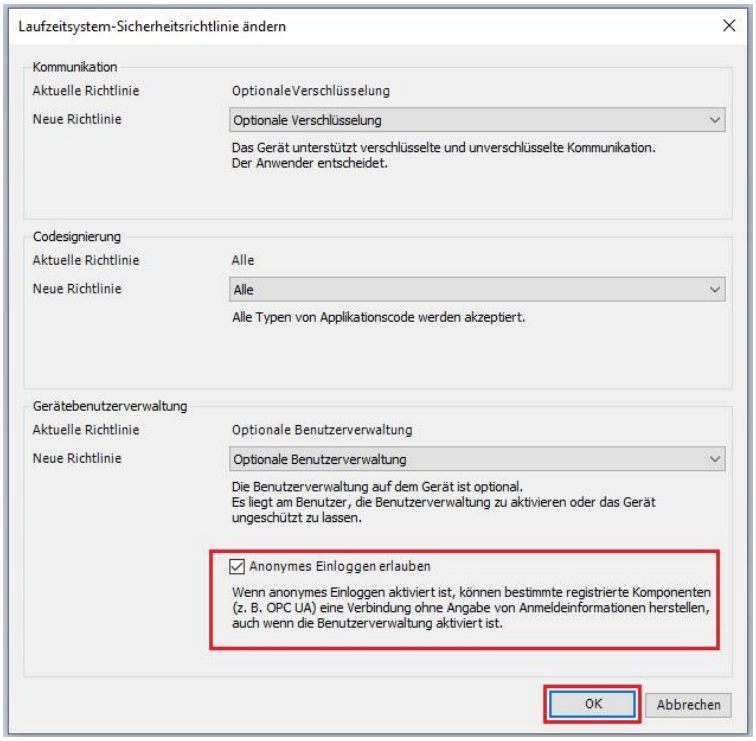


Abbildung 70: Anonymes Einloggen erlauben

Anschließend muss ein Bootprojekt erzeugt und die Steuerung neu gestartet werden, damit die Änderungen wirksam werden.

In Kapitel 5.2 OPC-UA Expert Client für Anonyme Verbindung konfigurieren wird die Konfiguration des UA-Expert Clients für anonyme Verbindungen beschrieben.

6 OPC-UA EXPERT CLIENT KONFIGURIEREN

6.1 OPC-UA EXPERT CLIENT FÜR VERSCHLÜSSELUNG MIT BENUTZERVERWALTUNG KONFIGURIEREN

Ein OPC-UA Client stellt alle notwendigen Funktionen bereit, um eine Verbindung zu einem OPC-UA Server aufzubauen, sich zu authentifizieren, Werte abzufragen und zu bearbeiten.

Ein geeigneter OPC-UA Client für den PC ist der **UA Expert Client** von der Firma Unified Automation GmbH. Unter der folgenden Adresse http://www.unifiedautomation.com kann der Client bezogen werden.

In dieser Anleitung wird der UA Expert Client für die Verbindung zum OPC-UA Server genutzt. Alle Konfigurationseinstellungen beziehen sich auf diesen Client in der Version 1.7.1 540. In neueren oder älteren Versionen können die Bezeichnungen und Einstellungen sich unterscheiden. Eventuell kann man die Einstellungen von dem UA Expert Client in anderen Clients verwenden oder man nutzt die Einstellungen zu Orientierung.

Starten Sie den UA Expert Client, um eine neue Verbindung zu dem OPC-UA Server auf der Steuerung anzulegen. Sobald der Client gestartet ist, können Sie über das **+** Symbol in der Menüleiste eine neue Verbindung zu einem Server hinzufügen. *Abbildung 71*.

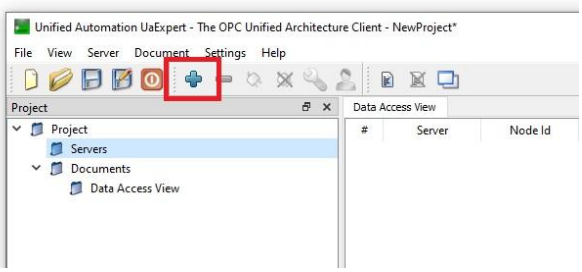


Abbildung 71: Eine neue Verbindung zu einem OPC-UA Server hinzufügen

In dem neuen Fenster in der Zeile **Custom Discovery** kann über einen Doppelklick auf **<Double click to Add Server>** eine neue Verbindung zu dem OPC-UA Server auf der Steuerung angelegt werden. *Abbildung 72*.

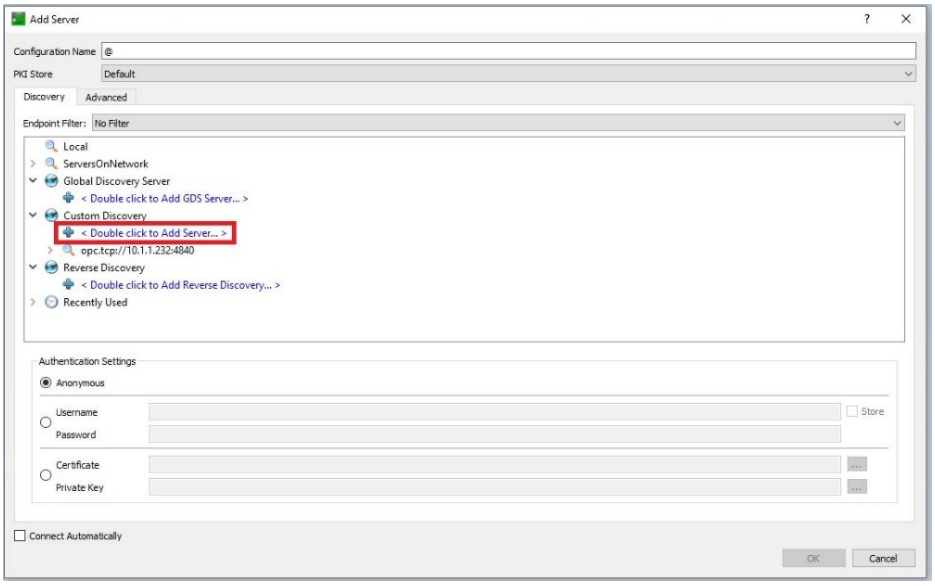


Abbildung 72: Verbindung anlegen

Bei der Frage nach der URL, geben Sie in das Feld hinter **opc.tcp://** die IP-Adresse der Steuerung ein, auf der der OPC-UA Server läuft, und fügen daran den OPC-UA Port **4840** an. Siehe *Abbildung 73*.

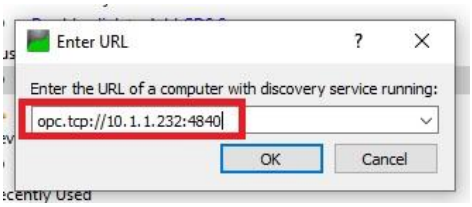


Abbildung 73: IP-Adresse des OPC-UA Servers angeben

In unserem Beispiel hat die Steuerung, auf dem der OPC-UA Server aktiv ist, die IP-Adresse 10.1.1.232. Mit **OK** wird die Verbindung hinzugefügt und das Fenster geschlossen.

In der Gesamtübersicht ist die Verbindung zu dem OPC-UA Server unter **Custom Discovery** zu sehen. Über das kleine **>** Zeichen vor dem Namen der Verbindung kann man sich die Details anzeigen lassen, wenn man draufklickt.
Hier sieht man den Namen des OPC-UA Servers auf der Steuerung und welche Verschlüsselungen von dem Server unterstützt werden. In dem Beispiel hat der Server den Namen **OPCUAServer@plm800** und unterstützt die Verschlüsselung **Basic256Sha256** und **Aes256_Sha256_RsaPss.**

Da auf der Steuerung die Verschlüsselung **Basic256Sha256** aktiv ist, entscheiden wir uns für diese und klicken anschließend auf **OK**. *Abbildung 74*.

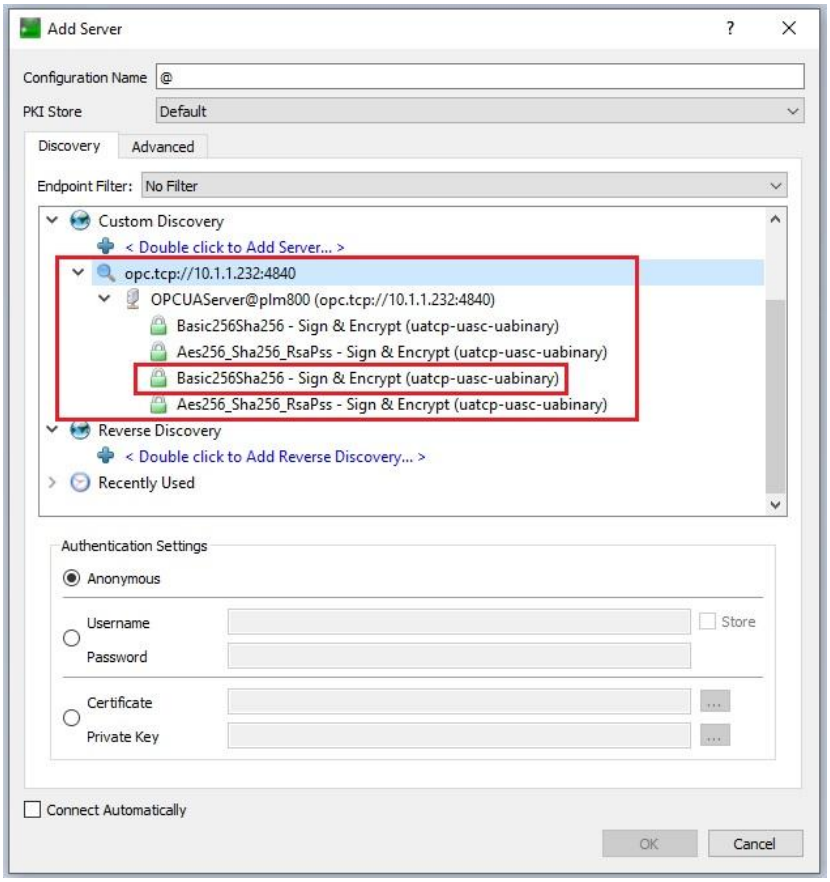


Abbildung 74: Verschlüsselung auswählen

Im Hauptfenster werden auf der linken Seite die bestehenden Serververbindungen angezeigt. Die neu erstellte Verbindung muss nun konfiguriert werden. Wählen Sie dazu die gewünschte

Verbindung durch Anklicken aus und klicken Sie anschließend auf das **Schraubenschlüssel-Symbol**, um die Konfiguration vorzunehmen. *Abbildung 75*.

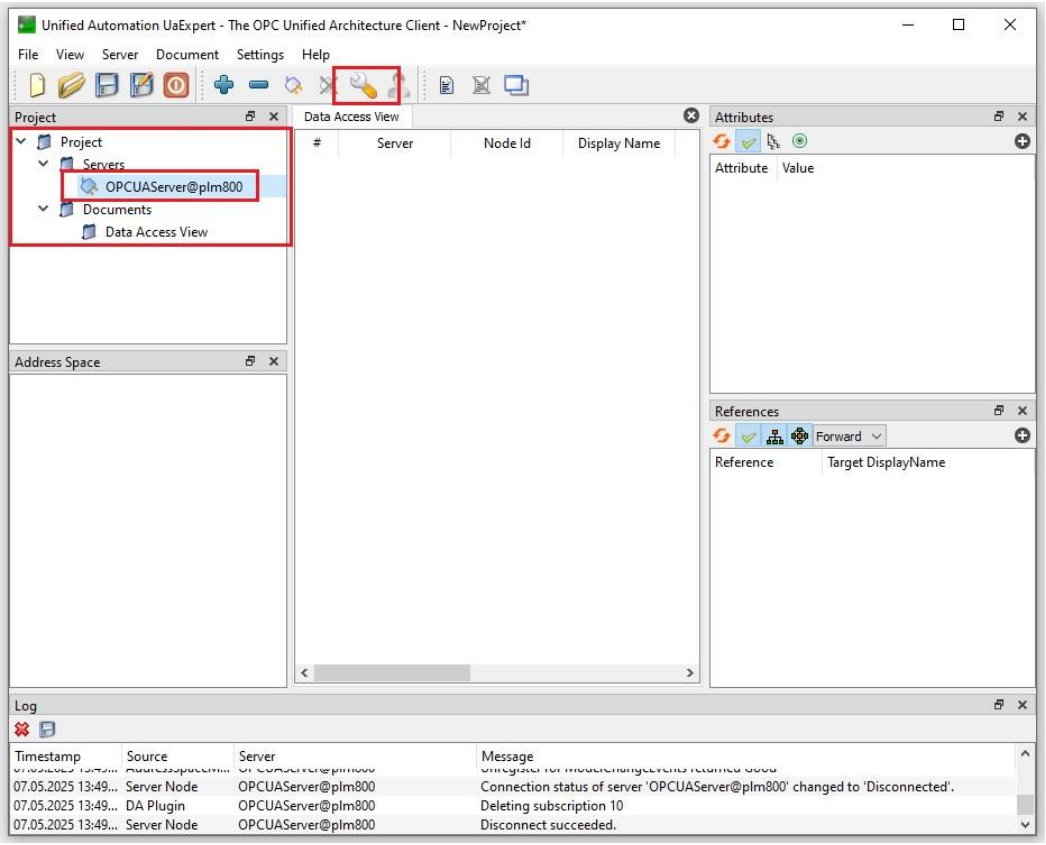


Abbildung 75: Verbindung konfigurieren

Im Konfigurationsfenster für die Verbindung müssen die Sicherheitseinstellungen angepasst werden.

Setzen Sie die **Security Policy** auf **Basic256Sha256** und den **Message Security Mode** auf **Sign & Encrypt**. Da auf der Steuerung ein Benutzerkonto mit Passwort für den OPC UA-Server eingerichtet wurde, ist als Authentifizierungsmethode **Username und Passwort** auszuwählen.

In unserem Beispiel ist der Username *my_server*, so wie er in der CODESYS IDE unter Benutzer und Gruppen angelegt wurde. Mit **OK** werden die Einstellungen übernommen und das Fenster wird geschlossen.

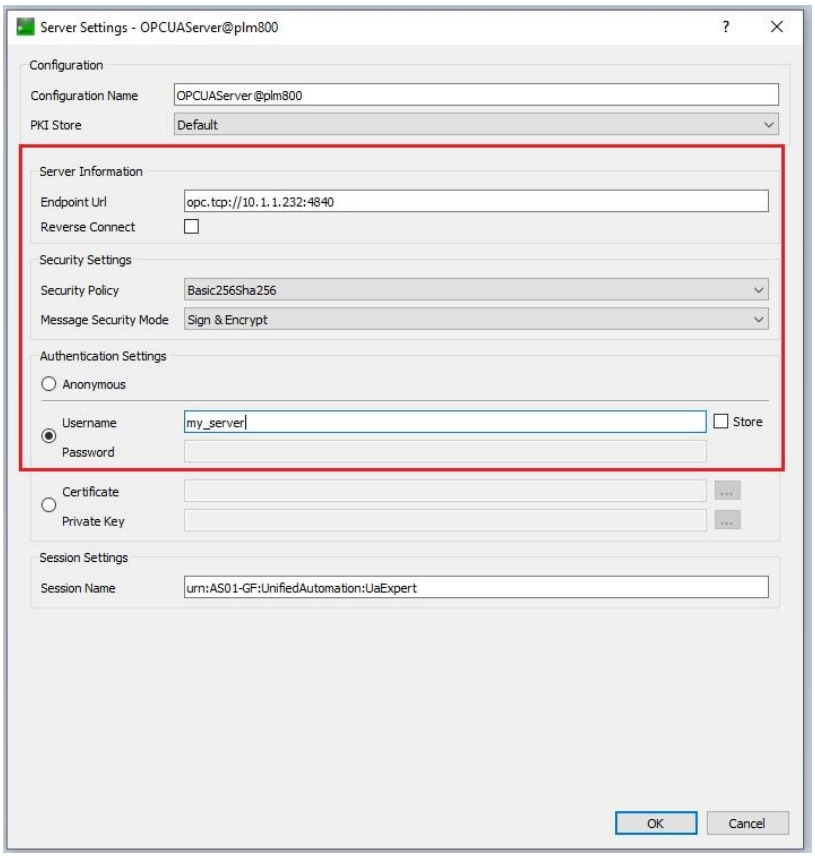


Abbildung 76: Einstellungen für die Verbindung zum Server

Durch die Verschlüsselte Kommunikation vom Client zum Server, dauert der Verbindungsaufbau etwas länger. Dadurch kann es zu einem Timeout Fehler kommen. Um dem vorzubeugen, müssen die Zeiten für die Kommunikation zwischen Client und Server erhöht werden.
Wählen Sie dazu aus der Menüleiste den Punkt **Settings** aus und klicken anschließend auf **Configure UaExpert**. *Abbildung 77*.

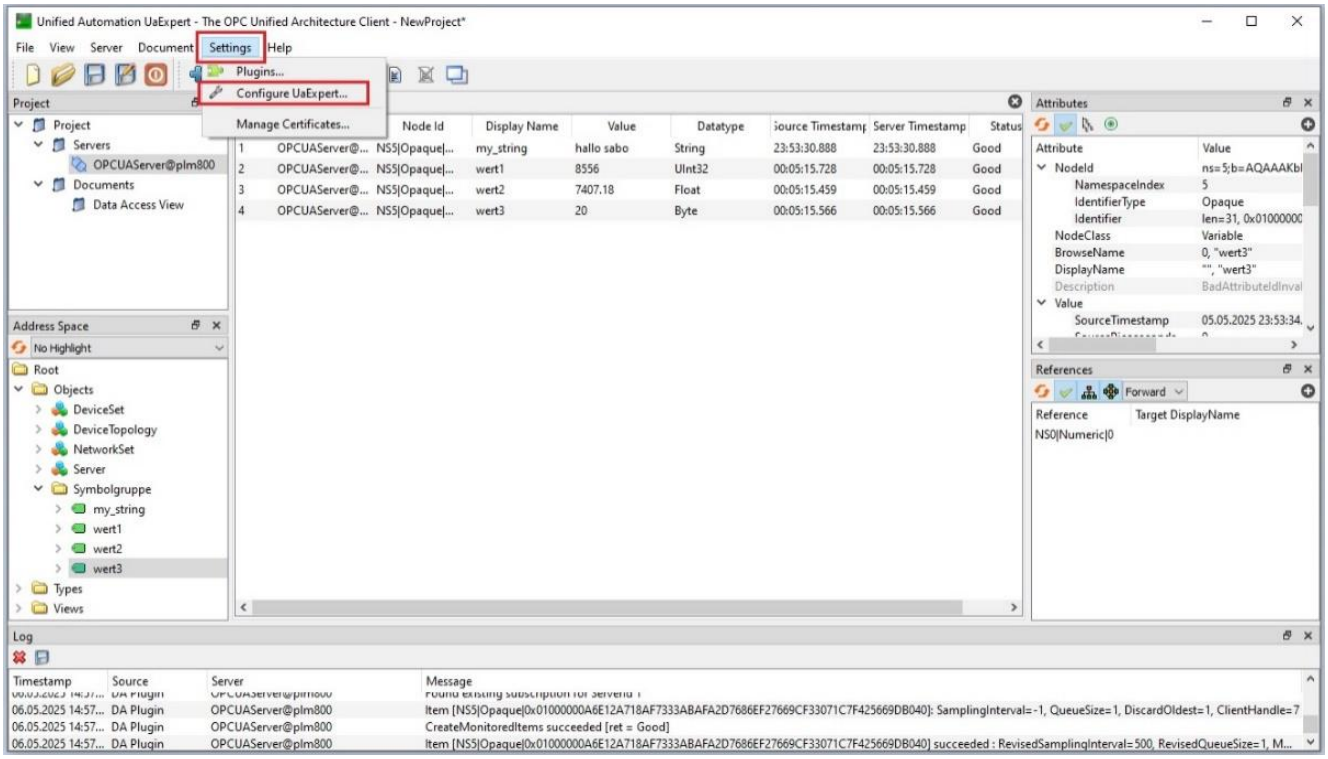


Abbildung 77: UA Expert konfigurieren

Es wird eine Tabelle mit Einstellungsmöglichkeiten für den Client angezeigt. Wichtig sind die Parameter **General.BrowseTimout**, **GeneralCallTimeout** und **General.ConnectTimeout**. Stellen Sie die Zeiten für diese Parameter genauso ein, wie in der *Abbildung 78* dargestellt.

Mit **OK** werden die Parameter übernommen und das Fenster geschlossen.

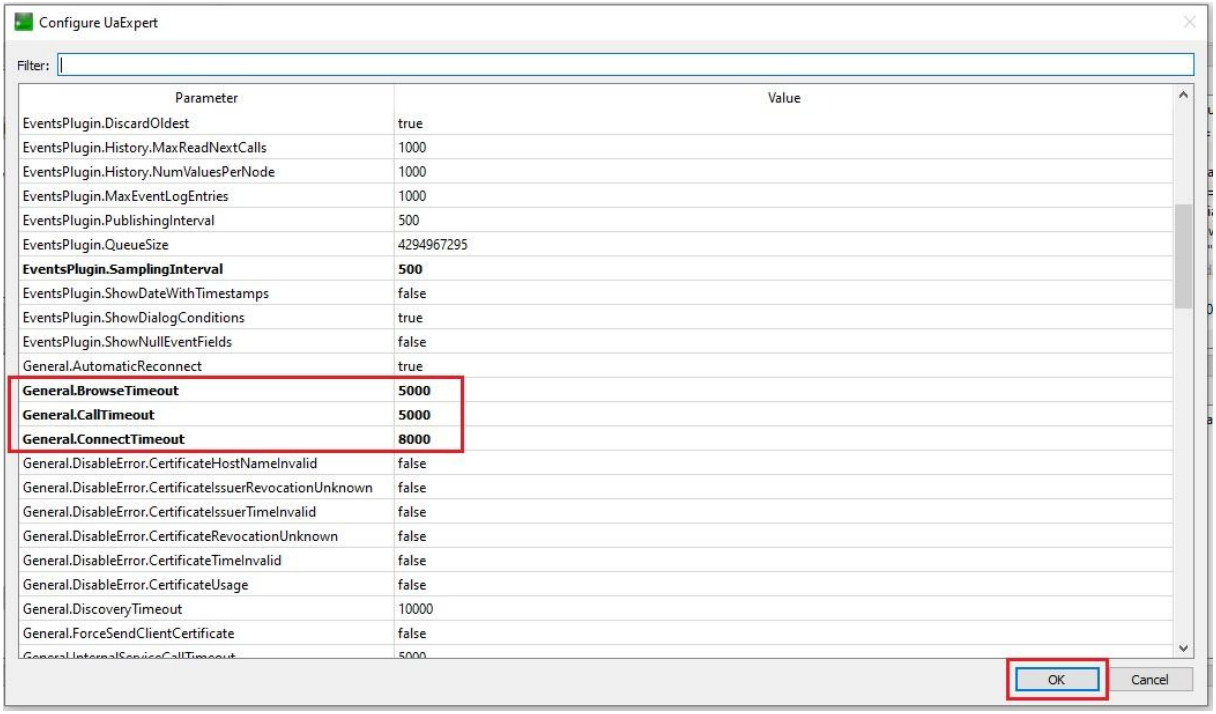


Abbildung 78: Konfiguration der Timeoutzeiten

Sollten Sie bei der Verbindung weiterhin einen Timeout Fehler erhalten, dann erhöhen Sie die Zeit für **General.ConnectTimeout** auf 10000. Die Zeit wird in Millisekunden angegeben.

Über das **Steckersymbol** in der Menüleiste, können Sie die Verbindung zu dem OPC-UA Server aufbauen.

Abbildung 79: Verbindung zum OPC-UA Server herstellen

Der Client verbindet sich jetzt mit dem Server. Da wir einen Benutzer mit Passwort für den OPC-UA Server angelegt haben, wird jetzt die Eingabe des Passwortes für den Benutzer verlangt. Geben Sie hier das Passwort, das Sie für den Benutzer vergeben haben, ein und bestätigen mit **OK** die Eingabe.

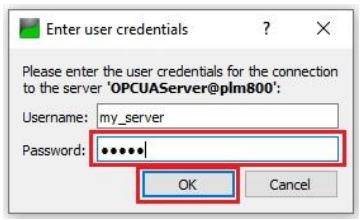


Abbildung 80: Passworteingabe für den OPC-UA Server Benutzer

Nach einer erfolgreichen Anmeldung erscheint eine Fehlermeldung, dass das Zertifikat der Steuerung nicht vertrauenswürdig ist. In diesem Fall kann man das Häkchen bei **Accept the server certificate temporarily for this session** setzen und mit **Continue** die Verbindung fortsetzen.

Alternativ besteht die Möglichkeit, **Trust Server Certificate** auszuwählen, um das Zertifikat dauerhaft als vertrauenswürdig zu markieren. *Abbildung 81*

.

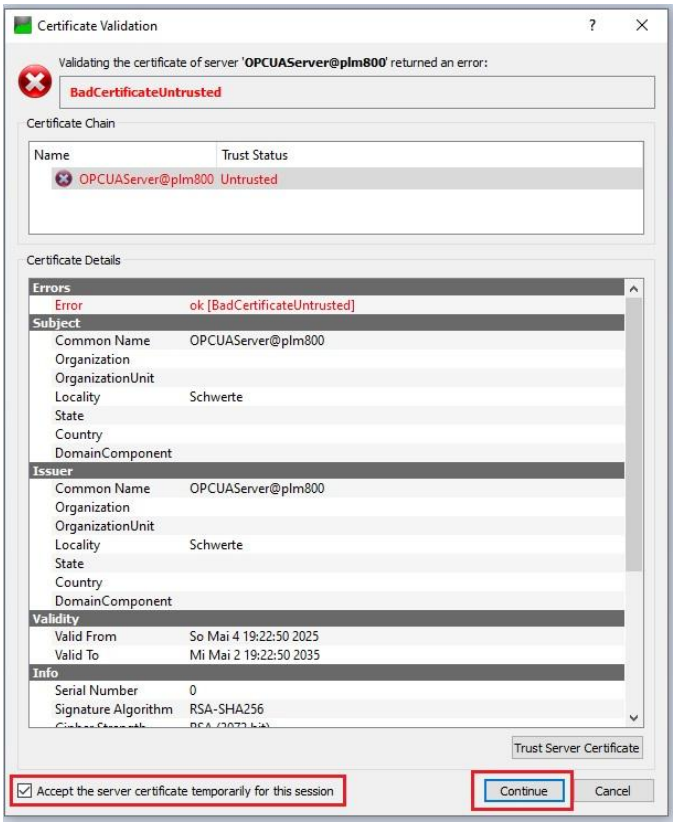


Abbildung 81: Fehlermeldung für nicht vertrauenswürdiges Zertifikat

Zusätzlich muss das Zertifikat vom UA Expert Client in der CODESYS IDE aus dem **Zertifikate in Quarantäne** Ordner in den Ordner für **Vertrauenswürdige Zertifikate** verschoben werden. Das geschieht in der CODESYS IDE über den **Security Screen**. Das Zertifikat vom UA Expert Client kann per Drag and Drop in den Ordner für Vertrauenswürdige Zertifikate verschoben werden. Siehe *Abbildung 82*.

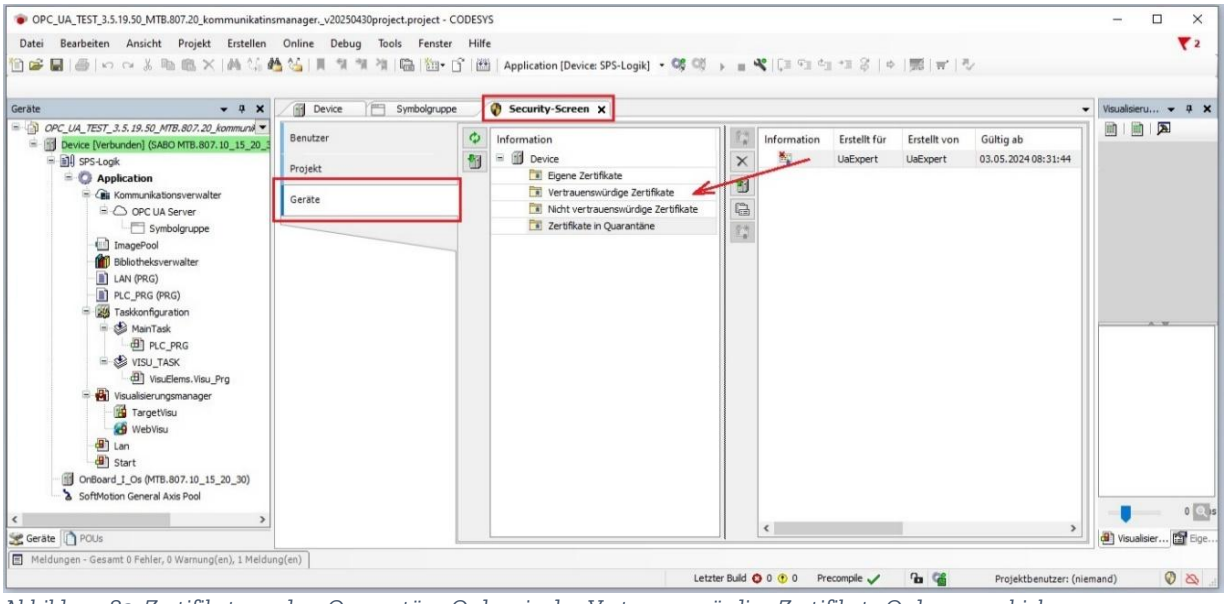


Abbildung 82: Zertifikat aus dem Quarantäne Ordner in den Vertrauenswürdige Zertifikate Ordner verschieben

Nach dem man das Zertifikat per Drag and Drop verschoben hat, wird es jetzt in dem Ordner für **Vertrauenswürdige Zertifikate** angezeigt. *Abbildung 83*.

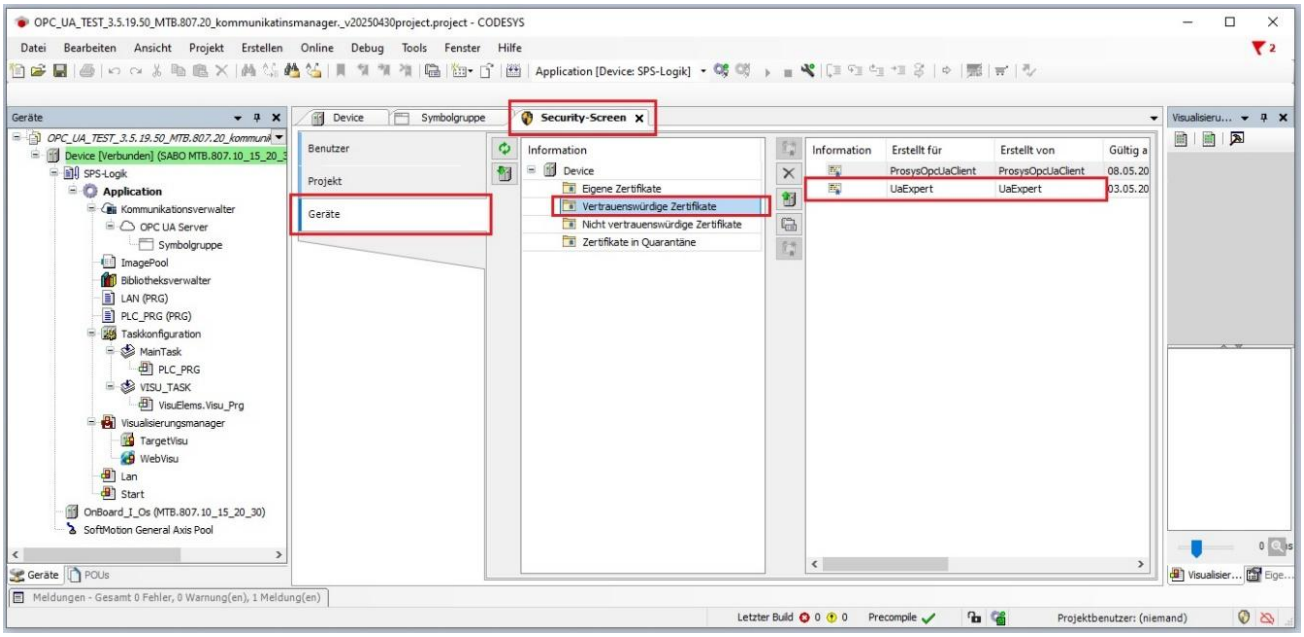


Abbildung 83: Anzeige der vertrauenswürdigen Zertifikate

Sollte bei dem Verbindungsaufbau vom Client zum Server folgende Fehlermeldung angezeigt werden, dann klicken Sie auf **Ignore**, damit die Verbindung trotz der Meldung aufgebaut wird.

Abbildung 84: Fehlermeldung während der Verbindung zum Server

Der Aufbau der Verbindung kann einige Sekunden in Anspruch nehmen. Sobald die Verbindung aufgebaut wurde, wird in dem Hauptfenster von dem UA Expert Client auf der linken Seite der **Server mit der Symbolgruppe**, die in der CODESYS IDE erstellt wurde, angezeigt. Die Symbolgruppe enthält die Variablen, die dem Kommunikationsverwalter und OPC-UA Server hinzugefügt wurden. Per Drag and Drop können die Variablen in das mittlere Fenster für **Data Access View** eingefügt werden, so dass man die Werte der Variablen ablesen und verändern kann. *Abbildung 85*.

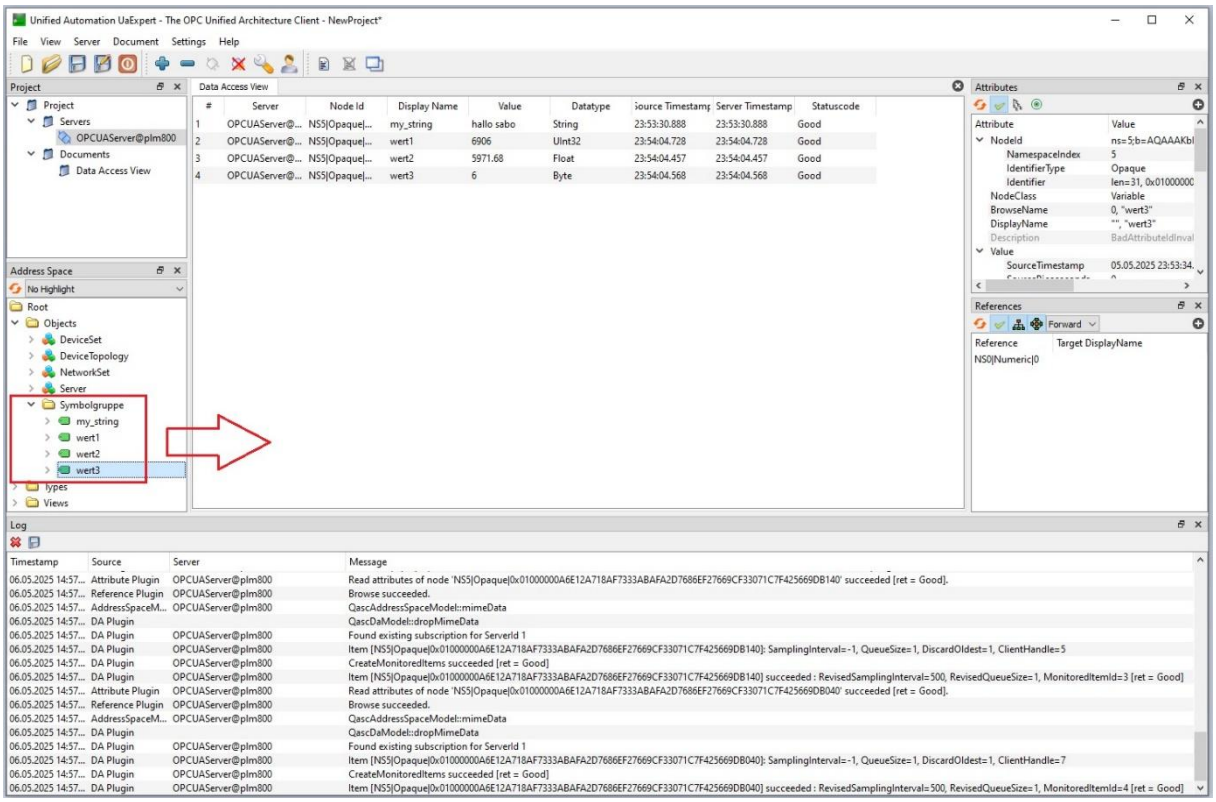


Abbildung 85: Anzeige der Variablen des OPC-UA Servers

Damit ist die Konfiguration des OPC-UA Servers und des UA Expert Clients abgeschlossen.

6.2 OPC-UA EXPERT CLIENT FÜR ANONYME VERBINDUNG KONFIGURIEREN

Ein OPC-UA Client stellt alle notwendigen Funktionen bereit, um eine Verbindung zu einem OPC-UA Server aufzubauen, sich zu authentifizieren, Werte abzufragen und zu bearbeiten. Ein geeigneter OPC-UA Client für den PC ist der **UA Expert Client** von der Firma Unified Automation GmbH.

Unter der folgenden Adresse http://www.unifiedautomation.com kann der Client bezogen werden.

In dieser Anleitung wird der UA Expert Client für die Verbindung zum OPC-UA Server genutzt. Alle Konfigurationseinstellungen beziehen sich auf diesen Client in der Version 1.7.2 584. In neueren oder älteren Versionen können die Bezeichnungen und Einstellungen differenzieren. Eventuell kann man die Einstellungen von dem UA Expert Client in anderen Clients verwenden oder man nutzt die Einstellungen zu Orientierung.

Starten Sie den UA Expert Client, um eine neue Verbindung zu dem OPC-UA Server auf der Steuerung anzulegen. Sobald der Client gestartet ist, können Sie über das + Symbol in der Menüleiste eine neue Verbindung zu einem Server hinzufügen. *Abbildung 86*.

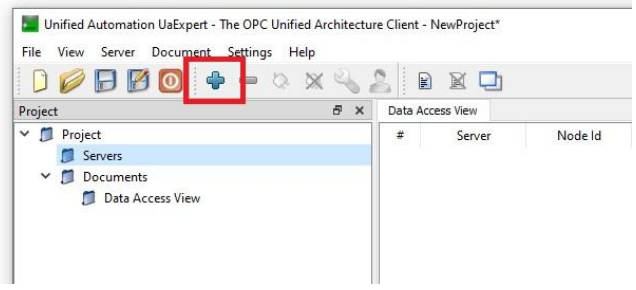


Abbildung 86: Neue Verbindung zum OPC-UA Server erstellen

In dem neuen Fenster in der Zeile **Custom Discovery** kann über einen Doppelklick auf **<Double click to Add Server>** eine neue Verbindung zu dem OPC-UA Server auf der Steuerung angelegt werden. *Abbildung 87*.

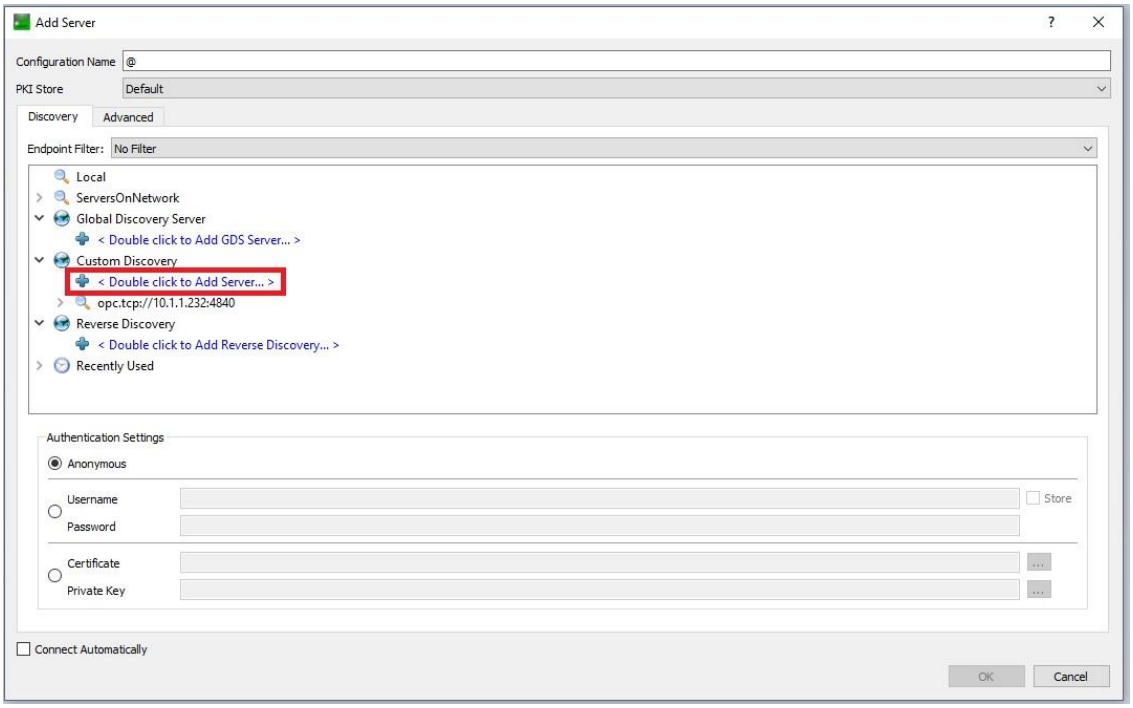


Abbildung 87: Neue Verbindung zum OPC-UA Server anlegen

Bei der Frage nach der URL, geben Sie in das Feld hinter **opc.tcp://** die IP-Adresse der Steuerung ein, auf der der OPC-UA Server läuft, und fügen daran den OPC-UA Port **4840** an. Siehe *Abbildung 88*.

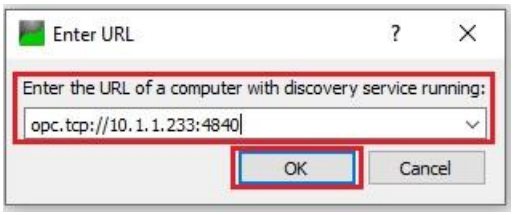


Abbildung 88: IP-Adresse des OPC-UA Servers eintragen

In unserem Beispiel hat die Steuerung, auf dem der OPC-UA Server aktiv ist, die IP-Adresse 10.1.1.233. Mit **OK** wird die Verbindung hinzugefügt und das Fenster geschlossen.

In der Gesamtübersicht ist die Verbindung zu dem OPC-UA Server unter **Custom Discovery** zu sehen. Über das kleine **>** Zeichen vor dem Namen der Verbindung kann man sich die Details anzeigen lassen, wenn man draufklickt. Hier sieht man den Namen des OPC-UA Servers auf der Steuerung und welche Verschlüsselungen von dem Server unterstützt werden. In dem Beispiel hat der Server den Namen **OPCUAServer@plm800**.

Da wir eine anonyme Verbindung zum OPC-UA Server nutzen möchten, muss in der Auswahl die Verbindung mit dem **roten offenen Schloss-Symbol** genutzt werden. Die Bezeichnung **None – None** gibt an, dass keine Verschlüsselung und keine Authentifizierung genutzt wird. In dem Feld **Authentication Setting** muss **Anonymous** angewählt sein. Mit der Schaltfläche **OK** wird die Verbindung übernommen. *Abbildung 89*.

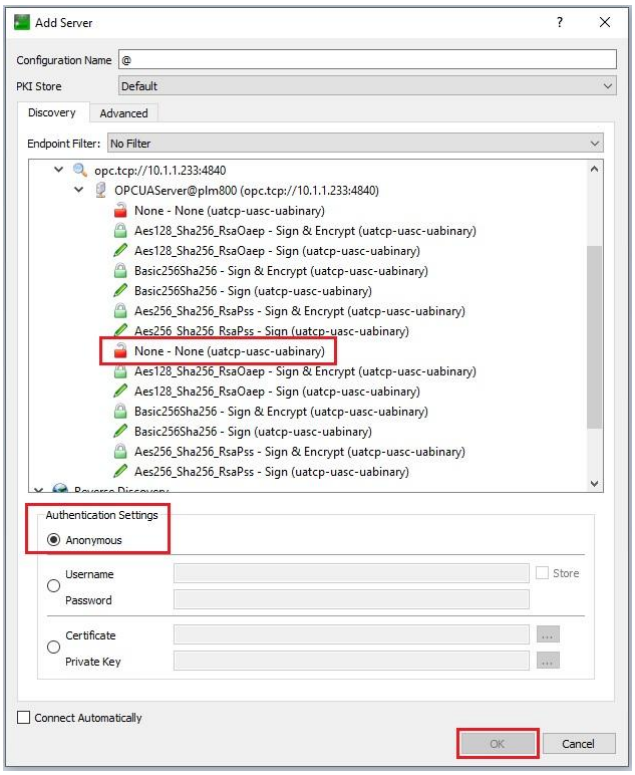


Abbildung 89: Verbindung auswählen

Über das **Steckersymbol** in der Menüleiste, können Sie die Verbindung zu dem OPC-UA Server aufbauen.

Abbildung 90: Mit dem Server verbinden

Bei erfolgreicher Verbindung zu OPC-UA Server sieht das Hauptfenster vom UA Expert Client wie folgt aus:

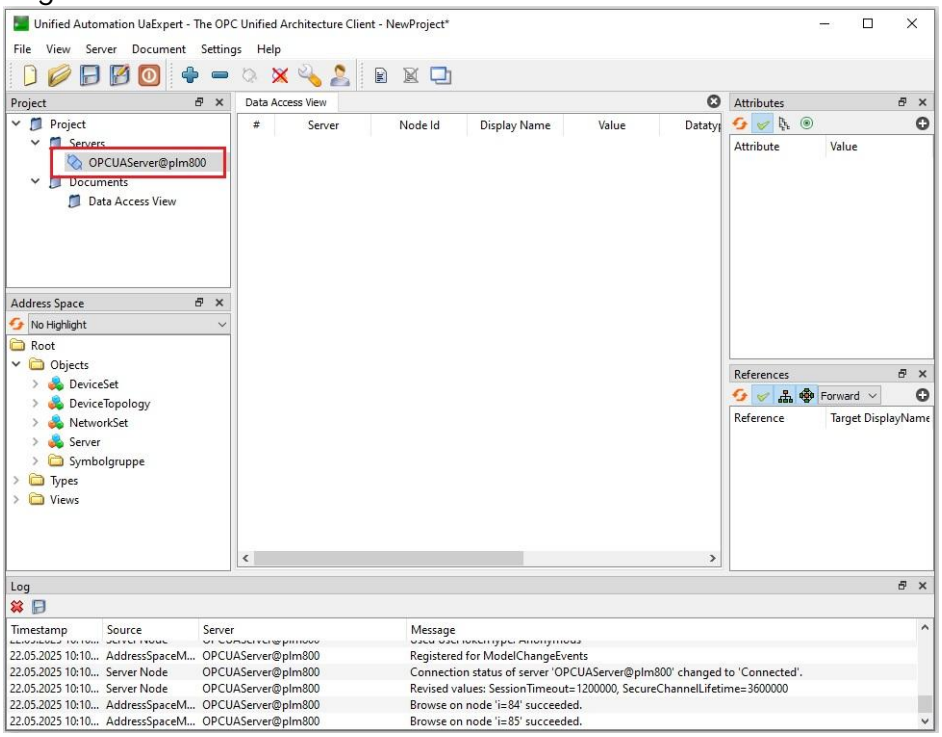


Abbildung 91: Aktive Verbindung zum OPC-UA Server

Per Drag and Drop können die Variablen aus der Symbolgruppe in das mittlere Fenster für **Data Access View** eingefügt werden, so dass man die Werte der Variablen ablesen und verändern kann. *Abbildung 92*.

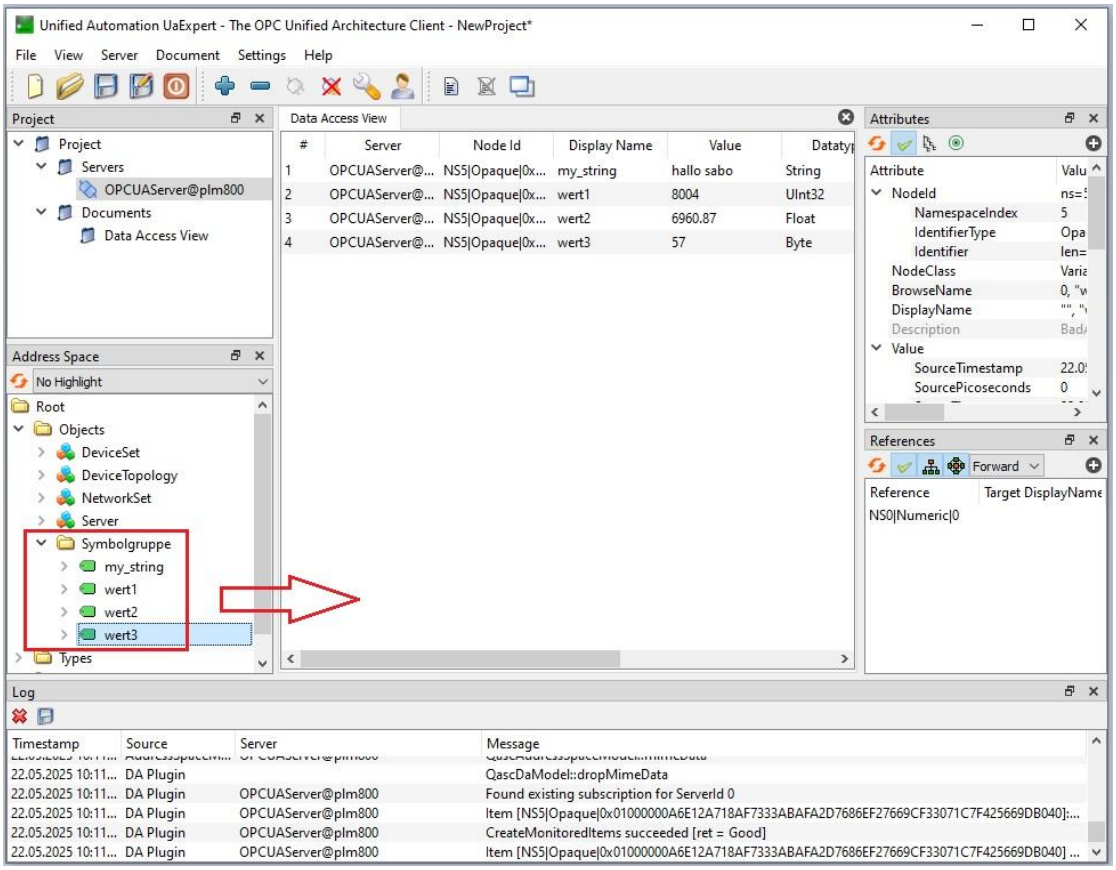


Abbildung 92: Datenansicht der OPC-UA Server Variablen

Damit ist die Konfiguration des OPC-UA Servers und des UA Expert Clients abgeschlossen.

7 SERIELLE SCHNITTSTELLEN

7.1 ALLGEMEINES

Die Steuerungen der Serie **PLM 800** verfügen über eine oder mehrere eingebaute serielle Schnittstellen vom Typ **RS232** oder **RS485**. Bei Bedarf können weitere serielle Schnittstellen in Form von Erweiterungsmodulen über den CAN-Systembus angeschlossen werden. Eine serielle Schnittstelle wird auch als COM-Port bezeichnet.

Die seriellen Schnittstellen sind gemäß folgendem Schema durchnummeriert:

COM 0 ... 3	Eingebaute serielle Schnittstellen (falls vorhanden)
COM 8	Virtuelle serielle Schnittstelle USB-Adapter
COM 9	Virtuelle serielle Schnittstelle TCP-Server
COM 10 ... 25	Virtuelle serielle Schnittstelle CAN-Erweiterungsmodule

Tabelle 4: Serielle Schnittstellen

Erläuterungen zu den Schnittstellenstandards RS232 und RS485 finden sich z.B. im Internet unter:

> https://de.wikipedia.org/wiki/RS-232 und
> https://de.wikipedia.org/wiki/EIA-485

Eine serielle Schnittstelle muss vor der Verwendung initialisiert werden. Hierfür stehen spezielle Bibliotheksfunktionen zur Verfügung, mit denen die Baudrate und andere Übertragungsparameter eingestellt werden. Die verfügbaren Parameter sind der jeweiligen Gerätedokumentation zu entnehmen.
Nach erfolgter Initialisierung erfolgt der Programmzugriff auf alle seriellen Schnittstellen in gleicher Weise.

7.2 INITIALISIERUNG DER EINGEBAUTEN SCHNITTSTELLEN (COM 0 ... 3)

Benötigt wird folgende Bibliothek:

PlmStandard > V3.5.16.28

Die angegebene Bibliothek muss vom Projekt geladen werden, damit die Funktionen zur Verfügung stehen. Klicken Sie dazu auf den ***Bibliotheksverwalter*** > ***Bibliothek hinzufügen*** > ***Sonstige*** > ***PlmStandard*** > ***OK***. Der benötigte Baustein zu Initialisierung der Schnittstelle befindet sich in dem Ordner ***SerialCommunication*** in der Bibliothek **PlmStandard**.

7.2.1 COMMSETPARAM (PLMSTANDARD.LIBRARY)

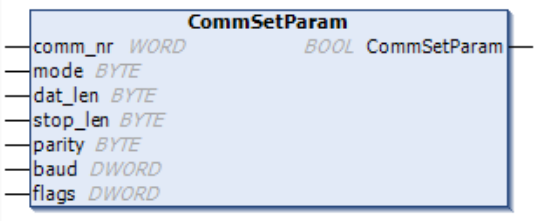


Abbildung 93: Funktion CommSetParam

Input Parameter:		
comm_nr	BYTE	Nummer des COM-Ports (z.B. 0 für erste eingebaute serielle Schnittstelle)
mode	BYTE	Betriebsart des COM-Ports: 0 = normale Verwendung durch das Anwenderprogramm Andere Werte sind für interne Verwendung
dat_len	BYTE	Anzahl der Datenbits pro Zeichen, 7 oder 8
stop_len	BYTE	Anzahl der Stopbits, 1 oder 2
parity	BYTE	0 = keine Parität (none) 1 = gerade Parität (even) 2 = ungerade Parität (odd)
baud	DWORD	Baudrate: 300, 600, 1200, 2400, 9600, 19200, 38400, 56000, 57600, 115200
flags	DWORD	Div. Steuerbits: 0 = RS232 Standardbetrieb 16#80 = Sende-/Empfangsumschaltung RS485 aktiv
Return-Wert:		
CommSetParam	BOOL	FALSE = Input-Parameter fehlerhaft TRUE = Parametrierung erfolgreich

Tabelle 5: Parameter CommSetParam

7.2.2 PROGRAMMIERBEISPIEL ZUR INITIALISIERUNG DER COM-PORTS 0 ... 3

Der in der *Abbildung 94* dargestellte Funktionsblock `CommInit()` wird vor der Verwendung der COM-Ports 0 ...3 im Programm aufgerufen. Die Initialisierung der COM-Ports sollte nur einmalig durchgeführt werden.

Die Reihenfolge der einzelnen Initialisierungsschritte sollte genau so gewählt werden, wie dargestellt.

Beim Aufrufen der Funktion muss zwingend der **Namenraum** der Bibliothek angegeben werden, in diesem Fall ist das `PlmStd`. Der Namensraum kann im **Bibliotheksverwalter** in der **Spalte Namensraum** abgelesen werden.

Beispiel in Strukturiertem Text:

CommInit

```
1  PROGRAM CommInit
2  VAR
3      initDone:bool;
4  END_VAR
5
```

100 %

```
1  IF NOT initDone THEN
2      initDone:=TRUE;
3
4      // Com-Port 0 für RS232 initialisieren
5      PlmStd.CommSetParam(comm_nr:=0,mode:=0,dat_len:=8,stop_len:=1,parity:=0,baud:=115200,flags:=0);
6
7      // Com-Port 1 für RS485 initialisieren
8      PlmStd.CommSetParam(comm_nr:=1,mode:=0,dat_len:=8,stop_len:=1,parity:=0,baud:=115200,flags:=16#80);
9  END_IF
```

100 %

Abbildung 94: Programmbeispiel zur Initialisierung der COM-Ports (ST)

Beispiel in FUP:

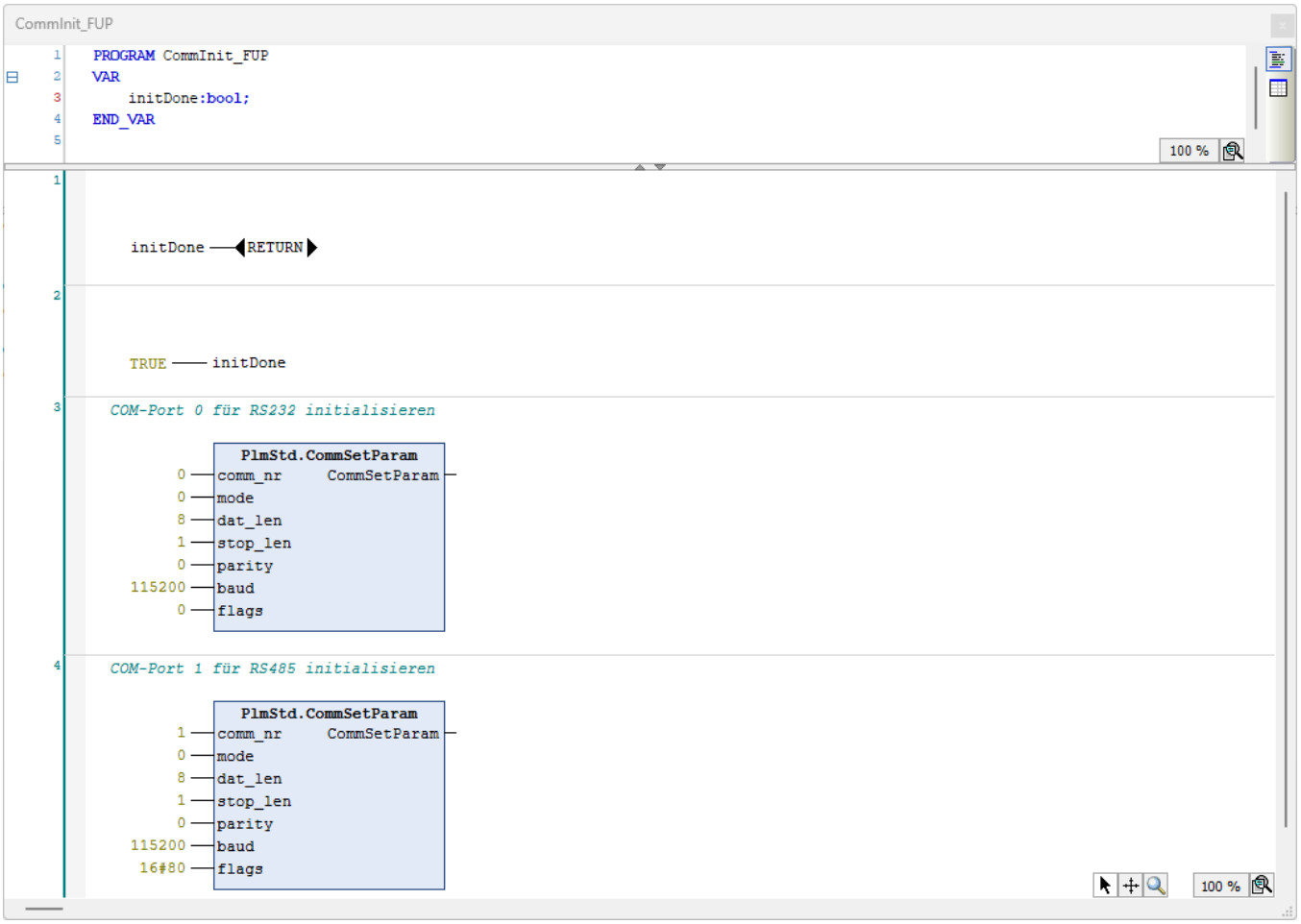


Abbildung 95: Programmbeispiel zur Initialisierung der COM-Ports (FUP)

7.3 INITIALISIERUNG DER VIRTUELLEN USB-SCHNITTSTELLE (COM 8)

Die Steuerung erlaubt den Anschluss von Geräten, die als USB-Serial-Adapter arbeiten. Falls ein solches Gerät an den USB-Port angeschlossen ist, kann es über die virtuelle serielle Schnittstelle COM 8 angesprochen werden.

Die Initialisierung erfolgt genau wie bei den eingebauten seriellen Schnittstellen COM 0...3 mittels der Funktion `CommSetParam()` (siehe *Abschnitt 7.2*).

Entsprechende Geräte erscheinen intern auf der Steuerung meistens unter dem Gerätenamen `ttyUSB0`, bevor sie intern der virtuellen Schnittstelle COM 8 zugewiesen werden können. Falls in Ausnahmefällen der interne Gerätename anders lautet, kann dies *vor* dem Aufruf der Funktion `CommSetParam()` durch Aufruf der Funktion

```
PlmStd.SystemSetParameter(10370,ADR(deviceName));
```

konfiguriert werden. `deviceName` muss dabei ein String sein, der den gewünschten Namen enthält, z.B.

```
deviceName:STRING:= 'ttyACM0';
```

Grundsätzlich gibt es Einschränkungen bezüglich der anschließbaren USB-Geräte. Im Zweifelsfall halten Sie bitte Rücksprache mit der SABO Elektronik GmbH.

7.4 INITIALISIERUNG DER VIRTUELLEN SCHNITTSTELLE FÜR CAN-ERWEITERUNGSMODULE (COM 10 ... 25)

Benötigt wird die folgende Bibliothek:

PlmProtocol > V3.5.16.35

Die angegebene Bibliothek muss vom Projekt geladen werden, damit die Funktionen zur Verfügung stehen. Klicken Sie dazu auf den ***Bibliotheksverwalter*** > ***Bibliothek hinzufügen*** > ***Sonstige*** > ***PlmProtocol*** > ***OK***. Der benötigte Baustein zu Initialisierung der Schnittstelle befindet sich in dem Ordner ***SimModul*** in der Bibliothek **PlmProtocol**.

Bei Verwendung von seriellen Schnittstellen, die durch PLM730 CAN-Bus-Module (z.B. SIM.730.10) bereitgestellt werden, erfolgt die Initialisierung durch den Funktionsblock `SIM_730_Com`.

7.4.1 SIM_730_COM (PLMPROTOCOL.LIBRARY)

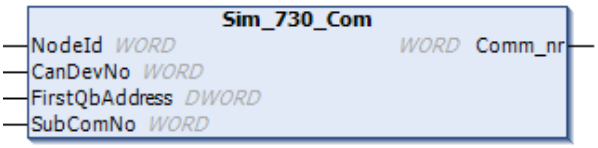


Abbildung 96: Funktionsbaustein Sim_730_Com

Input Parameter:		
NodeId	WORD	CAN-NodeID des SIM Moduls
CanDevNo	WORD	Nummer des zuständigen CAN-Masters, z.B. 0 oder 1
FirstQbAddress	DWORD	Adresse des ersten %QB des Moduls aus dem CANopen E/A-Abbild, z.B. ADR(%QB0)
SubComNo	WORD	Nummer der seriellen Schnittstelle, wenn auf der SIM-Baugruppe mehr als eine vorhanden ist, z.B.0
Return-Wert:		
Comm_nr	WORD	Zugewiesene COM-Nummer der Schnittstelle, z.B. 10

Tabelle 6: Parameter des SIM_730_Com Bausteins

Der Ausgabe-Parameter `Comm_nr` erhält die zugewiesene COM-Nummer der Schnittstelle und muss bei folgenden Aufrufen von `CommOut()` und `CommRead()` verwendet werden.

Der Eingabe-Parameter `FirstQbAddress` muss aus dem CANopen E/A Abbild abgelesen werden. Beispiel:

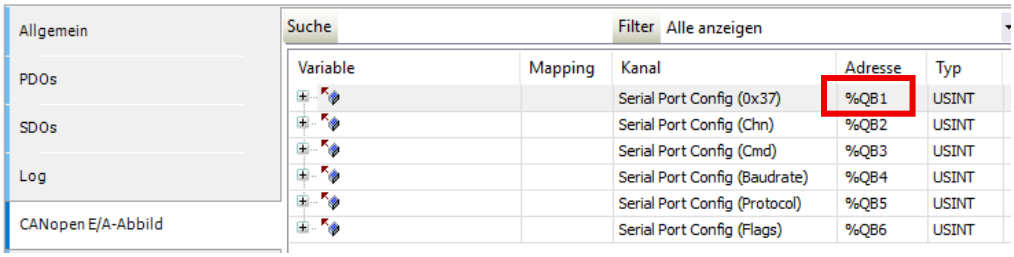


Abbildung 97: CANopen E/A-Abbild eines SIM Moduls

In diesem Fall ist `FirstQbAddres:= ADR(%QB1)` anzugeben.

7.5 VERWENDEN DER SERIELLEN SCHNITTSTELLEN

Benötigt wird folgende Bibliothek:

PlmStandard > V3.5.16.28

Die angegebene Bibliothek muss vom Projekt geladen werden, damit die Funktionen zur Verfügung stehen. Klicken Sie dazu auf den ***Bibliotheksverwalter*** > ***Bibliothek hinzufügen*** > ***Sonstige*** > ***PlmStandard*** > ***OK***. Die benötigten Funktionen für das Senden und das Empfangen auf einer seriellen Schnittstelle sind `CommOut` und `CommRead` und befinden sich in dem Ordner ***SerialCommunication*** in der Bibliothek **PlmStandard**.

7.5.1 COMMOUT (PLMSTANDARD.LIBRARY)

Abbildung 98: CommOut Funktion (PlmStandard)

Input Parameter:		
comm_nr	WORD	Nummer des COM-Ports, z.B. 0 für die erste serielle Schnittstelle
dat_adr	WORD	Adresse einer Variablen oder eines Arrays mit den zu sendenden Zeichen
dat_len	DWORD	Anzahl der zu sendenden Zeichen
Return-Wert:		
Comm_nr	WORD	Anzahl der tatsächlich versendeten Zeichen

Tabelle 7: Parameter der CommOut Funktion

Die Funktion `CommOut` übernimmt das Versenden von Zeichen über eine serielle Schnittstelle. Tatsächlich werden die Zeichen in einen internen Ringbuffer eingetragen, dessen Inhalt im Hintergrund durch die Steuerung versendet wird (vgl. *Abschnitt 7.6, Dauer der Datenübertragung*).

Der am Eingang `dat_adr` angegebene Datenspeicher (z.B. ein BYTE-Array) muss die durch `dat_len` bestimmte Anzahl an Zeichen bereitstellen. Die benötigte Speicheradresse wird üblicherweise mit der `ADR()`-Funktion ermittelt.

Wenn `dat_len` den Wert `0` hat, wird an `dat_adr` die Adresse einer String-Variablen erwartet. In diesem Fall werden so viele Zeichen versendet, wie der String momentan enthält. Das Versenden endet vor dem ersten Null-Zeichen im String, dies entspricht dem internen Speicherformat von CODESYS.

7.5.2 COMMREAD (PLMSTANDARD.LIBRARY)

Abbildung 99: Funktion CommRead (PlmStandard)

Input Parameter:		
comm_nr	WORD	Nummer des COM-Ports, z.B. 0 für die erste serielle Schnittstelle
dat_adr	DWORD	Adresse einer Variablen oder eines Arrays, in dem die Empfangszeichen gespeichert werden sollen
dat_len	DWORD	Maximale Anzahl der zu empfangenen Zeichen
Return-Wert:		
Comm_nr	DWORD	Anzahl der tatsächlich bei dat_adr gespeicherten Zeichen

Tabelle 8: Parameter der CommRead Funktion

Die Funktion `CommRead` liefert die von einer seriellen Schnittstelle empfangenen Zeichen.

Der am Eingang `dat_adr` angegebene Datenspeicher (z.B. ein BYTE-Array) muss die durch dat_len bestimmte Anzahl an Zeichen aufnehmen können. Die benötigte Speicheradresse wird üblicherweise mit der `ADR()`-Funktion ermittelt.

Bedingt durch die Dauer der Datenübertragung (vgl. *Abschnitt 7.6*) muss das Anwenderprogramm damit rechnen, die erwartete Anzahl von Zeichen nicht innerhalb eines einzigen Zyklus' von `CommRead` zu erhalten. Die übliche Vorgehensweise ist daher, die von `CommRead` erhaltenen Zeichen zunächst so lange zu sammeln, bis die Datenübertragung vollständig ist und erst dann die Daten auszuwerten (siehe auch Programmbeispiele in *Abschnitt 7.7* und *7.8*).

Wenn die empfangenen Daten als STRING verarbeitet werden sollen, ist zu beachten, dass Strings in CODESYS durch ein unsichtbares Null-Zeichen terminiert sind. Vor der Verarbeitung als String muss das Anwenderprogramm in jedem Fall sicherstellen, dass an die empfangenen Zeichen ein Null-Zeichen angehängt wird, da sonst die String-Länge undefiniert ist. Die Verarbeitung eines Strings mit undefinierter Länge kann zum Fehlverhalten des IEC-Programms oder sogar zum Absturz der Steuerung führen.

7.6 DAUER DER DATENÜBERTRAGUNG

Die Datenübertragung über RS232 oder RS485 erfolgt unsynchronisiert in Bezug auf die Zyklus- und Verarbeitungszeiten der Steuerung. Daher arbeiten sowohl Senden als auch Empfangen *gepuffert*, d.h. die Daten werden zunächst in Ringbuffer geschrieben, aus denen sie dann zur Verarbeitung ausgelesen werden.

Die Größe der Ringbuffer beträgt jeweils 5000 Zeichen.

Die Dauer der Datenübertragung muss vom Anwenderprogramm berücksichtigt werden:

- Beim Senden kann das Anwenderprogramm eine "große" Datenmenge (z.B. 500 Bytes) innerhalb eines Zyklus' zum Versenden bereitstellen. Die Sendefunktion `CommOut` schreibt diese Daten zunächst in einen Sende-Ringbuffer. Die Steuerung sorgt dann im Hintergrund dafür, dass die einzelnen Zeichen mit größtmöglicher Geschwindigkeit über die serielle Schnittstelle übertragen werden. Diese Übertragung kann jedoch mehrere Zykluszeiten in Anspruch nehmen.

- Empfangsdaten stehen häufig nicht sofort oder zunächst unvollständig zur Verfügung. Das Anwenderprogramm muss dies entsprechend berücksichtigen und darf erst mit der Verarbeitung der Daten beginnen, wenn alle erwarteten Zeichen eingetroffen sind. Das Eintreffen aller Zeichen im Empfangs-Ringbuffer kann dabei ebenfalls mehrere Zykluszeiten in Anspruch nehmen.

- Beim Empfang, insbesondere mit hohen Baudraten, werden pro Zyklus mehrere Zeichen empfangen. Bei großen Datenmengen müssen diese pro Zyklus vollständig abgearbeitet werden, um einen Überlauf des Empfangs-Ringbuffers zu vermeiden.

Die für eine Datenübertragung minimal benötigte Zeit (*Tmin*) lässt sich aus der Baudrate (*Baudrate*), der Anzahl der Bits pro Zeichen (*AnzBits*) und der Anzahl der übertragenen Zeichen (*Num*) berechnen:

$$Tmin = Num \times AnzBits / Baudrate$$

Die Anzahl der Bits pro Zeichen (*AnzBits*) berechnet sich aus 1 Startbit + Anzahl Datenbits + Anzahl Paritätbits + Anzahl Stopbits.

Beispiel 1: 200 Zeichen werden seriell übertragen mit 9600 Baud, 8 Datenbits, keine Parität, 1 Stopbit (9600, 8N1). Die minimale Übertragungszeit ist dann

$$Tmin = 200 \times 10 / 9600 = 208{,}3 \text{ ms}.$$

Dies entspricht bei einer Zykluszeit von 20 ms ca. 11 Zyklen. Wenn die Steuerung der Empfänger ist, müssen pro Zyklus 200 Zeichen / 11 Zyklen ≈ 19 Zeichen/Zyklus verarbeitet werden.

Beispiel 2: 5000 Zeichen werden seriell übertragen mit 115200 Baud, 8 Datenbits, keine Parität, 1 Stopbit (115200, 8N1). Die minimale Übertragungszeit ist dann

$$Tmin = 5000 \times 10 / 115200 = 434{,}0 \text{ ms}.$$

Dies entspricht bei einer Zykluszeit von 20 ms ca. 22 Zyklen. Wenn die Steuerung der Empfänger ist, müssen pro Zyklus 5000 Zeichen / 22 Zyklen ≈ 228 Zeichen/Zyklus verarbeitet werden.

7.7 PROGRAMMBEISPIELE (ST)

Die folgenden Programmbeispiele setzen voraus, dass die verwendete serielle Schnittstelle geeignet initialisiert ist (siehe *Abschnitte* 7.2 und 7.3).

In den Beispielen werden die Sende- und Empfangsfunktionen `CommOut` und `CommRead` (siehe *Abschnitte* 7.5.1 und 7.5.2) verwendet.

7.7.1 BEISPIEL SENDBYTES (ST)

SendBytes_ST

```
PROGRAM SendBytes_ST
VAR
    txData  :ARRAY[0..100] OF BYTE;
    txLen   :DWORD;
    do_send :BOOL;
END_VAR
```

```
// Bytes senden
IF do_send THEN
    do_send:= FALSE;
    txData[0]:= 65; // A
    txData[0]:= 66; // B
    txData[0]:= 67; // C
    txLen:= 3;
    PlmStd.CommOut(comm_nr:= 0, dat_adr:= ADR(txData), dat_len:= txLen);
END_IF
```

Abbildung 100: Senden von BYTE Daten über COM 0 (ST)

Beim Aufrufen der Funktion muss zwingend der **Namenraum** der Bibliothek angegeben werden, in diesem Fall ist das `PlmStd`. Der Namensraum kann im **Bibliotheksverwalter** in der **Spalte Namensraum** abgelesen werden.

Sobald `do_send` auf `TRUE` gesetzt ist, wird das BYTE-`Array txData[]` mit den drei Zeichen A – B – C (65 – 66 – 67) gefüllt und diese drei Zeichen über COM 0 übertragen.

7.7.2 BEISPIEL SENDSTRING (ST)

SendString_ST

```
PROGRAM SendString_ST
VAR
    txLine  :STRING(100):='ABC';
    cr_lf   :STRING(20):='$R$N';
    do_send :BOOL;
END_VAR
```

```
// String senden mit Zeilenende = CR LF (ASCII 13, ASCII 10)
IF do_send THEN
    do_send:= FALSE;
    PlmStd.CommOut(comm_nr:= 0, dat_adr:= ADR(txLine), dat_len:= LEN(txLine));
    PlmStd.CommOut(comm_nr:= 0, dat_adr:= ADR(cr_lf), dat_len:= 2);
END_IF
```

Abbildung 101: Senden eines Strings mit Zeilenendkennung über COM 0 (ST)

Beim Aufrufen der Funktion muss zwingend der **Namenraum** der Bibliothek angegeben werden, in diesem Fall ist das `PlmStd`. Der Namensraum kann im **Bibliotheksverwalter** in der **Spalte Namensraum** abgelesen werden.

Sobald `do_send` auf `TRUE` gesetzt ist, wird der `STRING txLine` über COM 0 übertragen. Anschließend wird im Beispiel noch eine Zeilenendekennung mit `CR LF (ASCII 13 10)` übertragen, anhand derer der Empfänger das Ende des Strings erkennen kann. Die Kennung muss jedoch zwischen Sender und Empfänger vereinbart werden.

7.7.3 BEISPIEL RECEIVEBYTES (ST)

ReceiveBytes_ST

```
1  PROGRAM ReceiveBytes_ST
2  VAR
3      rxData      :ARRAY[0..100] OF BYTE;
4      rxLen       :DWORD;
5      num         :DWORD;
6      wantedLen   :DWORD:=10;
7  END_VAR
```

100 %

```
1  // Bytes empfangen, Verarbeitung nach best. Anzahl
2  REPEAT
3      num:= PlmStd.CommRead(comm_nr:= 0, dat_adr:= ADR(rxData) + rxLen, dat_len:= 1);
4      rxLen:= rxLen + num;
5  UNTIL
6      num = 0 OR rxLen = wantedLen
7  END_REPEAT
8
9  IF rxLen = wantedLen THEN
10     // Verarbeitung von rxLen Bytes in rxData[0..rxLen-1] ...
11     rxLen:=0;
12 END_IF
```

100 %

Abbildung 102: Empfangen von BYTE Daten über COM 0 (ST)

Beim Aufrufen der Funktion muss zwingend der **Namenraum** der Bibliothek angegeben werden, in diesem Fall ist das `PlmStd`. Der Namensraum kann im **Bibliotheksverwalter** in der **Spalte Namensraum** abgelesen werden.

Im Beispiel wird eine feste Anzahl von Zeichen (hier: `wantedLen = 10`) von COM 0 gelesen und in einem BYTE-`Array rxData[]` gespeichert. `rxLen` enthält stets die Anzahl der gültigen Zeichen in `rxData[]`.

Sobald 10 Zeichen vorhanden sind, erfolgt die Verarbeitung der Array-Daten innerhalb des IF-Zweiges (Zeile 9). Nach der Verarbeitung muss `rxLen` wieder auf Null gesetzt werden (Zeile 11).

Falls insgesamt mehr als 10 Zeichen empfangen wurden, verbleiben die restlichen im Empfangs-Ringbuffer. Das Programm muss ggf. sicherstellen, dass durch einen Ringbuffer-Überlauf keine Zeichen verlorengehen, indem z.B. nach der Verarbeitung im IF-Zweig erneut Empfangsdaten abgerufen werden.

7.7.4 BEISPIEL RECEIVESTRING (ST)

ReceiveString_ST

```
1  PROGRAM ReceiveString_ST
2  VAR
3      rxLine  :STRING(20);
4      lineLen :DWORD;
5      rxPtr   :POINTER TO BYTE;
6      num     :DWORD;
7      rxChar  :BYTE;
8  END_VAR
```

100 %

```
1  // String empfangen, Verabeitung nach Zeilenende = CR LF (ASCII 13, ASCII 10)
2  rxPtr:= ADR(rxLine) + lineLen;
3  WHILE TRUE DO
4      num:= Plmstd.CommRead(comm_nr:= 0, dat_adr:= ADR(rxChar), dat_len:= 1);
5      IF num = 0 THEN
6          rxChar:= 0;
7          EXIT;
8      END_IF
9      IF rxChar = 10 OR lineLen >= 100 THEN
10         EXIT;
11     END_IF
12     IF rxChar <> 13 THEN
13         rxPtr^:= rxChar;
14         rxPtr:= rxPtr + num;
15         lineLen:= lineLen + num;
16     END_IF
17 END_WHILE
18 rxPtr^:= 0;
19
20 IF rxChar = 10 THEN
21     // Verarbeitung der Stringvariablen rxLine ...
22     lineLen:= 0;
23 END_IF
```

100 %

Abbildung 103: Empfangen eines Strings über COM 0 (ST)

Beim Aufrufen der Funktion muss zwingend der **Namenraum** der Bibliothek angegeben werden, in diesem Fall ist das `PlmStd`. Der Namensraum kann im **Bibliotheksverwalter** in der **Spalte Namensraum** abgelesen werden.

Das Beispiel implementiert vollständig den Empfang eines Strings, der mit einer Zeilenendekennung CR LF (ASCII 13 10) abgeschlossen ist. Der empfangene String wird ohne die Zeilenendekennung in der STRING-Variablen `rxLine` gespeichert.

Nach dem Empfang des letzten Zeichens (LF) erfolgt die String-Verarbeitung innerhalb des IF-Zweiges. Nach der Verarbeitung muss `lineLen` auf Null gesetzt werden.

Die String-Variable `rxLine` wurde mit `STRING(100)` deklariert und kann somit 100 Zeichen aufnehmen. Falls der empfangene String länger als 100 Zeichen ist, wird der Rest abgeschnitten. Die Verarbeitung erfolgt jedoch erst nach Empfang des LF Zeichens.

Strings werden in CODESYS intern durch ein unsichtbares Null-Zeichen terminiert. Eine mit `STRING(100)` deklarierte Variable bietet daher intern Platz für 101 Zeichen, um in jedem Fall das abschließende Null-Zeichen aufnehmen zu können. Wenn Empfangsdaten von der seriellen Schnittstelle als String verarbeitet werden sollen, ist vor der Verarbeitung das Null-Zeichen explizit anzuhängen. Dies geschieht im Beispiel in Zeile 18 durch `rxPtr^:=0`.

Falls mehrere Strings empfangen wurden, verbleiben die restlichen im Empfangs-Ringbuffer. Das Programm muss ggf. sicherstellen, dass durch einen Ringbuffer-Überlauf keine Zeichen verlorengehen, indem z.B. nach der Verarbeitung im IF-Zweig erneut Empfangsdaten abgerufen werden.

7.7.5 BEISPIEL SENDBYTES (FUP)

Die folgenden Programmbeispiele setzen voraus, dass die verwendete serielle Schnittstelle geeignet initialisiert ist (siehe *Abschnitte* 7.2 und 7.3).

In den Beispielen werden die Sende- und Empfangsfunktionen `CommOut` und `CommRead` (siehe *Abschnitte* 7.5.1 und 7.5.2) verwendet.

Eine komplexere Datenverarbeitung in FUP wird nicht empfohlen, da dies wesentlich übersichtlicher in ST gelöst werden kann.

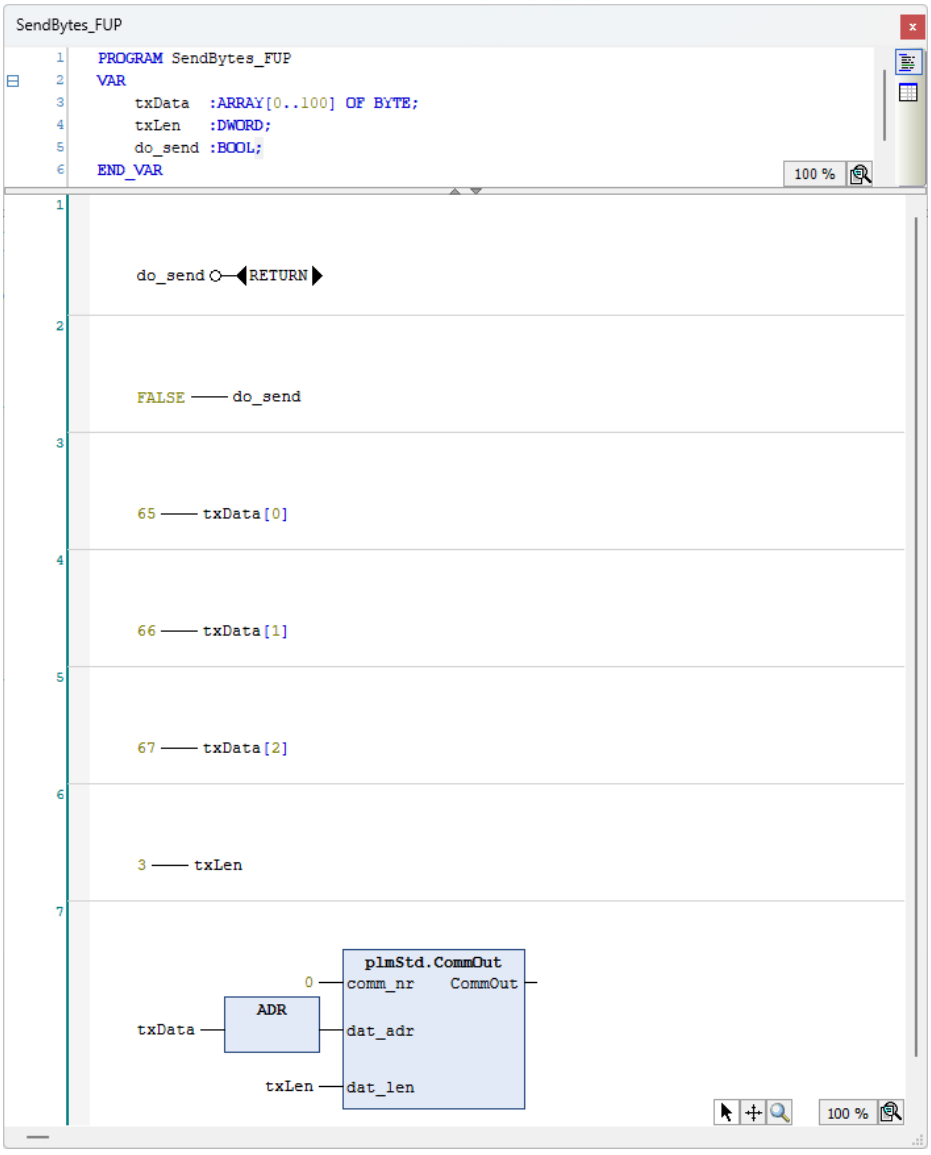


Abbildung 104: Senden von BYTE Daten über COM 0 (FUP)

Das Beispiel entspricht dem aus *Abschnitt* 7.7.1.

Beim Aufrufen der Funktion muss zwingend der **Namenraum** der Bibliothek angegeben werden, in diesem Fall ist das `PlmStd`. Der Namensraum kann im **Bibliotheksverwalter** in der **Spalte Namensraum** abgelesen werden.

Sobald `do_send` auf TRUE gesetzt ist, wird das BYTE-`Array txData[]` mit den drei Zeichen A – B – C (65 – 66 – 67) gefüllt und diese drei Zeichen über COM 0 übertragen.

7.7.6 BEISPIEL RECEIVEBYTES (FUP)

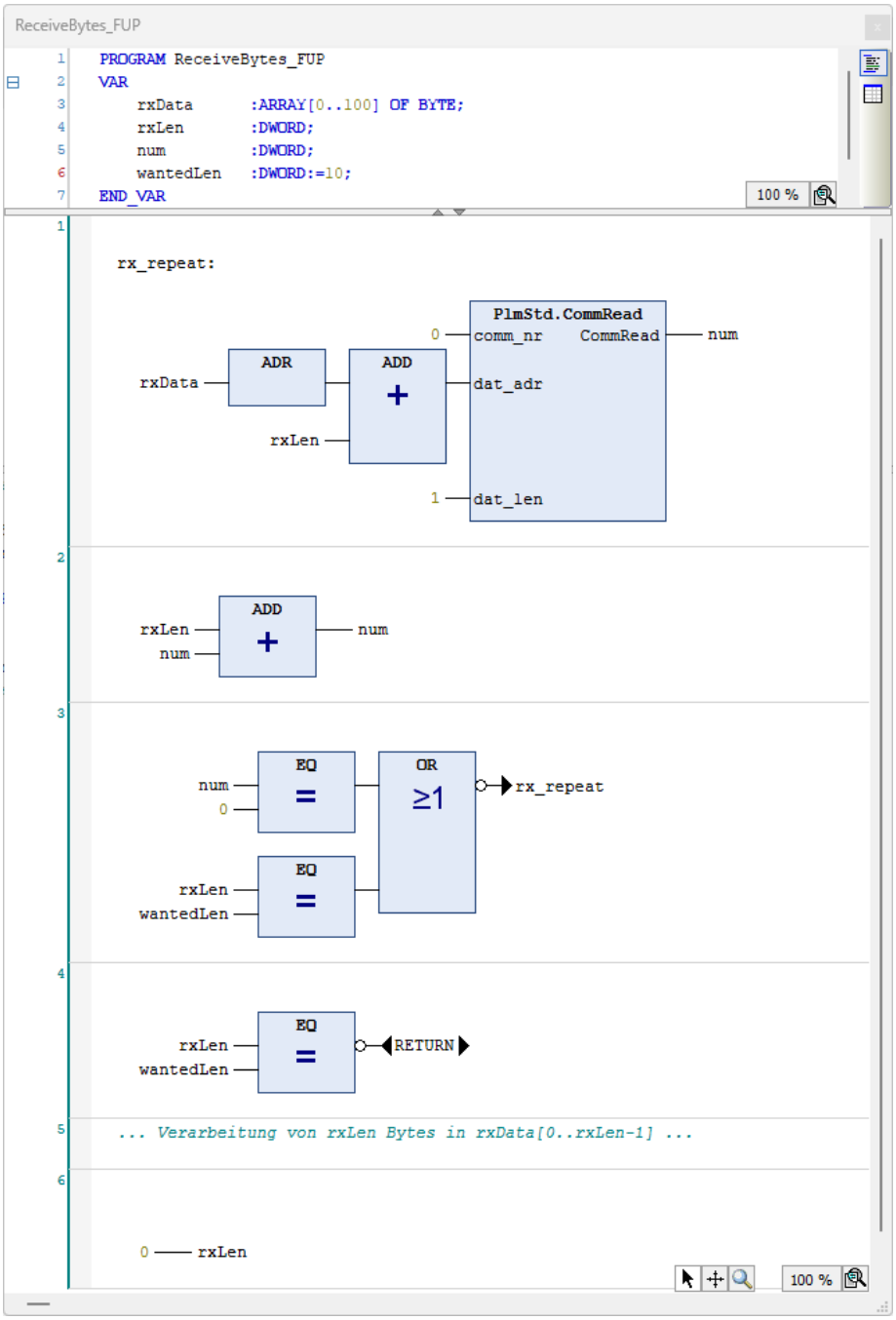


Abbildung 105: Empfangen von BYTE-Daten über COM 0 (FUP)

Das Beispiel entspricht dem aus *Abschnitt* 7.7.3.

Beim Aufrufen der Funktion muss zwingend der **Namenraum** der Bibliothek angegeben werden, in diesem Fall ist das `PlmStd`. Der Namensraum kann im **Bibliotheksverwalter** in der **Spalte Namensraum** abgelesen werden.

Es wird eine feste Anzahl von Zeichen (hier: `wantedLen = 10`) von COM 0 gelesen und in einem BYTE-Array `rxData[]` gespeichert. `rxLen` enthält stets die Anzahl der gültigen Zeichen in `rxData[]`.

Sobald 10 Zeichen vorhanden sind, erfolgt die Verarbeitung der Array-Daten (Netzwerk 5). Nach der Verarbeitung muss `rxLen` wieder auf Null gesetzt werden (Netzwerk 6).

Falls insgesamt mehr als 10 Zeichen empfangen wurden, verbleiben die restlichen im Empfangs-Ringbuffer. Das Programm muss ggf. sicherstellen, dass durch einen Ringbuffer-Überlauf keine Zeichen verlorengehen, indem z.B. der Programmbaustein mehrfach aufgerufen wird, bis keine Empfangsdaten in diesem Zyklus mehr vorliegen.

8 DATEIEN SCHREIBEN UND LESEN

8.1 ALLGEMEINES

Häufig besteht der Wunsch, Daten eines Programms in eine Datei zu schreiben oder Daten aus einer Datei zu lesen. Eine Anwendung ist, Parameter eines Programms in einer Datei zu speichern oder aus einer Datei zurückzulesen, z.B. zur Archivierung der Parameter oder des Einlesens von "Rezepten".

Eine Datei kann auf einem internen Laufwerk der Steuerung oder auf einem USB-Stick gespeichert sein.

Das Dateiformat, in dem Daten gespeichert werden, kann grundsätzlich frei gewählt werden. Häufig bietet sich die Speicherung in einem zeilenorientierten ASCII-Format an, welches zu Kontrollzwecken direkt mit einem Texteditor gelesen oder bearbeitet werden kann.

8.2 BENÖTIGTE BIBLIOTHEKEN

Benötigt werden die folgenden Bibliotheken:

PlmStandard > V3.5.16.28
PlmUtil > V3.5.16.47

Die angegebenen Bibliotheken müssen vom Projekt geladen werden, damit die Funktionen zur Verfügung stehen. Klicken Sie dazu auf den ***Bibliotheksverwalter*** > ***Bibliothek hinzufügen*** > ***Sonstige*** > ***PlmStandard*** > ***OK*** und wiederholen Sie den Schritt für die **PlmUtil** Bibliothek. Die benötigten Bausteine zum Schreiben von Dateien finden Sie in der Bibliothek **PlmStandard** und dem Ordner **FileWork** und der Bibliothek **PlmUtil** in dem Ordner **CSV-File**.

Die Steuerungen der Serie PLM 800 verfügen über die folgenden Laufwerke:

Laufwerk	Größe	Beschreibung
a/	500 MB	RAM-Disk, Dateien werden bei Neustart gelöscht
b/	100 MB	Interner Flash Speicher, Dateien werden persistent gespeichert, darf nur selten beschrieben werden
c/	Abhängig von Medium	USB-Stick
d/	Abhängig vom Medium	SD-Card

e/	Abhängig vom Medium	Zweites USB Laufwerk
p/	50 MB	Interner Flash Speicher, Dateien werden persistent gespeichert, darf nur selten beschrieben werden

Tabelle 9: Übersicht der Laufwerke der PLM 800 Serie

Laufwerk c/ steht erst nach dem Einstecken eines USB-Sticks bzw. einer USB-Festplatte zur Verfügung. Laufwerk d/ steht erst nach dem Einstecken einer SD-Card zur Verfügung.

Wichtig: Die Laufwerke b/ und p/ sind im **internen Flash**-Speicher der Steuerung realisiert.

Die Anzahl der **Schreibzyklen** von Flash-Speicherbausteinen ist generell **begrenzt** und liegt bei mehreren zehntausend. Danach sind Schäden am Flash-Speicher zu erwarten, dabei wird die Steuerung irreparabel beschädigt. Laufwerke b/ und p/ können für Logger-Daten o.ä. verwendet werden, aber es darf auf **keinen Fall zyklisch** in das Laufwerk geschrieben werden. Es sollte sichergestellt werden, dass Daten oder Parameter auf diesen Laufwerken z.B. durch eine manuelle Eingabe oder in größeren Zeitabständen gesichert werden.

8.2.1 LAUFWERK P/ KONFIGURIEREN

Zum Einrichten des Laufwerks p/ wird die Funktion `DrivePControl` benötigt, die sich in dem Ordner **System** der Bibliothek **PlmStandard** befindet.

Abbildung 106: Funktion DrivePControl zur Konfiguration des Laufwerks p/ (PlmStandard)

Input Parameter:		
`ctrl`	`WORD`	`16#03 = Laufwerk p/ anlegen ohne ftp-Zugriff` `16#07 = Laufwerk p/ anlegen mit ftp-Zugriff`
Return-Wert:		
`DrivePControl`	`DWORD`	

Die Funktion darf im IEC-Programm nur einmalig aufgerufen z.B. im Baustein `InitOnce()`. Der Parameter `ctrl` bestimmt die FTP-Zugriffsrechte.

8.2.2 BEISPIEL KONFIGURATION LAUFWERK P/ (ST)

Beispiel für das Einrichten des Laufwerks p/ ohne FTP-Zugriff:

Abbildung 107: Funktion DrivePControl ohne FTP-Zugriff (ST)

Beispiel für das Einrichten des Laufwerks p/ mit FTP-Zugriff:

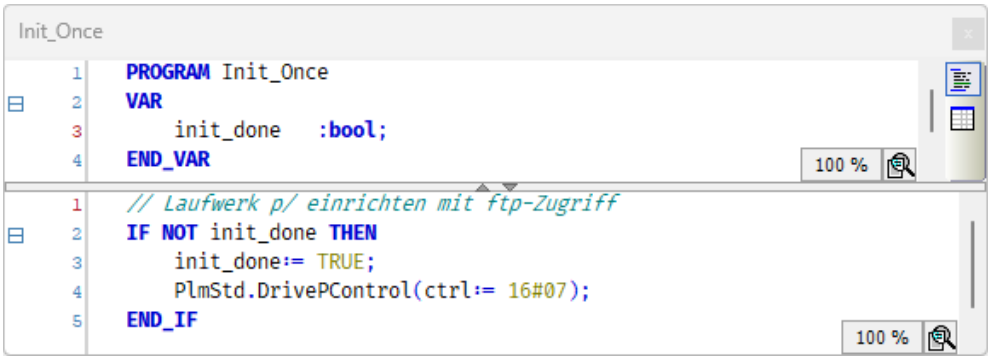


Abbildung 108: Funktion DrivePControl mit FTP-Zugriff (ST)

8.3 FTP-SERVER

Auf die Laufwerke kann mittels des internen FTP-Servers der Steuerung zugegriffen werden. Dazu ist eine Ethernet-Verbindung notwendig und die IP-Adresse der Steuerung muss bekannt sein.

Der Zugriff auf den FTP-Server erfolgt z.B. durch einen PC mit einem FTP-Client Programm. Unter Windows kann der standardmäßig vorhandene Windows-Explorer (Filemanager) als FTP-Client verwendet werden. Der Zugriff erfolgt über die folgende URL:

```
ftp://<ip-adresse>/
```

Anstelle von `<ip-adresse>` ist die vierstellige IP-Adresse der Steuerung einzusetzen, z.B. `ftp://10.1.1.231/`.

Bei der Verwendung von FTP-Client-Programmen ist darauf zu achten, dass die Datenübertragung im **Binärmodus** erfolgt, nicht im ASCII-Modus. Dies kann meistens im FTP-Client-Programm konfiguriert werden. Andernfalls besteht die Gefahr, dass Binärdateien fehlerhaft übertragen werden.

8.4 VORGEHENSWEISE ZUM SCHREIBEN EINER DATEI

Das Schreiben einer Datei im IEC-Programm geschieht grundsätzlich mit folgenden Befehlen aus der Bibliothek PlmStandard.Library:

```
VAR
      fid   :DWORD;
      num   :DINT;
      data  :STRING := 'Hallo Welt$R$N';
END_VAR

fid:= PlmStd.FileOpen(FileName:='a/test.txt', FileFlag:='w+');
If fid <> 0 then
      num:= PlmStd.FileWrite(FileId:= fid, Buffer:= ADR(data), Size:= LEN(data));
      PlmStd.FileClose(FileId:=fid);
END_IF
```

Der Befehl `PlmStd.FileOpen()` öffnet die Datei `test.txt` auf Laufwerk `a/`. Die Angabe von `FileFlag:='w+'` erstellt eine leere Datei, egal ob diese neu oder bereits vorhanden ist. Falls stattdessen an eine vorhandene Datei angehängt werden soll, ist `FileFlag:='a+'` zu verwenden.

Im Erfolgsfall dient `fid` den folgenden Dateifunktionen als Referenz auf die geöffnete Datei. Falls die Datei nicht erstellt werden kann, hat der Rückgabewert `fid` den Wert `0`; in diesem Fall dürfen keine weiteren Dateifunktionen mit `fid` ausgeführt werden.

Die vorgegebene Stringvariable `data` enthält eine ASCII-Zeichenkette, die mit `'$R$N'` (Zeilenumbruch CR-LF) abgeschlossen ist. Alternativ ist jeder andere Variablentyp denkbar, der auch Binärdaten enthalten kann.

Der Befehl `PlmStd.FileWrite()` liefert in `num` die Anzahl der tatsächlich geschriebenen Zeichen zurück. Die Anzahl entspricht normalerweise dem bei Size angegebenen Wert, kann jedoch im Fehlerfall kleiner oder sogar Null sein. Dies wird in obigem Beispiel nicht überprüft.

Nach dem Schreiben muss die Datei mit `PlmStd.FileClose()` geschlossen werden; dadurch wird `fid` ungültig und darf nicht weiter verwendet werden.

In vielen Fällen können die komfortablen Funktionen aus dem Ordner **FileWork** der Bibliothek **PlmStandard** verwendet werden, die hier allerdings nicht beschrieben werden.

8.5 VORGEHENSWEISE ZUM LESEN EINER DATEI:

Das Lesen einer Datei im IEC-Programm geschieht grundsätzlich mit folgenden Befehlen aus der Bibliothek PlmStandard.Library:

```
VAR
      fid   :DWORD;
      num   :DINT;
      data  :STRING;
      pBY   :POINTER TO BYTE;
END_VAR
fid:= PlmStd.FileOpen(FileName:='a/test.txt', FileFlag:='r');
If fid <> 0 then
      num:= PlmStd.FileRead(FileId:= fid, Buffer:= ADR(data), Size:= SIZEOF(data));
      pBY:= ADR(data) + num;
      pBy^:= 0;
      PlmStd.FileClose(FileId:=fid);
END_IF
```

Der Befehl `PlmStd.FileOpen()` öffnet die Datei `test.txt` auf Laufwerk a/. Die Angabe von `FileFlag:='r'` öffnet die Datei zum Lesen; dies setzt voraus, dass die Datei bereits vorhanden ist.

Im Erfolgsfall dient `fid` den folgenden Dateifunktionen als Referenz auf die geöffnete Datei. Falls die Datei nicht gelesen werden kann, hat der Rückgabewert `fid` den Wert `0`; in diesem Fall dürfen keine weiteren Dateifunktionen mit `fid` ausgeführt werden.

Der Befehl `PlmStd.FileRead()` erhält in Buffer die Adresse einer Variablen, in die die gelesenen Daten gespeichert werden. `Size` enthält in obigem Beispiel die Größe dieser Variablen, um einen Überlauf zu vermeiden. Es werden max. `Size` Bytes aus der Datei gelesen, jedoch weniger, falls die Datei kürzer ist. Der Rückgabewert in `num` enthält die Anzahl der tatsächlich gelesenen Zeichen; er ist beim Lesen normalerweise unbedingt auszuwerten.

Beim Einlesen von Strings ist zu beachten, dass ein ASCII-String aus einer Datei noch nicht dem internen Speicherformat von CODESYS entspricht und daher zunächst nicht direkt weiterverarbeitet oder in der Visu angezeigt werden darf. Die Umwandlung erfolgt hier durch die beiden Zeilen mit der Variablen `pBy`. Das Einlesen von mehreren Strings mit unbekannter Länge aus einer Datei kann auf diese Weise **nicht** erfolgen; dies ist jedoch nicht Thema dieses Abschnitts.

Nach dem Lesen muss die Datei mit `PlmStd.FileClose()` geschlossen werden; dadurch wird `fid` ungültig und darf nicht weiter verwendet werden.

In vielen Fällen können die komfortablen Funktionen aus dem Ordner **FileWork** der Bibliothek **PlmStandard** verwendet werden, die hier allerdings nicht beschrieben werden.

8.6 BEARBEITUNGSGESCHWINDIGKEIT

Die zum Schreiben und Lesen notwendigen Dateioperationen benötigen in einigen Fällen beträchtliche Bearbeitungszeiten durch das Betriebssystem der Steuerung, z.B. kann das Speichern einer großen Datei auf einem USB-Stick mehrere Sekunden dauern, bedingt durch den Flash-Speicher auf dem USB-Stick.

Eine Dateioperation, die länger als ein Steuerungszyklus (20 ms) dauert, hält den Zyklus auf, so dass auch alle anderen Bearbeitungen des Zyklus' aufgehalten werden.

Wir empfehlen daher, die entsprechenden Programmteile an einen eigenen-Task anzuhängen, so dass der MainTask, der das Hauptprogramm abarbeitet, nicht beeinflusst wird.

8.7 PARAMETER-DATEIEN LESEN UND SCHREIBEN

In diesem Abschnitt wird die Speicherung von Parametern als Binärdatei und als CSV-Datei behandelt.

Alternativ können in vielen Fällen die komfortablen Funktionen aus dem Ordner **FileWork** der Bibliothek **PlmStandard** verwendet werden oder aus der Bibliothek **PlmUtil** und dem Ordner **CSV-File**, die hier allerdings nicht beschrieben werden.

Binärdateien bieten die kompakteste und einfachste Möglichkeit der Datenspeicherung, allerdings mit den Nachteilen, dass erstens die Parameter nicht im Klartext lesbar sind und zweitens eine nachträgliche Erweiterung der zu speichernden Daten Probleme bereiten kann.

CSV-Dateien (CSV = Comma Separated Value) sind ebenfalls einfach zu erzeugen, können im Klartext gelesen und sogar nachträglich mit einem Text-Editor bearbeitet werden. Die Dateien sind meist etwas größer als entsprechende Binärdateien. Die Werte werden hier im Klartext (ASCII-Format) mit einem definierten

Trennzeichen gespeichert. Als Trennzeichen wird häufig ein Komma, ein Semikolon oder ein Tabulatorzeichen verwendet. Eine CSV-Datei könnte z.B. so aussehen:

```
Wert1 <TAB> Wert2 <TAB> Wert3 <CRLF>
Wert4 <TAB> Wert5 <TAB> Wert6 <CRLF>
Wert7 <TAB> Wert8 <TAB> Wert9 <CRLF>
```

Das aufwändigere XML-Format mit seinen Vor- und Nachteilen wird hier nicht behandelt.

Die Bibliothek **PlmUtil** mit dem Ordner **CSV-File** bietet Funktionen zum Schreiben und Lesen von CSV-Dateien.

8.7.1 CSVREAD (PLMUTIL.LIBRARY)

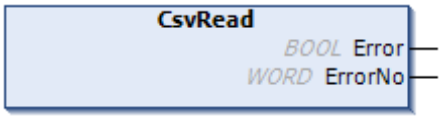


Abbildung 109: Funktion CsvRead (PlmUtil.Library)

Output-Parameter:		
Error	BOOL	TRUE = Fehler beim Verarbeiten des Bausteins aufgetreten
ErrorNo	WORD	1 = Fehler beim Lesen der Datei 2 = Undefiniertes Kontrollzeichen 3 = Zeilenende anstelle des Trennzeichens eingelesen 4 = Trennzeichen anstelle des Zeilenendes eingelesen 10 = Fehler beim Öffnen der Datei

Der Funktionsblock `CsvRead` wird über seine Methoden verwendet. Folgende Methoden können aufgerufen werden:

- `Csv_read.csv_to_bool(METH)`
- `Csv_read.csv_to_byte(METH)`
- `Csv_read.csv_to_dint(METH)`
- `Csv_read.csv_to_dword(METH)`
- `Csv_read.csv_to_int(METH)`
- `Csv_read.csv_to_lreal(METH)`
- `Csv_read.csv_to_real(METH)`
- `Csv_read.csv_to_sint(METH)`
- `Csv_read.csv_to_word(METH)`
- `Csv_read.file_open(METH)`
- `Csv_read.file_close(METH)`
- `Csv_read.get_file_pos(METH)`
- `Csv_read.next_line(METH)`
- `Csv_read.next_value(METH)`
- `Csv_read.set_file_pos(METH)`
- `Csv_read.set_ignore_char(METH)`

Als Beispiel sieht der Aufruf der Methode zum Einlesen eines Wertes aus einer bereits geöffneten CSV-Datei, der als WORD-Wert in der Variablen my_word gespeichert werden soll, wie folgt aus:

```
csv_read.csv_to_word(WordPtr:=ADR(my_word);
```

Zusätzlich kann man den Rückgabewert der Methode verwenden, um weitere Informationen zu erhalten.

```
result:=csv_read.csv_to_word(WordPtr:=ADR(my_word);
```

Die Variable result ist vom Typ WORD und kann enthält folgende Rückgabewerte der Methode:

Bit	Wert (hex)	Wert (dez)	Bedeutung
.0	16#0001	1	CR wurde nach dem Wert gefunden
.1	16#0002	2	LF wurde nach dem Wert gefunden
.2	16#0004	4	Trennzeichen wurde nach dem Wert gefunden
.3	16#0008	8	Eingelesener Wert ist gültig
.4	16#0010	16	Der Buffer enthält Ziffern (0…9)
.5	16#0020	32	Der Buffer enthält ein Komma
.6	16#0040	64	Der Buffer enthält einen Punkt
.7	16#0080	128	Der Buffer enthält Buchstaben (a-z, A-Z)
.8	16#0100	256	Dateiende erreicht (End of File)
.13	16#2000	8192	Der eingelesene Wert ist länger als die Buffergröße. Es wurden zu viele Zeichen eingelesen.
.14	16#4000	16384	Steuerzeichen (< ASCII 32) im eingelesenen Text gefunden.
.15	16#8000	32768	Es ist ein Fehler aufgetreten. Der Fehlercode ist bei CsvRead.ErrorNo zu entnehmen

Ein detailliertes Beispiel für das Schreiben und Einlesen von Werten aus einer CSV ist am Ende des Kapitels aufgeführt.

8.7.1.1 METHODE CSV_TO_BOOL

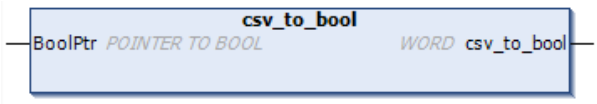


Abbildung 110: Methode csv_to_bool

Input Parameter:		
boolPtr	POINTER TO BOOL	Adresse einer Booleschen Variable, in die der eingelesene Wert aus der CSV-Datei geschrieben wird
Return-Wert:		
csv_to_bool	WORD	Rückgabewert der Methode: .0 (16#01) CR wurde nach dem Wert gefunden .1 (16#02) LF wurde nach dem Wert gefunden .2 (16#04) Trennzeichen wurde nach dem Wert gefunden .3 (16#08) Wert ist gültig .4 (16#10) Der Buffer enthält Ziffern (0…9) .5 (16#20) Der Buffer enthält ein Komma .6 (16#40) Der Buffer enthält einen Punkt

		.7 (16#80) Der Buffer enthält Buchstaben (a-z, A-Z) .8 (16#100) Dateiende erreicht (End of File) .13 (16#2000) Der eingelesene Wert ist länger als die Buffergröße. Es wurden zu viele Zeichen eingelesen .14 (16#4000) Steuerzeichen (< ASCII 32) im eingelesenen Text gefunden .15 (16#8000) Es ist ein Fehler aufgetreten. Der Fehlercode ist bei CsvRead.ErrorNo zu entnehmen

Diese Methode liest einen Wert aus einer CSV-Datei und speichert diesen in eine `BOOL` Variable, die mit dem Adress Operator `ADR` () an den Eingang `BoolPtr` angegeben wird.

```
csv_read.csv_to_bool(BoolPtr:=ADR(my_bool);
```

8.7.1.2 METHODE CSV_TO_BYTE

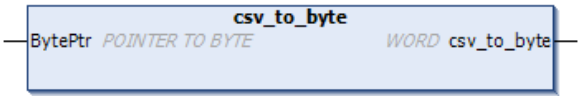


Abbildung 111: Methode csv_to_byte

Input Parameter:		
BytePtr	POINTER TO BYTE	Adresse einer Byte Variable, in die der eingelesene Wert aus der CSV-Datei geschrieben wird
Return-Wert:		
csv_to_byte	WORD	Rückgabewert der Methode: .0 (16#01) CR wurde nach dem Wert gefunden .1 (16#02) LF wurde nach dem Wert gefunden .2 (16#04) Trennzeichen wurde nach dem Wert gefunden .3 (16#08) Wert ist gültig .4 (16#10) Der Buffer enthält Ziffern (0…9) .5 (16#20) Der Buffer enthält ein Komma .6 (16#40) Der Buffer enthält einen Punkt .7 (16#80) Der Buffer enthält Buchstaben (a-z, A-Z) .8 (16#100) Dateiende erreicht (End of File) .13 (16#2000) Der eingelesene Wert ist länger als die Buffergröße. Es wurden zu viele Zeichen eingelesen .14 (16#4000) Steuerzeichen (< ASCII 32) im eingelesenen Text gefunden .15 (16#8000) Es ist ein Fehler aufgetreten. Der Fehlercode ist bei CsvRead.ErrorNo zu entnehmen

Diese Methode liest einen Wert aus einer CSV-Datei und speichert diesen in eine `BYTE` Variable, die mit dem Adress Operator `ADR`() an den Eingang `bytePtr` angegeben wird.

```
csv_read.csv_to_byte(bytePtr:=ADR(my_byte);
```

8.7.1.3 METHODE CSV_TO_DINT

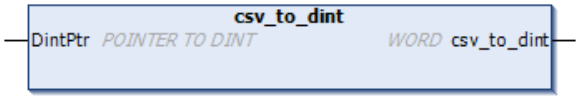


Abbildung 112: Methode csv_to_dint

Input Parameter:		
DintPtr	POINTER TO DINT	Adresse einer DINT Variable, in die der eingelesene Wert aus der CSV-Datei geschrieben wird
Return-Wert:		
csv_to_dint	WORD	Rückgabewert der Methode: .0 (16#01) CR wurde nach dem Wert gefunden .1 (16#02) LF wurde nach dem Wert gefunden .2 (16#04) Trennzeichen wurde nach dem Wert gefunden .3 (16#08) Wert ist gültig .4 (16#10) Der Buffer enthält Ziffern (0…9) .5 (16#20) Der Buffer enthält ein Komma .6 (16#40) Der Buffer enthält einen Punkt .7 (16#80) Der Buffer enthält Buchstaben (a-z, A-Z) .8 (16#100) Dateiende erreicht (End of File) .13 (16#2000) Der eingelesene Wert ist länger als die Buffergröße. Es wurden zu viele Zeichen eingelesen .14 (16#4000) Steuerzeichen (< ASCII 32) im eingelesenen Text gefunden .15 (16#8000) Es ist ein Fehler aufgetreten. Der Fehlercode ist bei CsvRead.ErrorNo zu entnehmen

Diese Methode liest einen Wert aus einer CSV-Datei und speichert diesen in eine `DINT` Variable, die mit dem Adress Operator `ADR()` an den Eingang `dintPtr` angegeben wird.

```
csv_read.csv_to_dint(DintPtr:=ADR(my_dint);
```

8.7.1.4 METHODE CSV_TO_DWORD

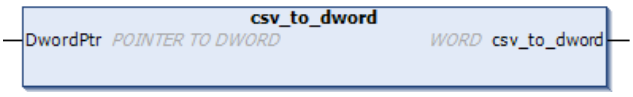


Abbildung 113: Methode csv_to_dword

Input Parameter:		
DwordPtr	POINTER TO DWORD	Adresse einer DWORD Variable, in die der eingelesene Wert aus der CSV-Datei geschrieben wird
Return-Wert:		
csv_to_dint	WORD	Rückgabewert der Methode: .0 (16#01) CR wurde nach dem Wert gefunden .1 (16#02) LF wurde nach dem Wert gefunden .2 (16#04) Trennzeichen wurde nach dem Wert gefunden .3 (16#08) Wert ist gültig

		.4 (16#10) Der Buffer enthält Ziffern (0…9) .5 (16#20) Der Buffer enthält ein Komma .6 (16#40) Der Buffer enthält einen Punkt .7 (16#80) Der Buffer enthält Buchstaben (a-z, A-Z) .8 (16#100) Dateiende erreicht (End of File) .13 (16#2000) Der eingelesene Wert ist länger als die Buffergröße. Es wurden zu viele Zeichen eingelesen .14 (16#4000) Steuerzeichen (< ASCII 32) im eingelesenen Text gefunden .15 (16#8000) Es ist ein Fehler aufgetreten. Der Fehlercode ist bei CsvRead.ErrorNo zu entnehmen

Diese Methode liest einen Wert aus einer CSV-Datei und speichert diesen in eine DWORD Variable, die mit dem Adress Operator ADR() an den Eingang DwordPtr angegeben wird.

```
csv_read.csv_to_dword(DwordPtr:=ADR(my_dword);
```

8.7.1.5 METHODE CSV_TO_INT

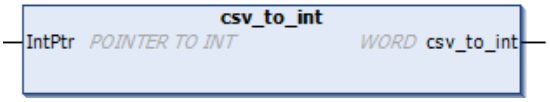


Abbildung 114: Methode csv_to_int

Input Parameter:		
IntPtr	POINTER TO INT	Adresse einer INT Variable, in die der eingelesene Wert aus der CSV-Datei geschrieben wird
Return-Wert:		
csv_to_int	WORD	Rückgabewert der Methode: .0 (16#01) CR wurde nach dem Wert gefunden .1 (16#02) LF wurde nach dem Wert gefunden .2 (16#04) Trennzeichen wurde nach dem Wert gefunden .3 (16#08) Wert ist gültig .4 (16#10) Der Buffer enthält Ziffern (0…9) .5 (16#20) Der Buffer enthält ein Komma .6 (16#40) Der Buffer enthält einen Punkt .7 (16#80) Der Buffer enthält Buchstaben (a-z, A-Z) .8 (16#100) Dateiende erreicht (End of File) .13 (16#2000) Der eingelesene Wert ist länger als die Buffergröße. Es wurden zu viele Zeichen eingelesen .14 (16#4000) Steuerzeichen (< ASCII 32) im eingelesenen Text gefunden .15 (16#8000) Es ist ein Fehler aufgetreten. Der Fehlercode ist bei CsvRead.ErrorNo zu entnehmen

Diese Methode liest einen Wert aus einer CSV-Datei und speichert diesen in eine INT Variable, die mit dem Adress Operator ADR() an den Eingang IntPtr angegeben wird.

```
csv_read.csv_to_int(IntPtr:=ADR(my_int);
```

8.7.1.6 METHODE CSV_TO_LREAL

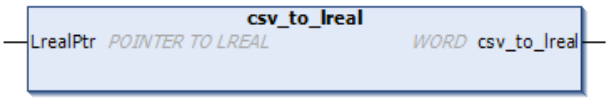


Abbildung 115: Methode csv_to_lreal

Input Parameter:		
LrealPtr	POINTER TO LREAL	Adresse einer LREAL Variable, in die der eingelesene Wert aus der CSV-Datei geschrieben wird
Return-Wert:		
csv_to_lreal	WORD	Rückgabewert der Methode: .0 (16#01) CR wurde nach dem Wert gefunden .1 (16#02) LF wurde nach dem Wert gefunden .2 (16#04) Trennzeichen wurde nach dem Wert gefunden .3 (16#08) Wert ist gültig .4 (16#10) Der Buffer enthält Ziffern (0…9) .5 (16#20) Der Buffer enthält ein Komma .6 (16#40) Der Buffer enthält einen Punkt .7 (16#80) Der Buffer enthält Buchstaben (a-z, A-Z) .8 (16#100) Dateiende erreicht (End of File) .13 (16#2000) Der eingelesene Wert ist länger als die Buffergröße. Es wurden zu viele Zeichen eingelesen .14 (16#4000) Steuerzeichen (< ASCII 32) im eingelesenen Text gefunden .15 (16#8000) Es ist ein Fehler aufgetreten. Der Fehlercode ist bei CsvRead.ErrorNo zu entnehmen

Diese Methode liest einen Wert aus einer CSV-Datei und speichert diesen in eine `LREAL` Variable, die mit dem Adress Operator `ADR()` an den Eingang `LrealPtr` angegeben wird.

```
csv_read.csv_to_Lreal(LrealPtr:=ADR(my_Lreal);
```

8.7.1.7 METHODE CSV_TO_REAL

Abbildung 116: Methode csv_to_real

Input Parameter:		
RealPtr	POINTER TO REAL	Adresse einer REAL Variable, in die der eingelesene Wert aus der CSV-Datei geschrieben wird
Return-Wert:		
csv_to_real	WORD	Rückgabewert der Methode: .0 (16#01) CR wurde nach dem Wert gefunden .1 (16#02) LF wurde nach dem Wert gefunden .2 (16#04) Trennzeichen wurde nach dem Wert gefunden .3 (16#08) Wert ist gültig

		.4 (16#10) Der Buffer enthält Ziffern (0…9) .5 (16#20) Der Buffer enthält ein Komma .6 (16#40) Der Buffer enthält einen Punkt .7 (16#80) Der Buffer enthält Buchstaben (a-z, A-Z) .8 (16#100) Dateiende erreicht (End of File) .13 (16#2000) Der eingelesene Wert ist länger als die Buffergröße. Es wurden zu viele Zeichen eingelesen .14 (16#4000) Steuerzeichen (< ASCII 32) im eingelesenen Text gefunden .15 (16#8000) Es ist ein Fehler aufgetreten. Der Fehlercode ist bei CsvRead.ErrorNo zu entnehmen

Diese Methode liest einen Wert aus einer CSV-Datei und speichert diesen in eine `REAL` Variable, die mit dem Adress Operator `ADR()` an den Eingang `RealPtr` angegeben wird.

```
csv_read.csv_to_real(RealPtr:=ADR(my_real);
```

8.7.1.8 METHODE CSV_TO_SINT

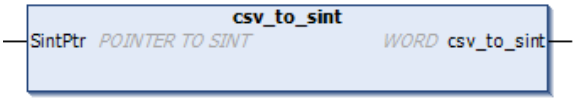


Abbildung 117: Methode csv_to_sint

Input Parameter:		
`SintPtr`	`POINTER TO SINT`	Adresse einer SINT Variable, in die der eingelesene Wert aus der CSV-Datei geschrieben wird
Return-Wert:		
`csv_to_sint`	`WORD`	Rückgabewert der Methode: .0 (16#01) CR wurde nach dem Wert gefunden .1 (16#02) LF wurde nach dem Wert gefunden .2 (16#04) Trennzeichen wurde nach dem Wert gefunden .3 (16#08) Wert ist gültig .4 (16#10) Der Buffer enthält Ziffern (0…9) .5 (16#20) Der Buffer enthält ein Komma .6 (16#40) Der Buffer enthält einen Punkt .7 (16#80) Der Buffer enthält Buchstaben (a-z, A-Z) .8 (16#100) Dateiende erreicht (End of File) .13 (16#2000) Der eingelesene Wert ist länger als die Buffergröße. Es wurden zu viele Zeichen eingelesen .14 (16#4000) Steuerzeichen (< ASCII 32) im eingelesenen Text gefunden .15 (16#8000) Es ist ein Fehler aufgetreten. Der Fehlercode ist bei CsvRead.ErrorNo zu entnehmen

Diese Methode liest einen Wert aus einer CSV-Datei und speichert diesen in eine `SINT` Variable, die mit dem Adress Operator `ADR()` an den Eingang `SintPtr` angegeben wird.

```
csv_read.csv_to_sint(SintPtr:=ADR(my_sint);
```

8.7.1.9 METHODE CSV_TO_WORD

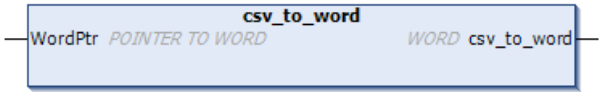


Abbildung 118: Methode csv_to_word

Input Parameter:		
WordPtr	POINTER TO WORD	Adresse einer WORD Variable, in die der eingelesene Wert aus der CSV-Datei geschrieben wird
Return-Wert:		
csv_to_word	WORD	Rückgabewert der Methode: .0 (16#01) CR wurde nach dem Wert gefunden .1 (16#02) LF wurde nach dem Wert gefunden .2 (16#04) Trennzeichen wurde nach dem Wert gefunden .3 (16#08) Wert ist gültig .4 (16#10) Der Buffer enthält Ziffern (0…9) .5 (16#20) Der Buffer enthält ein Komma .6 (16#40) Der Buffer enthält einen Punkt .7 (16#80) Der Buffer enthält Buchstaben (a-z, A-Z) .8 (16#100) Dateiende erreicht (End of File) .13 (16#2000) Der eingelesene Wert ist länger als die Buffergröße. Es wurden zu viele Zeichen eingelesen .14 (16#4000) Steuerzeichen (< ASCII 32) im eingelesenen Text gefunden .15 (16#8000) Es ist ein Fehler aufgetreten. Der Fehlercode ist bei CsvRead.ErrorNo zu entnehmen

Diese Methode liest einen Wert aus einer CSV-Datei und speichert diesen in eine `WORD` Variable, die mit dem Adress Operator `ADR()` an den Eingang `WordPtr` angegeben wird.

```
csv_read.csv_to_word(WordPtr:=ADR(my_word);
```

8.7.1.10 METHODE FILE_CLOSE

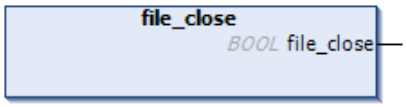


Abbildung 119: Methode file_close

Return-Wert:		
File_close	BOOL	TRUE = Datei konnte fehlerfrei geschlossen werden

```
result:=csv_read.file_close();
```

8.7.1.11 METHODE FILE_OPEN

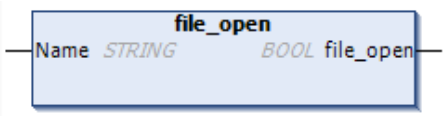


Abbildung 120: Methode file_open

Input Parameter:		
Name	STRING	Name der zu öffnenden CSV-Datei als String, z.B. 'a/my_params.csv'
Return-Wert:		
File_open	BOOL	TRUE = Die Datei konnte fehlerfrei geöffnet werden FALSE = Fehler beim Öffnen der Datei

```
result:=csv_read.file_open(Name:='a/my_params.csv');
```

8.7.2 CSVWRITE (PLMUTIL.LIBRARY)

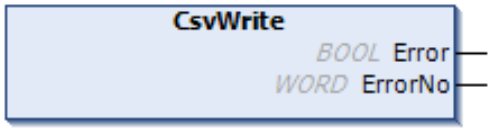


Abbildung 121: Funktion CsvWrite (PlmUtil.Library)

Output Paramter:		
Error	BOOL	Fehler beim Aufruf des Bausteins
ErrorNo	WORD	Fehlercode: 10 = Fehler beim Öffnen der Datei

Der Funktionsblock CsvWrite wird über seine Methoden verwendet. Folgende Methoden sind möglich:

- Csv_write.bool_to_csv(METH)
- Csv_write.byte_to_csv(METH)
- Csv_write.dint_to_csv(METH)
- Csv_write.dword_to_csv(METH)
- Csv_write.int_to_csv(METH)
- Csv_write.lreal_to_csv(METH)
- Csv_write.real_to_csv(METH)
- Csv_write.sint_to_csv(METH)
- Csv_write.string_to_csv(METH)
- Csv_write.stringPtr_to_csv(METH)
- Csv_write.word_to_csv(METH)
- Csv_write.file_open(METH)
- Csv_write.file_close(METH)
- Csv_write.set_open_file(METH)
- Csv_write.set_delimiter(METH)
- Csv_write.set_crlf(METH)

Als Beispiel sieht der Aufruf der Methode zum Schreiben eines WORD Wertes in eine bereits geöffnete CSV-Datei ohne ein CrLf aber mit Trennzeichen wie folgt aus:

```
csv_write.word_to_csv(Value:= my_word, CrLf:=FALSE);
```

Ein detailliertes Beispiel ist am Ende des Kapitels aufgeführt.

8.7.2.1 METHODE BOOL_TO_CSV

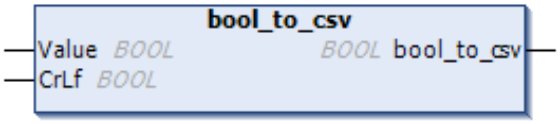


Abbildung 122: Methode bool_to_csv

Input Parameter:		
Value	BOOL	Boolesche Variable, die in die CSV-Datei geschrieben werden soll
CrLf	BOOL	FALSE = Es wird keine neue Zeile nach dem geschriebenen Wert erzeugt, aber es wird ein Trennzeichen geschrieben TRUE = Neue Zeile nach dem geschriebenen Wert
Return-Wert:		
Bool_to_csv	BOOL	TRUE = Der Wert konnte fehlerfrei in die CSV-Datei geschrieben werden FALSE = Fehler beim Schreiben des Wertes

Diese Methode schreibt einen BOOL Wert in eine CSV-Datei und fügt dahinter ein Trennzeichen ein:

```
csv_write.bool_to_csv (Value:=my_bool, CrLf:=FALSE);
```

8.7.2.2METHODE BYTE_TO_CSV

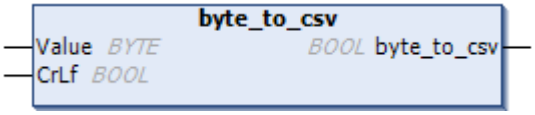


Abbildung 123: Methode byte_to_csv

Input Parameter:		
Value	BYTE	Eine Byte Variable, die in die CSV-Datei geschrieben werden soll
CrLf	BOOL	FALSE = Es wird keine neue Zeile nach dem geschriebenen Wert erzeugt, aber es wird ein Trennzeichen geschrieben TRUE = Neue Zeile nach dem geschriebenen Wert
Return-Wert:		
Bool_to_csv	BOOL	TRUE = Der Wert konnte fehlerfrei in die CSV-Datei geschrieben werden FALSE = Fehler beim Schreiben des Wertes

Diese Methode schreibt einen BYTE Wert in eine CSV-Datei und fügt dahinter ein Trennzeichen ein:

```
csv_write.byte_to_csv (Value:=my_byte, CrLf:=FALSE);
```

8.7.2.3 METHODE DINT_TO_CSV

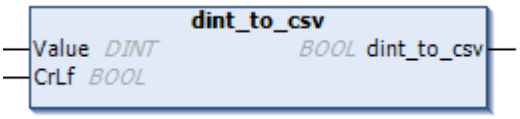


Abbildung 124: Methode dint_to_csv

Input Parameter:		
Value	DINT	Eine DINT Variable, die in die CSV-Datei geschrieben werden soll
CrLf	BOOL	FALSE = Es wird keine neue Zeile nach dem geschriebenen Wert erzeugt, aber es wird ein Trennzeichen geschrieben TRUE = Neue Zeile nach dem geschriebenen Wert
Return-Wert:		
dint_to_csv	BOOL	TRUE = Der Wert konnte fehlerfrei in die CSV-Datei geschrieben werden FALSE = Fehler beim Schreiben des Wertes

Diese Methode schreibt einen DINT Wert in eine CSV-Datei und fügt dahinter ein Trennzeichen ein:

```
csv_write.dint_to_csv (Value:=my_dint, CrLf:=FALSE);
```

8.7.2.4 METHODE DWORD_TO_CSV

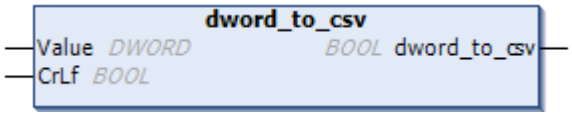


Abbildung 125: Methode dword_to_csv

Input Parameter:		
Value	dword	Eine DWORD Variable, die in die CSV-Datei geschrieben werden soll
CrLf	BOOL	FALSE = Es wird keine neue Zeile nach dem geschriebenen Wert erzeugt, aber es wird ein Trennzeichen geschrieben TRUE = Neue Zeile nach dem geschriebenen Wert
Return-Wert:		
dword_to_csv	BOOL	TRUE = Der Wert konnte fehlerfrei in die CSV-Datei geschrieben werden FALSE = Fehler beim Schreiben des Wertes

Diese Methode schreibt einen DWORD Wert in eine CSV-Datei und fügt dahinter ein Trennzeichen ein:

```
csv_write.dword_to_csv (Value:=my_dword, CrLf:=FALSE);
```

8.7.2.5 METHODE INT_TO_CSV

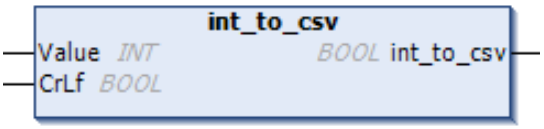


Abbildung 126: Methode int_to_csv

Input Parameter:		
Value	INT	Eine INT Variable, die in die CSV-Datei geschrieben werden soll
CrLf	BOOL	FALSE = Es wird keine neue Zeile nach dem geschriebenen Wert erzeugt, aber es wird ein Trennzeichen geschrieben TRUE = Neue Zeile nach dem geschriebenen Wert
Return-Wert:		
int_to_csv	BOOL	TRUE = Der Wert konnte fehlerfrei in die CSV-Datei geschrieben werden FALSE = Fehler beim Schreiben des Wertes

Diese Methode schreibt einen INT Wert in eine CSV-Datei und fügt dahinter ein Trennzeichen ein:

```
csv_write.int_to_csv (Value:=my_int, CrLf:=FALSE);
```

8.7.2.6 METHODE LREAL_TO_CSV

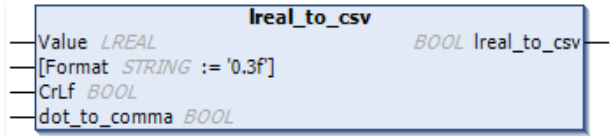


Abbildung 127: Methode lreal_to_csv

Input Parameter:		
Value	LREAL	Eine LREAL Variable, die in die CSV-Datei geschrieben werden soll
Format	STRING	Das Format, wie die LREAL Variable angezeigt wird. Z.B. entspricht '%0.3f' 3 Nachkommastellen ohne Begrenzung der Vorkommastellen
CrLf	BOOL	FALSE = Es wird keine neue Zeile nach dem geschriebenen Wert erzeugt, aber es wird ein Trennzeichen geschrieben TRUE = Neue Zeile nach dem geschriebenen Wert
Dot_to_comma	BOOL	TRUE = Der Punkt in der Zahl wird mit einem Komma ersetzt, 2.345 -> 2,345
Return-Wert:		
lreal_to_csv	BOOL	TRUE = Der Wert konnte fehlerfrei in die CSV-Datei geschrieben werden FALSE = Fehler beim Schreiben des Wertes

Diese Methode schreibt einen LREAL Wert in eine CSV-Datei und fügt dahinter ein Trennzeichen ein:

```
csv_write.lreal_to_csv (Value:=my_lreal,Format:='0.3f', CrLf:=FALSE,
                                dot_to_comma:=FALSE);
```

8.7.2.7 METHODE REAL_TO_CSV

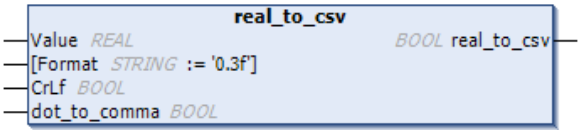


Abbildung 128: Methode real_to_csv

Input Parameter:		
Value	REAL	Eine REAL Variable, die in die CSV-Datei geschrieben werden soll
Format	STRING	Das Format, wie die REAL Variable angezeigt wird. Z.B. entspricht '%0.3f' 3 Nachkommastellen ohne Begrenzung der Vorkommastellen
CrLf	BOOL	FALSE = Es wird keine neue Zeile nach dem geschriebenen Wert erzeugt, aber es wird ein Trennzeichen geschrieben TRUE = Neue Zeile nach dem geschriebenen Wert
Dot_to_comma	BOOL	TRUE = Der Punkt in der Zahl wird mit einem Komma ersetzt, 2.345 -> 2,345
Return-Wert:		
real_to_csv	BOOL	TRUE = Der Wert konnte fehlerfrei in die CSV-Datei geschrieben werden FALSE = Fehler beim Schreiben des Wertes

Diese Methode schreibt einen REAL Wert in eine CSV-Datei und fügt dahinter ein Trennzeichen ein:

```
csv_write.real_to_csv (Value:=my_real,Format:='0.3f', CrLf:=FALSE,
                               dot_to_comma:=FALSE);
```

8.7.2.8 METHODE SINT_TO_CSV

Abbildung 129: Methode sint_to_csv

Input Parameter:		
Value	SINT	Eine SINT Variable, die in die CSV-Datei geschrieben werden soll
CrLf	BOOL	FALSE = Es wird keine neue Zeile nach dem geschriebenen Wert erzeugt, aber es wird ein Trennzeichen geschrieben TRUE = Neue Zeile nach dem geschriebenen Wert
Return-Wert:		
sint_to_csv	BOOL	TRUE = Der Wert konnte fehlerfrei in die CSV-Datei geschrieben werden FALSE = Fehler beim Schreiben des Wertes

Diese Methode schreibt einen SINT Wert in eine CSV-Datei und fügt dahinter ein Trennzeichen ein:

```
csv_write.sint_to_csv (Value:=my_sint, CrLf:=FALSE);
```

8.7.2.9 METHODE STRING_TO_CSV

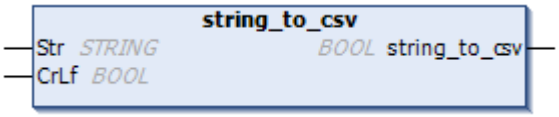


Abbildung 130: Methode string_to_csv

Input Parameter:		
Str	STRING	Eine STRING Variable, die in die CSV-Datei geschrieben werden soll
CrLf	BOOL	FALSE = Es wird keine neue Zeile nach dem geschriebenen Wert erzeugt, aber es wird ein Trennzeichen geschrieben TRUE = Neue Zeile nach dem geschriebenen Wert
Return-Wert:		
string_to_csv	BOOL	TRUE = Der Wert konnte fehlerfrei in die CSV-Datei geschrieben werden FALSE = Fehler beim Schreiben des Wertes

Diese Methode schreibt einen STRING in eine CSV-Datei und fügt dahinter ein Trennzeichen ein:

```
csv_write.string_to_csv (Value:=my_string, CrLf:=FALSE);
```

8.7.2.10 METHODE WORD_TO_CSV

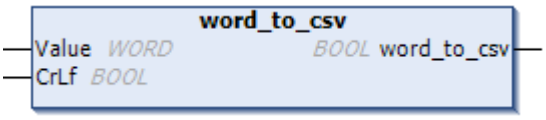


Abbildung 131: Methode word_to_csv

Input Parameter:		
Value	WORD	Eine WORD Variable, die in die CSV-Datei geschrieben werden soll
CrLf	BOOL	FALSE = Es wird keine neue Zeile nach dem geschriebenen Wert erzeugt, aber es wird ein Trennzeichen geschrieben TRUE = Neue Zeile nach dem geschriebenen Wert
Return-Wert:		
word_to_csv	BOOL	TRUE = Der Wert konnte fehlerfrei in die CSV-Datei geschrieben werden FALSE = Fehler beim Schreiben des Wertes

Diese Methode schreibt einen WORD Wert in eine CSV-Datei und fügt dahinter ein Trennzeichen ein:

```
csv_write.word_to_csv (Value:=my_word, CrLf:=FALSE);
```

8.7.2.11 METHODE SET_DELIMITER

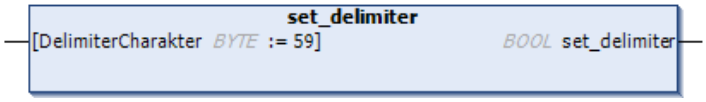


Abbildung 132: Methode set_delimiter

Input Parameter:		
DelimiterCharakter	BYTE	ASCII Code für das Trennzeichen, z.B. 59 = ;
Return-Wert:		
Set_delimiter	BOOL	

Diese Methode gibt das Trennzeichen zwischen den geschriebenen Werten in der CSV-Datei an.
`DelimiterCharakter: =59` steht für das Trennzeichen ';'.

```
csv_write.set_delimiter(DelimiterCharakter:=59);
```

8.7.2.12 METHODE FILE_OPEN

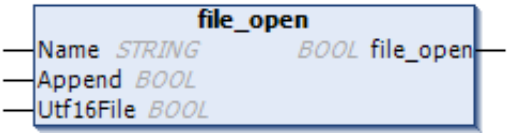


Abbildung 133: Methode file_open

Input Parameter:		
Name	STRING	Name der zu öffnenden CSV-Datei als STRING
Append	BOOL	FASLE = Neue CSV-Datei anlegen TRUE = Werte an eine bestehende CSV-Datei anhängen. Erstellt eine neue CSV-Datei, falls keine vorhanden ist
Utf16File	BOOL	Die CSV-Datei muss im Utf16 Dateiformat gespeichert sein
Return-Wert:		
File_open	BOOL	TRUE = CSV-Datei konnte fehlerfrei geöffnet werden FASLE = Fehler beim Öffnen der CSV-Datei

Diese Methode legt und öffnet eine neue CSV-Datei:

```
csv_write.file_open(Name:='a/my_file.csv', Append:=FALSE,
                    Utf16File:=FALSE);
```

8.7.2.13 METHODE FILE_CLOSE

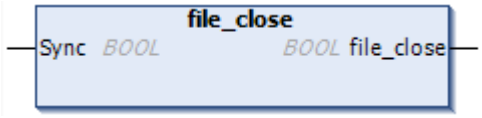


Abbildung 134: Methode file_close

Input Parameter:		
Sync	BOOL	TRUE = CSV-Datei schließen und ein SYNC ausführen
Return-Wert:		
File_close	BOOL	TRUE = Datei erfolgreich geschlossen

Diese Methode schließt eine geöffnete CSV-Datei:

```
csv_write.file_close(Sync:=TRUE);
```

8.7.2.14 METHODE SET_CRLF_CHARACTER

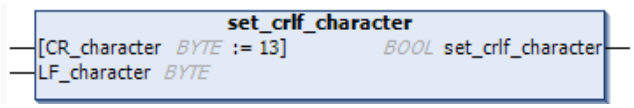


Abbildung 135: Methode set_crlf_character

Input Parameter:		
CR_character	BYTE	ASCII Code für Carraige Return, z.B. 13 = '$R'
LF_character	BYTE	ASCII Code für Line Feed, z.B. 10 = '$N'
Return-Wert:		
Set_crlf_character	BOOL	

8.8 PROGRAMMBEISPIEL PARAMETERDATEI LESEN/SCHREIBEN (BINÄRDATEI) (ST)

Zum Speichern eines Parametersatzes als Binärdatei bietet es sich an, alle Parameter zu einer STRUCT zusammenzufassen. Eine STRUCT ist eine fest definierte Zusammenfassung von verschiedenen Variablen, die in dieser Form zusammen als eine einzige Variable angesprochen werden können. STRUCTs werden unter CODESYS als DUT (Data Unit Type) angelegt und verwaltet.

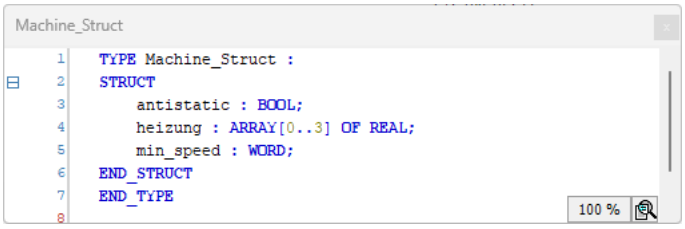


Abbildung 136: Definition Machine_Struct

Abbildung 136 zeigt als Beispiel eine STRUCT, die einen BOOL-Wert, ein Array von vier REAL-Werten und ein WORD zu einem neuen Variablentyp MACHINE_STRUCT zusammenfasst. Die globale Retain-Variable Maschine ist von diesem Typ und enthält den Parametersatz des Beispiels.

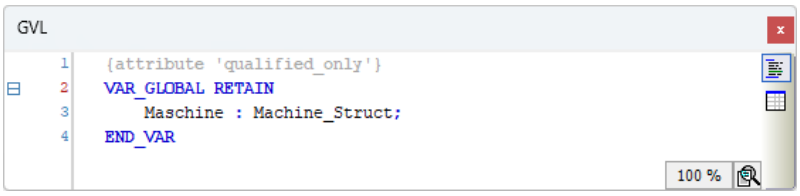


Abbildung 137:Maschine enthält den Parametersatz von Machine_Struct

Der Wert `min_speed` innerhalb von `Maschine` wird z.B. mit der Syntax `GVL.Maschine.min_speed` angesprochen, der vierte `REAL`-Wert aus dem Array `heizung` mit `GVL.Maschine.heizung[3]`.

Das Beispiel demonstriert sowohl den Umgang mit Binärdateien

Die Variablen `Do_Read_Bin` und `Do_Write_Bin` lösen die jeweiligen Dateioperationen aus.

Eine Schreiboperation läuft wie folgt ab:

1. Das Zyklusprogramm setzt zunächst `Write_Ok` und `Write_Err` auf `FALSE`
2. Das Zyklusprogramm setzt `Do_Write_Bin` auf `TRUE` und anschließend wieder auf `FALSE`
3. Der Programmblock führt die gewünschte Schreiboperation aus und bestätigt das Ende durch Setzen von `Write_Ok` oder `Write_Err`.
4. Das Zyklusprogramm wartet, bis entweder `Write_Ok` oder `Write_Err` `TRUE` ist. Bei `Write_Ok = TRUE` wurde der Parametersatz erfolgreich in die Datei geschrieben.

Eine Leseoperation läuft wie folgt ab:

1. Das Zyklusprogramm setzt zunächst `Read_Ok` und `Read_Err` auf `FALSE`
2. Das Zyklusprogramm setzt `Do_Read_Bin` auf `TRUE` und anschließend wieder auf `FALSE`.
3. Der Programmblock führt die gewünschte Leseoperation aus und bestätigt das Ende durch Setzen von `Read_Ok` oder `Read_Err`.
4. Das Zyklusprogramm wartet, bis entweder `Read_Ok` oder `Read_Err` `TRUE` ist. Bei `Read_Ok = TRUE` wurde der Parametersatz erfolgreich aus der Datei gelesen.

Während eine Lese- oder Schreiboperation läuft (d.h. gestartet und noch nicht mit `Ok` oder `Err` beendet wurde), dürfen die Daten in der `STRUCT`-Variablen `Maschine` nicht durch das Zyklusprogramm verändert werden.

Die Schreib- und Leseprogrammblöcke prüfen bei Aufruf zunächst, ob die Datei auf dem USB-Stick (c/) angelegt bzw. von diesem gelesen werden soll, und wenn ja, ob überhaupt ein USB-Stick vorhanden ist. Andernfalls kehren sie sofort mit Fehler zurück.

Bin_ST

```
1   PROGRAM Bin_ST
2   VAR
3       // Write
4       do_write    :BOOL;
5       write_ok    :BOOL;
6       write_err   :BOOL;
7
8       // Read
9       do_read     :BOOL;
10      read_ok     :BOOL;
11      read_err    :BOOL;
12
13      fid         :DWORD;
14      FilenameBin :String:='c/data.bin';
15  END_VAR
```

100 %

```
1   IF do_write THEN
2       do_write:=FALSE;
3       write_ok:=FALSE;
4       write_err:=FALSE;
5
6       IF NOT PlmStd.FileExist(FileName:='c/' ) THEN
7           write_err:=TRUE;
8           RETURN; // Kein USB-Stick eingesteckt/gemountet
9       END_IF
10
11      fid:=PlmStd.FileOpen(FileName:=FilenameBin, FileFlag:='w+');    // File öffnen
12      IF fid = 0 THEN
13          write_err:=TRUE;
14      ELSE
15          // Maschinenparameter als Block schreiben
16          PlmStd.FileWrite(FileId:=fid, Buffer:=ADR(gvl.Maschine), Size:=SIZEOF(GVL.Maschine));
17          PlmStd.FileClose(FileId:=fid);
18      END_IF
19  END_IF
20
21  IF do_read THEN
22      do_read:=FALSE;
23      read_ok:=FALSE;
24      read_err:=FALSE;
25
26      IF NOT PlmStd.FileExist(FileName:='c/' ) THEN
27          read_err:=TRUE;
28          RETURN;     // Kein USB_Stick eingesteckt/gemountet
29      END_IF
30
31      fid:=PlmStd.FileOpen(FileName:=FilenameBin, FileFlag:='r');     // File öffnen
32      IF fid = 0 THEN
33          read_err:=TRUE;
34      ELSE
35          // Maschienenparamter einlesen
36          PlmStd.FileRead(FileId:=fid, Buffer:= ADR(GVL.Maschine), Size:=SIZEOF(GVL.Maschine));
37          PlmStd.FileClose(FileId:=fid);
38          read_ok:=TRUE;
39      END_IF
40  END_IF
```

100 %

Abbildung 138: Programmblock zu Lesen und Schreiben einer Binärdatei (ST)

8.9 PROGRAMMBEISPIEL WERTE IN EINE CSV-DATEI SCHREIBEN UND ZURÜCKLESEN (ST)

csv_ST

```
PROGRAM csv_ST
VAR
    csv_write           :PlmUtil.CsvWrite;
    csv_read            :PlmUtil.CsvRead;
    result              :WORD;

    do_write, do_read   :BOOL;
    filename            :STRING:='a/daten.csv';

    my_word             :WORD:=123;
    my_dword            :DWORD:=12345;
    my_byte             :BYTE:=55;
    my_int              :INT:=456;
    my_real             :REAL:=12.3456;
    my_string           :STRING:='PLM800';

    read_word           :WORD;
    read_dword          :DWORD;
    read_byte           :BYTE;
    read_int            :INT;
    read_real           :REAL;
    read_string         :STRING;
END_VAR
```

100 %

```
IF do_write THEN
    do_write:=FALSE;
    //CSV-Datei öffnen
    IF csv_write.file_open(Name:=filename, Append:=FALSE, Utf16File:=FALSE) THEN
        //WORD Varible in CSV-Datei mit Trennzeichen schreiben
        csv_write.word_to_csv(Value:=my_word, CrLf:=FALSE);

        //DWORD Variable in CSV-Datei mit Trennzeichen schreiben
        csv_write.dword_to_csv(Value:=my_dword, CrLf:=FALSE);

        //BYTE Variable in CSV-Datei mit Trennzeichen schreiben
        csv_write.byte_to_csv(Value:=my_byte, CrLf:=FALSE);

        //INT Variable in CSV-Datei mit Trennzeichen schreiben
        csv_write.int_to_csv(Value:=my_int, CrLf:=FALSE);

        //REAL Variable in CSV-Datei mit Trennzeichen schreiben
        csv_write.real_to_csv(Value:=my_real, Format:='%0.3f', CrLf:=FALSE, dot_to_comma:=FALSE);

        //STRING Variable in CSV-Datei mit Zeilenende und neuer Zeile schreiben
        csv_write.string_to_csv(str:=my_string, CrLf:=TRUE);

        //CSV-Datei schließen, ohne einen Sync auszuführen
        csv_write.file_close(Sync:=FALSE);
    END_IF
END_IF
```

100 %

Abbildung 139: Beispiel Werte in eine CSV-Datei schreiben und auslesen in ST (Teil 1/2)

csv_ST

```
1   PROGRAM csv_ST
2   VAR
3       csv_write           :PlmUtil.CsvWrite;
4       csv_read            :PlmUtil.CsvRead;
5       result              :WORD;
6
7       do_write, do_read   :BOOL;
8       filename            :STRING:='a/daten.csv';
9
10      my_word             :WORD:=123;
11      my_dword            :DWORD:=12345;
12      my_byte             :BYTE:=55;
13      my_int              :INT:=456;
14      my_real             :REAL:=12.3456;
15      my_string           :STRING:='PLM800';
16
17      read_word           :WORD;
18      read_dword          :DWORD;
19      read_byte           :BYTE;
20      read_int            :INT;
21      read_real           :REAL;
22      read_string         :STRING;
23  END_VAR
```

100 %

```
28  IF do_read THEN
29      do_read:=FALSE;
30      //CSV-Datei öffnen
31      IF csv_read.file_open(Name:=filename) THEN
32          //Solange Werte aus CSV-Datei einlesen, bis Fehler aufgetreten ist, oder Ende erreicht wurde
33          REPEAT
34              //WORD Variable aus CSV-Datei lesen
35              result:=csv_read.csv_to_word(WordPtr:=ADR(read_word));
36              IF NOT result.3 OR NOT result.2 THEN    //Fehler, Wert einlesen nicht möglich oder kein Trennzeichen nach dem Wert
37                  CONTINUE;                           //In der Repeatschleife zu Until springen
38              END_IF
39
40              //DWORD Variable aus CSV-Datei lesen
41              result:=csv_read.csv_to_dword(DwordPtr:=ADR(read_dword));
42              IF NOT result.3 OR NOT result.2 THEN    //Fehler, Wert einlesen nicht möglich oder kein Trennzeichen nach dem Wert
43                  CONTINUE;                           //In der Repeatschleife zu Until springen
44              END_IF
45
46              //BYTE Variable aus CSV-Datei lesen
47              result:=csv_read.csv_to_byte(BytePtr:=ADR(read_byte));
48              IF NOT result.3 OR NOT result.2 THEN    //Fehler, Wert einlesen nicht möglich oder kein Trennzeichen nach dem Wert
49                  CONTINUE;                           //In der Repeatschleife zu Until springen
50              END_IF
51
52              //INT Variable aus CSV-Datei lesen
53              result:=csv_read.csv_to_int(IntPtr:=ADR(read_int));
54              IF NOT result.3 OR NOT result.2 THEN    //Fehler, Wert einlesen nicht möglich oder kein Trennzeichen nach dem Wert
55                  CONTINUE;                           //In der Repeatschleife zu Until springen
56              END_IF
57
58              //REAL Variable aus CSV-Datei lesen
59              result:=csv_read.csv_to_real(RealPtr:=ADR(read_real));
60              IF NOT result.3 OR NOT result.2 THEN    //Fehler, Wert einlesen nicht möglich oder kein Trennzeichen nach dem Wert
61                  CONTINUE;                           //In der Repeatschleife zu Until springen
62              END_IF
63
64              //STRING Variable aus CSV-Datei einlesen
65              result:=csv_read.next_value(BufferPtr:=ADR(read_string), BufferSize:=SIZEOF(read_string));
66              IF NOT result.3 OR (result <> 16#08) THEN   //Fehler, Wert einlesen nicht mögliche oder kein CR oder LF
67                  CONTINUE;                               //In der Repeatschleife zu Until springen
68              END_IF
69          UNTIL
70              TRUE
71          END_REPEAT
72          //CSV-Datei schließen
73          csv_read.file_close();
74      END_IF
75  END_IF
```

100 %

Abbildung 140: Beispiel für Werte in eine CSV-Datei schreiben und auslesen in ST (Teil2/2)

Eine Schreiboperation läuft wie folgt ab:

1. Das Zyklusprogramm setzt `do_write` auf `TRUE` und anschließend wieder auf `FALSE`
2. Das Zyklusprogramm öffnet mit `csv_write.file_open()` die CSV-Datei. Ist der Rückgabewert des Funktionsblocks `TRUE` wird die IF-Schleife ausgeführt.
3. Der Programmblock führt innerhalb der IF-Schleife die gewünschten Schreiboperationen aus
4. Nach dem Schreiben der Werte in die CSV-Datei wird die Datei mit `csv_write.fileclose()` geschlossen.

Eine Leseoperation läuft wie folgt ab:

1. Das Zyklusprogramm setzt zunächst `do_read` und `do_read` auf `FALSE`
2. Das Zyklusprogramm öffnet die CSV-Datei mit `csv_read.file_open()`. Ist der Rückgabewert des Funktionsblocks `TRUE` wird die REPEAT-Schleife ausgeführt.
3. Der Programmblock führt die gewünschten Leseoperationen innerhalb der REPEAT-Schleife aus. Bei einem Fehler wird die REPEAT-Schleife abgebrochen.
4. Nach dem Lesen der Werte aus der CSV-Datei wir die Datei mit `csv_read.file_close()` wieder geschlossen.

8.10 SCHREIBEN UND LESEN VON WERTEN ALS CSV-LISTE

Eine weitere Möglichkeit, Werte in eine CSV-Datei zu schreiben, besteht darin, diese zunächst in einer Liste zu sammeln und anschließend die gesamte Liste in die CSV-Datei zu schreiben. Der Vorteil dieses Vorgehens liegt darin, dass die Anzahl der Schreib- und Lesevorgänge deutlich reduziert wird. Anstatt jeden Wert einzeln in die CSV-Datei zu schreiben oder daraus zu lesen, werden alle in der Liste enthaltenen Werte in einem einzigen Schreib- bzw. Lesevorgang verarbeitet. Dadurch kann die Performance, insbesondere bei großen Datenmengen, erheblich verbessert werden.

Hierfür stehen zwei Funktionsbausteine zur Verfügung, `CsvListWrite_fb` und `CsvListRead_fb` aus dem Ordner **CSV by ConfigList**.

8.10.1 CSVLISTWRITE_FB

Abbildung 141: Funktionsblock CsvListWrite_fb (PlmUtil.Library)

Input Parameter:		
SingleStruct	BOOL	Es wird von dem angelegten Struct nur ein Datensatz in die CSV-Datei geschrieben, auch wenn der Struct als Array angelegt wurde. Der Wert val_count von

		der Struktur CSV_VAL_STRUCT wird in diesem Fall ignoriert, auch wenn dieser größer 0 ist.
Utf16_File	BOOL	Die CSV-Datei wird im Utf16 Format erstellt
Csv_List_Adr	DWORD	Adresse der CSV-Beschreibungsvariablen
DataAddress	DWORD	Adresse der Datenstruktur, die in die CSV-Datei gesichert werden soll
Csv_wr	CsvWrite	Eingang für die Funktion CsvWrite().
Return-Wert:		
Error	BOOL	TRUE = Fehler beim Schreiben der CSV-Liste
ErrMsg	STRING	Enthält den Fehlertext. 'No List Address'

8.10.2 CSVLISTREAD_FB

Abbildung 142: Funktionsblock CsvListRead_fb (PlmUtil.Library)

Input Parameter:		
SingleStuct	BOOL	Es wird von dem angelegten Struct nur ein Datensatz aus der CSV-Datei gelesen, auch wenn der Struct als Array angelegt wurde. Der Wert val_count von der Struktur CSV_VAL_STRUCT wird in diesem Fall ignoriert, auch wenn dieser größer 0 ist.
wStringStrictly	BOOL	
Csv_List_Adr	DWORD	Adresse der CSV-Beschreibungsvariablen
DataAddress	DWORD	Adresse der Datenstruktur, in die die eingelesenen Werte aus der CSV-Datei geschrieben werden. Es muss nicht zwingend die Datenstruktur sein, die am Eingang Csv_List_Adr angegeben wurde. Es kann hier eine zweite Datenstruktur genutzt werden, die den gleichen Aufbau hat.
Csv_rd	CsvRead	Eingang für die Funktion CsvRead().
LineCount	DWORD	Anzahl der eingelesenen Zeilen aus der CSV-Datei.
Return-Wert:		
Error	BOOL	TRUE = Fehler beim Einlesen von Werten aus der CSV-Datei
ErrMsg	STRING	Enthält den Fehlertext. 'No List Adress' 'Delimiter instead of Cr/Lf' 'Cr/Lf instead of Delimiter' 'Read Error (End of File)'
ErrLine	DWORD	Zeigt die Zeile mit dem Fehler an
ErrColumn	WORD	Zeigt die Spalte mit dem Fehler an
ValueCount	DWORD	Anzahl der eingelesenen Werte

8.11 PROGRAMMBEISPIEL WERTE ALS CSVLIST IN EINE CSV-DATEI SCHREIBEN (ST)

CONFIG_STRUCT

```
1  TYPE CONFIG_STRUCT :
2  STRUCT
3      name        :STRING:='Anlage';      // Beschreibung
4      temperatur  :REAL:=21.0;            // Temperaturwert
5      druck       :WORD:=2;               // Druck
6      setpoint    :ARRAY[0..9] OF REAL;   // Vorgaben
7  END_STRUCT
8  END_TYPE
9
```

100 %

Abbildung 143: Programmbeispiel Werte als CSV-Liste in eine CSV-Datei schreiben und auslesen in ST (Teil 1/3)

CsvList_ST

```
PROGRAM CsvList_ST
VAR
    init_done           :BOOL;
    i                   :WORD;
    do_write_list       :BOOL;
    do_read_list        :BOOL;

    rd_csv              :PlmUtil.CsvRead;
    wr_csv              :PlmUtil.CsvWrite;
    rd_list             :PlmUtil.CsvListRead_fb;
    wr_list             :PlmUtil.CsvListWrite_fb;

    csv_config          :ARRAY[0..20] OF Plmutil.CSV_VAL_STRUCT;

    config_data         :CONFIG_STRUCT;
    config_data2        :CONFIG_STRUCT;

    Lines               :DWORD;
END_VAR
```

```
IF NOT init_done THEN
    init_done:=TRUE;

    i:=0;
    (******* Speicherbeschreibung konfigurieren ******)
    csv_config[i].val_type:=16#8000;                // 16#8000 = Startadresse des Datensatzes. Hier fängt die Beschreibung der Datan an
    csv_config[i].val_size:=SIZEOF(config_data);    // Länge eines Datensatzes
    csv_config[i].val_count:=1;                     // Datensatz ist kein Array
    csv_config[i].val_adr:=ADR(config_data);        // Startadresse des Datensatzes
    i:=i+1;

    (****** erste Variable String ******)
    csv_config[i].val_type:=5;                          // 5 = Variable ist ein String
    csv_config[i].val_size:=SIZEOF(config_data.name);   // Länge des Strings
    csv_config[i].val_count:=1;                         // String-Variable ist kein Array
    csv_config[i].val_adr:=ADR(config_data.name);       // Adresse der String-Variablen
    i:=i+1;

    (****** zweite Variable Temperatur ******)
    csv_config[i].val_type:=16#303;                     // 16#003 = REAL Variable, 16#300 = 3 Nachkommastellen
    csv_config[i].val_size;                             // Nur bei Strings nötig
    csv_config[i].val_count:=1;                         // Temperatur-Variable ist kein Array
    csv_config[i].val_adr:=ADR(config_data.temperatur); // Adresse der Temperatur-Variablen
    i:=i+1;

    (****** dritte Varaibel Druck ******)
    csv_config[i].val_type:=1;                          // 1 = WORD Variable
    csv_config[i].val_size;                             // Nur bei String nötig
    csv_config[i].val_count:=1;                         // Drukc-Variable ist kein Array
    csv_config[i].val_adr:=ADR(config_data.druck);      // Adresse der Druck-Variablen
    i:=i+1;

    (****** vierte Varaible Array Setpoint ******)
    csv_config[i].val_type:=16#303;                     // 16#003 = REAL Variable, 16#300 = 3 Nachkommastellen
    csv_config[i].val_size;                             // Nur bei Strings nötig
    csv_config[i].val_count:=10;                        // Varaible Setpoint ist ein Array mit 10 Elementen
    csv_config[i].val_adr:=ADR(config_data.setpoint);   // Adresse der Setpoint-Variablen
    i:=i+1;

    (****** Ende der Liste ******)
    csv_config[i].val_type:=0;
    csv_config[i].val_size:=0;
    csv_config[i].val_count:=0;
    csv_config[i].val_adr:=0;
END_IF

IF do_write_list THEN
```

Abbildung 144: Programmbeispiel Werte als CSV-Liste in eine CSV-Datei schreiben und auslesen in ST (Teil 2/3)

CsvList_ST

```
1   PROGRAM CsvList_ST
2   VAR
3       init_done           :BOOL;
4       i                   :WORD;
5       do_write_list       :BOOL;
6       do_read_list        :BOOL;
7
8       rd_csv              :PlmUtil.CsvRead;
9       wr_csv              :PlmUtil.CsvWrite;
10      rd_list             :PlmUtil.CsvListRead_fb;
11      wr_list             :PlmUtil.CsvListWrite_fb;
12
13      csv_config          :ARRAY[0..20] OF Plmutil.CSV_VAL_STRUCT;
14
15      config_data         :CONFIG_STRUCT;
16      config_data2        :CONFIG_STRUCT;
17
18      Lines               :DWORD;
19  END_VAR
20
```

100 %

```
47  IF do_write_list THEN
48      do_write_list:=FALSE;
49      // Datei öffnen
50      IF wr_csv.file_open(Name:='a/config.csv', Append:=FALSE, Utf16File:=FALSE) THEN
51          wr_list(
52              SingleStruct:=,
53              Utf16_File:=,
54              Csv_List_Adr:=ADR(csv_config),      // Adresse der CSV-Datenbeschreibungsvariablen
55              DataAddress:=ADR(config_data),      // Adresse der Daten, welche gespeichert werden sollen
56              Error=>,
57              ErrMsg=>,
58              csv_wr:=wr_csv);                    // Schreiben ausführen
59          wr_csv.file_close(Sync:=FALSE);         // Datei schließen
60      END_IF
61  END_IF
62
63  IF do_read_list THEN
64      do_read_list:=FALSE;
65      // Datei öffnen
66      IF rd_csv.file_open(Name:='a/config.csv') THEN
67          rd_list(
68              SingleStruct:=,
69              wStringStrictly:=,
70              Csv_List_Adr:=ADR(csv_config),      // Adresse der CSV-Dataenbeschreibungsvariablen
71              DataAddress:=ADR(config_data2),     // Adresse der Daten, in die die eingelenen Werte abgelegt werden sollen
72              Error=>,
73              ErrMsg=>,
74              ErrLine=>,
75              ErrColumn=>,
76              ValueCount=>,
77              csv_rd:=rd_csv,
78              LineCount:=Lines);
79          rd_csv.file_close();
80      END_IF
81  END_IF
82
```

100 %

Abbildung 145: Programmbeispiel Werte als CSV-Liste in eine CSV-Datei schreiben und auslesen in ST (Teil 3/3)

Funktionsbeschreibung:

Zunächst wird eine Struktur vom Typ `CONFIG_STRUCT (STRUCT)` erstellt, welche alle Variablen enthält, die als Datensatz in einer CSV-Datei gespeichert werden sollen (siehe Abbildung 143).
In diesem Beispiel umfasst die Struktur die vier Variablen `name`, `temperatur`, `druck` und `setpoint`.
Anschließend werden im Programm zwei Instanzen dieser Struktur angelegt: `config_data` und `config_data2` (siehe Abbildung 144, Zeilen 15,16).
Die Struktur `config_data` enthält die Werte, die in die CSV-Datei geschrieben werden. Die Struktur `config_data2` dient hingegen als Zielstruktur für die Werte, die aus der CSV-Datei eingelesen werden.
Das Array `csv_config` enthält die Datenbeschreibung der Variablen aus der Struktur `CONFIG_STRUCT`.
Die einmalige Beschreibung der Variablen in das Array `csv_config` erfolgt zwischen den Zeilen 1 bis 45.

Für die Beschreibung einer Variable müssen grundsätzlich deren Datentyp sowie die Parameter Länge, Anzahl und Speicheradresse übergeben werden.

Mittels `do_write_list = TRUE` wird die CSV-Datei geöffnet und die „Liste" mit den Variablen aus der Datenstruktur `config_data` in die Datei geschrieben. Anschließend wird die CSV-Datei wieder geschlossen. Abbildung 145, Zeilen 47 – 61.

Mittels `do_read_list = TRUE` wird die CSV-Datei geöffnet und die gelesenen Variablen aus der CSV-Datei in die Datenstriktur `config_data2` geschrieben. Anschließend wird die CSV-Datei wieder geschlossen. Abbildung 145, Zeilen 63 – 81.

Der Vorteil hier ist, dass die Liste mit den Variablen in einem Vorgang geschrieben oder gelesen werden können, was Performance spart.

9 ETHERNET EINSTELLUNGEN ABFRAGEN UND ÄNDERN

9.1 ALLGEMEINES

Bei Verwendung der Ethernet-Schnittstellen der PLM-Steuerungen sind diese vorher geeignet zu parametrieren. Dies kann über das Webconfig erfolgen oder was wünschenswert ist, dass die Parametrierung der Ethernet-Schnittstellen aus dem IEC-Programm vom Anwender vorgenommen werden kann.

Hierfür steht die Bibliothek **PlmStandard.Library** mit dem Ordner **Network** zur Verfügung.

9.2 BENÖTIGTE BIBLIOTHEKEN

Benötigt wird die folgende Bibliothek:

PlmStandard >V3.5.16.28

Die angegebene Bibliothek muss vom Projekt geladen werden, damit die Funktionen zur Verfügung stehen. Klicken Sie dazu auf den ***Bibliotheksverwalter*** > ***Bibliothek hinzufügen*** > ***Sonstige*** > ***PlmStandard*** > ***OK***. Der benötigte Baustein zu Parametrierung der Ethernet-Schnittstelle befindet sich in dem Ordner ***Network.*** in der Bibliothek **PlmStandard**.

9.2.1 MANAGEINTERFACE (PLMSTANDARD.LIBRARY)

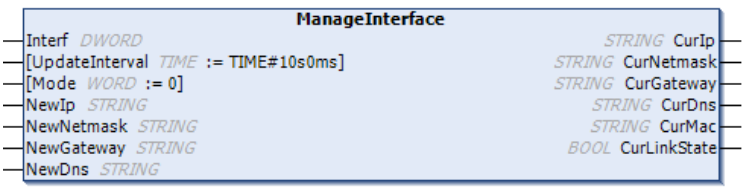


Abbildung 146: funktion ManageInterface (PlmStandard.Library)

Input Parameter:		
Interf	DWORD	Interface Index, 0 = ETH0, 1 = ETH1
UpdateInterval	TIME	Zeitangabe, wie oft die Werte für CurIp, CurNetmask, CurGateway, CurDns, CurMac und CurLinkState aktualisiert werden sollen. Das kann CPU-Last erzeugen, daher sollte die Zeit nicht zu klein gewählt werden. Standard sind T#10s
Mode	WORD	Folgende Modi sind möglich: 0 = Änderungen nach einem Neustart übernehmen 1 = Änderungen sofort übernehmen (trennt alle Netzwerkverbindungen) +2 = Änderungen werden gecancelt, Modus ist anschließend 0/1 +4 = Um Änderungen zu bestätigen, benötigt mindestens einen Zyklus, Modus ist anschließend 0/1
NewIp	STRING	String Variable für neue statische IP-Adresse.Z.B. '192.168.10.231' Wenn hier kein Wert angegeben wird, dann ist DHCP aktiv
NewNetmask	STRING	String Variable für neue Netmask
NewGateway	STRING	String Variable für neuen Gateway
NewDns	STRING	String Variable für neuen Dns Server
Return-Wert:		
CurIp	STRING	Ausgabe der aktuellen IP-Adresse der Steuerung
CurNetmask	STRING	Ausgabe der aktuellen Netmask der Steuerung
CurGateway	STRING	Ausgabe der aktuellen IP-Adresse vom Gateway der Steuerung
CurDns	STRING	Ausgabe der aktuellen IP-Adresse vom DNS Server der Steuerung
CurMac	STRING	Ausgabe der aktuellen Mac-Adresse der Steuerung
CurLinkState	BOOL	Aktueller Verbindungsstatus TRUE = Verbindungen FALSE = keine Verbindung

9.2.2 FUNKTIONSWEISE

Der Funktionsbaustein `ManageInterface` ist zyklisch aufzurufen. Am Eingang `Interf` wird die zu verwendende Ethernet-Schnittstelle der Steuerung parametriert.

Bei Steuerungen mit einer einzelnen Ethernet-Schnittstelle (`Eth0`) ist der Wert `0` an `Interf` zu übergeben. Verfügt die Steuerung über eine zusätzliche Schnittstelle (`Eth1`) und soll diese genutzt werden, ist der Wert `1` an `Interf` anzulegen.

Für den gleichzeitigen Betrieb beider Ethernet-Schnittstellen sind zwei Instanzen des Funktionsbausteins `ManageInterface` erforderlich. Dabei wird einer Instanz der Wert `0` (für `Eth0`) und der zweiten Instanz der Wert `1` (für `Eth1`) am jeweiligen Eingang `Interf` zugewiesen.

Über den `Mode` Eingang kann gesteuert werden, wann eine Änderung der IP-Adressen erfolgen soll.

Die Ausgänge `CurIp`, `CurNetmask`, `CurGateway` und `CurDns` stellen die aktuell konfigurierten Netzwerkparameter der Steuerung da. Der Ausgang `CurMac` liefert die MAC-Adresse der jeweiligen

Ethernet-Schnittstelle, während `CurLinkState` den aktuellen Verbindungsstatus (Link aktiv/inaktiv) signalisiert.

Diese Werte sind zur Anzeige in der Visualisierung der Steuerung vorzusehen.
An den Eingängen `NewIp`, `NewNetmask`, `NewGateway` und `NewDns` können neue Netzwerkparameter übergeben werden.

Eine Übernahme erfolgt ausschließlich bei Abweichung von den aktuellen Werten an den Ausgängen `CurIp`, `CurNetmask`, `CurGateway` und `CurDns`. Stimmen die Werte überein, wird keine Aktualisierung durchgeführt.

Für die Eingänge `NewIp`, `NewNetmask`, `NewGateway` und `NewDns` sind String-Variablen zu verwenden, die in der Visualisierung editierbar sind.

9.2.3 BEISPIEL KONFIGURIEREN DER ETH0 UND ETH1 (ST)

LAN

```
PROGRAM LAN
VAR
    mode: WORD := 1;
    eth0cfg : PlmStd.ManageInterface;
    eth1cfg : PlmStd.ManageInterface;
    newIp, newNetmask, newGateway, newDns: ARRAY[0..1] OF STRING;
END_VAR
```

100 %

```
// IP-Adressen ETH0 parametrieren
eth0cfg(
    Interf:=0 ,
    UpdateInterval:=T#10S ,
    Mode:= mode,
    CurIp=> ,
    CurNetmask=> ,
    CurGateway=> ,
    CurDns=> ,
    CurMac=> ,
    CurLinkState=> ,
    NewIp:= newIp[0],
    NewNetmask:= newNetmask[0],
    NewGateway:= newGateway[0],
    NewDns:= newDns[0]);

// IP-Adressen ETH1 parametrieren
eth1cfg(
    Interf:=1 ,
    UpdateInterval:=T#10S ,
    Mode:= mode,
    CurIp=> ,
    CurNetmask=> ,
    CurGateway=> ,
    CurDns=> ,
    CurMac=> ,
    CurLinkState=> ,
    NewIp:= newIp[1],
    NewNetmask:= newNetmask[1],
    NewGateway:= newGateway[1],
    NewDns:= newDns[1]);
```

100 %

Abbildung 147: Funktionsbaustein ManageInterface (PlmStandard.Library)

Das Beispiel zeigt für zwei Ethernet-Schnittstellen den Aufruf des Funktionsbausteins `ManageInterface`. Der Aufruf muss zyklisch erfolgen.

Wie die Ausgabe und Eingabe einer IP-Adresse aussehen kann ist in der folgenden *Abbildung 148* dargestellt.

Auf die gleiche Art und Weise können die anderen Netzwerk-Parameter angezeigt und editiert werden:

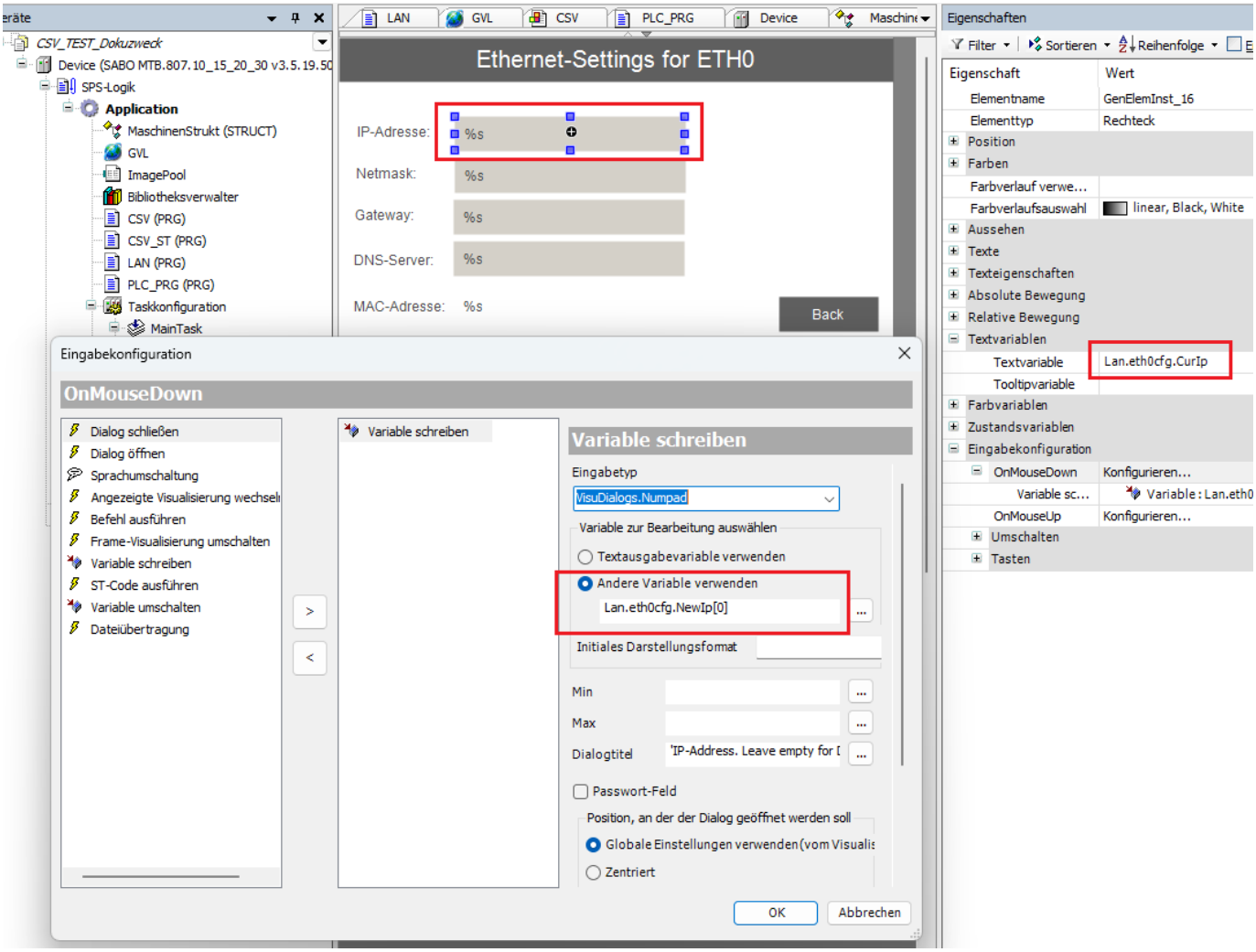


Abbildung 148: IP-Adresse in der Visualisierung darstellen und editieren

Für die Ausgabe der IP-Adresse kann die Textvariable `Lan.eth0cfg.CurIp` verwendet werden, zum Editieren der IP-Adresse wird die Variable `Lan.eth0cfg.NewIp[0]` genutzt. Das gleiche Prinzip kann auf die anderen IP-Adressen angewendet werden.

10 MODBUS RTU (SLAVE)

10.1 ALLGEMEINES

Modbus RTU erlaubt einen einfachen Datenaustausch zwischen einem Master und einem oder mehreren Slaves. Der Master spricht dabei die Slaves über eine serielle Schnittstelle an, üblicherweise RS485. Für Modbus-Verbindungen über Ethernet steht Modbus TCP zur Verfügung (siehe Abschnitte 13 und 14).

Dieser Abschnitt beschreibt die Arbeitsweise einer PLM-Steuerung als Modbus RTU Slave.

Weitere Informationen zum Modbus-Protokoll stehen auf der Website der Modbus-Nutzerorganisation unter http://modbus.org/ zur Verfügung.

Ein Modbus RTU-Slave stellt dem Master ein Werte-Array zur Verfügung. Die Werte des Arrays können vom Master abgefragt (Read) oder beschrieben werden (Write). Das Modbus-Protokoll erlaubt den Zugriff auf Words (Register) oder Bits (Coils). Die Art des jeweiligen Zugriffs wird über einen Function Code (FC) beim Zugriff festgelegt.

Das Anwenderprogramm stellt ein WORD-Array beliebiger Größe für Word-Zugriffe bereit, sowie ein BYTE-Array beliebiger Größe für Bit-Zugriffe. Mit diesen wird der Baustein `RtuSlave` zyklisch aufrufen. Anschließend steht der Slave-Dienst dem Master über die festgelegte serielle Schnittstelle zur Verfügung.

Die verwendete serielle Schnittstelle muss vor der Verwendung geeignet initialisiert werden (siehe Abschnitt 7).

10.2 SPEZIFIKATIONEN

- Modbus RTU Slave (Slave-ID 1...247)
- Verwendung einer beliebigen seriellen Schnittstelle, z.B. RS485
- Daten-Arrays: max. 65536 Words für Register-Zugriffe, max. 65536 Bytes für Bit-Zugriffe
- Unterstützte Function-Codes (FC):
  - 1 (Read Coils)
  - 2 (Read Discrete Inputs)
  - 3 (Read Holding Registers)
  - 4 (Read Input Registers)
  - 5 (Write Single Coil)
  - 6 (Write Single Register)
  - 15 (Write Multiple Coils)
  - 16 (Write Multiple Registers)

10.2.1 BENÖTIGTE BIBLIOTHEKEN

Benötigt werden die folgenden Bibliotheken:

PlmStandard > V3.5.16.28
PlmModbus > V3.5.16.17

Die angegebenen Bibliotheken müssen vom Projekt geladen werden, damit die Funktionen zur Verfügung stehen. Klicken Sie dazu auf den ***Bibliotheksverwalter*** > ***Bibliothek hinzufügen*** > ***Sonstige*** > ***PlmStandard*** > ***OK***. Wiederholen Sie den Schritt für die Bibliothek **PlmModbus**. Die benötigten Bausteine zu Parametrierung der seriellen-Schnittstelle befinden sich in dem Ordner **SerialCommuniction.** in der Bibliothek **PlmStandard**. Der benötigte Baustein für Modbus RTU Slave ist in dem Ordner **Modbus RTU Slave** der Bibliothek **PlmModbus**.

Abbildung 149: Funktionsblock RtuSlave (PlmModbus.Library)

Input Parameter:		
`ComPort`	`WORD`	`Nummer des ComPorts, z.B. 0 für erste serielle Schnittstelle`
`SlaveId`	`BYTE`	`Slave-Adresse (0…247)`

BitData	POINTER TO ARRAY[0..65535]OF BYTE	Adresse des Byte Arrays für Function Codes 1,2,5,15; kann mit Adresse des Word Arrays identisch sein
NumBitData	WORD	Größe des Byte Arrays (Anzahl der Bytes in BitData)
WordData	POINTER TO ARRAY[0..65535] OF WORD	Adresse des Word Arrays für Function Codes 3,4,6,16
NumWordData	WORD	Größe des Word Arrays (Anzahl der Words in WordData)
Mode	WORD	Wird momentan nicht benötigt
Enable	BOOL	Enable Flag TRUE = Slave aktiv FALSE = Slave inaktiv
Return-Wert:		
RequestOk	BOOL	TRUE, wenn während des letzten Aufrufs eine Anfrage erfolgreich bearbeitet wurde FALSE, wenn der Baustein nicht aufgerufen wurde oder die Bearbeitung nicht erfolgreich war

Der Funktionsblock `RtuSlave` muss zyklisch aufgerufen werden. Bei der Auswahl einer seriellen Schnittstelle (`ComPort`) ist sicherzustellen, dass diese nicht von anderen Diensten verwendet wird. Die verwendete serielle Schnittstelle muss vor der Verwendung geeignet initialisiert werden (siehe *Abschnitt 7*).

10.2.2 PROGRAMMBEISPIEL (FUP)

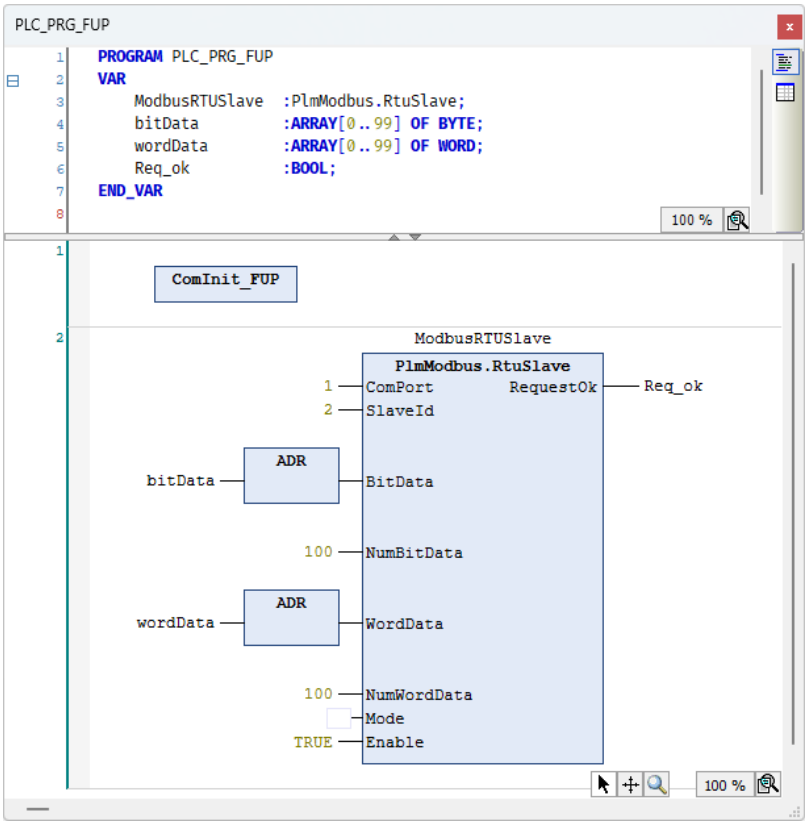


Abbildung 150: Realisierung eines Modbus RTU Slaves (FUP, Teil 1/2)

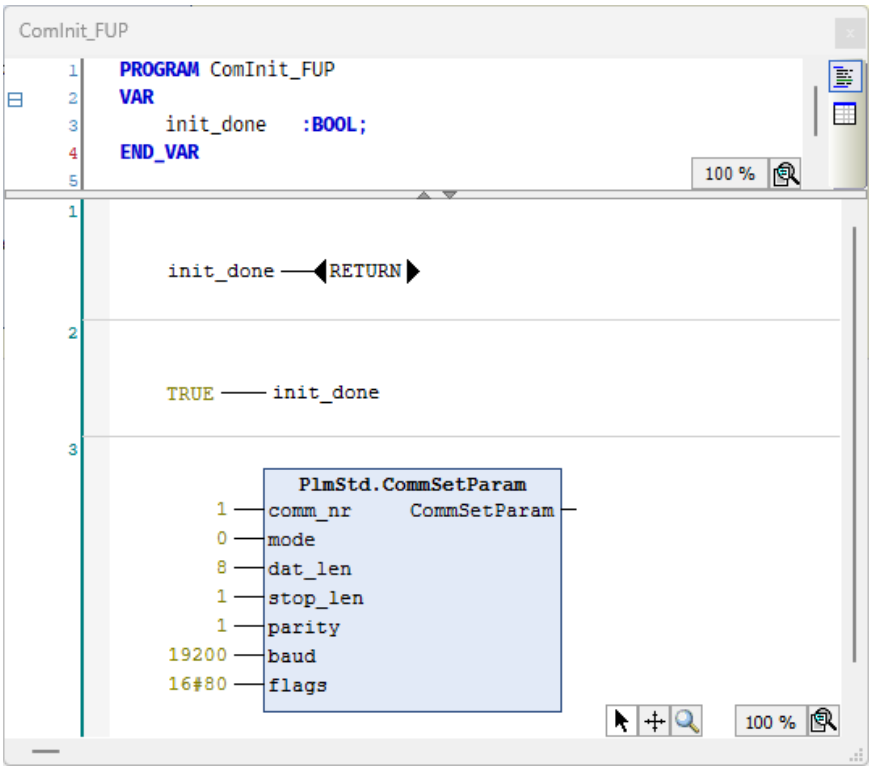


Abbildung 151: Realisierung eines Modbus RTU Slaves (Fup, Teil 2/2)

10.2.3 PROGRAMMBEISPIEL (ST)

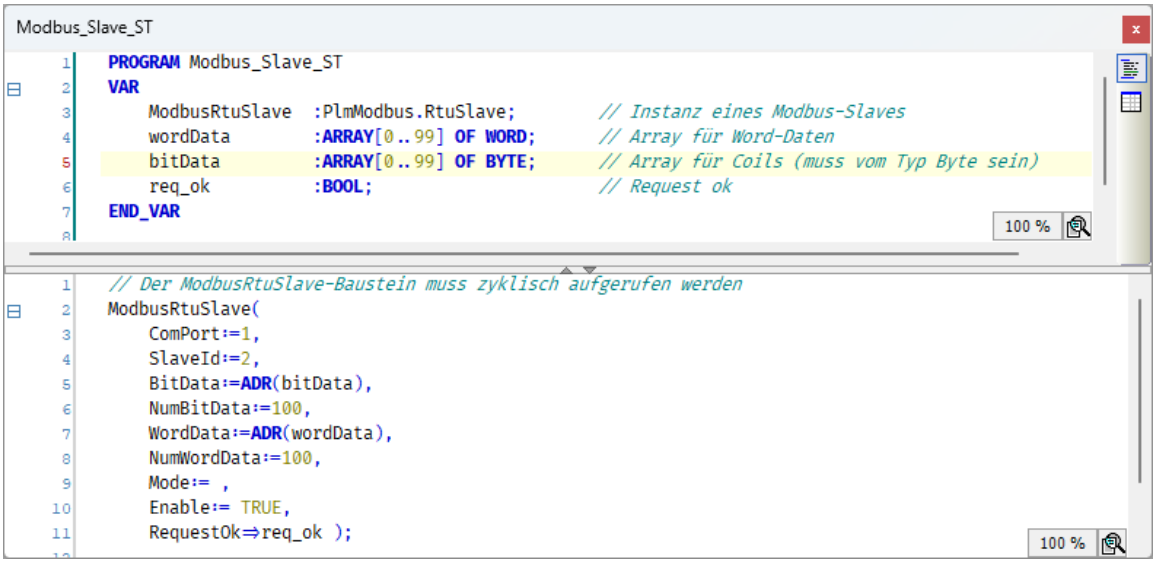


Abbildung 152: Realisierung eines Modbus RTU Slaves (ST, Teil 1/2)

Init_Once_ST

```
PROGRAM Init_Once_ST
VAR
    init_done:bool;
END_VAR
```

```
IF NOT init_done THEN
    init_done:=TRUE;

    // Initialisierung der Schnittstelle
    PlmStd.CommSetParam(
        comm_nr:=1,
        mode:=0,
        dat_len:=8,
        stop_len:=1,
        parity:=0,
        baud:=19200,
        flags:=16#80);
END_IF
```

Abbildung 153: Realisierung eines Modbus RTU Slaves (ST, Teil 2/2)

Die in *Abbildung 150 (FUP)*, *Abbildung 151 (FUP,)* Abbildung 152 *(ST)* und *Abbildung 153 (ST)* gezeigten Programmbeispiele realisieren einen Modbus RTU-Slave mit dem Baustein `RtuSlave`, der die Arrays `bitData[]` mit 100 Bytes (800 Bits) und `wordData[]` mit 100 Words auf COM-Port 1 unter der Modbus-Slave-Adresse 2 bereitstellt.

Nach dem Start wird zunächst die serielle Schnittstelle COM 1 (RS485 an einer PLM 800 Steuerung) mit den Einstellungen 19200 Baud, 8 Datenbits, 1 Stopbit und Even Parity initialisiert. Die Übertragung mit Even Parity ist häufig die Voreinstellung für Modbus RTU.

Je nach Anwendung muss das Anwenderprogramm dafür sorgen, dass die aktuellen Prozesswerte zyklisch in den Arrays `bitData[]` und `wordData[]` abgelegt bzw. aus diesen gelesen werden.

11 MODBUS RTU (MASTER)

11.1 ALLGEMEINES

Modbus RTU erlaubt einen einfachen Datenaustausch zwischen einem Master (PLMSteuerung) und einem oder mehreren Slaves. Der Master spricht dabei die Slaves über eine serielle Schnittstelle an, üblicherweise RS485. Für Modbus-Verbindungen über Ethernet steht Modbus TCP zur Verfügung (siehe *Abschnitte 12* und *13*).

Dieser Abschnitt beschreibt die Arbeitsweise einer PLM-Steuerung als Modbus RTUMaster.

Weitere Informationen zum Modbus-Protokoll stehen auf der Website der Modbus Nutzerorganisation unter *http://modbus.org/* zur Verfügung.

Ein Modbus RTU-Master greift auf eine Werte-Tabelle zu, welcher jeder Slave zur Verfügung stellt. Die Werte können physikalischen Ein- oder Ausgängen des Slaves entsprechen. Werte können gelesen (Read) oder beschrieben werden (Write). Das Modbus-Protokoll erlaubt den Zugriff auf Register (16-Bit-Words) oder Coils (Bits). Die Art des jeweiligen Zugriffs wird über einen Function Code (FC) beim Zugriff festgelegt.

Die PLM-Bibliothek für den Modbus RTU-Master stellt einen Funktionsblock `RtuMasterTransfer` zur Verfügung, der ein Array von Werten aus einem bestimmten Slave liest (`Read`) oder in diesen schreibt (`Write`).

Es kann eine beliebige Anzahl von Instanzen des Funktionsblocks verwendet werden, um z.B. mit verschiedenen Slaves zu kommunizieren oder um verschiedene Zugriffe auf einen Slave durchzuführen.

Der Zugriff auf die Slaves erfolgt normalerweise zyklisch, d.h. die Werte in den Slaves werden regelmäßig gelesen oder geschrieben. Ebenso ist jedoch auch ein programmgesteuerter einmaliger Zugriff möglich, z.B. Schreibzugriffe in der Initialisierungsphase.

Bei Verwendung des Funktionsblocks `RtuMasterTransfer` braucht der Anwender sich nicht um die zeitliche Abfolge der verschiedenen Slave-Zugriffe zu kümmern; der Ablauf wird komplett von der Bibliothek koordiniert.

Die verwendete serielle Schnittstelle muss vor der Verwendung geeignet initialisiert werden (siehe *Abschnitt* 7).

11.2 SPEZIFIKATIONEN

- Modbus RTU Master
- Verwendung einer beliebigen seriellen Schnittstelle, z.B. RS485
- Lese- und Schreibzugriffe auf beliebig viele Slaves (max. 246 gemäß Modbus Spezifikation)
- Automatische Steuerung des zeitlichen Ablaufs der Slave-Zugriffe
- Unterstützte Function-Codes (FC):

  1 (Read Coils)
  2 (Read Discrete Inputs)
  3 (Read Registers)
  4 (Read Input Registers)
  5 (Write Single Coil)
  6 (Write Single Register)
  15 (Write Multiple Coils)
  16 (Write Multiple Registers)

11.3 BENÖTIGTE BIBLIOTHEKEN

Benötigt werden die folgenden Bibliotheken:

PlmStandard > V3.5.16.28
PlmModbus > V3.5.16.17

Die angegebenen Bibliotheken müssen vom Projekt geladen werden, damit die Funktionen zur Verfügung stehen. Klicken Sie dazu auf den ***Bibliotheksverwalter*** > ***Bibliothek hinzufügen*** > ***Sonstige*** > ***PlmStandard*** > ***OK***. Wiederholen Sie den Schritt für die Bibliothek **PlmModbus**. Die benötigten Bausteine zu Parametrierung der seriellen-Schnittstelle befinden sich in dem Ordner **SerialCommuniction.** in der Bibliothek **PlmStandard**. Die benötigten Bausteine für Modbus RTU Master sind in dem Ordner **Modbus RTU Master** der Bibliothek **PlmModbus**.

11.3.1 RTUMASTERINIT (PLMMODBUS.LIBRARY)

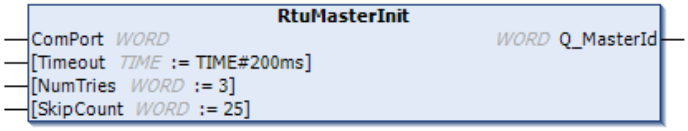


Abbildung 154: Funktionsblock RtuMasterInit (PlmModbus.Library)

Input Parameter:		
ComPort	WORD	Nummer des ComPorts, z.B. 0 für erste serielle Schnittstelle
Timeout	TIME	Timeout Zeit für Antwort vom Slave, z.B. t#200ms
NumTries	WORD	Anzahl der Versuche bevor ein Timeout gemeldet wird
SkipCount	WORD	Nach Ablauf des Slave-Timeouts: Diesen Slave für standardmäßig 25 Anforderungszyklen überspringen
Return-Wert:		
Q_MasterId	WORD	Handle von diesem Modbus-RTU-Masters für angehängte RtuMasterTransfer()-Blöcke (0 bedeutet Fehler)

Der Funktionsblock `RtuMasterInit` darf nur einmalig beim Start des Programms aufgerufen werden und initialisiert die internen Datenstrukturen. Das gelieferte Handle `Q_MasterId` wird anschließend für die Instanzen von `RtuMasterTransfer` benötigt. Bei Bedarf können mehrere Master auf verschiedenen Schnittstellen instanziert werden. Die eindeutige Zuordnung zu diesen Mastern erfolgt mittels des jeweils gelieferten Handles `Q_MasterId`.

Bei der Auswahl einer seriellen Schnittstelle `(ComPort)` ist sicherzustellen, dass diese nicht von anderen Diensten verwendet wird.

11.3.2 RTUMASTERTRANSFER (PLMMODBUS.LIBRARY)

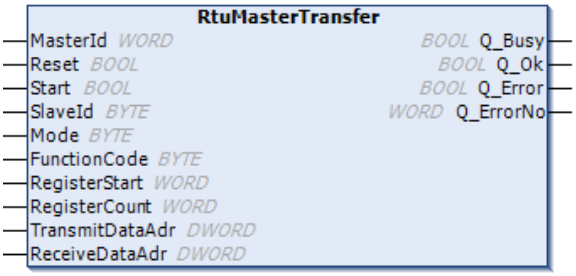


Abbildung 155: Funktionsblock RtuMasterTransfer (PlmModbus.Library)

Input Parameter:		
MasterId	WORD	Handle der Master-Instanz (RtuMasterInit), der dieser Zugriff zugeordnet werden soll
Reset	BOOL	TRUE = Bewirkt ein Rücksetzen des Bausteins
Start	BOOL	TRUE = Damit wir der Baustein abgearbeitet
SlaveId	BYTE	Adresse des angesprochenen Slaves (1…247) oder 0 für Broadcast
Mode	BYTE	Betriebsart des Bausteins: 0 = einzelner Slave Zugriff, wird erst

		wiederholt, wenn Start FALSE und wieder TRUE wird 1 = Slave Zugriff wird zyklisch ausgeführt +2 = Löscht das Q_Error Flag nach einem erfolgreichen Abarbeiten des Bausteins +4 = Gilt nur für die Function Codes 1, 2, 5 und 6. Die Register TransmitData und ReceiveData werden werden als ARRAY OF BOOL anstatt ARRAY OF BYTE behandelt +16#80 = Erweiterter EBM Modbus
FunctionCode	BYTE	Art des Slave Zugriffs. Mögliche Function Codes: 1 = Read Coils (1…2000 Bits lesen) 2 = Read Discrete Inputs (1…2000 Bits lesen) 3 = Read Registers (1…125 Words lessen) 4 = Read Input Registers (1…125 Word lesen) 5 = Write Single Coil (1 Bit schreiben) 6 = Write Single Register (1 Word schreiben) 15 = Write Multiple Coils (1…1968 Bits schreiben) 16 = Write Multiple Register (1…123 Register schreiben)
RegisterStart	WORD	Adresse des ersten zu lesenden/schreibenden Bits/Registers (0…65535)
RegisterCount	WORD	Anzahl der zu lesenden/schreibenden Bits/Register: Function Code 1 oder 2: 1…2000 Function Code 3 oder 4: 1…125 Function Code 5 oder 6: (keine Bedeutung) Function Code 15: 1…1968 Function Code 16: 1…123
TransmitDataAdr	DWORD	Adresse der Variablen/des Arrays mit den Sendedaten
ReceiveDataAdr	DWORD	Adresse der Variablen/des Arrays mit den Empfangsdaten
Return-Wert:		
Q_Busy	BOOL	TRUE = Slave Zugriff läuft noch
Q_Ok	BOOL	TRUE = Salve Zugriff erfolgreich beendet, Empfangsdaten gültig
Q_Error	BOOL	TRUE = Fehler bei Slave Zugriff aufgetreten, Fehlercode in Q_ErrorNo
Q_ErrorNo	WORD	Nummer des Fehlers: Slave-Meldungen: 16#xx01 = Unzulässiger FunctionCode 16#xx02 = Unzulässige Datenadresse 16#xx03 = Unzulässiger Datenwert Master-Meldungen: 16#81xx = Keine Antwort vom Slave nach Timeoutzeit, Retries fehlgeschlagen 16#82xx = Prüfsummen-Error in SlaveAntwort 16#83xx = Unzulässiger Function Code am Eingang FunctionCode 16#84xx = Function Code der Slave-Antwort falsch 16#85xx = unzulässige AnzahlRegister/Coils am Eingang RegisterCount

		16#86xx = interner Fehlercode 16#87xx = interner Fehlercode, keine Antwort vom Slave nach Timeoutzeit, versuche Retry 16#88xx = Falsche Adresse an Eingang TransmitDataAdr

Üblicherweise verwendet ein Programm mehrere Instanzen von `RtuMasterTransfer`, z.B. um mehrere Slaves anzusprechen oder um bei einem Slave sowohl Werte zu schreiben als auch zu lesen. Die Bibliothek sorgt dafür, dass die Slave-Zugriffe automatisch in Reihenfolge der Bausteine im Programm durchgeführt werden. Wenn alle vorhandenen Bausteine abgearbeitet sind, d.h. die Slave-Zugriffe stattgefunden haben, wird wieder mit dem ersten Baustein begonnen.

Bei Bausteinen mit `Mode = 0` wird der Zugriff nur einmalig ausgeführt; dies kann z.B. zur Initialisierung von Slaves nach dem Programmstart verwendet werden. Bausteine mit `Mode = 1` werden ständig zyklisch abgearbeitet; dies dient üblicherweise dem Übertragen von Prozessdaten im Betrieb.

Die am Eingang `TransmitDataAdr` und `ReceiveDataAdr` angegebenen Datenspeicher müssen in jedem Fall die durch `FunctionCode` und `RegisterCount` bestimmte Anzahl an Datenwerten aufnehmen können. Die benötigten Speicheradressen werden üblicherweise mit der `ADR()`-Funktion ermittelt.

Die von einem bestimmten Slave zugelassenen Function Codes sowie die verfügbaren Register sind der Geräte-Dokumentation des jeweiligen Slaves zu entnehmen

11.4 ABLAUF IM NORMALBETRIEB UND IM FEHLERFALL

Der Modbus-RTU-Master arbeitet alle Bausteine vom Typ `RtuMasterTransfer` nach einem internen Schema nacheinander zyklisch ab.

Der Master sendet dabei für jeden Transferbaustein zunächst eine Anfrage (Request) auf die serielle Schnittstelle und erwartet darauf eine Antwort (Response) des angesprochenen Slaves. Dabei kann es sich um eine `Ok`-Response oder um eine `Error`-Response handeln. Die Antwort wird dann dem entsprechenden Transferbaustein zugeordnet.

Falls im Fehlerfall innerhalb der Timeout-Zeit (Parameter *Timeout*) keine Antwort vom Slave empfangen wird, wiederholt der Master die Anfrage. Dies geschieht normalerweise bis zu dreimal (Parameter *NumTries* am *RtuMasterInit* Baustein).

Wenn dann noch immer keine Antwort empfangen wurde, wird der entsprechende Transferbaustein zurückgestellt und bei den nächsten Abfragen übersprungen. Dadurch wird verhindert, dass der Modbus jedes Mal durch den mehrfachen Ablauf der Timeout-Zeit blockiert wird.

Nach einigen Abfragezyklen (Parameter *`SkipCount`* am *`RtuMasterInit`* Baustein) wird der zurückgestellte Transferbaustein erneut abgefragt. Dadurch wird ein Slave, der wegen einer kurzzeitigen Störung keine Anfragen beantwortet, nach einiger Zeit automatisch wieder in den Abfrage-Zyklus integriert.

Der Parameter *Timeout* wird bei der Initialisierung im Baustein `RtuMasterInit` angegeben (siehe *Abschnitt 11.3.1*).

11.5 PROGRAMMBEISPIEL (ST)

Nach dem Start wird zunächst die serielle Schnittstelle COM 1 (RS485 an einer PLM 807) mit den Einstellungen 19200 Baud, 8 Databits, Even Parity und 1 Stoppbit initialisiert.

Die Übertragung mit Even Parity ist häufig die Voreinstellung für Modbus RTU. Anschließend wird eine Instanz von `RtuMasterInit` auf COM-Port 1 gebildet mit einer Timeout-Zeit von 2s.

Das Handle `mb_fh` wird anschließend am Eingang `MasterId` von `RtuMasterTransfer` angegeben.

Modbus_Master

```
PROGRAM Modbus_Master
VAR
    init_master_done    :BOOL;
    Modbus_Master       :PlmModbus.RtuMasterInit;       // ModbusMaster-Instanz anlegen
    mb_fh               :WORD;                          // ID der Master-Instanz
    read                :PlmModbus.RtuMasterTransfer;   // Transferbaustein zum Daten lesen
    write               :PlmModbus.RtuMasterTransfer;   // Transferbaustein zum Daten schreiben
    arr_read            :ARRAY[0..99] OF WORD;          // Array für die eingelesenen Daten vom Slave
    arr_write           :ARRAY[0..99] OF WORD;          // Array für die zu schreibenden Daten zum Slavve
    reset               :BOOL;
    start_rd, start_wr  :BOOL;
    busy_rd, busy_wr    :BOOL;
    ok_rd, ok_wr        :BOOL;
    err_rd, err_wr      :BOOL;
    err_nr_rd, err_nr_wr:WORD;
END_VAR
```

100 %

```
// einmalige Initialisierung des Masters
IF NOT init_master_done THEN
    init_master_done:=TRUE;
    // COM 1 RS485 für Modbus (19200 Baud, 8 Datenbits, 1 Stopbit, Even Parity)
    PlmStd.CommSetParam(comm_nr:= 1, mode:= 0, dat_len:= 8, stop_len:= 1, parity:= 1, baud:= 19200, flags:= 16#80);
    Modbus_Master(ComPort:=1, Timeout:=T#2S, NumTries:=3, SkipCount:=25, Q_MasterId=>mb_fh);
END_IF

// Daten vom Slave lesen
read(
    MasterId:=mb_fh,
    Reset:=reset,
    Start:=start_rd,
    SlaveId:=2 ,                    // Slave Adresse 2
    Mode:=1,                        // Zyklisch aufrufen
    FunctionCode:=3,                // FC = 3 Read Holding Registers
    RegisterStart:=0,               // Startregister 0
    RegisterCount:=10,              // 10 Werte abfragen
    TransmitDataAdr:= ,             // FC 3 braucht keine Sendedaten
    ReceiveDataAdr:=ADR(arr_read),  // Empfangsdaten in arr_read[] ablegen
    Q_Busy=> busy_rd,
    Q_Ok=> ok_rd,
    Q_Error=> err_rd,
    Q_ErrorNo=>err_nr_rd );

// Daten zum Slave schreiben
write(
    MasterId:=mb_fh,
    Reset:=reset,
    Start:=start_wr,
    SlaveId:=3 ,                    // Slave Adresse 3
    Mode:=1,                        // Zyklisch aufrufen
    FunctionCode:=16,               // FC = 16 Write Multiple Registers
    RegisterStart:=0,               // Startregister 0
    RegisterCount:=10,              // 10 Werte schreiben
    TransmitDataAdr:=ADR(arr_write),// Sendedaten aus dem arr_write[]
    ReceiveDataAdr:=,               // FC 16 braucht keine Empfangsdaten
    Q_Busy=> busy_wr,
    Q_Ok=> ok_wr,
    Q_Error=> err_wr,
    Q_ErrorNo=>err_nr_wr );
```

100 %

Abbildung 156: Realisierung eines Modbus RTU Masters (ST)

Im Beispiel erfolgt ein zyklischer Zugriff (Mode = 1) auf zwei Slaves mit den Adressen 2 und 3. Aus dem Slave mit der Adresse 2 werden 10 Datenworte ab der Registeradresse 0 ausgelesen (`FunctionCode = 3, RegisterStart = 0, RegisterCount = 10`). Die gelesenen Datenworte werden in das Array `arr_read[]` abgelegt. In den Slave mit der Adresse 3 werden 10 Datenworte ab Registeradresse 0 (`FunctionCode = 16, RegisterStart = 0, RegisterCount = 10`) mit dem Inhalt der ersten 10 Elementen des Arrays `arr_wr[]` beschrieben.

Die Transferbausteine werden von der Bibliothek automatisch so koordiniert, dass immer eine Slave-Anfrage vollständig abgeschlossen wird, bevor die nächste beginnt.

Der Function Code 3 (Read Holding Registers) benötigt keine Sendedaten, daher braucht am Eingang `TransmitDataAdr` keine Adresse angegeben zu werden. Dem Eingang `ReceiveDataAdr` wird die Adresse des jeweiligen Empfangsdaten-Arrays zugewiesen. Das Array muss groß genug sein, um die angeforderte Anzahl von Werten aufzunehmen. In diesem Fall (`FunctionCode = 3`, d.h. Word-Zugriff und `RegisterCount = 10`) muss das Array vom Typ WORD sein und mindestens 10 Werte aufnehmen können. Der Function Code 16 (Write Multiple Registers) benötigt keine Empfangsdaten, daher braucht am Eingang `ReceiveDataAdr` keine Adresse angegeben zu werden. Dem Eingang `ReceiveDataAdr` wird die Adresse des jeweiligen Sendedaten-Arrays zugewiesen. Aufgrund des Function Code 16 (Word-Zugriff) muss das Array mit den Sendedaten vom Typ Word sein.

Die Werte in `arr_read[]` und `arr_write[]` sind erst gültig, nachdem der jeweilige Zugriff mindestens einmal erfolgreich abgeschlossen wurde, was durch `Ok_rd = TRUE` und `Ok_wr= TRUE` angezeigt wird.

11.6 PROGRAMMBEISPIEL (FUP)

Abbildung 157: Realisierung eines Modbus RTU Masters (Fup, Teil 1/3)

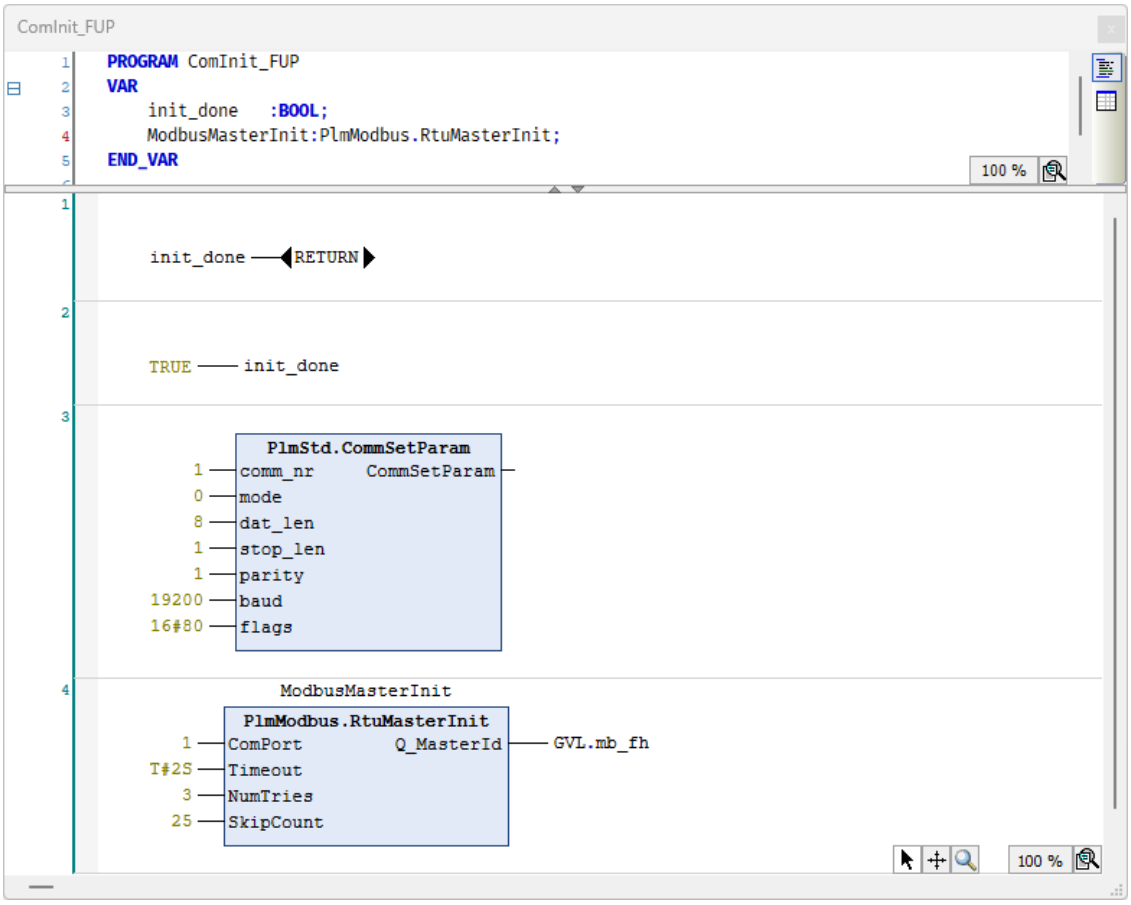


Abbildung 158: Realisierung eines Modbus RTU Masters (Fup, Teil 2/3)

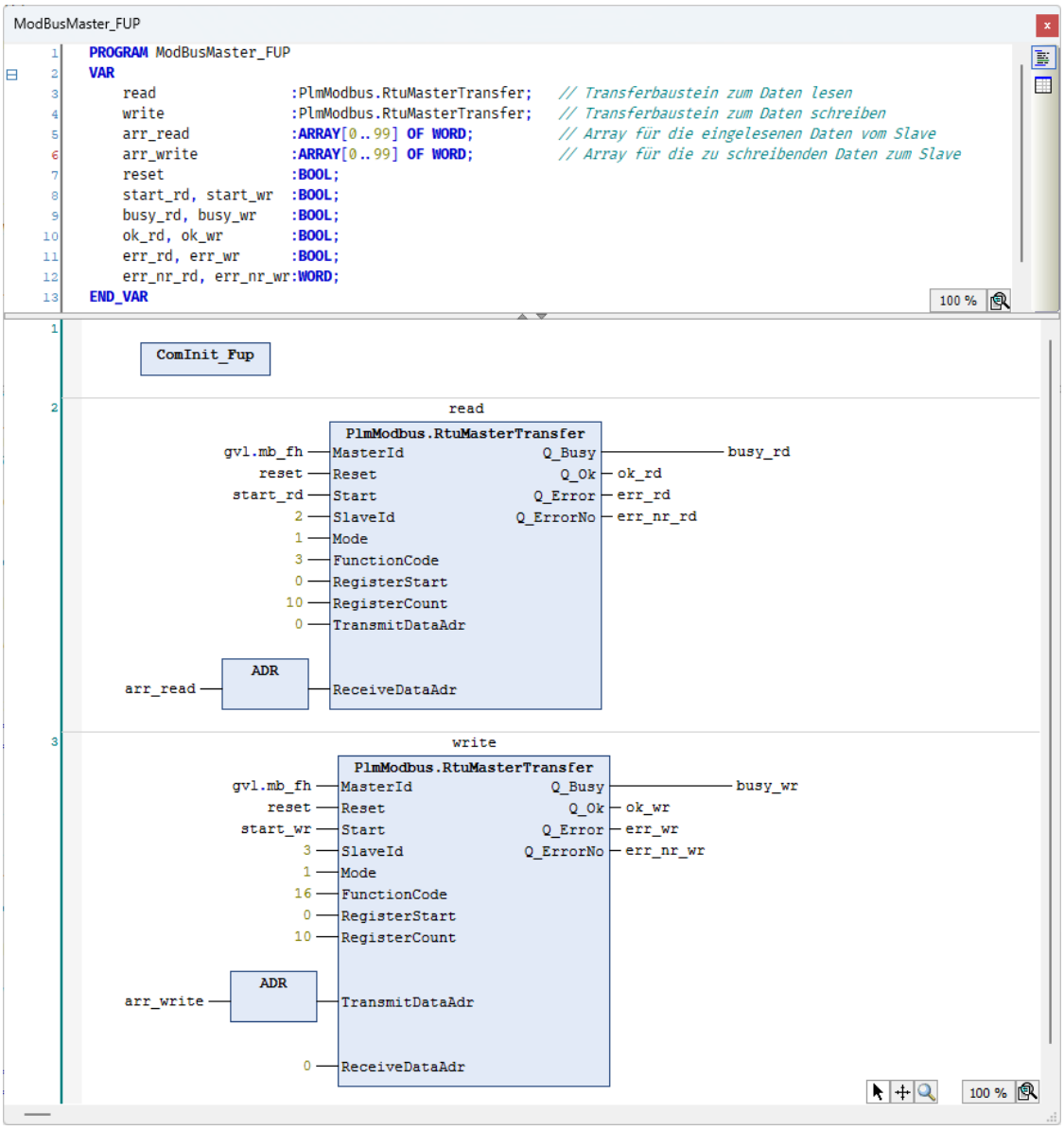


Abbildung 159: Realisierung eines Modbus RTU Masters (Fup, Teil 3/3)

Die Arbeitsweise ist analog zum Beispiel in ST (siehe Abschnitt 11.5). Allerdings ist die einmalige Initialisierung nach dem Start in einen eigenen Funktionsblock `ComInit_Fup` ausgelagert. Da das Handle Mb_fh sowohl in ComInit_Fup als auch in ModbusMaster_Fup verwendet wird, ist es hier als globale Variable in der Globalen Variablenliste (GVL) angelegt.

Auch hier werden die Transferbausteine von der Bibliothek automatisch so koordiniert, dass immer eine Slave-Anfrage vollständig abgeschlossen wird, bevor die nächste beginnt.

12 MODBUS TCP (SERVER)

12.1 ALLGEMEINES

Modbus TCP erlaubt einen einfachen Datenaustausch zwischen einem oder mehreren Clients und einem Server. Die Clients sprechen den Server dabei über eine Ethernet-Verbindung (TCP) an. Für

Modbus-Verbindungen über die serielle Schnittstelle (RS232/RS485) steht Modbus RTU zur Verfügung (siehe Abschnitt 7 und 10.1).

Dieser Abschnitt beschreibt die Arbeitsweise einer PLM-Steuerung als Modbus TCPServer. Weitere Informationen zum Modbus-Protokoll stehen auf der Website der Modbus Nutzerorganisation unter https://www.modbus.org/ zur Verfügung.

Für den Zugriff über Ethernet muss die IP-Adresse der Steuerung bekannt sein, auf der der Modbus TCP-Server läuft. Außerdem müssen sowohl der Server als auch die Clients so konfiguriert sein, dass sie miteinander über Ethernet kommunizieren können.

Ein Modbus-Server stellt den Clients ein Werte-Array zur Verfügung. Die Werte des Arrays können von den Clients abgefragt (Read) oder beschrieben werden (Write). Das Modbus-Protokoll erlaubt den Zugriff auf Words (Register) oder Bits (Coils). Die Art des jeweiligen Zugriffs wird über einen Function Code (FC) beim Zugriff festgelegt.

Das Anwenderprogramm stellt ein WORD-Array beliebiger Größe für Word-Zugriffe bereit, sowie ein BYTE-Array beliebiger Größe für Bit-Zugriffe. Mit diesen wird der Baustein `ModbusTcpServer` zyklisch aufrufen. Anschließend steht der Server-Dienst den Clients über Ethernet zur Verfügung (standardmäßig auf Port 502).

12.2 SPEZIFIKATION

- Modbus TCP Server, Port wählbar (standardmäßig 502)
- Max. Anzahl gleichzeitiger Client-Verbindungen: 64
- Daten-Arrays: max. 65536 Words und max. 65536 Bytes für Bit-Zugriffe
- Unterstützte Function-Codes (FC):

  1 (Read Coils)

  2 (Read Discrete Inputs)

  3 (Read Holding Registers)

  4 (Read Input Registers)

  5 (Write Single Coil)

  6 (Write Single Register)

  15 (Write Multiple Coils)

  16 (Write Multiple Registers)

  23 (Read/Write Multiple Registers)

12.3 BENÖTIGTE BIBLIOTHEKEN

Benötigt werden die folgenden Bibliotheken:

PlmStandard > V3.5.16.28
PlmModbus > V3.5.16.17

Die angegebenen Bibliotheken müssen vom Projekt geladen werden, damit die Funktionen zur Verfügung stehen. Klicken Sie dazu auf den ***Bibliotheksverwalter*** > ***Bibliothek hinzufügen*** > ***Sonstige*** > ***PlmStandard*** > ***OK***. Wiederholen Sie den Schritt für die Bibliothek **PlmModbus**. Die benötigten Bausteine zu Parametrierung der Ethernet-Schnittstelle befinden sich in dem Ordner **Network** in der Bibliothek **PlmStandard**. Der benötigte Baustein `ModbusTcpServer` ist in dem Ordner **Modbus TCP Server** der Bibliothek **PlmModbus**.

12.3.1MODBUSTCPSERVER (PLMMODBUS.LIBRARY)

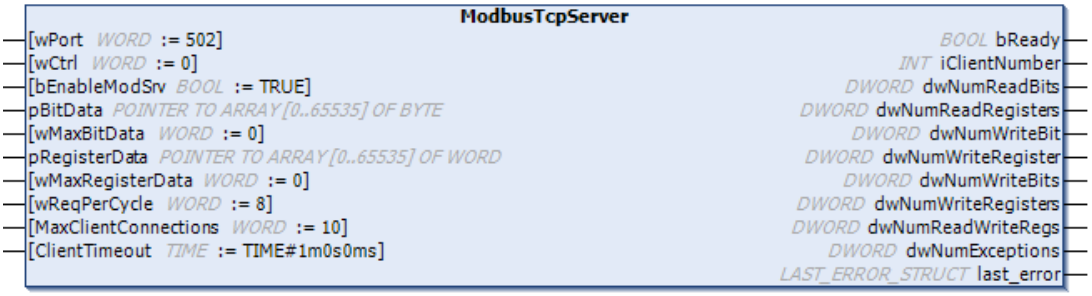


Abbildung 160: Funktionsblock ModbusTcpServer (PlmModbus.Library)

Input Parameter:		
wPort	WORD	TCP-Port auf dem der Dienst bereitgestellt wird
wCtrl	WORD	16#0001 = ID Memory nutzen
bEnableModSrv	BOOL	TRUE = Baustein wird abgearbeitet
pBitData	POINTER TO ARRAY[0…65534] OF BYTE	Adresse des Byte Arrays für Coil Zugriffe (Function Codes 1, 2, 5 und 15)
wMaxBitData	WORD	Anzahl der Bytes in pBitData
pRegisterData	POINTER TO ARRAY[0…65535] OF WORD	Adresse des Word Arrays für Register Zugriffe (Function Codes 3, 4, 6 und 16)
wMaxRegisterData	WORD	Anzahl der Word in pRegisterData
wReqPerCycle	WORD	Maximale Anzahl der Antworten pro verbundenen Client in einem Zyklus
MaxClientConnections	WORD	Maximale Anzahl von Clients
ClientTimeout	TIME	Zeitangabe für den Timeout für die Client-Verbindung, Standard T#1m
Return-Wert:		
bReady	BOOL	TRUE = Server ist bereit für Client-Verbindungen
iClientNumber	INT	Anzahl der verbunden Clients
dwNumberReadBits	DWORD	Event-Counter für Modbus Client-Zugriffe

dwNumReadRegisters	DWORD	Event-Counter für Modbus Client-Zugriffe
dwNumWriteBit	DWORD	Event-Counter für Modbus Client-Zugriffe
dwNumWriteRegister	DWORD	Event-Counter für Modbus Client-Zugriffe
dwNumberWriteBits	DWORD	Event-Counter für Modbus Client-Zugriffe
dwNumWriteRegisters	DWORD	Event-Counter für Modbus Client-Zugriffe
dwNumReadWriteRegs	DWORD	Event-Counter für Modbus Client-Zugriffe
dwNumExceptions	DWORD	Zähler für die Ausnahmeantworten des Clients
Last_error	LAST_ERROR_STRUCT	Informationen vom Client zum letzten Übertragungsfehler

Die Bibliothek **PlmModbus** stellt einen Funktionsblock als globale Variable `ModbusTcpServer[0]` zur Verfügung. Diese Variable muss für den zyklischen Aufruf verwendet werden (siehe Programmbeispiel).

Coil-Werte werden auf dem Modbus immer in Achtergruppen zu einem Byte zusammengefasst, wobei immer acht Coil-Werte in einem Byte von `pBitData` gespeichert werden. Unabhängig davon können aber mit den entsprechenden Modbus-Function-Codes auch einzelne Coil-Werte beschrieben oder abgefragt werden.

Der Parameter `wMaxBitData` enthält die Array-Größe von `pBitData` in Bytes, nicht die Anzahl der Coils.

Coil-Adressen können auf dem Modbus zwischen 0 und 65535 liegen, somit ist eine maximale Array-Größe von `pBitData` von 8192 Bytes (=65536 Coils) sinnvoll.

In der momentanen Version der Bibliothek sind Coil-Zugriffe mit den Function-Codes 1, 2 und 15 (Read/Write Multiple Coils) nur auf Byte-Grenzen möglich, d.h. die Coil-Startadresse im Request muss durch 8 teilbar sein (0, 8, 16, 24 etc.)

Wenn der Server durch einen CODESYS-Event (z.B. *Reset Kalt*) gestoppt und anschließend wieder gestartet wird, kann es vorkommen, dass der angegebene Server-Port (üblicherweise 502) vom Betriebssystem der Steuerung ca. 1 Minute lang blockiert ist. Der Server wartet dann bei `bEnableModSrv = TRUE` solange mit `bReady = FALSE`, bis der Port wieder zur Verfügung steht. Anschließend wird der Port wieder automatisch belegt und Client-Verbindungen werden zugelassen (`bReady = TRUE`).

Falls alle 64 möglichen Client-Verbindungen belegt sind und eine weitere Verbindungsanfrage von einem Client eintrifft, wird die älteste Verbindung (auf der am längsten keine Modbus-Anfrage mehr bearbeitet wurde) geschlossen und dem neuen Client zugewiesen.

Der Server verwendet intern ein `Array Client[1..64]`. In diesem Array werden die bis zu 64 möglichen Client-Verbindungen gespeichert.

12.4 PROGRAMMBEISPIEL (ST)

ModbusTcpServer_ST

```
PROGRAM ModbusTcpServer_ST
VAR
    CoilData     :ARRAY[0..15] OF BYTE;
    RegisterData:ARRAY[0..15] OF WORD;
END_VAR
```

```
PlmModbus.ModbusTcpServer[0](
    wPort                := 502,
    wCtrl                := 0 ,
    bEnableModSrv        := TRUE,
    pBitData             := ADR(CoilData),
    wMaxBitData          := 16,
    pRegisterData        := ADR(RegisterData),
    wMaxRegisterData     := 16);
```

Abbildung 161: Realisierung eines Modbus TCP Servers (ST)

12.5 PROGRAMMBEISPIEL (FUP)

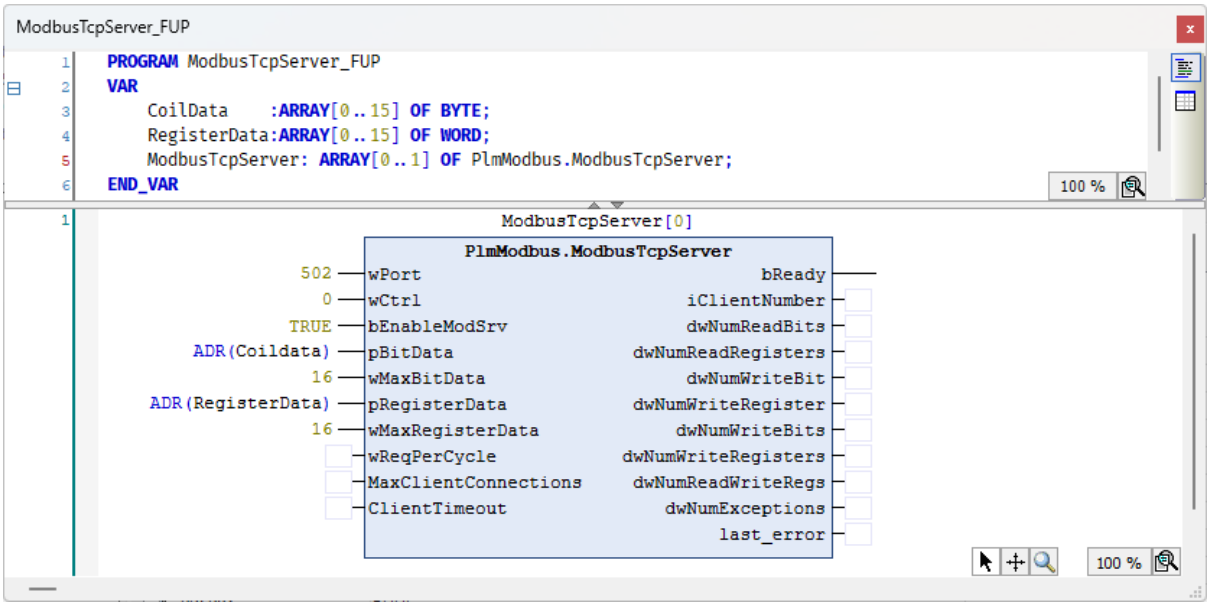


Abbildung 162: Realisierung eines Modbus TCP Servers (FUP)

Das in *Abbildung 161* und *Abbildung 162* gezeigte Programmbeispiel realisiert einen Modbus TCP-Server, der das Array `RegisterData[]` mit 16 Words und das Array `CoilData[]` mit 16 Bytes (= 128 Coils) auf Port 502 bereitstellt.

Der Funktionsbaustein `ModbusTcpServer[0]` wird durch die Bibliothek **PlmModbus** als globale Variable bereitgestellt und muss zyklisch aufgerufen werden.

Je nach Anwendung muss das Anwenderprogramm dafür sorgen, dass die aktuellen Prozesswerte zyklisch in den Arrays `RegisterData[]` und `CoilData[]` abgelegt bzw. aus diesen gelesen werden.

13 MODBUS TCP (CLIENT)

13.1 ALLGEMEINES

Modbus TCP erlaubt einen einfachen Datenaustausch zwischen einem oder mehreren Clients und einem Server. Die Clients sprechen den Server dabei über eine Ethernet-Verbindung (TCP) an. Für Modbus-Verbindungen über die serielle Schnittstelle (RS232/RS485) steht Modbus RTU zur Verfügung (siehe Abschnitt 7 und Abschnitt 10).

Dieser Abschnitt beschreibt die Arbeitsweise einer PLM-Steuerung als Modbus TCPClient.

Weitere Informationen zum Modbus-Protokoll stehen auf der Website der Modbus Nutzerorganisation unter https://www.modbus.org/ zur Verfügung.

Für den Zugriff über Ethernet muss die IP-Adresse der Steuerung bekannt sein, auf der der Modbus TCP-Server läuft. Außerdem müssen sowohl der Server als auch die Clients so konfiguriert sein, dass sie miteinander über Ethernet kommunizieren können.

Ein Modbus-Server stellt den Clients ein Werte-Array zur Verfügung. Die Werte des Arrays können von den Clients abgefragt (Read) oder beschrieben werden (Write). Das Modbus-Protokoll erlaubt den Zugriff auf Words (Register) oder Bits (Coils). Die Art des jeweiligen Zugriffs wird über einen Function Code (FC) beim Zugriff festgelegt.

Der PLM Modbus-TCP-Client wird durch zyklischen Aufruf des eigentlichen Client-Bausteins `ModbusTcpClient` und der daran angehängten Datentransferbausteine `ModbusTcpClientTransfer` realisiert. Der Client-Baustein stellt die Netzwerk-Verbindung zum Modbus TCP Server her, während die Transferbausteine darüber ihren Datenaustausch abwickeln.

13.2 SPEZIFIKATION

- Modbus TCP Client
- Unterstützte Function-Codes (FC):
  - 1 (Read Coils)
  - 2 (Read Discrete Inputs)
  - 3 (Read Holding Registers)
  - 4 (Read Input Registers)
  - 5 (Write Single Coil)
  - 6 (Write Single Register)
  - 15 (Write Multiple Coils)
  - 16 (Write Multiple Registers)

13.3 BENÖTIGTE BIBLIOTHEKEN

Benötigt werden die folgenden Bibliotheken:

PlmStandard > V3.5.16.28
PlmModbus > V3.5.16.17

Die angegebenen Bibliotheken müssen vom Projekt geladen werden, damit die Funktionen zur Verfügung stehen. Klicken Sie dazu auf den ***Bibliotheksverwalter*** > ***Bibliothek hinzufügen*** > ***Sonstige*** > ***PlmStandard*** > ***OK***. Wiederholen Sie den Schritt für die Bibliothek **PlmModbus**. Die benötigten Bausteine zu Parametrierung der Ethernet-Schnittstelle befinden sich in dem Ordner **Network** in der Bibliothek **PlmStandard**. Der benötigte Baustein `ModbusTcpServer` ist in dem Ordner **Modbus TCP Server** der Bibliothek **PlmModbus**.

13.3.1MODBUSTCPCLIENT (PLMMODBUS.LIBRARY)

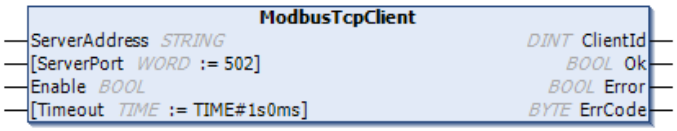


Abbildung 163: Funktionsblock Modbus TCP Client (PlmModbus.Library)

Input Parameter:		
`ServerAddress`	`STRING`	`IP-Adresse des Modbus TCP Servers, zu dem eine Verbindung hergestellt werden soll`
`ServerPort`	`WORD`	`TCP-Port, auf dem der Server den Dienst bereitstellt; Standard 502`
`Enable`	`BOOL`	`TRUE = Verbindung zum Server wird aufgebaut` `FALSE = Verbindung zum Server wird getrennt`
`Timeout`	`TIME`	`Timout Zeit für alle Server Reaktionen`
Return-Wert:		
`ClientId`	`DINT`	`Referenz (Handle) für das Anhängen der Transferbausteine`
`Ok`	`BOOL`	`TRUE = Verbindung zum Server hergestellt`
`Error`	`BOOL`	`TRUE = Fehler beim Herstellen der Verbindung zum Server, siehe ErrCode`

`ErrCode`	`BYTE`	`0 = Kein Fehler`
		`1 = Socket create Error`
		`2 = Kann zum Server keine Verbindung herstellen (z.B. falsche IP-Adresse vom Modbus TCP Server, keine Verbindung)`
		`3 = Keinen Transferbaustein angehangen`
		`4 = Verbindung wurde vom Server beendet`

Der Baustein `ModbusTcpClient` muss zyklisch aufgerufen werden.

Das Aufbauen der Verbindung zum Modbus-TCP-Server geschieht, indem der Eingang `Enable` auf `TRUE` gesetzt wird. Ist der anschließende Verbindungsaufbau innerhalb der vorgegebenen Timeout-Zeit erfolgreich,

wird der Ausgang `Ok` auf `TRUE` gesetzt. Wenn ein Fehler beim Verbindungsaufbau auftritt, wird `Err` auf `TRUE` gesetzt und ein entsprechender Fehlercode in `ErrCode` ausgegeben.

Solange `Enable` auf `TRUE` ist, bleibt die Verbindung zum Server bestehen. Sollte der Server seinerseits die Verbindung beenden, so wird `Err` auf `TRUE` gesetzt und ein entsprechender Fehlercode in `ErrCode` ausgegeben. Ein solcher Verbindungsabbruch kann allerdings erst beim nächsten Datentransfer erkannt werden. Solange `Enable` den Wert `TRUE` hat, versucht der Client automatisch, die Verbindung wieder aufzubauen.

13.3.2 MODBUSTCPCLIENTTRANSFER (PLMMODBUS.LIBRARY)

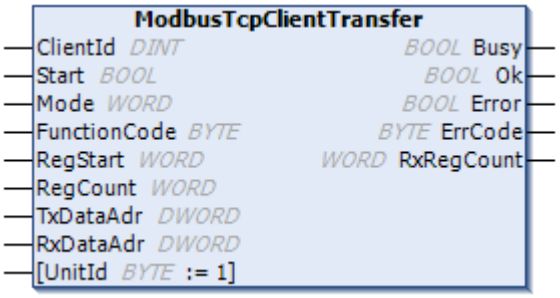


Abbildung 164: Funktionsblock ModbusTcpClientTransfer (PlmModbus.Library)

Input Parameter:		
ClientId	DINT	Referenz (Handle) des zugehörigen Clients
Start	BOOL	TRUE = Baustein wird abgearbeitet. Wird Mode = 0 benutzt, erfolgt der Einzeltransfer bei steigender Flanke (FALSE->TRUE) an Start
Mode	WORD	0 = einzelner Transfer 1 = zyklischer Transfer
FunctionCode	BYTE	Art des Transfers. Mögliche Function Codes: 1 = Read Coils (1…2000 Bits lesen) 2 = Read Discrete Inputs (1…2000 Bits lesen) 3 = Read Registers (1…125 Words lesen) 4 = Read Input Registers (1…125 Words lesen) 5 = Write Single Coil (1 Bit schreiben) 6 = Write Single Register (1 Word schreiben) 15 = Write Multiple Coils (1…1968 Bits schreiben) 16 = Write Multiple Registers (1…123 Words schreiben)
RegStart	WORD	Modbus Index des ersten zu lesenden/schreibenden Registers bzw. Coils
RegCount	WORD	Anzahl der zu lesenden/schreibenden Register bzw. Coils FC 1 oder FC 2: 1…2000 FC 3 oder FC 4: 1…125FC 5 oder FC 6: (keine Bedeutung) FC 15: 1…1968FC 16: 1…123
TxDataAdr	DWORD	Adresse der Variablen/des Arrays mit den Sendedaten für FC 5,6,15 und 16
RxDataAdr	DWORD	Adresse der Variablen/des Arrays mit den Empfangsdaten für FC 1,2,3 und 4

UnitId	BYTE	Optionaler Sub-Unit Identifier für Modbus TCP Server
Return-Wert:		
Busy	BOOL	TRUE = Transfer wird ausgeführt
Ok	BOOL	TRUE = Transfer erfolgreich ausgeführt
Error	BOOL	TRUE = Fehler beim Transfer, siehe ErrCode
ErrCode	BYTE	0 = Kein Fehler 1 = Server exception response, illegal request FunctionCode 2 = Server exception response, illegal request Data Address 3 = Server exception response, illegal request Data Value 81 = Request Timeout, no server response 82 = Request Timeout, incomplete server response 83 = Unzulässiger FunctionCode 84 = Function Code der Response falsch 85 = RegCount unzulässig 86 = ungültige TxDataAdr bzw. RxDataAdr 87 = Transaction Identifier der Response falsch 88 = Unit Identifier der Response falsch 89 = RxRegCount der Response (Antwort) falsch
RxRegCount	WORD	Anzahl der tatsächlich empfangenen Register/Coils bei FC 1,2,3 und 4

Alle Bausteine `ModbusTcpClientTransfer` müssen mindestens einmal beim Programmstart aufgerufen werden, oder zyklisch.

Jeder Transfer-Baustein muss an einen Client-Baustein vom Typ `ModbusTcpClient` angehängt werden, indem der Eingang `ClientId` mit dem entsprechenden Ausgang des Client-Bausteins verbunden wird.

Falls mehrere Transferbausteine mit einem Client-Baustein verbunden sind, erfolgt die Abarbeitung automatisch nacheinander in der Aufrufreihenfolge der Transferbausteine.

Die Eingänge `TxDataAdr` und `RxDataAdr` erwarten die Angabe einer Variablenadresse, die als Ergebnis der Adressfunktion `ADR()` geliefert wird. Der Typ der Variablen hängt vom Function-Code ab.

Bei den Function-Codes 3 und 4 (Read Registers) muss am Eingang `RxDataAdr` die Adresse eines WORD-Arrays mit mindestens `RegCount` WORDs angegeben werden.

Bei den Function-Codes 6 und 16 (Write Single/Multiple Registers) muss am Eingang `TxDataAdr` die Adresse eines WORD-Arrays mit mindestens `RegCount` WORDs angegeben werden.

Coil-Werte werden auf dem Modbus immer in Achtergruppen zu einem Byte zusammengefasst. Entsprechend erwarten die Transferbausteine bei den Coil-Funktionen BYTE-Daten, wobei immer acht Coil-Werte in einem Byte gespeichert werden. Unabhängig davon können aber auch weniger als acht Coil-Werte beeinflusst werden (entsprechender Wert von `RegCount`), oder Coil-Werte, die nicht auf einer Byte-Grenze beginnen, so dass sich nur die angegebenen Bits in den entsprechenden Bytes ändern.

Bei den Function-Codes 1 und 2 (Read Coils) muss am Eingang `RxDataAdr` die Adresse eines BYTE-Arrays mit mindestens `(RegCount+7)/8` Bytes angegeben werden. `RegCount` enthält die gewünschte Anzahl Coils, nicht Bytes. Die Verwendung von BOOL-Arrays ist nicht möglich.

Bei Function-Code 5 (Write Single Coil) muss am Eingang `TxDataAdr` die Adresse einer BYTE- oder BOOL-Variablen angegeben werden. Ist der Wert `0` (oder `FALSE`), so wird das Coil ausgeschaltet (auf `0` gesetzt), ist der

Wert ungleich 0 (oder TRUE), so wird das Coil eingeschaltet (auf 1 gesetzt). Der Eingang RegCount wird hierbei ignoriert.

Bei Function-Code 15 (Write Multiple Coils) muss am Eingang TxDataAdr die Adresse eines BYTE-Arrays mit mindestens (RegCount+7)/8 Bytes angegeben werden. RegCount enthält die gewünschte Anzahl Coils, nicht Bytes. Die Verwendung von BOOL-Arrays ist nicht möglich.

13.4 PROGRAMMBEISPIEL (FUP)

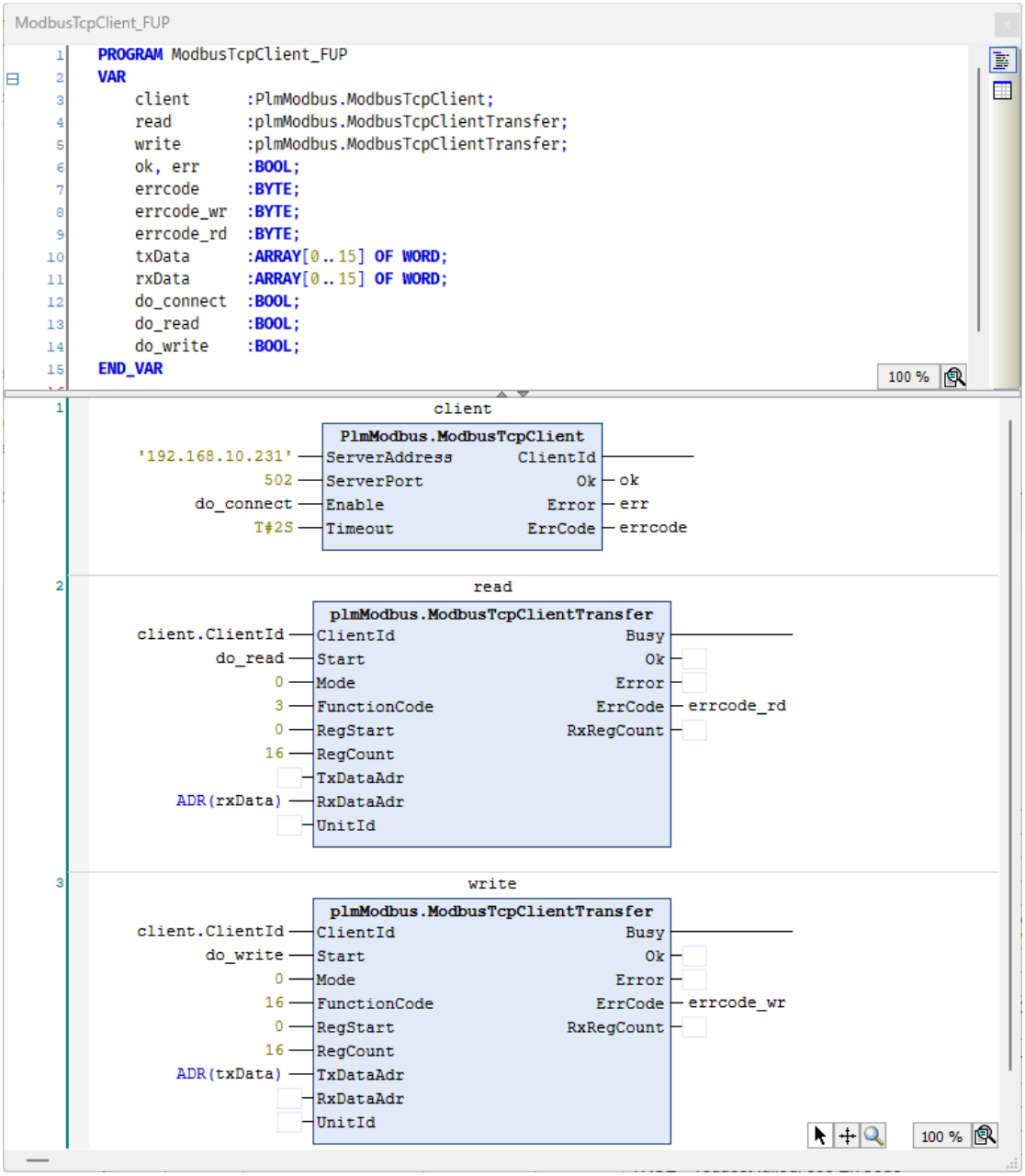


Abbildung 165: Realisierung eines Modbus TCP Clients (FUP)

Das in Abbildung 165 dargestellte Beispiel realisiert einen Modbus-TCP-Client. Zum Aufbau einer Verbindung muss zunächst do_connect auf TRUE gesetzt werden. Der Client versucht dann, eine Verbindung zu einem Modbus-TCP-Server mit der IP Adresse 192.168.10.231 auf Port 502 aufzubauen. Gelingt dies, wird Ok auf TRUE gesetzt und die angehängten Transferbausteine können mit do_read und do_write aktiviert werden. Der

angegebene Timeout-Wert muss bei sehr langsamen Verbindungen und stark belasteten Servern evtl. erhöht werden.

Die Eingänge `ClientId` beider Transferbausteine sind mit dem entsprechenden Ausgang `ClientId` des Client-Bausteins verbunden. Dies kann alternativ auch durch Zuweisung zu einer entsprechenden Variablen geschehen, die dann am Ausgang des Clients und an den Eingängen aller Transferbausteine angeschaltet wird.

Im Beispiel hat `Mode` bei beiden Transferbausteinen den Wert `0`, so dass bei einem Wechsel von `do_read` oder `do_write` von `FALSE` auf `TRUE` jeweils ein einziger Transfer (Modbus-Request) ausgeführt wird.

Der erste Transferbaustein `read` führt im Beispiel einen Lesevorgang von 16 Registerwerten ab Registerindex `0` (also Register 0...15) auf den Server aus. Anschließend werden die gelesenen Registerwerte im WORD-Array `rxData[]` gespeichert.

Der zweite Transferbaustein `write` führt im Beispiel einen Schreibvorgang von 16 Registerwerten ab Registerindex `0` (also Register 0...15) auf den Server aus. Dabei werden die Registerwerte aus dem WORD-Array `txData[]` an den Server übertragen.

13.5 PROGRAMMBEISPIEL (ST)

ModbusTcpClient_ST

```
PROGRAM ModbusTcpClient_ST
VAR
    client      :PlmModbus.ModbusTcpClient;
    read        :plmModbus.ModbusTcpClientTransfer;
    write       :plmModbus.ModbusTcpClientTransfer;
    ok, err     :BOOL;
    errcode     :BYTE;
    errcode_wr  :BYTE;
    errcode_rd  :BYTE;
    txData      :ARRAY[0..15] OF WORD;
    rxData      :ARRAY[0..15] OF WORD;
    do_connect  :BOOL;
    do_read     :BOOL;
    do_write    :BOOL;
END_VAR
```

100 %

```
// Verbindung zum Server
Client(
    ServerAddress   :='192.168.10.231',
    ServerPort      :=502,
    Enable          := do_connect,
    Timeout         :=T#2S,
    Ok⇒             ok,
    Error⇒          err,
    ErrCode⇒        errcode);

// Register lesen
read(
    ClientId        :=client.ClientId,
    Start           :=do_read,
    Mode            :=0,
    Functioncode    :=3,
    RegStart        :=0,
    RegCount        :=16,
    RxDataAdr       :=ADR(rxData),
    ErrCode⇒        errcode_rd);

// Register schreiben
Write(
    ClientId        :=client.ClientId,
    Start           :=do_write,
    Mode            :=0,
    Functioncode    :=16,
    RegStart        :=0,
    RegCount        :=16,
    TxDataAdr       :=ADR(txData),
    ErrCode⇒        errcode_wr);
```

100 %

Abbildung 166: Realisierung eines Modbus TCP Clients (ST)

14 UHRZEIT UND DATUM

14.1 ALLGEMEINES

Alle PLM-Steuerungen besitzen eine eingebaute Echtzeituhr (engl. RTC, Real TimeClock), die, einmal gestellt, die korrekte Uhrzeit und das Datum auch bei abgeschalteter Steuerung weiterführt.

Die Verwendung der Echtzeituhr im IEC-Programm und Darstellungen von Datum und Uhrzeit werden durch die Bibliothek PlmStandard.library erleichtert. Die Bibliothek enthält Programmbausteine zum einfachen Abfragen und Setzen der Echtzeituhr, zur Berücksichtigung von Sommer-/Winterzeitumschaltung (engl. DST, Daylight Saving Time) und zum Erzeugen von Strings mit Uhrzeit und Datum in verschiedenen Formaten.

Die Bibliothek enthält auch einen Programmbaustein zur Abfrage von NTP/SNTPServern. Dabei handelt es sich um Zeit-Server im Internet, die die genaue Uhrzeit bereitstellen.

14.2 LOKALE ZEIT UND SOMMER-/WINTERZEITUMSCHALTUNG

Zeitangaben beziehen sich international immer auf die sogenannte "Koordinierte Weltzeit" (engl. UTC, Universal Time Coordinated), die der Greenwich Mean Time am Nullmeridian entspricht. Die UTC wird auch von Zeit-Servern im Internet geliefert.

Als Ortszeit oder "lokale Zeit" (engl. Local Time) wird die Zeit am Aufstellort der PLM Steuerung bezeichnet. Diese lokale Zeit weicht, bedingt durch geografische Lage und Sommer-/Winterzeitumschaltung, von der UTC ab. Während die Abweichung durch die geografische Lage konstant ist, hängt die Abweichung durch Sommer-/Winterzeitumschaltung vom Datum und von der zugrundeliegenden Gesetzgebung ab (in Deutschland z.B. basierend auf dem Zeitgesetz vom 25. Juli 1978 und div. Zusatzverordnungen). Beispielsweise besteht zwischen der lokalen Zeit in Berlin/Deutschland und der UTC im Winter eine Zeitdifferenz von 1 Stunde (deutsche Winterzeit), im Sommer von 2 Stunden (deutsche Sommerzeit).

Damit die Zeitdifferenz zwischen der lokalen Zeit und der UTC richtig berücksichtigt wird, muss auf der Steuerung die korrekte Zeitzone eingestellt werden, z.B. Europe/Berlin für Deutschland. Dies kann entweder mit dem Baustein ManageTimezone() durch das IEC-Programm erfolgen oder über die Konfigurations-Website der Steuerung.

Diese Vorgehensweise hat den Vorteil, dass die Echtzeituhr nicht manuell auf Sommer-/Winterzeit umgestellt werden muss und ein Uhrzeitabgleich durch einen NTP/SNTP-Server einfach möglich ist. Die entsprechend anzugebenden Parameter können den untenstehenden Beispielen entnommen werden.

14.3 BENÖTIGTE BIBLIOTHEKEN

Benötigt wird die folgende Bibliothek:

PlmStandard > V3.5.16.28

Die angegebene Bibliothek muss vom Projekt geladen werden, damit die Funktionen zur Verfügung stehen. Klicken Sie dazu auf den ***Bibliotheksverwalter*** > ***Bibliothek hinzufügen*** > ***Sonstige*** > ***PlmStandard*** > ***OK***. Die benötigten Bausteine ManageLocalTime() und ManageTimeZone() zum Stellen des Datums und der Uhrzeit befinden sich in dem Ordner **Time+Date** in der Bibliothek **PlmStandard**.

14.3.1MANAGELOCALTIME (PLMSTANDARD.LIBRARY)

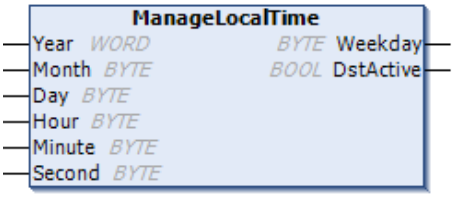


Abbildung 167: Funktionsbaustein ManageLocalTime (PlmStandard.Library)

Input Parameter:		
Year	WORD	Jahr (0…99 für die Jahre 2000…2099)
Month	BYTE	Monat (1…12)
Day	BYTE	Tag (1…31)
Hour	BYTE	Stunden (0…23)
Minute	BYTE	Minuten (0…59)
Second	BYTE	Sekunden (0…59)
Return-Wert:		
Weekday	BYTE	Wochentag (0 = Montag, 1 = Dienstag, 2 = Mittwoch, 3 = Donnerstag, 4 = Freitag, 5 = Samstag, 6 = Sonntag)
DstActive	BOOL	TRUE = Sommerzeit aktiv FALSE = Winterzeit

14.3.2 MANAGETIMEZONE (PLMSTANDARD.LIBRARY)

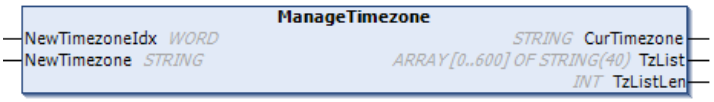


Abbildung 168: Funktionsbaustein ManageTimezone (PlmStandard.Library)

Input Parameter:		
NewTimezoneIdx	WORD	Neuer Index der Zeitzone in TzList
NewTimezone	STRING	Neue Zeitzone, muss ein String aus dem Array TzList[] sein
Return-Wert:		
CurTimezone	STRING	String mit aktueller Zeitzone, z.B. 'Europe/Berlin'
TzList	ARRAY[0..600] OF STRING(40)	Ein String-Array, das alle verfügbaren Zeitzonen enthält
TzListLen	INT	Anzahl der vorhandenen Strings in TzList

14.4 PROGRAMMBEISPIEL (ST)

Time_Date

```
PROGRAM Time_Date
VAR
    mgTime: PlmStd.ManageLocalTime;
    yr: WORD;
    mt, dy, hr, mi, sc: BYTE;
    daytab: ARRAY[0..6] OF STRING := [ 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun' ];
    dayname: STRING;
    uhrzeit:STRING;
    datum:STRING;

    tzcfg: PlmStd.ManageTimezone;
    newTz: STRING;
    newTzIdx: WORD;
END_VAR
```

100 %

```
// Baustein zum Einlesen und Schreiben von Datum und Uhrzeit
mgTime(
    Weekday=> ,
    DstActive=> ,
    Year:= yr,
    Month:= mt,
    Day:= dy,
    Hour:= hr,
    Minute:= mi,
    Second:= sc);

// Wochentag als String
dayname:=daytab[mgTime.Weekday];

// Format für Uhrzeit und Datum festlegen
Uhrzeit:=PlmStd.FormatTime(TimeHandle:=PlmStd.GetLocalTime() , FormatIdx:=30 ); // hh:mm:ss
Datum:=PlmStd.FormatTime(TimeHandle:=PlmStd.GetLocalTime() , FormatIdx:=21 );   // dd.mm.yyyy

// Baustein zum Einlesen und Schreiben der Zeitzone
tzcfg(
    CurTimezone=> ,
    TzList=> ,
    TzListLen=> ,
    NewTimezoneIdx:= newTzIdx,
    NewTimezone:= newTz);
```

100 %

Abbildung 169: Einlesen und Stellen von Datum und Uhrzeit (ST)

Der Funktionsbaustein `mgTime()` führt eine zyklische Abfrage von Systemdatum und Systemzeit durch. Die erfassten Zeitinformationen werden in den Variablen `yr` (Jahr), `mt` (Monat), `dy` (Tag), `hr` (Stunde), `mi` (Minute) sowie `sc` (Sekunde) gespeichert. Diese Variablen dienen als Schnittstelle zur Visualisierung und ermöglichen dem Anwender die Anpassung von Datum und Uhrzeit. Siehe Abbildung 170.

Der Wochentag (0...6) wird von dem Funktionsbaustein `mgTime()` an den Ausgang `Weekday` geschrieben. Der Variable `dayname` wird der Wochentag (0...6) entsprechend aus dem String-Array `daytab[]` übergeben.

Um die Uhrzeit oder das Datum als String in einem bestimmten Format in der Visualisierung anzuzeigen, kann der Baustein `FormatTime()` genutzt werden. Dem Eingang `TimeHandle` muss die Lokale Zeit über die Funktion `GetLocalTime()` übergeben werden. Der Eingang `FormatIdx` definiert das Ausgabeformat für die Darstellung von Datum bzw. Uhrzeit als String. Beispielsweise bewirkt der Wert 30 die Formatierung der Uhrzeitanzeige im Format `Stunde:Minute:Sekunde (hh:mm:ss)`. Der Wert 21 steht für die Datumsdarstellung `Tag.Monat.Jahr (dd.mm.yyyy)`.

Das Einstellen der Zeitzone erfolgt über den Baustein `tzcfg()`. Über eine Combobox in der Visualisierung kann die Zeitzone festgelegt werden. Siehe Abbildung 171.

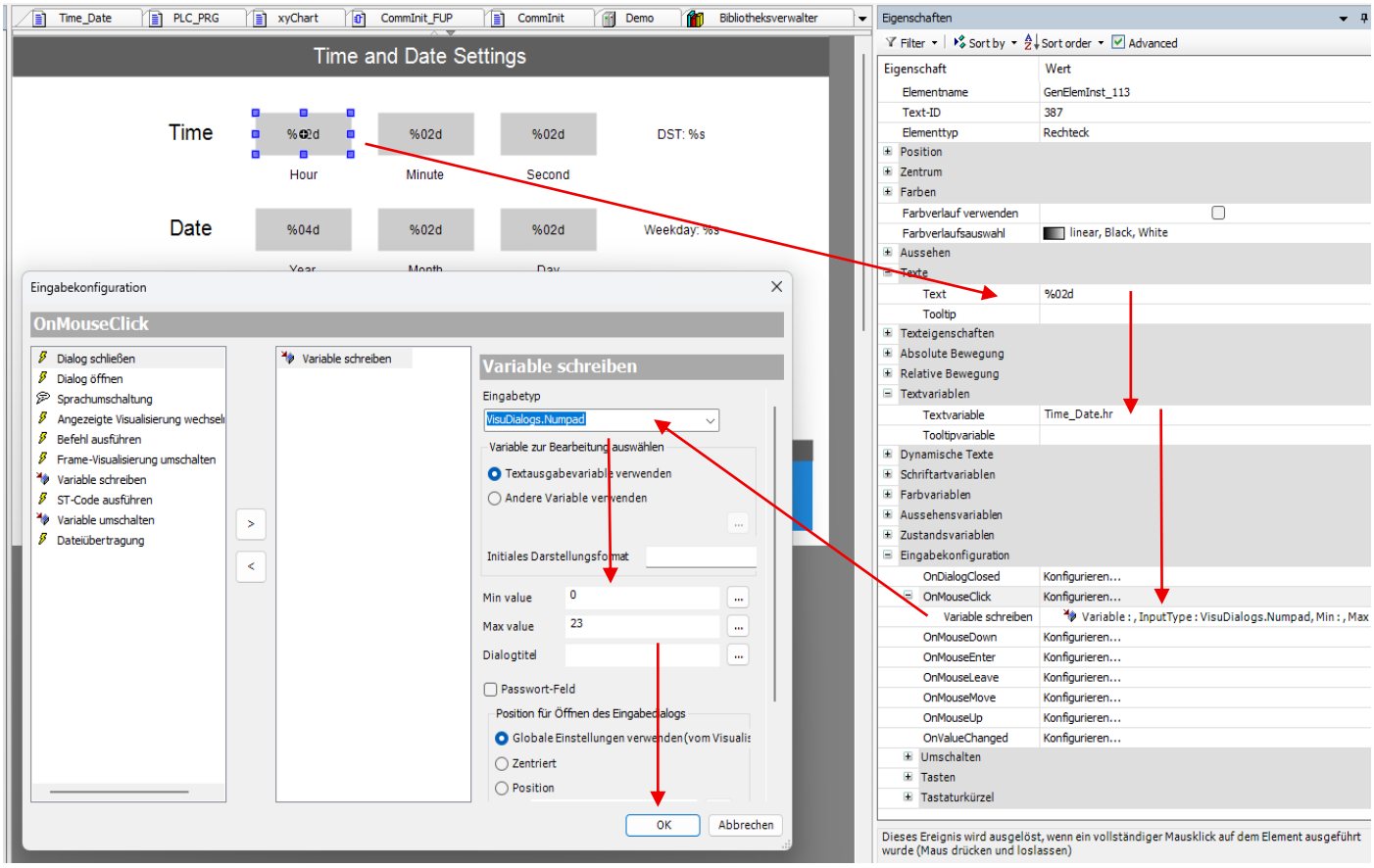


Abbildung 170: Konfiguration der Eingabe in der Visualisierung

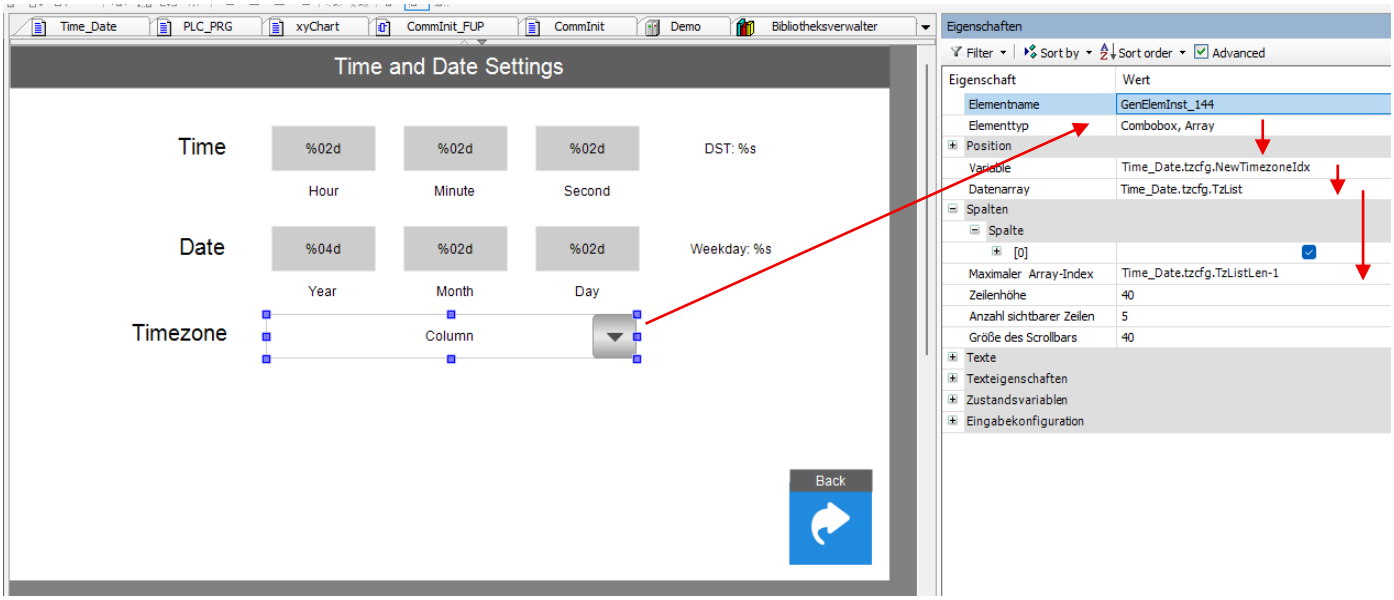


Abbildung 171: Konfiguration der Zeitzone in der Visualisierung

14.5 SNTP

Die aktuelle Uhrzeit kann bei einem NTP/SNTP-Server abgefragt werden und stellt damit im Erfolgsfall die Echtzeituhr (RTC). Eine derartige Abfrage kann z.B. einmal am Tag durchgeführt werden, um eine genaue Synchronisation der RTC mit anderen Uhren sicherzustellen. Hierfür wird der Baustein `SntpSet()` aus der Bibliothek **PlmStandard** benötigt.

Listen der verfügbaren NTP/SNTP-Server finden sich im Internet. Die Nutzung der Server ist fast immer kostenlos, einige Serverbetreiber bitten jedoch um Benachrichtigung, wenn der Dienst regelmäßig in Anspruch genommen wird.

14.5.1SNTPSET (PLMSTANDARD.LIBRARY)

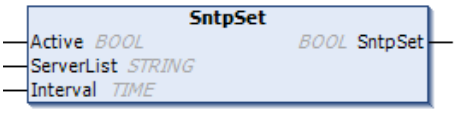


Abbildung 172: Funktion SntpSet (PlmStandard.Library)

Input Parameter:		
Active	BOOL	TRUE = fragt zyklisch die Systemzeit von einem SNTP-Server ab FALSE = Inaktiv
ServerList	STRING	Eine Liste von SNTP-Server, die Server können mit einem Semikolon oder Komma getrennt vorgegeben werden. Z.B.: '192.53.103.108;192.53.103.104' oder 'ptbtime1.ptb.de;ptbtime2.ptb.de'
Interval	TIME	Zeitangabe für das Intervall, mit dem die SNTP-Zeitserver abgefragt werden, z.B. t#12h. 0 deaktiviert die Synchronisation. Minimum sind t#5m.
Return-Wert:		
SntpSet	BOOL	

14.6 PROGRAMMBEISPIEL SNTP (ST)

Das in Abbildung 173 gezeigte Beispiel fragt die Uhrzeit bei einem NTP/SNTP-Server ab und stellt automatisch bei Erfolg die Uhrzeit. Die Abfrage erfolgt, wenn der Eingang `Active` von `FALSE` auf `TRUE` gesetzt wird.

Eine Liste von NTP/SNTP-Servern kann als String an den Eingang `ServerList` übergeben werden. Die Liste kann als IP-Adressen oder Serveradressen übergeben werden.

Das Intervall sollte nicht zu klein gewählt werden. In der Praxis reicht es aus, wenn die Abfrage einmal am Tag erfolgt oder auch noch seltener.

Time_Date

```
PROGRAM Time_Date
VAR
    mgTime: PlmStd.ManageLocalTime;
    yr: WORD;
    mt, dy, hr, mi, sc: BYTE;
    daytab: ARRAY[0..6] OF STRING := [ 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun' ];
    dayname: STRING;
    uhrzeit:STRING;
    datum:STRING;

    tzcfg: PlmStd.ManageTimezone;
    newTz: STRING;
    newTzIdx: WORD;
    start_sntp:bool;
END_VAR
```

100 %

```
// Baustein zum Einlesen und Schreiben von Datum und Uhrzeit
mgTime(
    Weekday=> ,
    DstActive=> ,
    Year:= yr,
    Month:= mt,
    Day:= dy,
    Hour:= hr,
    Minute:= mi,
    Second:= sc);

// Wochentag als String
dayname:=daytab[mgTime.Weekday];

// Format für Uhrzeit und Datum festlegen
Uhrzeit:=PlmStd.FormatTime(TimeHandle:=PlmStd.GetLocalTime() , FormatIdx:=30 ); // hh:mm:ss
Datum:=PlmStd.FormatTime(TimeHandle:=PlmStd.GetLocalTime() , FormatIdx:=21 );   // dd.mm.yyyy

// Baustein zum Einlesen und Schreiben der Zeitzone
tzcfg(
    CurTimezone=> ,
    TzList=> ,
    TzListLen=> ,
    NewTimezoneIdx:= newTzIdx,
    NewTimezone:= newTz);

//  alle 12 Stunden einen Abgleich der Uhrzeit über einen SNTP-Server durchführen
PlmStd.SntpSet(Active:=start_sntp, ServerList:='ptbtime1.ptb.de', Interval:=T#12h);
```

100 %

Abbildung 173: Abfrage eines SNTP-Servers und Stellen der Systemzeit

15 INI-FILES SCHREIBEN UND EINLESEN

15.1 ALLGEMEINES

INI-Dateien eignen sich zur strukturierten Speicherung von Parametern, Konfigurationsdaten und Rezepturen. Im CODESYS-Programm kann die gewünschte Struktur der INI-Datei durch definierte Schlüsselwörter (Sections) vorgegeben werden. Dadurch wird festgelegt, welche Daten enthalten sein müssen und in welcher Form diese organisiert werden.

Ähnlich wie CSV-Dateien können INI-Dateien mit einem beliebigen Texteditor komfortabel erstellt, angepasst und verwaltet werden. Nach der Bearbeitung lassen sich die Daten von der Steuerung einlesen und weiterverarbeiten. Durch die klare Gliederung in Abschnitte und Schlüssel-Wert-Paare bleibt die Datei auch bei umfangreichen Datensätzen übersichtlich und leicht wartbar.

Ein wesentlicher Vorteil der INI-Datei gegenüber der CSV-Datei besteht im vereinfachten Zugriff auf die gespeicherten Daten. Für das Lesen und Schreiben einer INI-Datei wird in der Regel lediglich ein einzelner Funktionsbaustein benötigt, der die Verwaltung aller Parameter übernimmt. Bei CSV-Dateien müssen

hingegen häufig mehrere Funktionsaufrufe oder zusätzliche Verarbeitungsroutinen implementiert werden, um einzelne Datensätze zu parsen und den entsprechenden Variablen zuzuordnen.

Insbesondere bei Anwendungen mit einer großen Anzahl von Parametern oder Rezepturwerten reduziert sich dadurch der Programmier- und Wartungsaufwand erheblich.

15.2 BENÖTIGTE BIBLIOTHEKEN

Benötigt werden die folgenden Bibliotheken:

PlmStandard > V3.5.16.28
PlmUtil > V3.5.16.42

Die angegebenen Bibliotheken müssen vom Projekt geladen werden, damit die Funktionen zur Verfügung stehen. Klicken Sie dazu auf den ***Bibliotheksverwalter*** > ***Bibliothek hinzufügen*** > ***Sonstige*** > ***PlmStandard*** > ***OK***. Wiederholen Sie den Schritt für die Bibliothek **PlmUtil**. Die benötigten Bausteine `IniFile()`, `IniFileListReadExtFb()` und `IniFileListWriteExtFb()` befinden sich in dem Ordner **INI-File** in der Bibliothek **PlmUtil**.

15.2.1 INIFILE (PLMUTIL.LIBRARY)

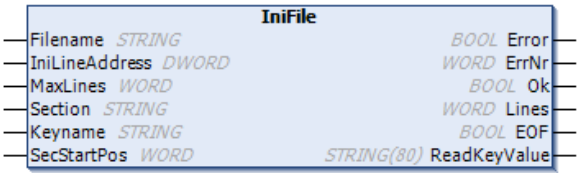


Abbildung 174: Funktionsblock IniFile (PlmUtil.Library)

Input Parameter:		
`Filename`	`STRING`	`Name der Ini-Datei mit Pfadangabe, z.B.: 'c/setup.ini'`
`IniLineAddress`	`DWORD`	`Adresse eines Arrays vom Typ INI_LINE_STRUCT`
`MaxLines`	`WORD`	`Maximale Anzahl von Werten, die verarbeitet werden sollen`
`Section`	`STRING`	`Name einer Section, z.B.: 'Rezeptur'`
`Keyname`	`STRING`	`Schlüsselwort innerhalb der Section, z.B.: 'Temperatur'`
`SecStartPos`	`WORD`	
Return-Wert:		
`Error`	`BOOL`	`TRUE = Es ist ein Fehler aufgetreten. Siehe ErrNr.` `FALSE = Baustein wurde erfolgreich abgearbeitet`

ErrNr	WORD	1 = Die angegebene Datei konnte zum Lesen nicht geöffnet werden 5 = Es wird versucht Daten von einer Datei einzulesen, die nicht geöffnet wurde 11 = Es wird versucht eine Datei zu öffnen, obwohl die Instanz bereits eine Datei geöffnet hat 12 = Es wird versucht in eine Datei zu schreiben, obwohl diese nicht geöffnet ist 101, 201, 203 = Dateiende Erreicht (EOF) und das EOF-Bit ist zusätzlich gesetzt 310 = Es wird versucht mehr Werte zu verarbeiten als in MaxLines angegeben
Ok	BOOL	TRUE = Verarbeitung des Bausteins durchgeführt
Lines	WORD	Anzahl der Verarbeiten Werte
EOF	BOOL	TRUE = End Of File, Dateiende erreicht
ReadKeyValue	STRING	Wert des Schlüsselwortes aus der Section

15.2.2 INIFILELISTREADEXTFB (PLMUTIL.LIBRARY)

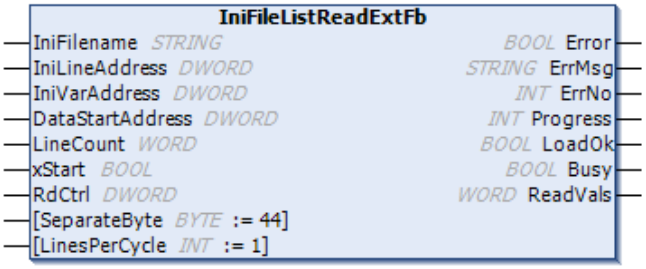


Abbildung 175: Funktionsblock IniFileListReadExtFb (PlmUtil.Library)

Input Parameter:		
IniFilename	STRING	Dateiname der INI-Datei mit Pfadangabe, z.B.: '*c/setup.ini*'
IniLineAddress	DWORD	Adresse eines Arrays vom Typ INI_LINE_STRUCT, das die Namen von Section (Abschnitt) und Keyname (Schlüsselwort)beschreibt
IniVarAddress	DWORD	Adresse eines Arrays vom Typ INI_VAR_STRUCT, das die Datenadressen für die Schlüsselwort-Werte beschreibt
DataStartAddress	DWORD	0 = Die gelesenen Daten werden in die Speicherbereiche geschrieben, deren Adressen im Array *IniVarAddress* hinterlegt sind
LineCount	WORD	Gibt an, wie viele gültige Einträge die Arrays *IniLineAddress* und *IniVarAddress* enthalten
xStart	BOOL	TRUE = Lesevorgang ausführen
RdCtrl	DWORD	RdCtrl.0 = TRUE oder RdCtrl:=16#0001, Lesen beenden, wenn ein in der Datei gelesener Abschnitt (Section) nicht im Array *IniLineAddress* definiert ist RdCtrl.1 = TRUE oder RdCtrl:=16#0002, Lesen beenden, wenn eine Section aus der Datei nicht im Array *IniLineAddress* definiert ist

		RdCtrl.2 = TRUE oder RdCtrl:=16#0004, Erlaubt Schlüsselwörter mit Werten am Anfang der Datei, die keiner Section zugeordnet sind. Die Schlüsselwörter werden in den definierten Sections (*IniLineAddress* / *IniVarAddress*) gesucht, bei denen zu *IniVarAddress*[].count das Bit 16#4000 hinzugefügt wurde
SeparateByte	BYTE	Trennzeichen wenn mehrere Werte hintereinander eingelesen werden. Standard ist das Komma(,) (ASCII Code 44). Beispiel 21,45,33…
LinesPerCycle	INT	Gibt die Lesegeschwindigkeit an. Mit diesem Wert gibt man an, wie viele Zeilen aus der Ini-Datei pro Aufruf eingelesen werden. Standard ist 1.
Return-Wert:		
Error	BOOL	TRUE = Fehler beim Einlesen
ErrMsg	STRING	Fehlertext (in Arbeit)
ErrNo	INT	1 = Keine Werte im *IniVarAddress*[] Array gefunden 10 = Datei kann nicht geöffnet werden 20 = Unbekannte Section 30 = Die Datei endet unerwartet mitten in einer Zeile 31 = Die Datei endet während des Einlesens eines Schlüsselwertes. Beispiel: *Port=* steht am Ende der Datei, ohne dass der eigentliche Wert vorhanden ist. 32 = Die Datei endet innerhalb einer Section-Definition. Beispiel: Die letzte Zeile enthält nur *[Settings* ohne schließende Klammer *]*
Progress	INT	Fortschritt in 0-100%
LoadOk	BOOL	Die Daten wurden erfolgreich eingelesen
Busy	BOOL	TRUE = Baustein wird verarbeitet
ReadVals	WORD	Anzahl der eingelesenen Werte

15.2.3 INIFILELISTWRITEEXTFB (PLMUTIL.LIBRARY)

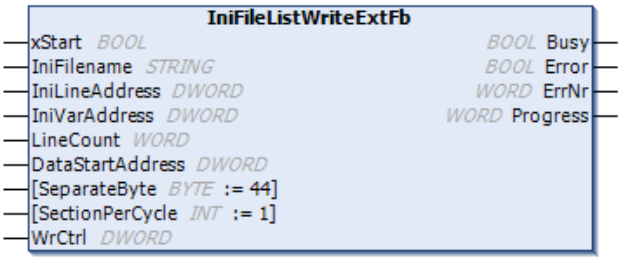


Abbildung 176: Funktionsblock IniFileListWriteExtFb (PlmUtil.Library)

Input Parameter:		
xStart	BOOL	TRUE = Schreibvorgang auslösen
IniFilename	STRING	Name der zu schreibenden INI-Datei, z.B.: *'c/setup.ini'*

IniLineAddress	DWORD	Adresse eines Arrays vom Typ INI_LINE_STRUCT, das die Namen von Section (Abschnitt) und Keyname (Schlüsselwort)beschreibt
IniVarAddress	DWORD	Adresse eines Arrays vom Typ INI_VAR_STRUCT, das die Datenadressen für die Schlüsselwort-Werte beschreibt
LineCount	WORD	Gibt an, wie viele gültige Einträge die Arrays *IniLineAddress* und *IniVarAddress* enthalten
DataStartAddress	DWORD	0 = Die Funktion schreibt die Daten aus den Variablen, deren Adressen im Array *IniVarAddress* hinterlegt sind. >0 = Adresse einer Strukturvariablen, deren Typ der in *IniVarAddress* beschriebenen Struktur entspricht.
SeparateByte	BYTE	Trennzeichen wenn mehrere Werte hintereinander geschrieben werden. Standard ist das Komma (,) (ASCII Code 44). Beispiel: 21,45,33…
SectionsPerCycle	INT	Gibt an, wie viele Sections pro Aufruf der Funktion geschrieben werden. Standard ist 1.
WrCtrl	DWORD	WrCtrl.0 = TRUE gibt an, dass eine boolesche Variable als 0/1 anstatt FALSE/TRUE in die INI-Datei geschrieben wird
Return-Wert:		
Busy	BOOL	TRUE = Baustein wird abgearbeitet
Error	BOOL	TRUE = Fehler beim Schreiben der INI-Datei
ErrNr	WORD	32 = Fehler beim Anlegen der Datei
Progress	WORD	Fortschritt in 0-100%

15.3 PROGRAMMBEISPIEL (ST)

Das Beispiel soll aufzeigen, wie man eine INI-Datei aufbaut, diese abspeichert und einliest. Dazu wird eine Struktur `CONFIG_STRUCT` angelegt, die für das Beispiel einen `STRING`, `REAL`, `WORD` und ein `ARRAY OF REAL` enthält.

CONFIG_STRUCT

```
1  TYPE CONFIG_STRUCT :
2  STRUCT
3      name        :STRING:='Anlage';      // Beschreibung
4      temperatur  :REAL:=21.0;            // Temperaturwert
5      druck       :WORD:=2;               // Druck
6      setpoint    :ARRAY[0..9] OF REAL;   // Vorgaben
7  END_STRUCT
8  END_TYPE
9
```

100 %

Abbildung 177: Beispiel-Struktur mit verschiedenen Werten (ST)

Um zehn unterschiedliche Konfigurationen auf Basis der Struktur `CONFIG_STRUCT` in einer INI-Datei abzulegen, ist ein globales Array mit zehn Einträgen zu definieren. Dieses Array wird mit dem Namen `Setup` angelegt und enthält die einzelnen Konfigurationsdatensätze.

GVLini

```
1  {attribute 'qualified_only'}
2  VAR_GLOBAL
3      Setup       :ARRAY[0..9] OF CONFIG_STRUCT;
4  END_VAR
```

100 %

Abbildung 178: Das Array Setup mit 10 Konfigurationssätzen (ST)

Ini_File

```
1  PROGRAM Ini_File
2  VAR
3      init_done           :BOOL;
4      IniDaten            :ARRAY[0..30]OF PlmUtil.INI_LINE_STRUCT;    // Beschreibung der Sections und Keywords
5      IniVal              :ARRAY[0..30]OF PlmUtil.INI_VAR_STRUCT;     // Beschreibung des Wertes
6      IniLinesMax         :WORD;
7      i                   :WORD;
8      do_read, do_write   :BOOL;
9  END_VAR
10
```

100 %

```
1  IF NOT init_done THEN
2      init_done:=TRUE;
3
4      (* Legt die Section Konfiguration an *)
5      IniDaten[i].section:=TRUE;                  // Section anlegen
6      IniDaten[i].name:='Konfiguration';          // Name der Section
7      IniVal[i].address:=ADR(GVLini.Setup);       // Die Strukur Setup aus den globalen Variablen angeben
8      IniVal[i].size:=SIZEOF(GVLini.Setup[0]);    // Größe der Struktur übergeben
9      IniVal[i].start_no:=0;                      // Anfangsnummer, die hinter den Namen "Konfiguration" angehangen wird
10     IniVal[i].count:=10;                        // Anzahl Datesätze
11     i:=i+1;                                     // Zähler für die Anzahl von Einträgen erhöhen
12
13     (* Bei Strings muss der Parameter size angegeben werden, da sonst zu viele Bytes
14         in einen zu kleinen Speicher kopiert werden können *)
15     IniDaten[i].section:=FALSE;                         // Keine Section anlegen
16     IniDaten[i].name:='Name';                           // Schlüsselwort anlegen
17     IniVal[i].address:=ADR(GVLini.Setup[0].name);       // Speicheradresse der Variablen "name" aus der Struktur Setup
18     IniVal[i].size:=SIZEOF(GVLini.Setup[0].name);       // Größe des String-Feldes ermitteln
19     IniVal[i].var_type:=PlmUtil.IniVar.INI_TYPE_STRING; // "name" ist vom Typ STRING
20     IniVal[i].count:=0;                                 // Keine zusätzliche Optionen
21     IniVal[i].default:='Testname';                      // Defaultname, falls kein Wert in der Struktur Setup.name existiert
22     i:=i+1;                                             // Zähler für die Anzahl von Einträgen erhöhen
23
24     IniDaten[i].section:=FALSE;                         // Keine Section anlegen
25     IniDaten[i].name:='Druck';                          // Schlüsselwort anlegen
26     IniVal[i].address:=ADR(GVLini.Setup[0].druck);      // Speicheradresse der Variablen "druck" aus der Struktur Setup
27     IniVal[i].var_type:=PlmUtil.IniVar.INI_TYPE_WORD;   // "druck" ist vom Typ WORD
28     IniVal[i].count:=0;                                 // Keine zusätzliche Optionen
29     IniVal[i].default:='1';                             // Defaultwert, falls kein Wert in der Struktur Setup.druck existiert
30     i:=i+1;                                             // Zähler für die Anzahl von Einträgen erhöhen
31
32     IniDaten[i].section:=FALSE;                         // Keine Section anlegen
33     IniDaten[i].name:='Temperatur';                     // Schlüsselwort anlegen
34     IniVal[i].address:=ADR(GVLini.Setup[0].temperatur); // Speicheradresse der Variablen "temperatur" aus der Struktur Setup
35     IniVal[i].var_type:=PlmUtil.IniVar.INI_TYPE_REAL;   // "temperatur" ist vom Typ REAL
36     IniVal[i].count:=0;                                 // Keine zusätzliche Optionen
37     IniVal[i].default:='10.5';                          // Defaultwert, falls kein Wert in der Struktur Setup.temperatur existiert
38     i:=i+1;                                             // Zähler für die Anzahl von Einträgen erhöhen
39
40     IniDaten[i].section:=FALSE;                         // Keine Section anlegen
41     IniDaten[i].name:='Setpoint';                       // Schlüsselwort anlegen
42     IniVal[i].address:=ADR(GVLini.Setup[0].setpoint);   // Speicheradresse des Arrays "setpoint" aus der Struktur Setup
43     IniVal[i].var_type:=PlmUtil.IniVar.INI_TYPE_REAL;   // "setpoint" ist vom Typ REAL
44     IniVal[i].count:=10;                                // Anzahl der Werte, welche in eine Zeile geschrieben werden - mit Komma getrennt
45     IniVal[i].default:='0.5';                           // Defaultwert, falls kein Wert in der Struktur Setup.setpoint existiert
46     i:=i+1;                                             // Zähler für die Anzahl von Einträgen erhöhen
47
48     IniLinesMax:=i;
49 END_IF
50
51 // INI-Datei einlesen
52 IF do_read THEN
53     do_read:=FALSE;
54     PlmUtil.IniFileListReadExt.xStart:=TRUE;
```

100 %

Abbildung 179: Programmbeispiel für ein INI-File (ST, Teil 1/2)

Die Struktur der zu erzeugenden INI-Datei wird in den Programmzeilen 1 bis 49 definiert. Dieser Programmabschnitt wird einmalig während der Initialisierung ausgeführt und dient der

Konfiguration der Sections, Schlüsselwörter (Keywords) sowie der Zuordnung der zu speichernden Variablen aus der globalen Struktur `Setup`.

In Zeile 5 wird durch die Zuweisung `IniDaten[i].section := TRUE` eine neue Section angelegt. Der Name der Section wird über `IniDaten[i].name` festgelegt und lautet in diesem Beispiel „`Konfiguration`".

In den Zeilen 15, 24, 32 und 40 wird `IniDaten[i].section := FALSE` gesetzt. Dadurch wird kein neuer Section-Eintrag erzeugt, sondern ein Schlüsselwort innerhalb der aktuell definierten Section angelegt. Der Name des jeweiligen Schlüsselwortes wird ebenfalls über `IniDaten[i].name` festgelegt.

Die Variable `IniVal[i].address` enthält die Speicheradresse der zugehörigen Strukturvariable `Setup`, deren Wert in die INI-Datei geschrieben bzw. aus dieser gelesen werden soll. Über `IniVal[i].var_type` wird der Datentyp der Variablen angegeben.

Mit `IniVal[i].default` kann ein Standardwert angegeben werden. Dieser wird verwendet, wenn für die entsprechende Variable kein gültiger Wert vorhanden ist oder beim Einlesen der INI-Datei kein Eintrag gefunden wird.

```
42      IniVal[i].address:=ADR(GVLini.Setup[0].setpoint);   // Speicheradresse des Arrays "setpoint" aus der Struktur Setup
43      IniVal[i].var_type:=PlmUtil.IniVar.INI_TYPE_REAL;   // "setpoint" ist vom Typ REAL
44      IniVal[i].count:=10;                                // Anzahl der Werte, welche in eine Zeile geschrieben werden - mit Komma getrennt
45      IniVal[i].default:='0.5';                           // Defaultwert, falls kein Wert in der Struktur Setup.setpoint existiert
46      i:=i+1;                                             // Zähler für die Anzahl von Einträgen erhöhen
47
48      IniLinesMax:=i;
49  END_IF
50
51  // INI-Datei einlesen
52  IF do_read THEN
53      do_read:=FALSE;
54      PlmUtil.IniFileListReadExt.xStart:=TRUE;
55  END_IF
56
57  PlmUtil.IniFileListReadExt(
58      IniFilename:= '/c/ext_setup.ini' ,      // Speicherort der INI-Datei mit Namen
59      IniLineAddress:=ADR(IniDaten) ,
60      IniVarAddress:=ADR(IniVal),
61      DataStartAddress:=0,
62      LineCount:=IniLinesMax ,
63      xStart:= ,
64      RdCtrl:= ,
65      SeparateByte:= ,
66      LinesPerCycle:= 10,
67      Error=> ,
68      ErrMsg=> ,
69      ErrNo=> ,
70      Progress=> ,
71      LoadOk=> ,
72      Busy=> ,
73      ReadVals=> );
74
75  // Ini-Datei schreiben
76  IF do_write THEN
77      do_write:=FALSE;
78      PlmUtil.IniFileListWriteExt.xStart:=TRUE;
79  END_IF
80
81  PlmUtil.IniFileListWriteExt(
82      xStart:= ,
83      IniFilename:='/c/ext_setup.ini' ,       // Speicherort der INI-Datei mit Namen
84      IniLineAddress:=ADR(IniDaten) ,
85      IniVarAddress:=ADR(IniVal),
86      LineCount:= IniLinesMax,
87      DataStartAddress:=0,
88      SeparateByte:= ,
89      SectionPerCycle:=10 ,
90      WrCtrl:= ,
91      Busy=> ,
92      Error=> ,
93      ErrNr=> ,
94      Progress=> );
```

100 %

Abbildung 180: Programmbeispiel für ein INI-File (ST, Teil 2/2)

Durch Setzen der Variablen `do_write` bzw. `do_read` auf `TRUE` wird das Schreiben bzw. Einlesen der INI-Datei ausgelöst. Der Dateipfad wird den Funktionsbausteinen `PlmUtil.IniFileListWriteFb()` und `PlmUtil.IniFileListReadFb()` über den Eingang `IniFilename` übergeben. Im vorliegenden Beispiel befindet sich die Datei `ext_setup.ini` auf einem USB-Datenträger.

Die Eingänge `IniLineAddress` und `IniVarAddress` erhalten die Speicheradressen der Variablen `IniDaten` und `IniVal`. Dabei handelt es sich um Arrays der Typen `PlmUtil.INI_LINE_STRUCT` bzw. `PlmUtil.INI_VAR_STRUCT`, welche die Struktur der zu schreibenden bzw. zu lesenden INI-Datei beschreiben.

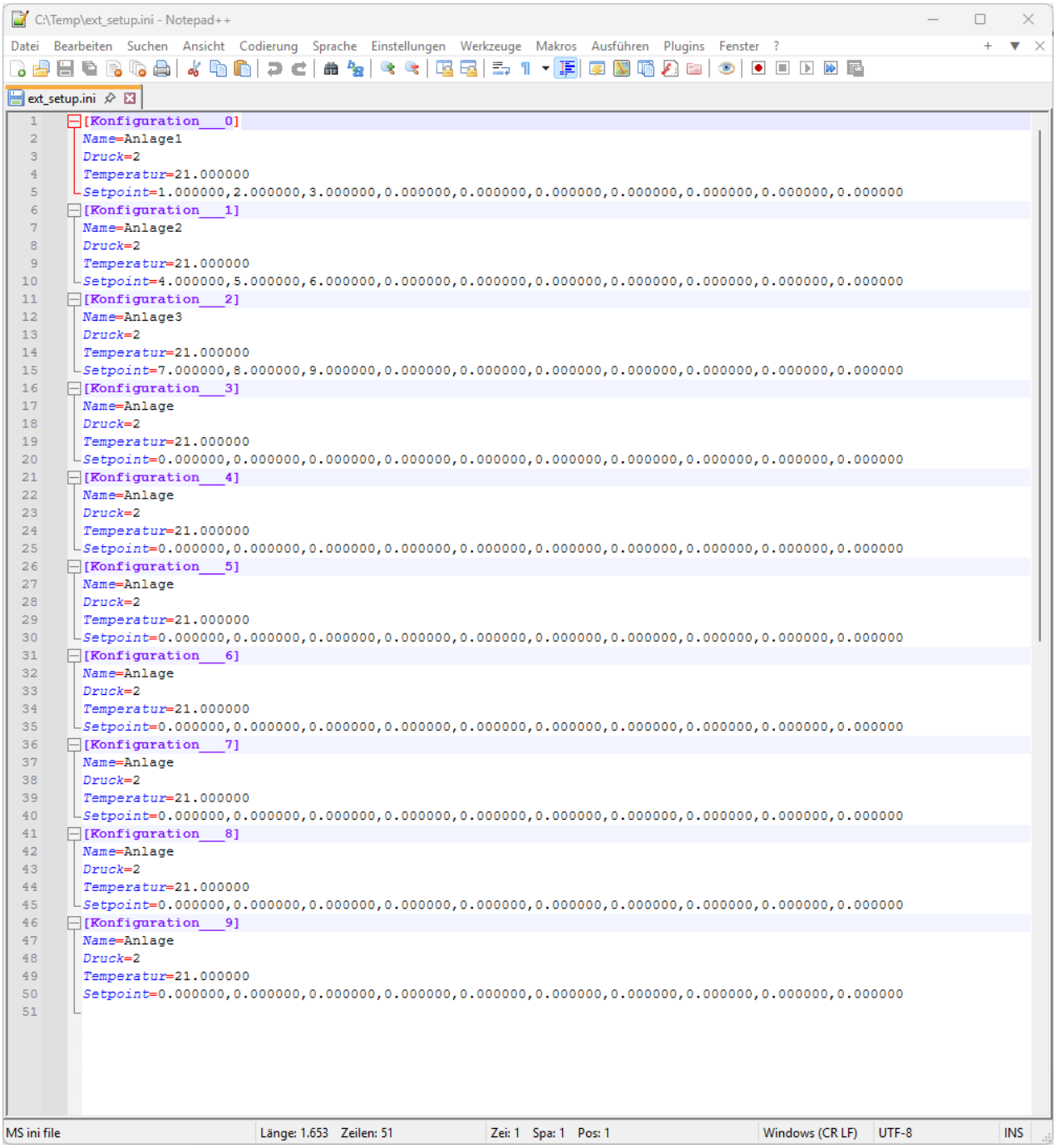


Abbildung 181: Beispiel einer Ini-Datei

In der Abbildung 181 ist die INI-Datei aus dem Programmbeispiel zu sehen, in der den ersten drei Parametersätzen einige Werte hinzugefügt wurden.

16 E-MAILS VERSENDEN

16.1 ALLGEMEINES

Alle PLM 800-Steuerungen mit Ethernet-Anschluss bieten die Möglichkeit, E-Mails durch das IEC-Programm zu versenden.
Voraussetzung ist, dass die Steuerung einen Internet-Zugang besitzt. Hierfür sind u.a. einzustellen:

- **IP-Adresse**
- **Netzmaske**
- **Default-Gateway**
- **ggf. DNS**

Default-Gateway ist die IP-Adresse des Routers, über den die Steuerung ins Internet gelangt, sofern sie nicht direkt mit dem Internet verbunden ist und eine öffentliche IP-Adresse besitzt.

Der DNS (Domain Name Service) löst Namen in IP-Adressen auf. Da einige Mail Provider keine feste IP-Adresse verwenden, sollte der Zugriff auf den SMTP-Server über einen Namen erfolgen (z.B. `smtp.web.de`). Hierzu muss ein DNS-Server auf der Steuerung bekannt sein.

Des Weiteren muss der Anwender eine Zugangsberechtigung auf einem SMTP-Server (Mail-Server) haben, der die Mail der Steuerung weiterleitet. Dabei kann es sich um einen kommerziellen Account handeln oder um einen kostenlosen Mail-Dienst, wie er von vielen Providern angeboten wird

16.2 BENÖTIGTE BIBLIOTHEKEN

Benötigt wird die folgende Bibliothek:

PlmStandard > V3.5.16.28

Die angegebene Bibliothek muss vom Projekt geladen werden, damit die Funktionen zur Verfügung stehen. Klicken Sie dazu auf den ***Bibliotheksverwalter*** > ***Bibliothek hinzufügen*** > ***Sonstige*** > ***PlmStandard*** > ***OK***. Der benötigte Baustein `SmtpSendMail()` zum Versenden von E-Mails befinden sich in dem Ordner **Mail** in der Bibliothek **PlmStandard**.

16.2.1 SMTPSENDMAIL (PLMSTANDARD.LIBRARY)

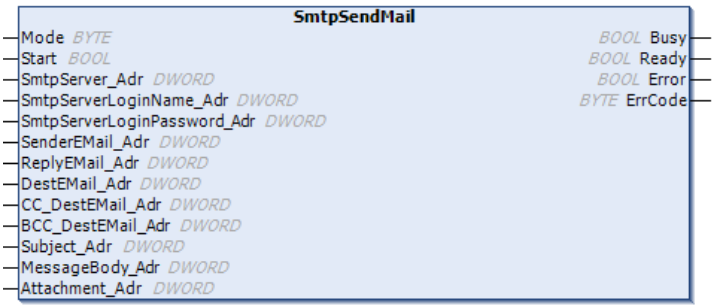


Abbildung 182: Funktionsblock SmtpSendMail (PlmStandard.Library)

Input Parameter:		
Mode	BYTE	16#00 = Keine Authentifizierung beim SMTP-Server 16#01 = SMTP Plain Authentification verwenden (benötigt SmtpServerLoginName und SmtpServerLoginPassword) 16#02 = STARTTLS verwenden (benötigt SmtpServerLoginName und SmtpServerLoginPassword) 16#03 = SSL verwenden (benötigt SmtpServerLoginName und SmtpServerLoginPassword) +16#10 = Unicode: Subject und MessageBody sind WSTRING (UTF-16) +16#20 = "Content-Transfer-Encoding: 8bit" für MessageBody erzwingen (sonst: "quoted-printable") +16#40 = "Content-Type: multipart" erzwingen, auch wenn keine Attachments angegeben sind +16#80 = Debug-Ausgabe speichern in 'a/mail_debug.txt'
Start	BOOL	Wechsel von FALSE nach TRUE starten den E-Mailversand
SmtpServer_Adr	DWORD	Adresse eines Strings mit IP-Adresse und ggf. Port des SMTP-Servers, z.B. 'my.mail.hoster.de' oder 'my.mail.hoster.de:25'
SmtpServerLoginName_Adr	DWORD	Adresse eines Strings mit Login-Name des Mailkontobesitzers, z.B. 'erika.mustermann@web.de'
SmtpServerLoginPassword_Adr	DWORD	Adresse eines Strings mit Password für das Mailkonto, z.B. 'erikaspasswd'
SenderEmail_Adr	DWORD	Adresse eines Strings mit Absenderadresse, die in der Mail erscheint, z.B. 'erika.mustermann@web.de', muss häufig mit SmtpServerLoginName identisch sein
ReplyEmail_Adr	DWORD	Adresse eines Strings mit Mailadresse, an die Antworten geschickt werden sollen, z.B. 'support@anlage.de'
DestEmail_Adr	DWORD	Adresse eines Strings mit Mailadressen der Empfänger (durch Semikolon getrennt), z.B. 'hausmeister@schule.de'
CC_DestEmail_Adr	DWORD	Adresse eines Strings mit Mailadressen der CC Empfänger (durch Semikolon getrennt)

BCC_DestEmail_Adr	DWORD	Adresse eines Strings mit Mailadressen der BCC Empfänger (durch Semikolon getrennt)
Subject_Adr	DWORD	Adresse eines Strings mit Betreffzeile, z.B. 'Anlagenfehler'
MessageBody_Adr	DWORD	Adresse eines Strings mit Text der Mail (Zeilenumbrüche mit '$N'), z.B. 'Fehler:$NHeizkreis 3 ausgefallen'
Attachment_Adr	DWORD	Adresse eines Strings mit Dateinamen der Attachments (durch Semikolon getrennt), z.B. 'a/log.txt;a/test.dat', oder 0
Return-Werte:		
Busy	BOOL	TRUE = E-Mailversand wird ausgeführt
Ready	BOOL	TRUE = Baustein fertig abgearbeitet. Mail wurde erfolgreich versendet (an SMTP-Server übergeben)
Error	BOOL	TRUE = Fehler beim versenden aufgetreten, siehe ErrCode
ErrCode	BYTE	Fehlercode: 0 = ok (kein Fehler) 230 = internal error 231 = message data rejected by server 232 = one or more recipient email addresses rejected by server 233 = sender email address rejected by server 234 = not enough memory 235 = command rejected by server 236 = connect() or select() failed 237 = read() failed 238 = write() failed 239 = socket() failed 240 = close() failed 241 = mail was cancelled 242 = illegal parameters 243 = could not deliver mail before timeout 244 = internal error 245 = authentication failed (wrong user/password) 247 = attachment filename empty 248 = cannot read attachment file 250 = no license

Zum Versenden einer E-Mail muss dieser Funktionsblock durch eine Flanke von FALSE auf TRUE am Eingang Start ausgelöst werden. Für jede weitere E-Mail ist der Funktionsblock erneut mit einer steigenden Flanke (FALSE → TRUE) am Eingang Start aufzurufen.

Alle String-Eingänge, deren Namen mit . . . _Adr enden, erwarten die Adresse einer String-Variablen; hierzu bietet CODESYS den Operator ADR(). An unbenutzte String-Eingänge muss als Adresse der Wert 0 gelegt werden.

Solange der Versand läuft, werden die Werte an den String-Eingängen intern verwendet und dürfen deshalb nicht geändert werden.

Das in `Mode` eingestellte Authentifizierungsverfahren für die Anmeldung am SMTP Server wird vom Server vorgegeben und ist ggf. beim Provider nachzufragen. Keine Authentifizierung wird aus Sicherheitsgründen nur von sehr wenigen Mailservern unterstützt.

Wenn kein TCP-Port hinter dem Server-Namen angegeben wird, verwendet der Baustein als Standard den SMTP-Port 25. Für STARTTLS ist es üblich, entweder den Port 25 oder 578 zu verwenden, für SLL den Port 465.

Der Versand der E-Mail beginnt, wenn der Eingang Start von FALSE auf TRUE wechselt und kann mehrere Sekunden dauern, abhängig von der Menge der zu übertragenden Daten und der Reaktionsgeschwindigkeit des SMTP-Servers. Nach Beginn des Versands kann Start sofort wieder auf FALSE zurückwechseln, ohne dadurch den Versand zu beeinträchtigen.

Bei Erfolg (d.h. der SMTP-Server hat die Mail entgegengenommen) geht `Ready` für mind. einen Zyklus auf `TRUE`. Andernfalls geht `Error` für mind. einen Zyklus auf `TRUE` und `ErrorNo` enthält einen entsprechenden Fehlercode. `Ready` und `Error` bleiben anschließend `TRUE`, solange Start noch `TRUE` ist.

In `DestEMail_Adr`, `CC_DestEMail_Adr` und `BCC_DestEMail_Adr` können jeweils eine oder mehrere E-Mail-Adressen angegeben werden. Bei mehreren E-Mail-Adressen müssen diese in einem String durch Komma oder Semikolon getrennt werden.

`SenderEMail_Adr` wird meist von SMTP-Servern überprüft und kann dann nicht beliebig gewählt werden.

Falls der Empfänger eine Antwort auf die Mail verfasst, soll diese normalerweise an eine bestimmte Person gehen und nicht zurück an die Steuerung, daher ist eine entsprechende Angabe in `ReplyEMail_Adr` möglich. Diese Angabe wird ggf. vom Mail-Empfänger ausgewertet.

Der String `MessageBody_Adr` enthält den eigentlichen Text der E-Mail. Zeilenumbrüche im Text können durch Einfügen der String-Konstanten `'$N'` erzeugt werden. Der Text kann maximal ca. 1024 Zeichen lang sein. Längere Texte können als Attachment geschickt werden.

Sollen ein oder mehrere Attachments (Anhangsdateien) mitgeschickt werden, so sind diese in einem String aufzulisten, dessen Adresse an `Attachment_Adr` angelegt wird. Jeder Dateiname muss einen Laufwerksbuchstaben enthalten, z.B. `'a/test.dat'`. Bei mehreren Dateien müssen die Dateinamen durch Komma oder Semikolon getrennt werden, z.B. `'a/test.dat;a/xy.log'`. Optional kann hinter jedem Dateinamen ein Doppelpunkt und dahinter der MIME-Typ der Datei angegeben werden. Ohne explizite Angabe ist die Voreinstellung *application/octet-stream*. Diese Angabe wird ggf. vom Mail-Empfänger ausgewertet.

16.3 DAUER DER DATENÜBERTRAGUNG

Die Dauer des E-Mail-Versands hängt davon ab, mit welchen Parametern `SmtpSendMail` aufgerufen wird, ob Attachments mitgeschickt werden sollen, wie groß diese sind, sowie von der Reaktionsgeschwindigkeit des SMTP-Servers.

Da die Übertragung auf keinen Fall in einem einzigen IEC-Zyklus (Standardwert 20 ms) abgeschlossen werden kann, sind mehrere Aufrufe des Bausteins `SmtpSendMail()` notwendig, bis der Baustein an den Ausgängen `Ready` oder `Error` den Abschluss der Datenübertragung meldet.

Die minimale Übertragungsdauer bei Versand ohne STARTTLS oder SSL beträgt 35 Zykluszeiten, d.h. eine Mail mit kurzem Message Body und ohne Attachments, die an einen einzigen Empfänger adressiert ist, benötigt

mind. 35 Aufrufe von `SmtpSendMail()`. Bei einem Aufrufintervall von 20 ms entspricht dies ca. 0,7 Sekunden, vorausgesetzt, dass der SMTP-Server innerhalb von 20 ms die jeweiligen Anfragen beantwortet.

Bei langsamer Reaktion des SMTP-Servers werden automatisch Wartezyklen ausgeführt, die die Dauer der Datenübertragung verlängern. Antwortet der SMTP Server auf eine Anfrage nicht innerhalb von 20 Sekunden, so wird der Versand mit Timeout abgebrochen.

Die Übertragung von Attachments erfolgt bei Versand ohne STARTTLS oder SSL in Blöcken mit je 702 Bytes pro Zyklus. Ein Attachment von 10 kByte benötigt daher mind. 15 Zyklen für die Übertragung bei ausreichend schneller Reaktion des SMTP Servers.

16.4 PROGRAMMBEISPIEL (ST)

Mailer

```
1  PROGRAM Mailer
2  VAR
3      start, busy, ready, errror      :BOOL;
4      errorNo                         :BYTE;
5      sendmail                        :Plmstd.SmtpSendMail;
6      server                          :STRING:='217.72.192.157';
7      user                            :STRING:='user@web.de';
8      pass                            :STRING:='mypass';
9      sendfrom                        :STRING:='user@web.de';
10     sendto                          :STRING:='personal@leitwarte.de';
11     subject                         :STRING:='Mail von PLM800';
12     message                         :STRING:='Zeile 1$NZeile 2$NZeile3';
13 END_VAR
14
```

100 %

```
1  sendmail(
2      Mode:= 1,
3      Start:= start,
4      SmtpServer_Adr:= ADR(server),
5      SmtpServerLoginName_Adr:= ADR(user),
6      SmtpServerLoginPassword_Adr:= ADR(pass),
7      SenderEMail_Adr:= ADR(sendfrom),
8      ReplyEMail_Adr:= 0,
9      DestEMail_Adr:= ADR(sendto),
10     CC_DestEMail_Adr:= 0,
11     BCC_DestEMail_Adr:= 0,
12     Subject_Adr:= ADR(subject),
13     MessageBody_Adr:= ADR(message),
14     Attachment_Adr:= 0,
15     Busy=> busy,
16     Ready=> ready,
17     Error=> error,
18     ErrCode=> errorNo);
```

100 %

Abbildung 183: Beispielprogramm für das Versenden von E-Mails (ST)

Das in Abbildung 183 gezeigte Beispiel demonstriert den Versand von E-Mails.

Zum Starten muss die Variable `start` auf `TRUE` gesetzt werden. Die Variable `busy` ist `TRUE`, solange der Versand läuft.

Nach Ende des Versands hat entweder `ready` oder `error` den Wert `TRUE`. Wenn `start` bereits wieder auf `FALSE` zurückgesetzt wurde, liegen die Werte an `ready` bzw. `error` nur für einen einzigen Zyklus an, andernfalls so lange, wie `start` noch auf `TRUE` steht.

Der SMTP-Server hat im Beispiel die IP-Adresse 217.72.192.157 (alte IP-Adresse von *smtp.web.de*) und erwartet eine unverschlüsselte Authentifizierung (`Mode=1`) mit Name (hier: *user@web.de)* und Password (hier: *mypass*). Mit den gleichen Anmeldedaten ist eine Anmeldung am Freemail-Webinterface von Web.de möglich. Bei Web.de müssen Loginname (`LoginName_Adr`) und Absender der E-Mail (`SenderEMail_Adr`) identisch sein.

Der Empfänger der E-Mail ist *personal@leitwarte.de*, die Betreffzeile lautet *Mail von PLM*. Der eigentliche Text der E-Mail besteht hier aus drei Zeilen, die durch Zeilenumbrüche `($N)` getrennt sind.

An die nicht verwendeten Eingänge wird die Adresse 0 angelegt.

Im Erfolgsfall geht `ready` für einen Zyklus auf `TRUE`. Andernfalls geht `error` für einen Zyklus auf `TRUE` und `ErrorNo` liefert einen Fehlercode, der Rückschlüsse auf die Art des Fehlers zulässt.

16.5 VERBINDUNGSPARAMETER EINIGER E-MAIL ANBIETER

Die folgenden Einstellungen wurden Stand Mai 2023 getestet.

16.5.1 WEB.DE

Zunächst muss auf der Webseite des Anbieters im Mail-Account der Zugriff über IMAP und POP3 aktiviert werden. Dazu im Postfach im Menü links auf *Einstellungen*, dann unter *POP3/IMAP-Abruf* das Häkchen *POP3 und IMAP Zugriff erlauben* setzen und *Speichern*.
Parameter am Baustein `SmtpSendMail`:

`Mode`	`2 (STARTTLS)`
`SmtpServer`	`Smtp.web.de:587`
`LoginName`	`Web.de User oder vollständige E-Mail Adresse`
`LoginPassword`	`Passwort des web.de Users`

16.5.2 T-ONLINE

Zunächst muss auf der Webseite des Anbieters *Ihr speziell für E-Mail-Programme angelegtes Passwort* angelegt werden. Dafür oben im Header links auf das Zahnrad und im Drop Down Menü ganz unten auf *Alle Einstellungen*. Dann in der linken Menüleiste auf *Passwörter* und *Passwort für E-Mail-Programme* auswählen, dort das Passwort definieren. IMAP und POP3 explizit zu aktivieren ist nicht notwendig, da standardmäßig aktiviert.
Parameter am Baustein `SmtpSendMail`:

`Mode`	`3 (SSL)`
`SmtpServer`	`Securesmtp.t-online.de:465`
`LoginName`	`Vollständige T-Online E-Mail Adresse`

LoginPassword	Das im Web-Interface speziell angelegte Passwort

16.5.3 GMX

Zunächst muss auf der Webseite des Anbieters im Mail-Account der Zugriff über IMAP und POP3 aktiviert werden. Dazu im Postfach im Menü links auf *Einstellungen*, dann unter *POP3/IMAP-Abruf* das Häkchen *POP3 und IMAP Zugriff erlauben* setzen und *Speichern*.
Parameter am Baustein `SmtpSendMail`:

Mode	2 (STARTTLS) oder 3 (SSL)
SmtpServer	securesmtp.t-online.de:587 oder securesmtp.t-online.de:465
LoginName	Vollständige GMX E-Mail Adresse
LoginPassword	Das Passwort des GMX-Benutzers

16.5.4 OUTLOOK

Zunächst muss auf der Webseite des Anbieters im Mail-Account der Zugriff über IMAP und POP3 aktiviert werden. Dazu auf das Zahnrad (*Einstellungen*) oben rechts im Postfach-Header, dort ganz unten auf *Alle Outlook-Einstellungen anzeigen*, dort *EMail → E-Mail-Einstellungen*, im Abschnitt *POP und IMAP* unter *POP-Optionen Geräten und Apps die Verwendung von POP gestatten* auf *Ja* einstellen.
Parameter am Baustein `SmtpSendMail`:

Mode	2 (STARTTLS)
SmtpServer	Smtp.office365.com:587
LoginName	Vollständige E-Mail Adresse
LoginPassword	Das Passwort des Benutzers

16.6 FEHLERSUCHE

Der Baustein `SmtpSendMail()` liefert im Fehlerfall einen Fehlercode, der eine erste Fehlerdiagnose erlaubt.

Bei Fehler 236 (Verbindung zum SMTP-Server konnte nicht hergestellt werden) ist zu prüfen, ob die Internet-Verbindung der Steuerung richtig konfiguriert wurde und funktioniert (insbesondere Default-Gateway prüfen).

Bei Fehler 236 oder 243 (Timeout) ist zunächst die IP-Adresse des SMTP-Servers zu kontrollieren. Öffnen Sie dazu unter Windows eine Eingabekonsole (z.B. *Start → Ausführen → cmd*) und geben Sie den Befehl *nslookup servername* ein, z.B. *nslookup smtp.web.de*. Sie erhalten dann eine Antwort wie z.B. *Nicht autorisierte Antwort: Name: smtp.web.de, Address: 217.72.192.157*. Die *Address* ist die benötigte IP-Adresse des SMTP-Servers. Den Namen des SMTP-Servers (z.B. *smtp.web.de*) erfahren Sie von Ihrem Provider.

In manchen Fällen ist es hilfreich, die Kommunikation mit dem SMTP-Server im Detail zu beobachten. Hierzu kann am Eingang `Mode` der Wert 128 (+16#80) addiert werden, wodurch eine Log-Datei mit dem Namen `mail_debug.txt` auf FTP-Laufwerk a/ angelegt wird. Diese Textdatei kann über FTP von der Steuerung heruntergeladen und mit einem Texteditor geöffnet werden. Die Textdatei enthält mitunter Klartextfehlermeldungen des SMTP-Servers, aus denen häufig die genaue Fehlerursache diagnostiziert werden kann. Die Datei wird bei jedem neuen Mail Versand überschrieben.

Da die Log-Datei, insbesondere beim Versand von Attachments, sehr groß werden kann, sollte diese Funktion nach dem Debuggen unbedingt wieder abgeschaltet werden.

17 M-BUS

17.1 ALLGEMEINES

Der M-Bus (Meter-Bus) wird in der Gebäudeautomation zur Fernabfrage von Verbrauchszählern, wie z.B. Wasser-, Strom- oder Wärmemengenzählern, eingesetzt. Eine Beschreibung erfolgt in der EN13757. Weitere Informationen sind auf der Website http://www.m-bus.com/ verfügbar.

Die Datenübertragung erfolgt seriell mit niedriger Baudrate auf einer verpolungssicheren Zweidrahtleitung. Dabei übernimmt ein Master die Abfrage von bis zu 250 Slaves, die durch ihre Primäradresse ausgewählt werden. Üblicherweise erfolgt die Abfrage in großen Zeitintervallen, z.B. einmal pro Stunde oder einmal pro Monat.

Die von einem Slave zur Verfügung gestellten Datenwerte hängen vom Gerätehersteller und vom Slave-Typ ab und können bei einigen Slaves zusätzlich konfiguriert werden. Ein Datenwert besteht immer aus dem Zahlenwert, z.B. "8,23" und einer codierten Einheit, z.B. "× 0,1 kWh". Innerhalb einer Abfrage werden üblicherweise mehrere Datenwerte übertragen.

Für das PLM-System steht ein M-Bus-Mastermodul SIM.730.20 zur Verfügung, an welches bis zu 20 Slaves angeschlossen werden können. Bei Bedarf können mehrere Mastermodule gleichzeitig betrieben werden.

Die Programmierung in CODESYS erfolgt mittels der Bibliothek PlmProtocol und den im Ordner M-Bus SIM.730.20 enthaltenen Funktionsbausteinen, die im Folgenden erläutert werden. Darüber hinaus wird auf das Datenblatt zum SIM.730.20 verwiesen.

17.2 SPEZIFIKATION

- M-Bus Master, bestehend aus M-Bus-Mastermodul SIM.730.20 und CODESYS Bibliothek PlmProtocol
- Beliebige Baudrate, u.a. 300, 2400 und 9600 Baud
- Unterstützt M-Bus-Slaves mit Primäradressierung
- Anschluss von bis zu 20 Standard-Slaves an einem M-Bus-Mastermodul SIM.730.20, mehrere Module können gleichzeitig verwendet werden
- Bibliotheksbausteine zum automatischen Abwickeln der M-Bus-Zugriffe und zur Auswertung der Slave-Antworten
- Vorgefertigte Bausteine für Wärmemengenzähler, Elektrizitätszähler, Wasserzähler, Gaszähler

- Optionale Initialisierungssequenz mit SND_NKE
- Auswertung der Antworttypen *Variable Data Structure* und *Fixed Data Structure*
- Debug-Möglichkeiten zur Fehlersuche

17.3 VORGEHENSWEISE

Die Vorgehensweise wird anhand eines Programmierbeispiels illustriert.

Das M-Bus-Mastermodul SIM.730.20 wird wie eine serielle Schnittstellenerweiterung in die Steuerungskonfiguration von CODESYS eingebunden. Die M-Bus-Baudrate wird in den *Service Data Objects* des Moduls festgelegt.

Beim Start des Programms muss zunächst die Schnittstelle einmalig mit dem Baustein SIM_730_COM initialisiert werden (vgl. Abschnitt 7.3). Das dabei gelieferte Handle (comm_nr) dient als Kennung für alle Bausteine, die den M-Bus-Master benötigen.

Die Bausteine aus dem Ordner M-Bus SIM.730.20 der Bibliothek PlmProtocol.Library sind in der *Abbildung 184* dargestellt.

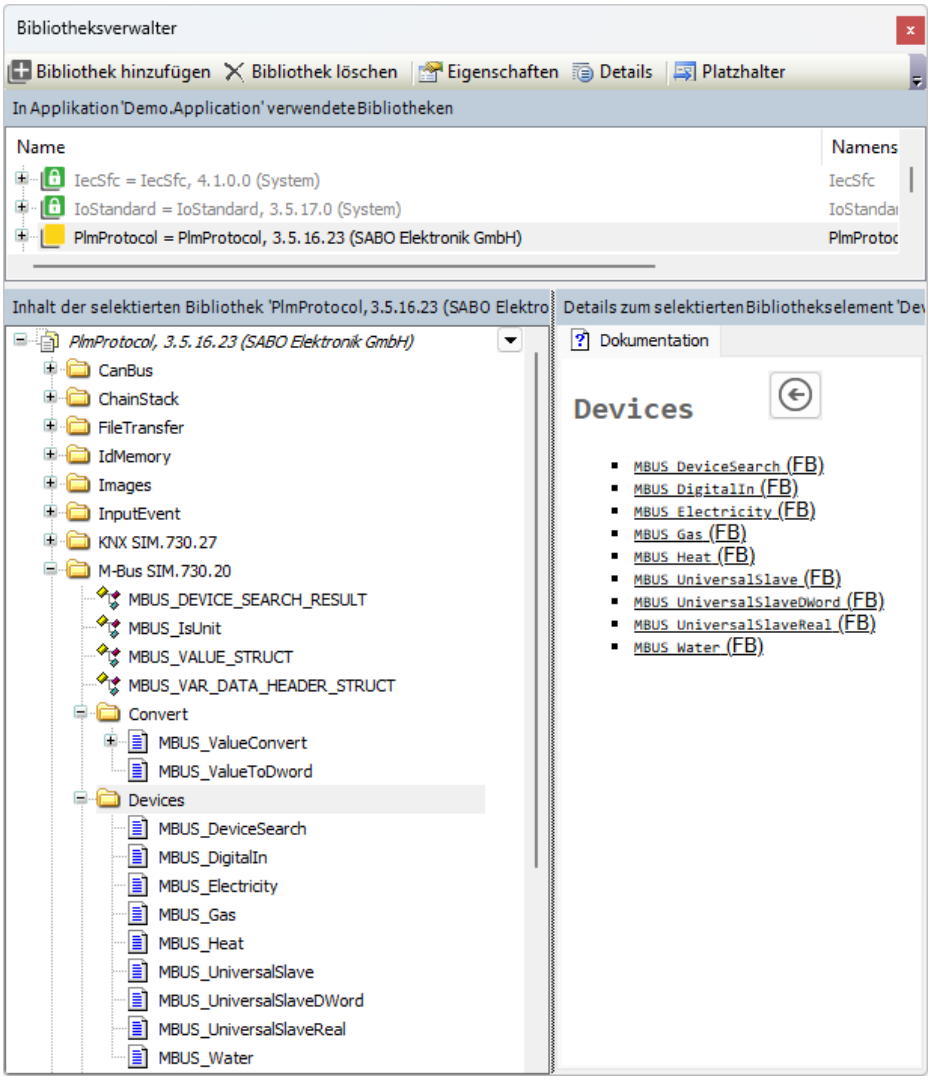


Abbildung 184: Bausteine des M-Bus SIM.730.20 Ordners

Baustein	Typ	Ausgänge
MBUS_DigitalIn	Digital Input	Zustand der Digitaleingänge im Format MBUS_VALUE_STRUCT
MBUS_Electricity	Elektrizitätszähler	• Energie • Leistung Im Format MBUS_VALUE_STRUCT
MBUS_Gas	Gaszähler	• Volumen • Durchfluss Im Format MBUS_VALUE_STRUCT
MBUS_Heat	Wärmemengenzähler	• Energie • Leistung • Volumen • Durchfluss • Vorlauftemperatur • Rücklauftemperatur • Differenztemperatur Im Format MBUS_VALUE_STRUCT
MBUS_UniversalSlave	Universal	30 Ausgangswerte des Slaves im Format MBUS_VALUE_STRUCT
MBUS_UniversalSlaveDword	Universal	30 Ausgangswerte des Slaves im Format DWORD
MBUS_UniversalSlaveReal	Universal	30 Ausgangswerte des Slaves im Format REAL
MBUS_Water	Wasserzähler	• Volumen • Durchfluss Im Format MBUS_VALUE_STRUCT

Die tatsächlich gelieferten Werte hänge von dem jeweiligen Slave ab.

Für jeden Slave wird ein Funktionsblock aus dem Ordner *Devices* eingefügt. Dieser ruft intern Bausteine aus dem Ordner *Util* auf und wickelt den M-Bus-Zugriff für seinen Slave ab.

Die Slave-Funktionsblöcke einigen sich automatisch untereinander, so dass immer nur ein Baustein den M-Bus belegt und die Bus-Zugriffe sequenziell abgearbeitet werden.

17.4 MBUS_VALUE_STRUCT

An den Ausgängen der Slave-Funktionsblöcke stehen nach erfolgreichem M-Bus Zugriff die vom Slave gelieferten Datenwerte als `MBUS_VALUE_STRUCT` zur Verfügung, siehe *Abbildung 185*.

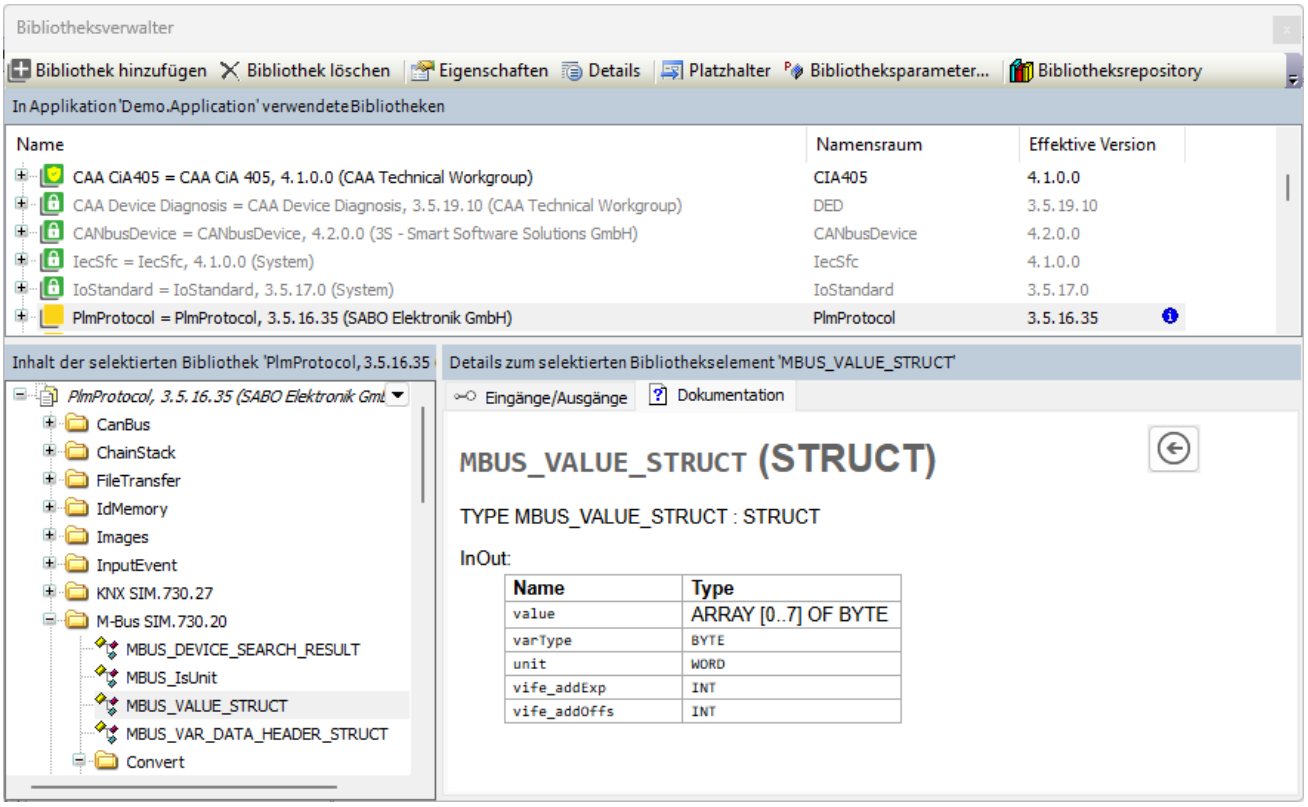


Abbildung 185: Der Datentyp MBUS_VALUE_STRUCT

Ein `MBUS_VALUE_STRUCT` entspricht dem Format, in dem die Daten vom M-Bus-Slave geliefert werden und enthält neben dem Datenwert noch Angaben zur Codierung und zur Einheit des Datenwerts.

Ein `MBUS_VALUE_STRUCT` muss durch Nachschalten eines Bausteins vom Typ `MBUS_ValueConvert()` in eine REAL-Zahl in der gewünschten Einheit, z.B. kWh oder °C, umgewandelt werden, siehe weiter unten. Für spezielle Zwecke steht auch ein Baustein `MBUS_ValueToDword()` zur Verfügung.

17.5 BENÖTIGTE BIBLIOTHEKEN

Benötigt werden die folgenden Bibliotheken:

PlmStandard > V3.5.16.28
PlmProtocol > V3.5.16.35

Die angegebenen Bibliotheken müssen vom Projekt geladen werden, damit die Funktionen zur Verfügung stehen. Klicken Sie dazu auf den ***Bibliotheksverwalter*** > ***Bibliothek hinzufügen*** > ***Sonstige*** > ***PlmStandard*** > ***OK***. Wiederholen Sie den Schritt für die Bibliothek **PlmProtocol**. Die benötigten Bausteine befinden sich in dem Ordner **M-BUS SIM.730.20** in der Bibliothek **PlmProtocol**.

17.5.1 MBUS_HEAT, ..._GAS, ..._ELECTRICITY, ..._WATER, ..._DIGITALIN (PLMPROTOCOL.LIBRARY)

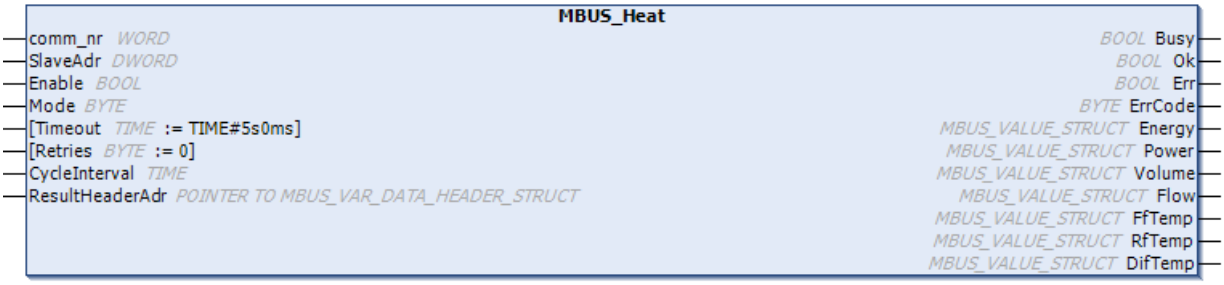


Abbildung 186: Funktionsbaustein MBUS_Heat (PlmProtocol.Library)

Der Funktionsblock `Mbus_Heat` dient der Abfrage eines Wärmemengenzählers und wird stellvertretend für die Funktionsblöcke `Mbus_Electricity`, `Mbus_Gas`, `Mbus_Water` und `Mbus_DigitalIn` erläutert, die sich lediglich durch die anderen Ausgangswerte unterscheiden.

Input Parameter:		
Comm_nr	WORD	Zugewiesene COM-Nummer des M-Bus Masters aus SIM_730_COM
SlaveAdr	DWORD	Primäradresse des Slaves z.B. 2
Enable	BOOL	TRUE = Slave abfragen
Mode	BYTE	0 = Einmalige Abfrage bei Enable FALSE -> TRUE auslösen 1 = Zyklische Slaveabfrage gemäß CycleInterval solange Enable = TRUE +2 = Sekundäre Adressierung +32 = ggf. zusätzliche Antwort-telegramme anfordern, erste Abfrage erfolgt mit FCB = 0 +64 = ggf. zusätzliche Antwort-telegramme anfordern, erste Abfrage erfolgt mit FCB = 1 +128 = vor der Abfrage SND_NKE senden (Reset Slave-Kommunikation)
Timeout	TIME	Timeout für Slave-Abfrage, z.B. T#5s
Retries	BYTE	Anzahl der Wiederholungen nach Timeout, z.B. 2
CycleInterval	TIME	Zyklisches Abfrageintervall bei Mode = 1, z.B. T#1h
ResultHeaderAdr	POINTER TO MBUS_VAR_DATA_HEADER_STRUCT	Adresse einer Variablen vom Typ

		MBUS_VAR_DATA_HEADER_STRUCT
Return-Werte:		
Busy	BOOL	TRUE = M-Bus Übertragung aktiv
Ok	BOOL	TRUE = Message Übertragung und Auswertung erfolgreich
Err	BOOL	TRUE = Fehler bei der Übertragung oder Auswertung
ErrCode	BYTE	Fehlercode: 0 = ok 1 = Slave Timeout 2 = Invalid Frame Boundary 3 = Frame Length Error, Längenangaben im Frame nicht konsistent 4 = Frame Checksum Error 5 = Rx Buffer overflow 6 = Interner Parameterfehler 7 = Frame Inconsistency Error bei Long Frame 11 = Konfigurationsfehler SIM.730.20 12 = Kein RxBuffer 20 = Unbekannter Antworttyp, weder Fixed Data Structure noch Variable Data Structure 22 = Data Struct zu kurz 23 = Unbekanntes Datenformat bei Variable Length Data 24 = Kein Datenframe empfangen 25 = Fixed Data Header fehlt 27 = Var Length Data zu lang 28 = Weitere Datenübertragung erforderlich
Energy Power Volume Flow FfTemp RfTemp DifTemp	MBUS_VALUE_STRUCT	Vom Slave gelieferte Werte als Datentyp Mbus_Value_Struct. Zur Umwandlung in eine Zahl mit einer bestimmten Einheit ist der Baustein MBUS_ValueConvert() nachzuschalten

Als SlaveAdr wird die M-Bus-Primäradresse des Slaves angegeben, diese liegt im Bereich 1...250 und wird am Slave üblicherweise durch Codierschalter oder mittels Konfigurations-Software eingestellt.

Die Adressen 254 und 255 werden von den Slaves als Broadcast interpretiert, der an alle Slaves gerichtet ist. Bei `SlaveAdr` = 254 antworten alle angeschlossenen Slaves; diese Adresse kann deshalb nur verwendet werden, wenn nur ein einziger Slave am M-Bus angeschlossen ist. Bei `SlaveAdr` = 255 antwortet kein Slave, diese Adresse kann daher nicht für die Datenabfrage verwendet werden.

Der Eingang Mode sollte bei den meisten M-Bus-Slaves den Wert 192 bzw. 16#C0 (Einzelabfrage + FCB=1 + SND_NKE) oder 193 bzw. 16#C1 (Zyklische Abfrage + FCB=1 + SND_NKE) haben. Diese Einstellung führt bei den

meisten Slaves dazu, dass die gewünschten Daten in der Antwort erscheinen. Details sind der M-Bus Dokumentation des jeweiligen Slaves zu entnehmen.

Der Baustein versucht, aus der Slave-Antwort die gewünschten Datenwerte zu extrahieren, z.B. bei `MBUS_Heat()` die Werte für Energie, Leistung, Volumen, Durchfluss, Vorlauftemperatur, Rücklauftemperatur und Differenztemperatur. Ob dies möglich ist, hängt vom jeweiligen Slave ab und ist der M-Bus-Dokumentation des jeweiligen Slaves zu entnehmen.

Bei erfolgreicher Übertragung werden die vom Slave empfangenen Datenwerte in die entsprechenden Ausgangsvariablen kopiert. Diese haben den Datentyp `MBUS_VALUE_STRUCT` und müssen mit Bausteinen vom Typ `MBUS_ValueConvert()` in die gewünschte Einheit umgewandelt werden.

Nicht in der Slave-Antwort enthaltene Ausgangswerte werden auf 0 gesetzt; ein nachgeschalteter Baustein vom Typ `MBUS_ValueConvert()` zeigt in diesem Fall einen Fehler an.

Der Eingang `ResultHeaderAdr` kann in den meisten Fällen auf 0 gesetzt werden.

Der Baustein muss zyklisch aufgerufen werden, auch wenn gerade keine Abfrage läuft, damit die internen Abläufe korrekt bearbeitet werden.

17.5.2 MBUS_UNIVERSALSLAVE (PLMPROTOCOL.LIBRARY)

Abbildung 187: Funktionsblock MBUS_UniversalSlave (PlmProtocol.Library)

Dieser Funktionsblock dient zur Abfrage unbekannter Slaves oder Slaves, die keinem der vorhandenen Standardtypen zuzuordnen sind.

Er wird stellvertretend für die Funktionsblöcke MBUS_UniversalSlaveDWord und MBUS_UniversalSlaveReal erläutert

Input Parameter:		
Comm_nr	WORD	Zugewiesene COM-Nummer des M-Bus Masters aus SIM_730_COM
SlaveAdr	DWORD	Primäradresse des Slaves z.B. 2
Enable	BOOL	TRUE = Slave abfragen
Mode	BYTE	0 = Einmalige Abfrage bei Enable FALSE -> TRUE auslösen 1 = Zyklische Slaveabfrage gemäß CycleInterval solange Enable = TRUE +2 = Sekundäre Adressierung +32 = ggf. zusätzliche Antwort-telegramme anfordern, erste Abfrage erfolgt mit FCB = 0 +64 = ggf. zusätzliche Antwort-telegramme anfordern, erste Abfrage erfolgt mit FCB = 1 +128 = vor der Abfrage SND_NKE senden (Reset Slave-Kommunikation)
Timeout	TIME	Timeout für Slave-Abfrage, z.B. T#5s
Retries	BYTE	Anzahl der Wiederholungen nach Timeout, z.B. 2
CycleInterval	TIME	Zyklisches Abfrageintervall bei Mode = 1, z.B. T#1h
ResultHeaderAdr	POINTER TO MBUS_VAR_DATA_HEADER_STRUCT	Adresse einer Variablen vom Typ MBUS_VAR_DATA_HEADER_STRUCT
Return-Werte:		
Busy	BOOL	TRUE = M-Bus Übertragung aktiv
Ok	BOOL	TRUE = Message Übertragung und Auswertung erfolgreich
Err	BOOL	TRUE = Fehler bei der Übertragung oder Auswertung
ErrCode	BYTE	0 = ok 1 = Slave Timeout 2 = Invalid Frame Boundary

		3 = Frame Length Error, Längenangaben im Frame nicht konsistent 4 = Frame Checksum Error 5 = Rx Buffer overflow 6 = Interner Parameterfehler 7 = Frame Inconsistency Error bei Long Frame 11 = Konfigurationsfehler SIM.730.20 20 = Unbekannter Antworttyp, weder Fixed Data Structure noch Variable Data Structure 23 = Unbekanntes Datenformat bei Variable Length Data 24 = Kein Datenframe empfangen 25 = Fixed Data Header fehlt 27 = Var Length Data zu lang 28 = Weitere Datenübertragung erforderlich
MbusVal0 … MbusVal29	MBUS_VALUE_STRUCT	Vom Slave gelieferte Werte als Datentyp MBUS_Value_STRUCT. Zur Umwandlung in eine Zahl mit einer bestimmten Einheit ist der Baustein MBUS_ValueConvert() nachzuschalten

Als SlaveAdr wird die M-Bus-Primäradresse des Slaves angegeben, diese liegt im Bereich 1…250 und wird am Slave üblicherweise durch Codierschalter oder mittels Konfigurations-Software eingestellt.

Die Adressen 254 und 255 werden von den Slaves als Broadcast interpretiert, der an alle Slaves gerichtet ist. Bei SlaveAdr = 254 antworten alle angeschlossenen Slaves; diese Adresse kann deshalb nur verwendet werden, wenn nur ein einziger Slave am M-Bus angeschlossen ist. Bei SlaveAdr = 255 antwortet kein Slave, diese Adresse kann daher nicht für die Datenabfrage verwendet werden.

Der Eingang Mode sollte bei den meisten M-Bus-Slaves den Wert 192 (Einzelabfrage + FCB=1 + SND_NKE) oder 193 (Zyklische Abfrage + FCB=1 + SND_NKE) haben. Diese Einstellung führt bei den meisten Slaves dazu, dass die gewünschten Daten in der Antwort erscheinen. Details sind der M-Bus-Dokumentation des jeweiligen Slaves zu entnehmen.

Der Baustein versucht, aus der Slave-Antwort die ersten 30 Datenwerte zu extrahieren. Ob dies überhaupt möglich ist, hängt vom jeweiligen Slave ab und ist der M-Bus-Dokumentation des jeweiligen Slaves zu entnehmen. Nicht in der Slave Antwort enthaltene Ausgangswerte werden auf 0 gesetzt.

Bei erfolgreicher Übertragung werden die vom Slave empfangenen Datenwerte in die Ausgangsvariablen MbusVal0…MbusVal29 kopiert. Diese haben beim Baustein MBUS_UniversalSlave den Datentyp MBUS_VALUE_STRUCT und müssen mit Bausteinen vom Typ MBUS_ValueConvert() in die gewünschte Einheit umgewandelt werden.

Beim Baustein MBUS_UniversalSlaveDWord() erfolgt bereits intern eine Umwandlung in den Zahlentyp DWORD, ohne Berücksichtigung einer Einheit.

Beim Baustein MBUS_UniversalSlaveReal() erfolgt bereits intern eine Umwandlung in den Zahlentyp REAL, ohne Berücksichtigung einer Einheit.

Der Baustein muss zyklisch aufgerufen werden, auch wenn gerade keine Abfrage läuft, damit die internen Abläufe korrekt bearbeitet werden.

17.5.3 MBUS_DEVICESEARCH (PLMPROTOCOL.LIBRARY)

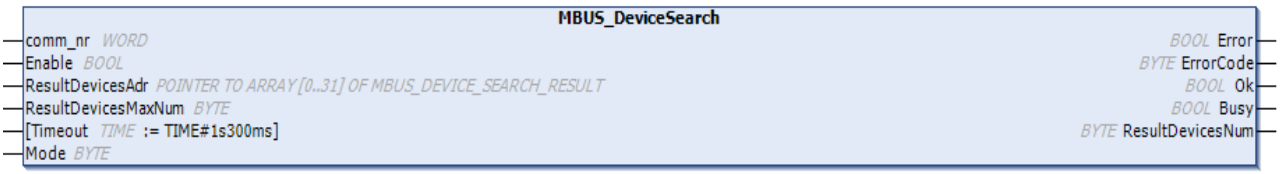


Abbildung 188: Funktionsblock MBUS_DeviceSearch (PlmProtocol.Library)

Ist die Adresse eines M-Bus Gerätes unbekannt, so kann mit dem Funktionsbaustein `MBUS_DeviceSearch` kann nach der Primär- oder Sekundäradresse eines M-Bus Gerätes gesucht werden. Das Ergebnis der Suche wird in dem Array vom Typ `MBUS_DEVICE_SEARCH_RESULT` des Eingangs `ResultDevicesAdr` hinterlegt.

Input Parameter:		
Comm_nr	WORD	Zugewiesene COM-Nummer des M-Bus Masters aus SIM_730_COM
Enable	BOOL	TRUE = Baustein abarbeiten
ResultDevicesAdr	POINTER TO ARRAY OF MBUS_DEVICE_SEARCH_RESULT	Adresse eines Arrays vom Typ MBUS_DEVICE_SEARCH_RESULT, in das das Ergebnis der Suche abgelegt wird
ResultDevidesMaxNum	BYTE	Maximale Anzahl der M-Bus Geräte, nach denen gesucht werden kann
Timeout	TIME	Timeoutzeit, z.B. T#300ms
Mode	BYTE	+1 = Primäradresse suchen +2 = Sekundäradresse suchen
Return-Werte:		
Error	BOOL	True = Fehler aufgetreten
ErrorCode	BYTE	Fehlercode
Ok	BOOL	TRUE = Baustein wurde erfolgreich abgearbeitet
Busy	BOOL	TRUE = Baustein wird momentan abgearbeitet
ResultDevicesNum	BYTE	Anzahl der gefunden M-Bus Geräte

17.5.4 MBUS_VALUECONVERT (PLMPROTOCOL.LIBRARY)

Abbildung 189: Funktionblock MBUS_ValueConvert (PlmProtocl.Library)

Input Parameter:		
MbusValue	MBUS_VALUE_STRUCT	Ausgabewert eines M-Bus Funktionsblocks
WantedUnit	INT	Gewünschte physikalische Einheit, in der der Wert ausgegeben werden soll (siehe unten)
Return-Werte:		
Value	REAL	Zahlenwert im Format REAL in der durch WantedUnit vorgegebenen physikalischen Größe
Err	BOOL	TRUE = Umwandlung konnte nicht durchgeführt werden. Die globale Variable MBUS_GVL enthält den Fehlercode. 0 = kein Fehler 50 = angeforderter Wert ist nicht in Antwort vom Mbus Device enthalten 51 = Fehlerhaft BCD-codierter Wert 52 = Umwandlung in gewünschte Einheit nicht möglich 53 = unbekanntes Codierungsformat (MBUS_VALUE_STRUCT.varType)

Der Baustein `MBUS_ValueConvert()` wandelt Datenwerte aus dem Format `MBUS_VALUE_STRUCT` in einen Zahlenwert in der gewünschten Einheit.

Der Baustein erlaubt nur sinnvolle Umwandlungen, z.B. kann ein Datenwert, der in der Einheit "Liter/Minute" geliefert wird, in "Kubikmeter/Stunde" umgewandelt werden aber nicht in "Kilowatt".

Im Fehlerfall wird der Ausgang `Value` auf den Wert 0 und der Ausgang `Err` auf `TRUE` gesetzt. Zusätzlich enthält die globale Bibliotheksvariable `MBUS_GVL` einen Fehlercode.

Am Eingang `WantedUnit` wird ein fester Wert vorgegeben, der die gewünschte Ausgabeeinheit des Parameters `Value` definiert. Statt eines numerischen Wertes kann auch eine passende globale Konstante aus der globalen Variablenliste `MBUS_GVL` der Bibliothek **PlmProtocol** im Ordner **M-Bus SIM.730.20** verwendet werden. Dies verbessert die Lesbarkeit und Nachvollziehbarkeit der Anwendung. Die verfügbaren Konstanten sind in der Variablenliste `MBUS_GVL` aufgeführt (`CONVERT_Mul1` ... `CONVERT_kA`).

Folgende Ausgabeeinheiten `WantedUnit` sind möglich:

Konstanter Faktor:
0 → ×1, 1 → ×10, 2 → ×100, 3 → ×1000, 4 → ×1e4, 5 → ×1e5, 6 → ×1e6, 7 → ×1e7, 8 → ×1e8, 9 → ×1e9, -1 → ÷10, -2 → ÷100, -3 → ÷1000, -4 → ÷1e4, -5 → ÷1e5, -6 → ÷1e6, -7 → ÷1e7, -8 → ÷1e8, -9 → ÷1e9
Energie:
20 → mJ, 21 → J, 22 → kJ, 23 → MJ, 24 → GJ, 25 → mW×s, 26 → mW×min, 27 → mW×hour,

28 → W×s, 29 → W×min, 30 → W×hour, 31 → kW×s, 32 → kW×min, 33 → kW×hour, 34 → MW×s, 35 → MW×min, 36 → MW×hour, 37 → GW×s, 38 → GW×min, 39 → GW×hour
Leistung:
50 → mW, 51 → W, 52 → kW, 53 → MW, 54 → GW, 55 → mJ/s, 56 → mJ/min, 57 → mJ/hour, 58 → J/s, 59 → J/min, 60 → J/hour, 61 → kJ/s, 62 → kJ/min, 63 → kJ/hour, 64 → MJ/s, 65 → MJ/min, 66 → MJ/hour, 67 → GJ/s, 68 → GJ/min, 69 → GJ/hour
Volumen:
80 → ml, 81 → l, 82 → m^3
Volumenfluss:
90 → ml/s, 91 → ml/min, 92 → ml/hour, 93 → l/s, 94 → l/min, 95 → l/hour, 96 → m^3/s, 97 → m^3/min, 98 → m^3/hour
Masse:
110 → mg, 111 → g, 112 → kg, 113 → ton
Massenfluss:
120 → mg/s, 121 → mg/min, 122 → mg/hour, 123 → g/s, 124 → g/min, 125 → g/hour, 126 → kg/s, 127 → kg/min, 128 → kg/hour, 129 → ton/s, 130 → ton/min, 131 → ton/hour
Temperatur:
140 → °C, 141 → °F
Temperaturdifferenz:
145 → mK, 146 → K
Druck:
150 → mbar, 151 → bar, 152 → kbar, 153 → mPa, 154 → Pa, 155 → hPa, 156 → kPa, 157 → psi, 158 → atm
Zeitintervall:
170 → s, 171 → min, 172 → hours, 173 → days

17.5.5 MBUS_VALUETODWORD (PLMPROTOCOL.LIBRARY)

Abbildung 190: Baustein MBUS_ValueToDword (PlmProtocol.Library)

Input Parameter:		
MbusValue	MBUS_VALUE_STRUCT	Ausgebwert eines Mbus Funktionsblocks
Return-Werte:		
ValueDword	DWORD	Zahlenwert im Format DWORD ohne weitere Umwandlung

Dieser Baustein wandelt den am Eingang angelegten `MbusValue` direkt in einen Wert vom Typ `DWORD`, ohne Berücksichtigung einer Einheit.

Der Baustein kann zu Debugging-Zwecken verwendet werden, oder um große Integer-Werte, wie z.B. Seriennummern, zu konvertieren.

Falls die Umwandlung nicht möglich ist, hat `ValueDWord` den Wert 0; eine weitergehende Fehlerdiagnose ist mit diesem Baustein nicht möglich.

17.6 PROGRAMMBEISPIEL (FUP)

Das folgende Beispiel demonstriert die Verwendung des M-Bus-Masters in einem IEC-Programm. Das Beispiel wurde in FUP ausgeführt, kann aber ebenso in CFC oder ST implementiert werden.

Die Beispiel-Bausteine `InitCom()` und `TestHeat()` müssen zyklisch aus `PLC_PRG()` heraus aufgerufen werden.

Zunächst wird das M-Bus-Mastermodul SIM.730.20 in die Steuerungskonfiguration eingetragen.

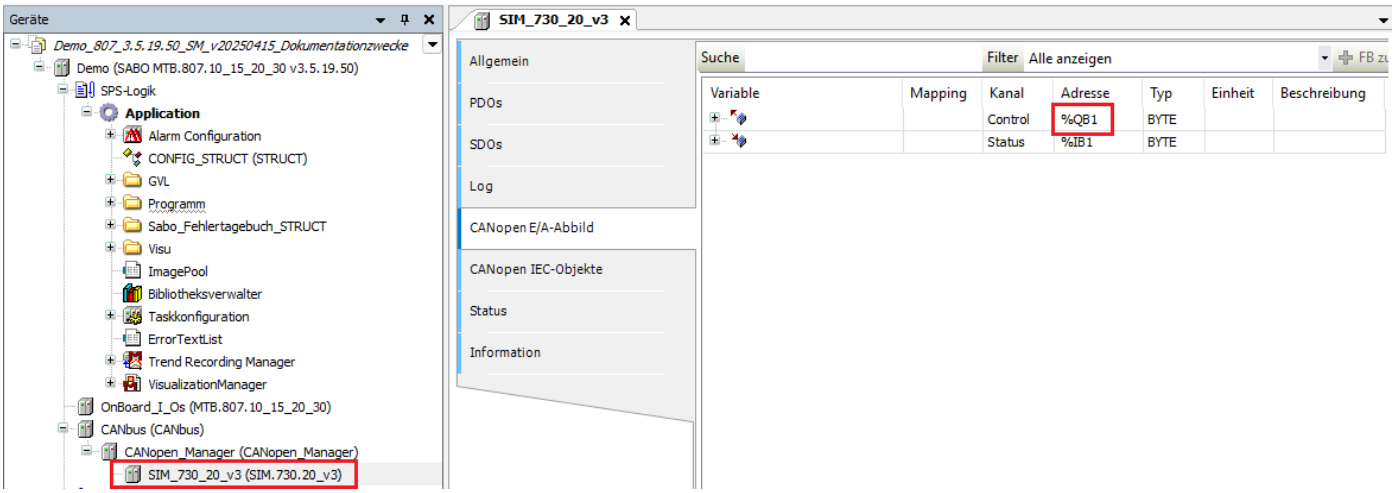


Abbildung 191: Steuerungskonfiguration mit M-Bus Master SIM.730.20

Der Baustein `InitCom()` initialisiert die Verbindung zum SIM.730.20, siehe *Abbildung 192*.

`SIM_730_COM()` erhält am Eingang `NodeID` die CAN-ID des M-Bus-Mastermoduls, die in der Steuerungskonfiguration unter dem Reiter *Allgemein* eingestellt ist (hier: 2). Am Eingang `FirstQbAddress` wird die erste QB-Adresse des Moduls aus der Steuerungskonfiguration angelegt (hier: `%QB1`., vgl. *Abbildung 191*). Der Eingang `CanDevNo` erhält die Nummer des CAN-Masters (CANBus), in diesem Fall die `0`.

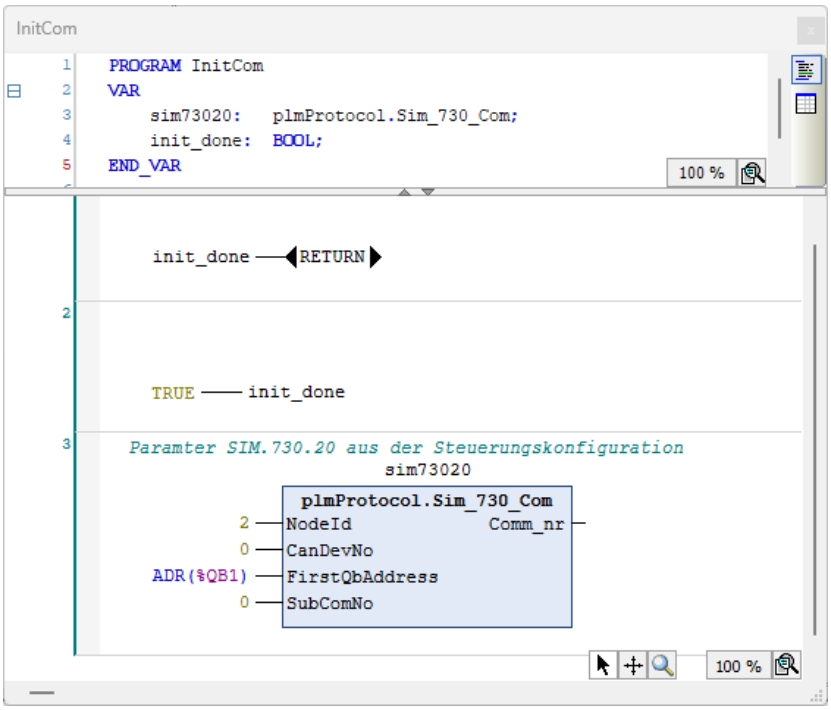
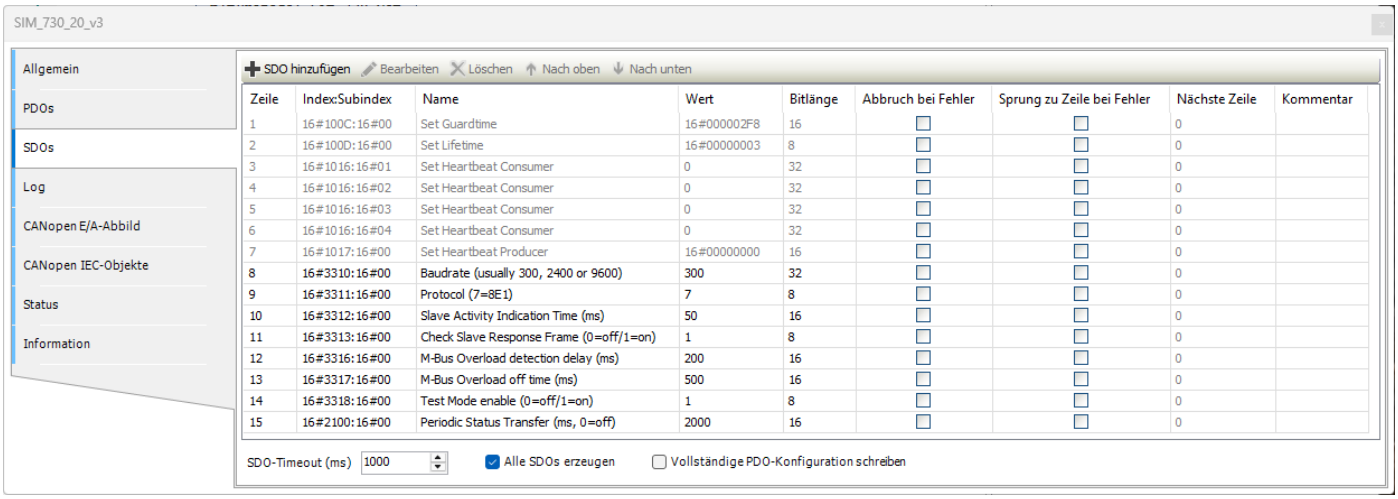


Abbildung 192: Einmalige Initialisierung der Kommunikation mit dem M-Bus Mastermodul SIM.730.20

Die Baudrate auf dem M-Bus wird unter *Service Data Objects* eingestellt. Die üblichen Baudraten sind 300, 2400 und 9600 Baud. Viele M-Bus-Slaves erkennen die Baudrate automatisch. Im Zweifelsfall ist die M-Bus-Dokumentation des entsprechenden Slaves heranzuziehen.



Zeile	Index:Subindex	Name	Wert	Bitlänge	Abbruch bei Fehler	Sprung zu Zeile bei Fehler	Nächste Zeile	Kommentar
1	16#100C:16#00	Set Guardtime	16#000002F8	16	☐	☐	0	
2	16#100D:16#00	Set Lifetime	16#00000003	8	☐	☐	0	
3	16#1016:16#01	Set Heartbeat Consumer	0	32	☐	☐	0	
4	16#1016:16#02	Set Heartbeat Consumer	0	32	☐	☐	0	
5	16#1016:16#03	Set Heartbeat Consumer	0	32	☐	☐	0	
6	16#1016:16#04	Set Heartbeat Consumer	0	32	☐	☐	0	
7	16#1017:16#00	Set Heartbeat Producer	16#00000000	16	☐	☐	0	
8	16#3310:16#00	Baudrate (usually 300, 2400 or 9600)	300	32	☐	☐	0	
9	16#3311:16#00	Protocol (7=8E1)	7	8	☐	☐	0	
10	16#3312:16#00	Slave Activity Indication Time (ms)	50	16	☐	☐	0	
11	16#3313:16#00	Check Slave Response Frame (0=off/1=on)	1	8	☐	☐	0	
12	16#3316:16#00	M-Bus Overload detection delay (ms)	200	16	☐	☐	0	
13	16#3317:16#00	M-Bus Overload off time (ms)	500	16	☐	☐	0	
14	16#3318:16#00	Test Mode enable (0=off/1=on)	1	8	☐	☐	0	
15	16#2100:16#00	Periodic Status Transfer (ms, 0=off)	2000	16	☐	☐	0	

Abbildung 193: Service Data Objects (SDO) vom SIM.730.20 Modul

TestHeat

```
PROGRAM TestHeat
VAR
    heat                            :PlmProtocol.MBUS_Heat;
    err                             :ARRAY[0..6] OF BOOL;
    transfBusy, transfOk, transfErr :BOOL;
    transfErrCode                   :BYTE;
    energy, power, volume, flow     :REAL;
    fftemp, rftemp, diftemp         :REAL;
END_VAR
```

100 %

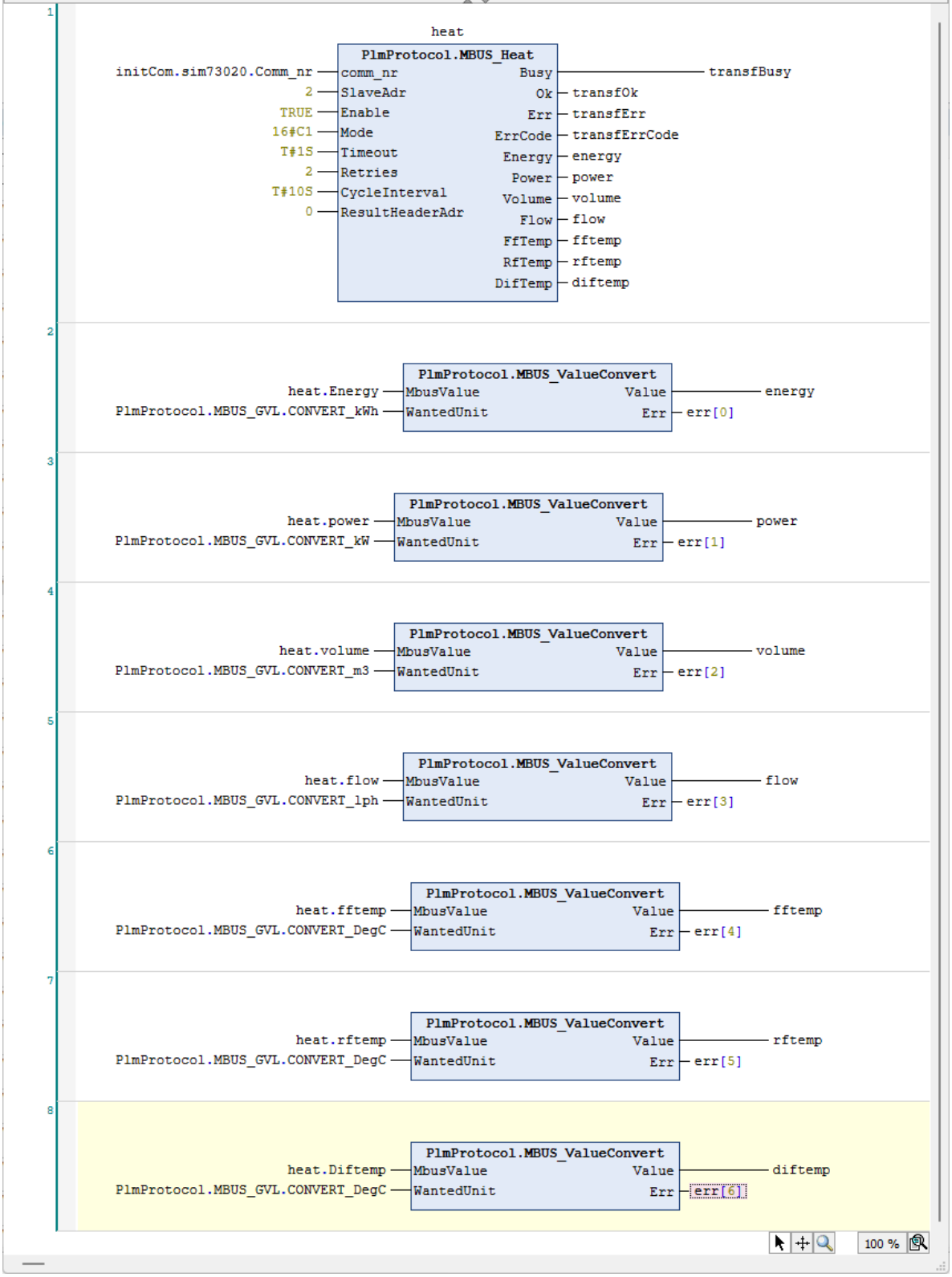


Abbildung 194: Beispiel für Abfrage eines M-Bus Wärmemengenzähler (FUP)

17.7 FEHLERSUCHE

Im Fehlerfall, d.h. es erscheinen keine Werte an den Ausgängen der M-Bus Device Bausteine, ist zunächst der Ausgang `ErrCode` zu betrachten, der häufig Aufschluss über die Art des Fehlers gibt. Die Fehlercodes sind in der Bibliothek vermerkt.

- Falls es sich um Timeout-Fehler handelt, ist zu prüfen, ob überhaupt eine Kommunikation mit dem betreffenden Device zustande kommt.

- Die Antwort eines M-Bus Device ist beim SIM.730.20 am kurzzeitigen Blinken der Status LED *Device-Aktivität* erkennbar.

- Eine testweise Kommunikation kann direkt am SIM.730.20 ausgelöst werden, indem der Fronttaster am Modul vier Sekunden lang gedrückt wird. Dadurch wird eine Datenabfrage an alle M-Bus-Devices ausgelöst (an Broadcast-Adresse 254). Die Status-LED *Device-Aktivität* des Moduls muss blinken als Zeichen einer M-Bus Device-Antwort. Falls die LED nicht blinkt: M-Bus falsch angeschlossen, falsche Baudrate, M-Bus Device falsch konfiguriert (z.B. antwortet nicht auf Primäradressierung), M-Bus Device zu häufig abgefragt (s.u.).

- Prüfen Sie, ob die Baudrate korrekt eingestellt ist. Falls ein M-Bus Device keine automatische Baudratenerkennung besitzt, muss diese im M-Bus Device passend konfiguriert oder beim M-Bus-Master (in der Steuerungskonfiguration) passend eingestellt werden.

- Bei einigen M-Bus Devices ist nur eine begrenzte Anzahl von Abfragen pro Zeitintervall möglich (z.B. max. 10 Abfragen pro Tag), vornehmlich bei M-Bus Devices mit Batteriespeisung. Dies kann dazu führen, dass die Abfrage zunächst funktioniert, später aber scheinbar nicht mehr. Abhilfe: Abfrageintervall ausreichend groß wählen, siehe M-Bus Device-Dokumentation.

- Falls die Datenübertragung mit dem Device-Baustein funktioniert (`Ok = TRUE`), aber die Werteumwandlung mit dem Baustein `MBUS_ValueConvert()` einen Fehler meldet (`Err = TRUE`), kann der Wert kann nicht in der gewünschten Weise aus den übertragenen Daten extrahiert werden. Hinweise auf die Ursache liefert in diesem Fall die globale Bibliotheksvariable `MBUS_GVL`; diese muss direkt nach dem Aufruf des betreffenden Bausteins `MBUS_ValueConvert()` ausgewertet werden. Die möglichen Fehlercodes sind als globale Konstanten in der Bibliothek definiert.

Debug-Logfile einschalten: globale Bibliotheksvariable `PlmProtocol.MBUS_GVL.LogEnable` in `PLC_PRG` auf `TRUE` setzen, dies erzeugt ein Logfile `a/mbus_debug.txt`. Diese Datei kann mit einem FTP-Client von der Steuerung geladen und mit einem Texteditor untersucht werden. Senden Sie uns die Datei ggf. nach Rücksprache zu Diagnosezwecken zu. Unbedingt `PlmProtocol.MBUS_GVL.LogEnable` im späteren Betrieb wieder deaktivieren, da das Logfile sonst unbegrenzt wächst.

18 FTP-CLIENT

18.1 ALLGEMEINES

Das FTP-Protokoll wird verwendet, um beliebige Dateien über eine TCP-Netzwerkverbindung zu übertragen. FTP basiert auf einem Client-Server-Modell. Dabei meldet sich der Client zunächst auf

dem Server an und führt dann eine Folge von Befehlen aus, um Dateien zum Server oder von diesem zu übertragen.

Das ursprüngliche FTP-Protokoll stammt aus dem Jahre 1971. Die Verbindung zwischen Client und Server ist weder verschlüsselt noch authentifiziert. Anmeldename und Passwort werden im Klartext übertragen, daher genügt das Protokoll für viele Zwecke nicht den heutigen Sicherheitsanforderungen.

Die meisten Server erlauben einen anonymen Zugang unter dem Anmeldenamen "anonymous" mit beliebigem Passwort. Im Internet existieren zahlreiche FTP-Server, die unter diesem anonymen Zugang den Download von Software und Daten erlauben, daher ist das FTP-Protokoll immer noch populär und wird von allen gängigen Browsern unterstützt (URLs beginnen mit `ftp://`...).

Eine Besonderheit des FTP-Protokolls ist die Verwendung von getrennten Verbindungen für die Befehls- und Datenübertragung. Vorgesehen ist, dass zunächst die Befehlsverbindung vom Client zum Server aufgebaut wird. Wenn dann durch einen entsprechenden FTP-Befehl eine Datenübertragung angefordert wird, baut entweder der Server eine zusätzliche Verbindung zum Client auf ("Active Mode"), oder der Client eine zusätzliche Verbindung zum Server ("Passive Mode"). Da die Rückwärtsverbindung im Active Mode häufig durch Firewall-Einstellungen blockiert wird, ist der Passive Mode vorzuziehen und wird vom hier dargestellten Client unterstützt.

Der FTP-Client für PLM-Steuerungen unterstützt die wichtigsten FTP-Befehle und erlaubt damit eine Datenübertragung zwischen den internen Laufwerken der Steuerung (Local) und einem externen FTP-Server (Remote).

Der FTP-Client wird durch zyklischen Aufruf des eigentlichen Client-Bausteins `FtpClient` und der daran angehängten Befehlsbausteine realisiert. Der Client Baustein stellt die Netzwerk-Verbindung zum FTP-Server her und übernimmt die Anmeldung (Login). Anschließend werden die Befehlsbausteine in der Programmreihenfolge ausgeführt.

18.2 VORGEHENSWEISE

Im einfachsten Fall werden mind. zwei Bausteine angelegt: Der Client-Baustein vom Typ `FtpClient` und ein Befehlsbaustein z.B. vom Typ `FtpGet`. Der Befehlsbaustein erhält an seinem Eingang `FtpClientId` den Wert des gleichnamigen Ausgangs `FtpClientId` des Client-Bausteins und wird diesem dadurch zugeordnet.

Durch Aktivieren des Eingangs `Enable` am Client-Baustein wird die Verbindung aufgebaut und anschließend die angehängten Befehlsbausteine ausgeführt, der Client-Ausgang `Busy` ist solange `TRUE`. Nach Beendigung aller Befehle geht `Busy` auf `FALSE`. Wenn alle Befehle fehlerfrei ausgeführt werden konnten, hat der Client Ausgang `Err` den Wert `FALSE`, andernfalls den Wert `TRUE` und in `ErrCode` steht ein Fehlercode, der Rückschlüsse auf den Fehler zulässt.

Alle Befehlsbausteine verfügen ebenfalls über einen `Enable`-Eingang. Wenn dieser konstant auf `TRUE` gesetzt ist, werden nach dem Herstellen der Server-Verbindung automatisch alle Bausteine der Reihe nach abgearbeitet. Alternativ können die `Enable`-Eingänge der Befehlsbausteine durch das Programm gesteuert werden. In diesem Fall wird ein Befehlsbaustein einmalig ausgeführt, sobald er an der Reihe ist und sein `Enable`-Eingang von `FALSE` auf `TRUE` gewechselt hat.

Im Detail verhalten sich verschiedene FTP-Server oft etwas unterschiedlich, daher kann das Verhalten des Clients durch Bausteine vom Typ `FtpConfig` modifiziert werden. Außerdem stehen verschiedene Möglichkeiten zur Fehlersuche zur Verfügung.

18.3 SPEZIFIKATION

- FTP Client gemäß RFC 959
- Unterstützte Befehle: Change Remote Directory, Create Remote Directory, Delete Remote Directory, List Remote Directory, Put File (Local → Remote), Get File (Remote → Local), Delete Remote File, Rename Remote File, Quit
- Datentransfer im "Passive Mode"
- Beliebige Anzahl von gleichzeitig aktiven FTP-Clients
- Standard-Port für Kontrollverbindung: 21 (konfigurierbar)
- Wahlweise vollautomatische Abarbeitung einer FTP-Befehlssequenz oder Ausführen von FTP-Einzelbefehlen

18.4 BENÖTIGTE BIBLIOTHEKEN

Benötigt werden die folgenden Bibliotheken:

PlmStandard > V3.5.16.28
PlmProtocol > V3.5.16.35

Die angegebenen Bibliotheken müssen vom Projekt geladen werden, damit die Funktionen zur Verfügung stehen. Klicken Sie dazu auf den ***Bibliotheksverwalter*** > ***Bibliothek hinzufügen*** > ***Sonstige*** > ***PlmStandard*** > ***OK***. Wiederholen Sie den Schritt für die Bibliothek **PlmProtocol**. Die benötigten Bausteine befinden sich in dem Ordner **FileTransfer -> FTP->FtpClient** und **FtpClientCommands** in der Bibliothek **PlmProtocol**.

18.4.1 FTPCLIENT

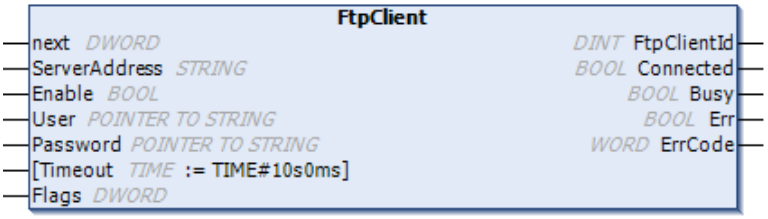


Abbildung 195: Funktionsblock FtpClient (PlmProtocol.Library)

Input Parameter:		
Next	DWORD	Interne Variable
ServerAddress	STRING	IP-Adresse des FTP-Servers, zu dem eine Verbindung hergestellt werden soll, z.B. '10.1.1.156'; nach einem Doppelpunkt kann

		eine Portnummer angegeben werden, z.B. '10.1.1.156:21'
Enable	BOOL	Enable = TRUE, es wird eine Verbindung zum Server aufgebaut, bei FALSE wird diese wieder beendet
User	POINTER TO STRING	Adresse eines Strings mit dem Anmeldenamen. Wenn keine StringAdresse angegeben wird oder String leer ist, wird als User "anonymous" verwendet
Password	POINTER TO STRING	Adresse eines Strings mit dem Anmeldekennwort. Wenn keine String-Adresse angegeben wird, wird ein leeres Password verwendet. Allerdings akzeptieren einige FTP-Server beim anonymen Login zwar ein beliebiges, jedoch kein leeres Password
Timeout	TIME	Timeout-Zeit für alle Server-Reaktionen, z.B. t#10s
Flags	DWORD	Momentan nur 16#8000 zum Einschalten des Debug-Log Modus
Return-Werte:		
FtpClientId	DINT	Referenz (Handle) für das Anhängen der Befehlsbausteine
Connected	BOOL	TRUE = Verbindung zum Ftp-Server hergestellt
Busy	BOOL	TRUE = Verbindung wird hergestellt, Anmeldung läuft oder es wird ein Befehlsbaustein ausgeführt
Err	BOOL	TRUE = Fehler beim Herstellen der Verbindung zum Server, beim Anmelden (Login) oder bei der Abarbeitung eines Befehlsbausteins, ErrCode enthält Fehlernummer
ErrCode	WORD	0 = kein Fehler 1 = TIMEOUT: Der Server hat nicht innerhalb der eingestellten Timeout-Zeit geantwortet 2 = SERVDISCONNECT: Die Verbindung wurde durch den Sever getrennt 3 = NOFTPCLIENTVISUTASK: Der Baustein FtpClientVisuTask wird nicht vom Programm aufgerufen 4 = SOCKCREATE: Es konnte kein Socket für die Verbindung zum Server angelegt werden 5 = NOCMDBLOCKS: An den Client sind keine Befehlsblöcke angehängt 6 = CONFIGPAR: Es wurde ein Config-Block mit einem ungültigen Wert am Eingang Par ausgeführt 7 = CONFIGVAL: Es wurde ein Config-Block mit einem ungültigen Wert am Eingang Val ausgeführt 8 = UNKNOWNCMD: interner Fehler 9 = CMDFAILED: Serverantwort war negativ, letzter FTP-Befehl fehlgeschlagen 10 = CONNECT1: 11 = CONNECT2: 12 = CONNECT3: 13 = CONNECT4:

		14 = CONNECTTIMEOUT: Es konnte keine Verbindung zum FTP-Server unter ServerAddress hergestellt werden. 15 = READ: Lesen vom Server fehlgeschlagen 16 = WRITE: Schreiben zum Server fehlgeschlagen 17 = WELCOME: negative Serverantwort in Welcome-Message 18 = LOGINFAILED: Anmeldung mit User und Password fehlgeschlagen 19 = PASVONLY: Datenübertragung nur im PASV-Modus möglich 20 = DATACONFAILED: Datenübertragung wurde unterbrochen 21 = PASVFAILED: Server lehnt Umschaltung der Datenverbindung in den Passive-Mode ab 22 = TYPEFAILED: Server lehnt Umschaltung der Datenverbindung mit TYPE ab 23 = LOCALFILEREADERROR: lokale Datei auf PLM-Steuerung kann nicht gelesen werden 24 = LOCALFILEWRITEERROR: lokale Datei auf PLM-Steuerung kann nicht geschrieben werden 25 = NOLOCALFILENAME: lokale Datei nicht angegeben 26 = NOREMOTEFILENAME: Remote-Datei nicht angegeben 27 = NOREMOTEDIR: Remote-Directory nicht angegeben 28 = NOFILENAMEFROM: FilenameFrom Parameter nicht angegeben 29 = NOFILENAMETO: FilenameTo Parameter nicht angegeben 30 = ILLEGALSTATE: interner Fehler

Der Baustein `FTPCLIENT` muss zyklisch aufgerufen werden.

Die angehängten Befehlsbausteine müssen mindestens einmal (z.B. in `InitOnce`) aufgerufen werden, können jedoch auch zyklisch aufgerufen werden, ohne nennenswert Ressourcen zu verschwenden.

Die Befehlsbausteine werden in genau der Reihenfolge ausgeführt, in der sie beim ersten Aufruf ausgeführt wurden, d.h. in der Programmreihenfolge.

Das Aufbauen der Verbindung zum FTP-Server geschieht, indem der Eingang `Enable` auf `TRUE` gesetzt wird. Sind der anschließende Verbindungsaufbau und das Anmelden erfolgreich, werden die an diesen Client-Baustein angehängten Befehlsblöcke ausgeführt. Anschließend muss die Verbindung wieder beendet werden, indem der Eingang `Enable` auf `FALSE` gesetzt wird.

Sobald ein Fehler auftritt, werden die folgenden Befehlsbausteine nicht mehr ausgeführt und die Verbindung automatisch beendet. Dieses Verhalten kann durch den Config-Parameter *StopOnError* modifiziert werden. Das ist insbesondere sinnvoll, wenn die `Enable`-Eingänge einzelner Befehlsbausteine nicht konstant auf `TRUE` gesetzt sind, sondern vom Programm gesteuert werden.

Der Ausgang `Busy` ist `TRUE`, wenn die Verbindung zum FTP-Server aufgebaut wird und solange Befehlsbausteine abgearbeitet werden. Wenn alle Befehlsbausteine abgearbeitet sind, geht `Busy` auf `FALSE`; das Anwenderprogramm sollte dann `Enable` auf `FALSE` setzen und damit die Verbindung beenden.

Der Ausgang `ErrCode` zeigt den zuletzt aufgetretenen Fehlercode des Clients oder eines Befehlsbausteins an.

Das Verhalten des Ausgangs `Err` hängt vom Config-Parameter *StopOnError* ab. Bei *StopOnError* = TRUE (Voreinstellung) zeigt `Err` an, dass entweder im Client oder in einem der Befehlsbausteine ein Fehler aufgetreten ist, der zum vorzeitigen Beenden der FTP-Verbindung geführt hat. Bei *StopOnError* = FALSE zeigt `Err` nur Fehler des Clients an. Fehler in einem der Befehlsbausteine können dann nur am `Err`-Ausgang des entsprechenden Befehlsbausteins erkannt werden.

Das FTP-Protokoll sieht vor, dass sich der Client vor dem Beenden der Verbindung mit dem Befehl QUIT beim Server abmeldet. Der QUIT-Befehl wird daher automatisch gesendet, sobald `Enable` auf `FALSE` gesetzt wird. Dieses Verhalten kann durch den Config-Parameter *AutoQuit* modifiziert werden.

Manche FTP-Server beenden die Verbindung nach Ablauf einer bestimmten Zeit (z.B. 15 Minuten) automatisch. Dies kann vom Client erst bemerkt werden, wenn die Ausführung eines weiteren Befehls fehlschlägt. Die Verbindung wird in diesem Fall auch vom Client beendet, der Fehlercode am Ausgang `ErrCode` ist dann 2 (SERVDISCONNECT). Es empfiehlt sich generell, die Verbindung zum Server sofort nach Abarbeitung der gewünschten Befehle wieder zu trennen.

Am Eingang `ServerAddress` wird die IP-Adresse des FTP-Servers angelegt, z.B. '10.1.1.156'. Der Client versucht dann, auf Port 21 die Verbindung zum FTP-Server aufzubauen. Falls der FTP-Server einen anderen Port verwendet, kann dieser nach einem Doppelpunkt an die IP-Adresse angehängt werden, z.B. für Port 2223 '10.1.1.156:2223'.

Die Fehlermeldungen sind als Konstanten in der globalen Variablenliste `PlmProtocol.GlbFtpClient` von `PLM_FTPERR_TIMEOUT` bis `PLM_FTPERR_NOLIC` definiert.

Alle Eingänge des Clients und der Befehlsbausteine, die einen String benötigen, erwarten die Angabe einer String-Adresse `ADR(str)`. Dadurch wird die Ausführungsgeschwindigkeit der Bibliotheksfunktionen erhöht.

18.4.2 FTPCHDIR (PLMPROTOCOL.LIBRARY)

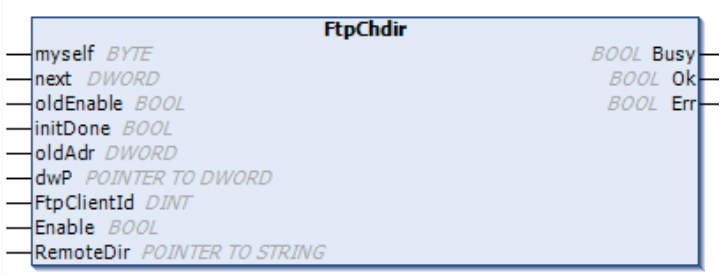


Abbildung 196: Funktionsblock FtpChdir (PlmProtocol.Library)

Input Parameter:		
FtpClientId	DINT	Referenz (Handle) des zugehörigen Client-Bausteins
Enable	BOOL	Enable-Flag, bei steigender Flanke FALSE → TRUE wird der Baustein einmalig ausgeführt; oder fest mit TRUE verbinden
RemoteDir	POINTER TO STRING	Adresse eines Strings, der den Namen des Remote-Verzeichnisses enthält, in das gewechselt werden soll
Return-Werte:		
Busy	BOOL	TRUE = Baustein wird abgearbeitet
Ok	BOOL	TRUE = Abarbeitung war erfolgreich

Err	BOOL	TRUE = Fehler bei Abarbeitung, Ausgang ErrCode am Client Baustein enthält Fehlernummer

Der Baustein FtpChdir veranlasst den FTP-Server, das aktuelle Remote Arbeitsverzeichnis zu wechseln. Auf dieses Verzeichnis beziehen sich alle folgenden Dateioperationen auf dem Server, sofern bei diesen kein absoluter Pfad angegeben ist.

Der angegebene Pfad RemoteDir muss als ADR(str) angegeben werden.

Als Pfadtrennzeichen ist in jedem Fall der Slash (/) zu verwenden, nicht der unter Windows übliche Backslash (\).

18.4.3 FTPMKDIR (PLMPROTOCOL.LIBRARY)

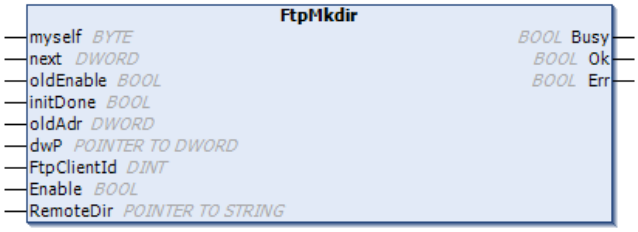


Abbildung 197: Funktionsblock FtpMkdir (PlmProtocol.Library)

Input Parameter:		
FtpClientId	DINT	Referenz (Handle) des zugehörigen Client-Bausteins
Enable	BOOL	Enable-Flag, bei steigender Flanke FALSE → TRUE wird der Baustein einmalig ausgeführt; oder fest mit TRUE verbinden
RemoteDir	POINTER TO STRING	Adresse eines Strings, der den Namen des Remote-Verzeichnisses enthält, das erzeugt werden soll
Return-Werte:		
Busy	BOOL	TRUE = Baustein wird abgearbeitet
Ok	BOOL	TRUE = Abarbeitung war erfolgreich
Err	BOOL	TRUE = Fehler bei Abarbeitung, Ausgang ErrCode am Client Baustein enthält Fehlernummer

Der Baustein FtpMkdir veranlasst den FTP-Server, das angegebene Remote Verzeichnis zu erstellen. Der angegebene Pfad RemoteDir muss als ADR(str) angegeben werden.

Als Pfadtrennzeichen ist in jedem Fall der Slash (/) zu verwenden, nicht der unter Windows übliche Backslash (\).

18.4.4 FTPRMDIR (PLMPROTOCOL.LIBRARY)

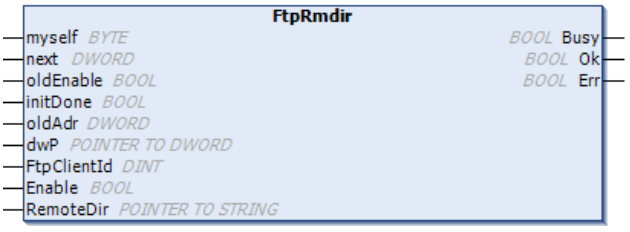


Abbildung 198: Funktionsblock FtpRmdir (PlmProtocol.Library)

Input Parameter:		
FtpClientId	DINT	Referenz (Handle) des zugehörigen Client-Bausteins
Enable	BOOL	Enable-Flag, bei steigender Flanke FALSE → TRUE wird der Baustein einmalig ausgeführt; oder fest mit TRUE verbinden
RemoteDir	POINTER TO STRING	Adresse eines Strings, der den Namen des Remote-Verzeichnisses enthält, das gelöscht werden soll
Return-Werte:		
Busy	BOOL	TRUE = Baustein wird abgearbeitet
Ok	BOOL	TRUE = Abarbeitung war erfolgreich
Err	BOOL	TRUE = Fehler bei Abarbeitung, Ausgang ErrCode am Client Baustein enthält Fehlernummer

Der Baustein `FtpRmdir` veranlasst den FTP-Server, das angegebene Remote Verzeichnis zu löschen. Der angegebene Pfad `RemoteDir` muss als `ADR(str)` angegeben werden.

Als Pfadtrennzeichen ist in jedem Fall der Slash (/) zu verwenden, nicht der unter Windows übliche Backslash (\).

18.4.5 FTPDELETE (PLMPROTOCOL.LIBRARY)

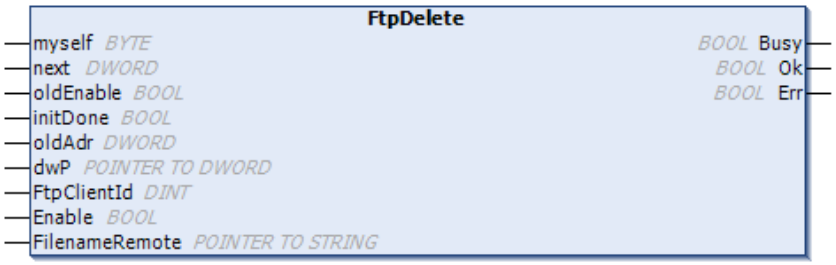


Abbildung 199: Funktionsblock FtpDelete (PlmProtocol.Library)

Input Parameter:		
FtpClientId	DINT	Referenz (Handle) des zugehörigen Client-Bausteins
Enable	BOOL	Enable-Flag, bei steigender Flanke FALSE → TRUE wird der Baustein einmalig ausgeführt; oder fest mit TRUE verbinden
FilenameRemote	POINTER TO STRING	Adresse eines Strings, der den Namen der Remote-Datei enthält, die gelöscht werden soll
Return-Werte:		
Busy	BOOL	TRUE = Baustein wird abgearbeitet
Ok	BOOL	TRUE = Abarbeitung war erfolgreich
Err	BOOL	TRUE = Fehler bei Abarbeitung, Ausgang ErrCode am Client Baustein enthält Fehlernummer

Der Baustein `FtpDelete` veranlasst den FTP-Server, die angegebene Datei auf dem FTP-Server zu löschen.

Falls der Dateiname keinen absoluten Pfad enthält, bezieht sich `FilenameRemote` auf das aktuelle Remote-Arbeitsverzeichnis.

Der angegebene Dateiname `FilenameRemote` muss als `ADR(str)` angegeben werden.

Ob der Dateiname eine Pfadangabe enthalten darf oder nicht hängt vom FTP-Server ab. Als Pfadtrennzeichen ist in jedem Fall der Slash (/) zu verwenden, nicht der unter Windows übliche Backslash (\).

18.4.6 FTPRENAME (PLMPROTOCOL.LIBRARY)

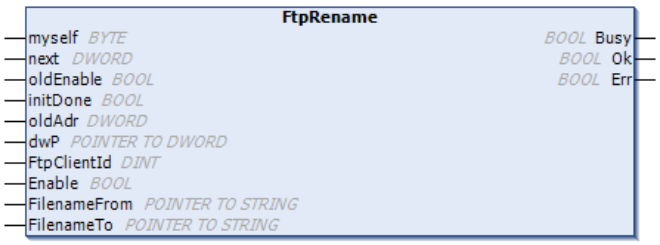


Abbildung 200: Funktionsblock FtpRename (PlmProtocol.Library)

Input Parameter:		
FtpClientId	DINT	Referenz (Handle) des zugehörigen Client-Bausteins
Enable	BOOL	Enable-Flag, bei steigender Flanke FALSE → TRUE wird der Baustein einmalig ausgeführt; oder fest mit TRUE verbinden
FilenameFrom	POINTER TO STRING	Adresse eines Strings, der den alten Namen der Remote-Datei enthält
FilenameTo	POINTER TO STRING	Adresse eines Strings, der den neuen Namen der Remote-Datei enthält
Return-Werte:		
Busy	BOOL	TRUE = Baustein wird abgearbeitet
Ok	BOOL	TRUE = Abarbeitung war erfolgreich
Err	BOOL	TRUE = Fehler bei Abarbeitung, Ausgang ErrCode am Client Baustein enthält Fehlernummer

Der Baustein `FtpRename` veranlasst den FTP-Server, die angegebene Datei auf dem FTP-Server von `FilenameFrom` in `FilenameTo` umzubenennen.

Falls keine absoluten Pfade angegeben sind, beziehen sich `FilenameFrom` und `FilenameTo` auf das aktuelle Remote-Arbeitsverzeichnis. Die angegebenen Dateinamen `FilenameFrom` und `FilenameTo` müssen als `ADR(str)` angegeben werden.

Ob durch das Umbenennen die Datei in ein anderes Directory verschoben werden kann oder nicht hängt vom FTP-Server ab. Als Pfadtrennzeichen ist in jedem Fall der Slash (/) zu verwenden, nicht der unter Windows übliche Backslash (\).

18.4.7 FTPGET (PLMPROTOCOL.LIBRARY)

Abbildung 201: Funktionsblock FtpGet (PlmProtocol.Library)

Input Parameter:		
FtpClientId	DINT	Referenz (Handle) des zugehörigen Client-Bausteins
Enable	BOOL	Enable-Flag, bei steigender Flanke FALSE → TRUE wird der Baustein einmalig ausgeführt; oder fest mit TRUE verbinden
FilenameRemote	POINTER TO STRING	Adresse eines Strings, der den Namen der Datei auf dem FTP-Server enthält
FilenameLocal	POINTER TO STRING	Adresse eines Strings, mit dem Namen, unter dem die Datei auf der PLM-Steuerung gespeichert werden soll
Return-Werte:		
Busy	BOOL	TRUE = Baustein wird abgearbeitet
Ok	BOOL	TRUE = Abarbeitung war erfolgreich
Err	BOOL	TRUE = Fehler bei Abarbeitung, Ausgang ErrCode am Client Baustein enthält Fehlernummer

Der Baustein `FtpGet` veranlasst die Übertragung einer Datei vom FTP-Server auf die PLM-Steuerung.

Die Datei wird als Binärdatei übertragen, d.h. es erfolgt keine Umwandlung von Zeilenendesequenzen etc.

Falls der `FilenameRemote` keinen absoluten Pfad enthält, bezieht er sich auf das aktuelle Remote-Arbeitsverzeichnis.

Die angegebenen Dateinamen `FilenameRemote` und `FilenameLocal` müssen als `ADR(str)` angegeben werden.

Ob `FilenameRemote` eine Pfadangabe enthalten darf oder nicht hängt vom FTPServer ab. Als Pfadtrennzeichen ist in jedem Fall der Slash (/) zu verwenden, nicht der unter Windows übliche Backslash (\).

Die Remote-Datei ersetzt (überschreibt) die lokale Datei. Falls stattdessen die Remote-Datei an die lokale Datei angehängt werden soll, kann dies durch Setzen des Config-Parameter *AppendMode* auf 1 erreicht werden.

18.4.8 FTPLIST (PLMPROTOCOL.LIBRARY)

Abbildung 202: Funktionsblock FtpList (PlmProtocol.Library)

Input Parameter:		
FtpClientId	DINT	Referenz (Handle) des zugehörigen Client-Bausteins
Enable	BOOL	Enable-Flag, bei steigender Flanke FALSE → TRUE wird der Baustein einmalig ausgeführt; oder fest mit TRUE verbinden

Arg	POINTER TO STRING	Adresse eines Strings, der den Namen des Remote-Verzeichnisses enthält, das gelistet werden soll
List	POINTER TO ARRAY OF STRING	Adresse eines String-Arrays, in das die Listing-Zeilen geschrieben werden. Kann 0 sein, in diesem Fall werden die Listing-Zeilen nicht gespeichert
ListLen	WORD	Länge des List-Arrays, d.h. max. Anzahl von Listing-Zeilen, die gespeichert werden können. Kann 0 sein
Return-Werte:		
Busy	BOOL	TRUE = Baustein wird abgearbeitet
Ok	BOOL	TRUE = Abarbeitung war erfolgreich
Err	BOOL	TRUE = Fehler bei Abarbeitung, Ausgang ErrCode am Client Baustein enthält Fehlernummer
ActLen	WORD	Tatsächliche Anzahl von Listingzeilen (unabhängig von ListLen)

Der Baustein `FtpList` veranlasst den FTP-Server, die Dateien im durch `Arg` angegebenen Verzeichnis aufzulisten.

Die Auswertung des `Arg`-Parameters und das Listing-Ergebnis hängen stark vom jeweiligen FTP-Server ab.

Wenn der Parameter `List` angegeben ist, muss der Parameter `ListLen` angegeben sein und die max. Länge des Arrays enthalten. In diesem Fall wird das Array mit den Ergebniszeilen des FTP-Servers gefüllt.

Wenn der Parameter `List` nicht oder mit `0` angegeben ist, wird `ListLen` nicht ausgewertet. Die Ergebniszeilen des FTP-Servers werden in diesem Fall nicht gespeichert.

In beiden Fällen enthält der Ausgang `ActLen` die tatsächliche Anzahl der Ergebniszeilen und kann in jedem Fall ausgewertet werden. Dies kann z.B. sinnvoll sein, wenn `Arg` den Namen einer bestimmten Remote-Datei enthält und lediglich geprüft werden soll, ob die Datei auf dem FTP-Server vorhanden ist oder nicht. In diesem Fall brauchen die Listing-Zeilen nicht gespeichert zu werden und `ActLen` enthält entweder den Wert `0` oder `1`.

Das Listing-Format kann mit dem Config-Parameter *WideListing* eingestellt werden: Bei *WideListing* = 0 werden nur die Dateinamen des Remote-Verzeichnisses aufgelistet. Bei *WideListing* = 1 werden weitere Angaben mit aufgelistet, die vom Betriebssystem des FTP-Servers abhängen.

Achtung: Das genaue Ergebnis des Bausteins variiert stark zwischen verschiedenen FTP-Servern und ist auch abhängig vom Listing-Format (Config-Parameter *WideListing*). Dies betrifft die Anzahl der Ergebniszeilen, die Auflistung von Verzeichnissen und das Auflisten von fehlenden Dateien. Es muss daher experimentell ermittelt werden!

18.4.9 FTPPUT (PLMPROTOCOL.LIBRARY)

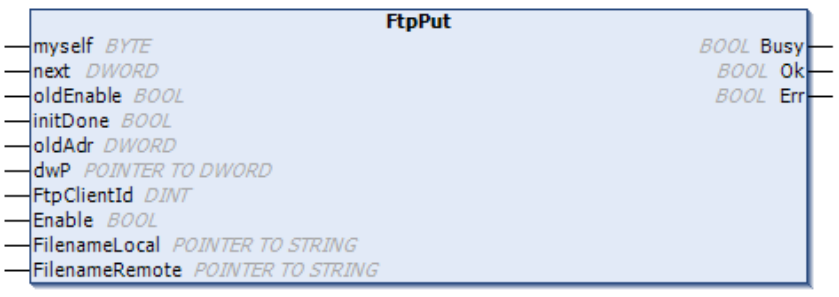


Abbildung 203: Funktionsblock FtpPut (PlmProtocol.Library)

Input Parameter:		
FtpClientId	DINT	Referenz (Handle) des zugehörigen Client-Bausteins
Enable	BOOL	Enable-Flag, bei steigender Flanke FALSE → TRUE wird der Baustein einmalig ausgeführt; oder fest mit TRUE verbinden
FilenameLocal	POINTER TO STRING	Adresse eines Strings, der den Namen der Datei auf der PL-Steuerung enthält
FilenameRemote	POINTER TO STRING	Adresse eines Strings mit dem Namen, unter dem die Datei auf dem FTP-Server gespeichert werden soll
Return-Werte:		
Busy	BOOL	TRUE = Baustein wird abgearbeitet
Ok	BOOL	TRUE = Abarbeitung war erfolgreich
Err	BOOL	TRUE = Fehler bei Abarbeitung, Ausgang ErrCode am Client Baustein enthält Fehlernummer

Der Baustein `FtpPut` veranlasst die Übertragung einer Datei von der PLM-Steuerung auf den FTP-Server.

Die Datei wird als Binärdatei übertragen, d.h. es erfolgt keine Umwandlung von Zeilenendesequenzen etc.

Falls der `FilenameRemote` keinen absoluten Pfad enthält, bezieht er sich auf das aktuelle Remote-Arbeitsverzeichnis.

Die angegebenen Dateinamen `FilenameLocal` und `FilenameRemote` müssen als `ADR(str)` angegeben werden.

Ob `FilenameRemote` eine Pfadangabe enthalten darf oder nicht hängt vom FTPServer ab. Als Pfadtrennzeichen ist in jedem Fall der Slash (/) zu verwenden, nicht der unter Windows übliche Backslash (\).

Die lokale Datei ersetzt (überschreibt) die Remote-Datei auf dem FTP-Server. Falls stattdessen die lokale Datei an die Remote-Datei angehängt werden soll, kann dies durch Setzen des Config-Parameter *AppendMode* auf 1 erreicht werden.

18.4.10 FTPQUIT (PLMPROTOCOL.LIBRARY)

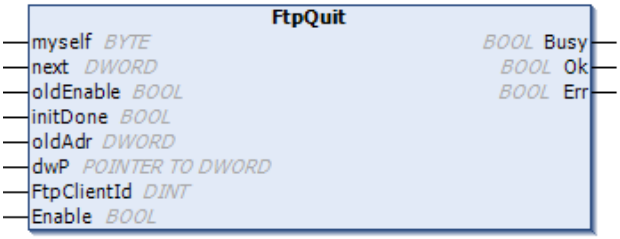


Abbildung 204: Funktionsblock FtpQuit (PlmProtocol.Library)

Input Parameter:		
FtpClientId	DINT	Referenz (Handle) des zugehörigen Client-Bausteins
Enable	BOOL	Enable-Flag, bei steigender Flanke FALSE → TRUE wird der Baustein einmalig ausgeführt; oder fest mit TRUE verbinden

Return-Werte:		
Busy	BOOL	TRUE = Baustein wird abgearbeitet
Ok	BOOL	TRUE = Abarbeitung war erfolgreich
Err	BOOL	TRUE = Fehler bei Abarbeitung, Ausgang ErrCode am Client Baustein enthält Fehlernummer

Der Baustein `FtpQuit` veranlasst eine Abmeldung auf dem FTP-Server. Anschließend muss die Verbindung durch den Client geschlossen werden.

Der QUIT-Befehl wird automatisch ausgeführt, sobald der `Enable`-Eingang am Client-Baustein auf `FALSE` gesetzt wird, um die Verbindung zu beenden. Dieses Verhalten kann durch den Config-Parameter *AutoQuit* modifiziert werden.

18.4.11 FTPCONFIG (PLMPROTOCOL.LIBRARY)

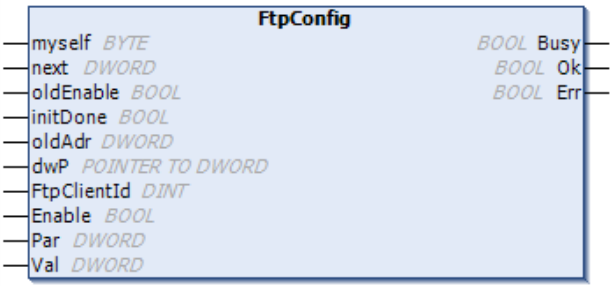


Abbildung 205: Funktionsblock FtpConfig (PlmProtocol.Library)

Input Parameter:		
FtpClientId	DINT	Referenz (Handle) des zugehörigen Client-Bausteins
Enable	BOOL	Enable-Flag, bei steigender Flanke FALSE → TRUE wird der Baustein einmalig ausgeführt; oder fest mit TRUE verbinden
Par	DWORD	Nummer des Config-Parameters, der geändert werden soll (siehe globale Konstanten in GblFtpClient)
Val	DWORD	Neuer Wert des Config-Parameters
Return-Werte:		
Busy	BOOL	TRUE = Baustein wird abgearbeitet
Ok	BOOL	TRUE = Abarbeitung war erfolgreich
Err	BOOL	TRUE = Fehler bei Abarbeitung, Ausgang ErrCode am Client Baustein enthält Fehlernummer

Der Baustein `FtpConfig` ermöglicht das Einstellen verschiedener Parameter des FTP-Clients. Im Gegensatz zu den anderen Befehlsbausteinen führt er keinen Befehl auf dem FTP-Server aus.

Wenn mehrere Parameter umgestellt werden sollen, ist für jeden ein eigener Config Baustein zu instanzieren und einzusetzen. Dies gilt auch, wenn ein Parameter mehrfach umgestellt werden soll.

Für die Nummer des gewünschten Config-Parameters am Eingang `Par` existieren globale Konstantendefinitionen in der globalen Variablenlist `GlbFtpClient`. Die möglichen Werte am Eingang Val hängen vom gewählten Config-Parameter ab.

Die folgende Tabelle zeigt, welche Config-Parameter eingestellt werden können:

Par	Val	Voreinst.	Funktion
Apendmode (101)	DWORD = 0/1	0 (aus)	Schaltet um zwischen Überschreiben/Anhängen bei FtpGet und FtpPut
WideListing (102)	DWORD = 0/1	0 (aus)	Schaltet das Listing-Format für FtpList um
AutoQuit (103)	DWORD = 0/1	1 (ein)	Veranlasst das automatische Senden eines Quit-Befehls vor dem Trennen der Verbindung durch den Client
StopOnError (104)	DWORD = 0/1	1 (ein)	Stoppt nach einem Fehler die Ausführung weiterer Befehlsbausteine. Wenn DisconnectOnError (109) TRUE ist, wird zusätzlich die Verbindung beendet.
PassiveMode (105)	DWORD = 0/1	1 (ein)	Legt fest, ob Datenübertragungen im Active oder Passive Mode ausgeführt werden. Momentan wird nur der Passive Mode unterstützt.
TransferMode (106)	ADR(string)= 'i/a'	'i' (binär)	Legt den Transfer Mode fest. Momentan wird nur 'i' (binär) unterstützt.
ControlPort(107)	DWORD = Port	21	Legt den FTP-Port fest
DataPort (108)	DWORD = Port	20	Legt den TCP-Port für Datenübertragung fest (nur Active Mode)
DisconnectOnError (109)	DWORD = 0/1	1 (ein)	Wenn StopOnError (104) TRUE ist und ein Fehler auftritt, wird die Verbindung getrennt

18.5 PROGRAMMBEISPIEL (FUP)

Das folgende Programmbeispiel realisiert einen FTP-Client, der sich nach dem Setzen von `do_connect` auf `TRUE` als User `'anonymous'` und mit leerem Password mit dem FTP-Server an der IP-Adresse 127.0.0.1 verbindet.

Bei erfolgreicher Anmeldung wird die Datei `'a/local.txt'` übertragen und auf dem FTP-Server unter `'a/remote.txt'` gespeichert.

Nachdem der Ausgang `client.busy` auf `FALSE` gegangen ist, kann `do_connect` wieder auf `FALSE` gesetzt und die Verbindung zum FTP-Server dadurch beendet werden.

Die im Beispiel gewählte IP-Adresse 127.0.0.1 (localhost) erlaubt die Verwendung des FTP-Servers derselben PLM-Steuerung, auf der der FTP-Client getestet wird.

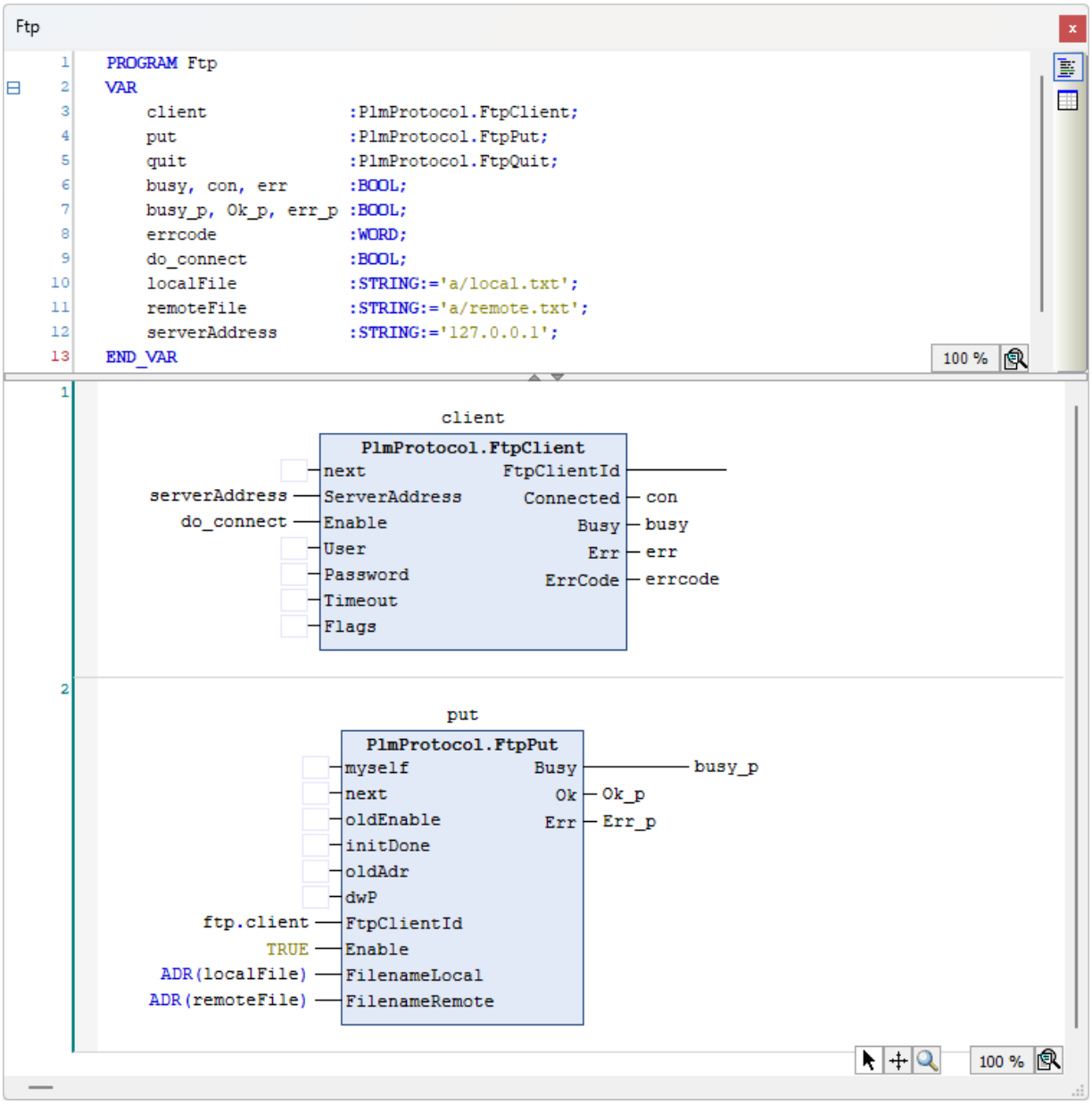


Abbildung 206: Beispiel für Datenübertragung durch FTP-Client (FUP)

18.6 PROGRAMMBEISPIEL (ST)

Ftp_ST

```
1   PROGRAM Ftp_ST
2   VAR
3       client              :PlmProtocol.FtpClient;
4       put                 :PlmProtocol.FtpPut;
5       quit                :PlmProtocol.FtpQuit;
6       busy, con, err      :BOOL;
7       busy_p, Ok_p, err_p :BOOL;
8       errcode             :WORD;
9       do_connect          :BOOL;
10      localFile           :STRING:='a/local.txt';
11      remoteFile          :STRING:='a/remote.txt';
12      serverAddress       :STRING:='127.0.0.1';
13  END_VAR
```

100 %

```
1   Client(
2       ServerAddress:= serverAddress,
3       Enable:= do_connect,
4       FtpClientId=> ,
5       Connected=> con,
6       Busy=> busy,
7       Err=> err,
8       ErrCode=> errcode);
9
10  Put(
11      FtpClientId:= client.FtpClientId,
12      Enable:= TRUE,
13      FilenameLocal:= ADR(localFile),
14      FilenameRemote:= ADR(remoteFile),
15      Busy=> busy_p,
16      Ok=> ok_p,
17      Err=> err_p);
18
```

100 %

Abbildung 207: Beispiel für Datenübertragung durch FTP-Client (ST)

18.7 FEHLERSUCHE

Für die Fehlersuche und Problemdiagnose stehen zwei Möglichkeiten zur Verfügung.

1. Die zuletzt aufgetretene negative Serverantwort kann als String gespeichert und dann vom Anwender ausgewertet werden. Dazu ist in der globalen Variablenliste `GlbFtpClient` der Variable `Plm_FtpLastServerErrMsg` die Adresse einer String-Variablen zuzuweisen, in der die negative Serverantwort im Klartext gespeichert wird.

Beispiel:

```
VAR
      serverMsg: STRING(80);
END_VAR
PlmProtocol.GlbFtpClient.Plm_FtpLastServerErrMsg := ADR(serverMsg);
```

Nach der erfolgreichen Ausführung eines Befehlsbausteins bleibt der String in `serverMsg` leer. Falls jedoch der FTP-Server einen Befehl nicht ausführen konnte, erscheint die entsprechende Serverantwort in `serverMsg`, z.B. nach fehlgeschlagenem `FtpChdir`:

```
'550 mydir: No such file or directory.'
```

Die genauen Serverantworten hängen vom Typ des FTP-Servers ab.

Dieses Verfahren ersetzt nicht die Auswertung von `ErrCode`, da nur diejenigen Fehler im Klartext erscheinen, die vom FTP-Server gemeldet werden. Lokale Fehler des FTP-Clients können nur anhand des Wertes von `ErrCode` identifiziert werden.

Es werden maximal 80 Zeichen in die String-Variable geschrieben; eine längere Fehlermeldung des FTP-Servers wird abgeschnitten.

2. Die Kommunikation zwischen FTP-Client und FTP-Server kann im Klartext aufgezeichnet und anschließend analysiert werden (Log-File).

Zum Einschalten der Protokollierung ist am Eingang `Flags` des FTP-Clients der Wert `16#8000` anzulegen. Die globale Variable `Plm_FtpLogFilename` aus der globalen Variablenliste `GlbFtpClient` enthält den Namen der anzulegenden Log-Datei. Der Name ist voreingestellt auf

```
'a/ftp_debug.txt',
```

er kann jedoch vor dem ersten Aufruf des FTP-Client-Bausteins geändert werden.

Nach dem Ausführen des FTP-Clients kann die Log-Datei per FTP von der PLM Steuerung auf einen PC heruntergeladen und dort mit einem Texteditor analysiert werden.

Die Aufzeichnung der Log-Datei sollte nur in der Entwicklungsphase verwendet werden, da dies erheblich zusätzliche Rechenzeit benötigt und dadurch evtl. den IEC Zyklus verlangsamt.

19 WEBVISUALISIERUNG MIT HTTPS VERWENDEN

Die Nutzung der Webvisualisierung über eine **HTTPS**-Verbindung zwischen einem Webbrowser auf einem PC und dem Webserver der Steuerung ist grundsätzlich möglich. Voraussetzung hierfür ist, dass zunächst mithilfe der CODESYS IDE Zertifikate für den Webserver auf der Steuerung erstellt werden. Anschließend müssen diese Zertifikate auch auf dem Windows-PC installiert werden, von dem aus auf die Webvisualisierung der Steuerung zugegriffen werden soll.

Für den sicheren Zugriff auf die Webvisualisierung müssen die Sicherheitseinstellungen der Steuerung entsprechend konfiguriert werden. Hierzu ist als Kommunikationsprotokoll **HTTPS** zu aktivieren. Zusätzlich muss eine aktuelle und unterstützte **TLS-Version (Transport Layer Security)** ausgewählt werden, um eine verschlüsselte und sichere Datenübertragung zwischen Webbrowser und Steuerung zu gewährleisten. Da die für den Webserver verwendeten Zertifikate in der Regel selbst erstellt werden, muss außerdem die Option zum **Zulassen selbsterstellter Zertifikate** aktiviert werden. Nur wenn diese Einstellungen korrekt vorgenommen wurden, kann eine verschlüsselte Verbindung mit dem Webserver der Steuerung aufgebaut werden.

19.1 SICHERHEITSEINSTELLUNGEN AUF DER STEUERUNG FÜR HTTPS

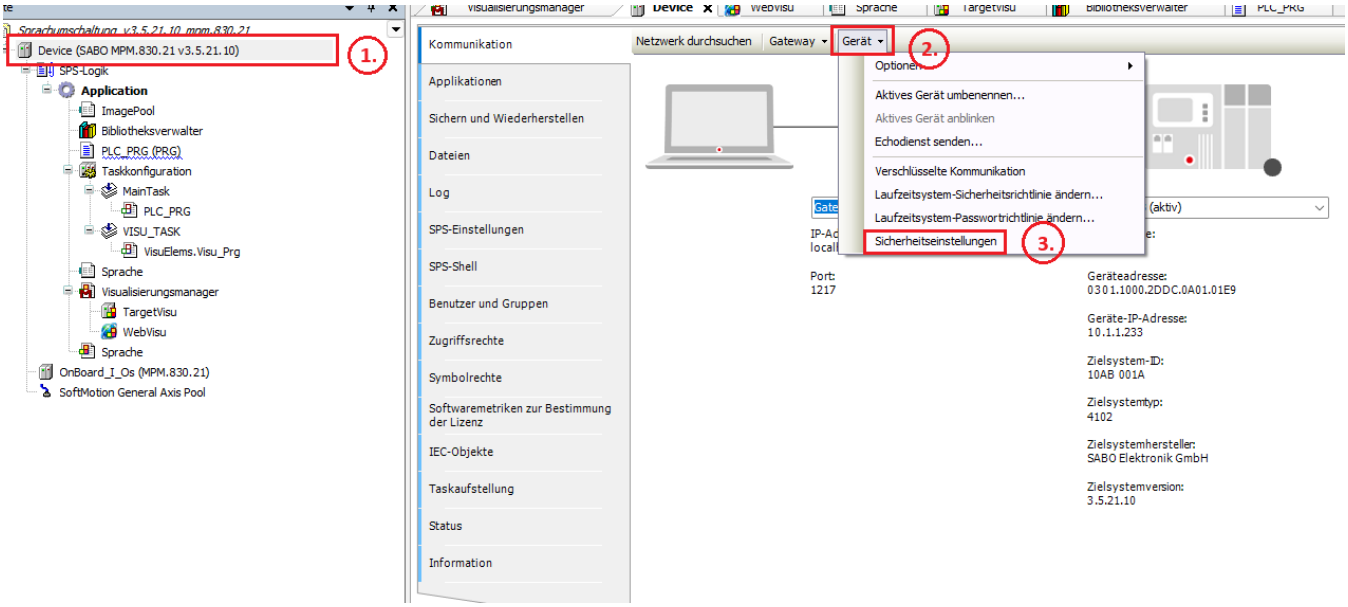


Abbildung 208: Sicherheitseinstellungen der Steuerung für den Webserver

Um auf die **Sicherheitseinstellungen** zugreifen zu können, verbinden Sie sich mit der Steuerung und klicken im nächsten Schritt das **Device** (1) an, anschließend klicken Sie auf den Menüpunkt **Gerät** (2) und wählen aus der Dropdownliste den Punkte **Sicherheitseinstellungen** (3) aus. *Abbildung 208.*

Die nötigen Einstellungen für die HTTPS-Verbindung können in dem Fenster Gerätesicherheitseinstellungen vorgenommen werden. Der Gerätename ist für den Aufruf im Webbrowser wichtig und sollte notiert werden. In der Abbildung 209 und Tabelle sind die Einstellungen dargestellt.

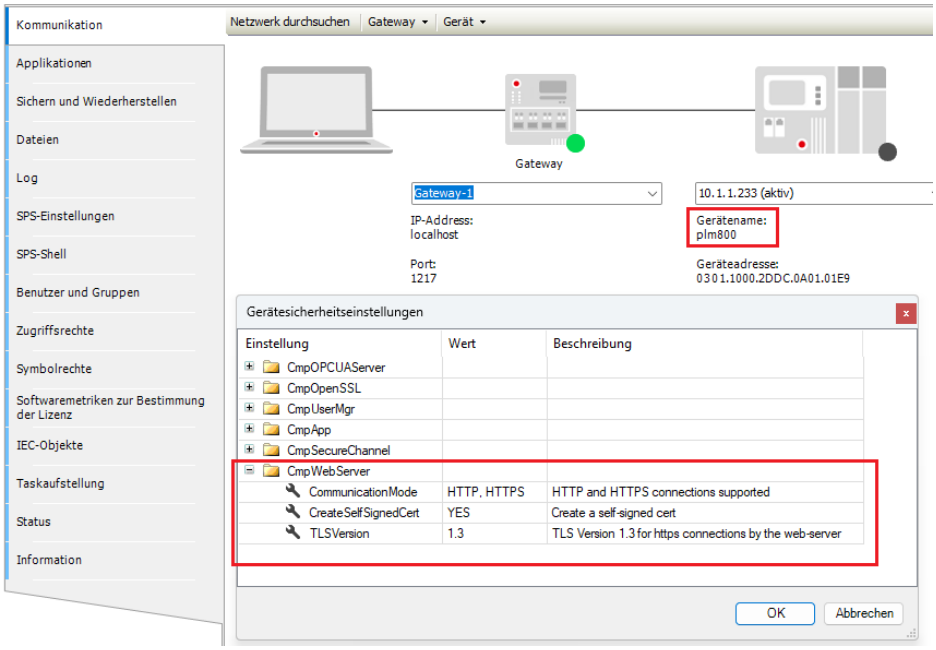


Abbildung 209: Gerätesicherheitseinstellungen

CommunicationMode	HTTP, HTTPS	HTTP and HTTPS connection supported
CreateSelfSignedCert	YES	Create a self signed cert
TLSVersion	1.3	TLS Version 1.3 for https connections by the web-server

Tabelle 10: Sicherheitseinstellungen für HTTPS Verbindungen

19.2 ZERTIFIKATSERSTELLUNG

Die Erstellung des Zertifikats erfolgt über die CODESYS IDE im **Security Screen**. Hierfür muss eine Verbindung zur Steuerung bestehen oder zunächst hergestellt werden. Der **Security Screen** kann über die Menüleiste der CODESYS IDE geöffnet werden. Wählen Sie dazu den Menüpunkt **Ansicht** und anschließend **Security Screen** aus.

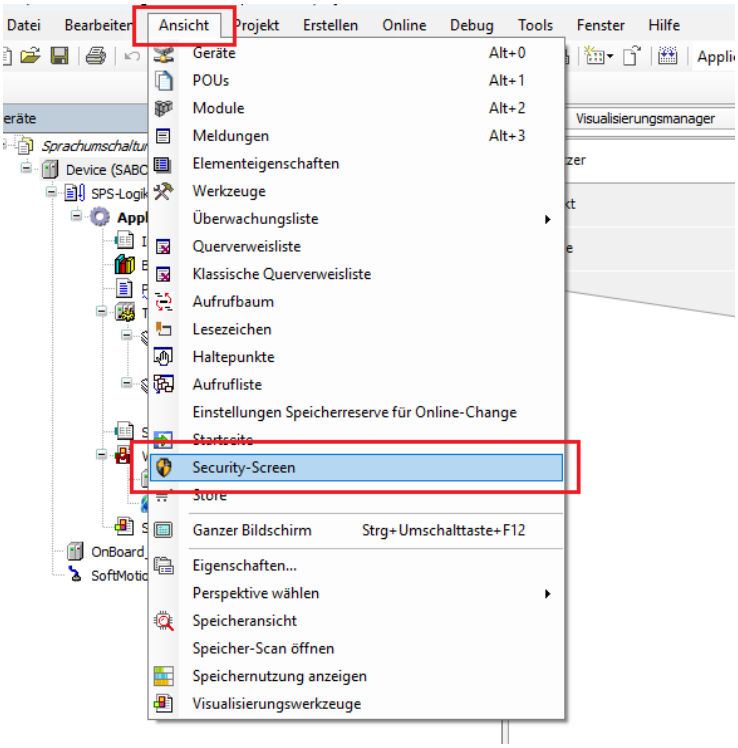


Abbildung 210: Security Screen

Über den Security Screen kann das Zertifikat für den Webserver erstellt werden.

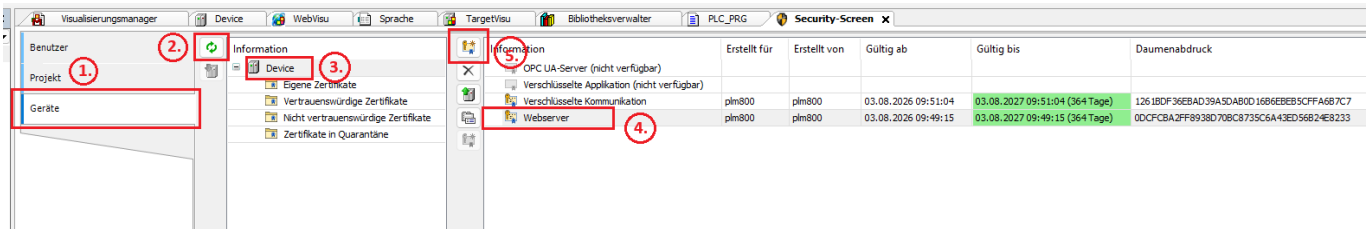


Abbildung 211: Zertifikaterstellug im Security Screen

Um das Zertifikat zu erstellen, muss über den Reiter **Geräte (1)** und das **Aktualisieren-Symbol (2)** zuerst eingelesen werden, welche Zertifikate auf der Steuerung möglich sind und ob schon eventuell Zertifikate auf der Steuerung vorhanden sind.
Nachdem die Informationen aktualisiert wurden, kann über das **Device (3)** eingesehen werden, welche Zertifikate auf der Steuerung vorhanden sind, bzw. welche Zertifikate erstellt werden können.
Um das Webserverzertifikat zu erstellen, wählen Sie in der rechten Informationsansicht den **Webserver (4)** aus und markieren diesen mit einem Linksklick. Über die Schaltfläche **Zertifikat erstellen (5)** wird anschließend das Zertifikat erstellt.
Wichtig ist hier, dass die **Uhrzeit** und das **Datum** auf der **Steuerung richtig gesetzt** sind, da sonst das Zertifikat mit einem falschen Datum erstellt wird.

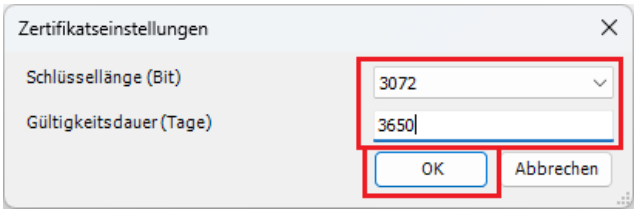


Abbildung 212; Zertifikatseinstellungen

Bei den Zertifikatseinstellungen muss die Schlüssellänge für die Verschlüsselung angegeben werden und die Gültigkeitsdauer für das Zertifikat in Tagen. CODESYS trägt eine Gültigkeitsdauer von 365 Tagen (1 Jahr) automatisch ein. Diese kann aber auf bis zu 3650 Tage (10 Jahre) erhöht werden.
Mit OK werden die Einstellungen übernommen und die Steuerung berechnet das Zertifikat. Das kann je nach Leistung der Steuerung etwas dauern.

19.3 ZERTIFIKATSINSTALLATION

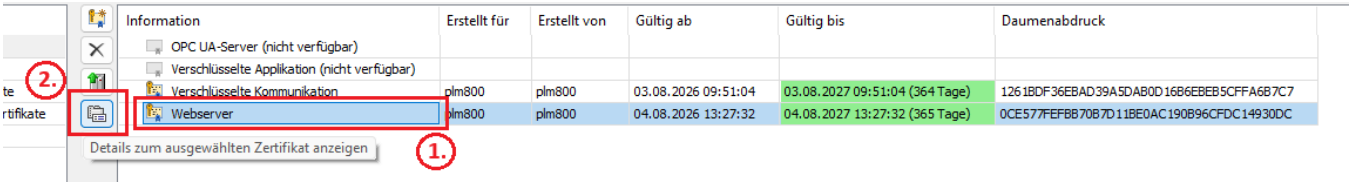


Abbildung 213: Zertifikat im Security Screen

Das **erstelle Zertifikat (1)** wird im Security Screen mit der Gültigkeit angezeigt. Über die Schaltfläche **Details zum Zertifikat (2)**, kann das Zertifikat auf dem Windowsrechner installiert werden, auf dem die Webvisualisierung aufgerufen werden soll.

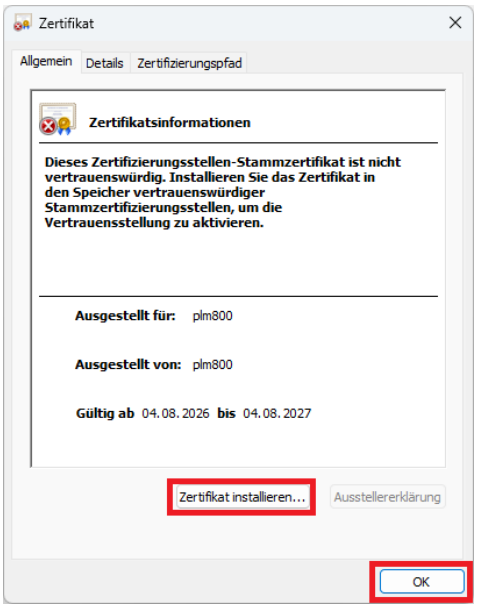


Abbildung 214: Zertifikatsinstallation

Über **Zertifikat installieren...** kann das Zertifikat auf dem aktuellen Windowsrechner installiert werden. Mit **OK** wir das Fenster wieder verlassen.

Damit das Zertifikat auf dem Rechner richtig installiert wird, muss bei der Abfrage nach dem Speicherort der **Lokale Computer** angegeben werden, *Abbildung 215*. Hierfür sind eventuell erhöhte Benutzerrechte nötig.

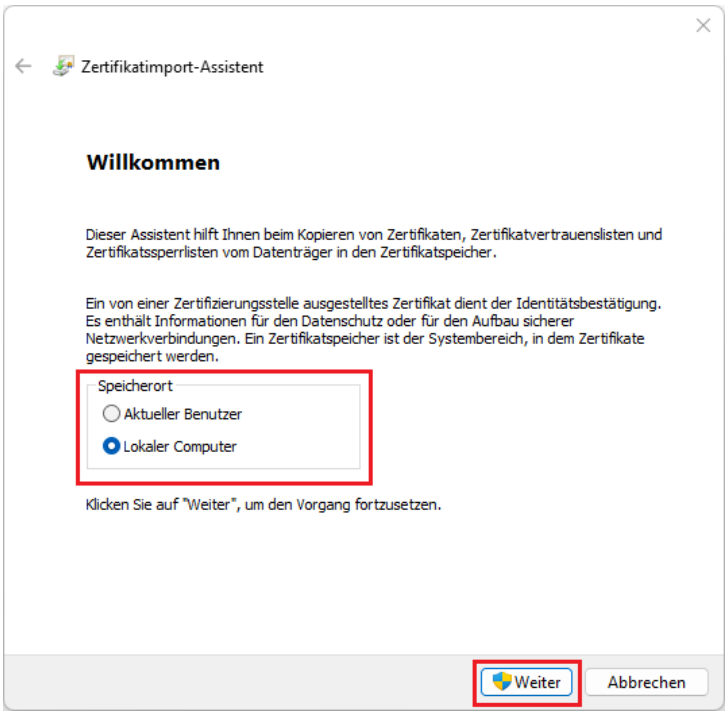


Abbildung 215: Assistent für das Importieren von Zertifikaten

Über die Schaltfläche **Weiter** gelangt man zu dem Punkt, in dem angegeben werden muss, wo das Zertifikat abgelegt werden soll.

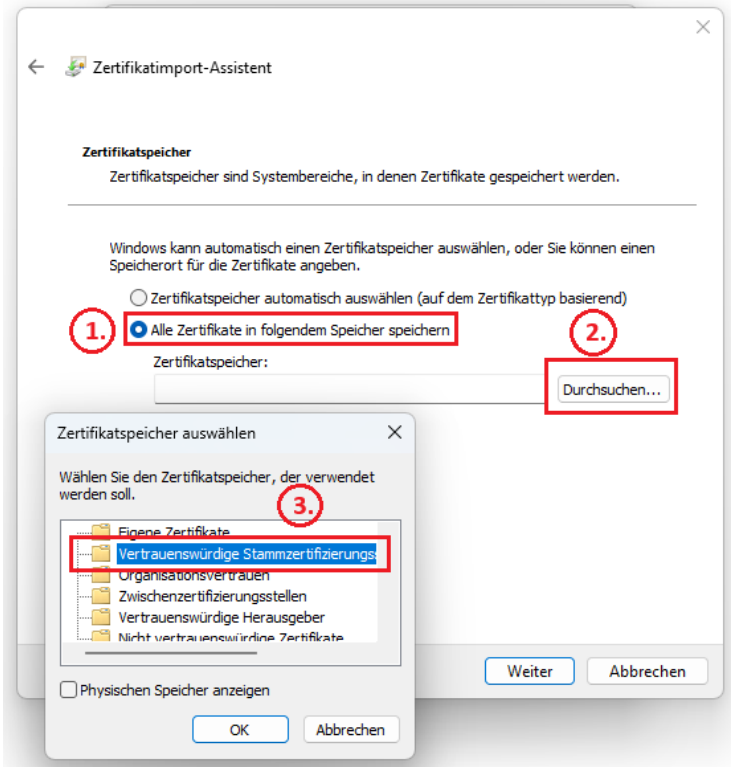


Abbildung 216: Zertifikatsspeicher

Für die Ablage des Zertifikats ist die Option **Alle Zertifikate in folgendem Speicher speichern (1)** auszuwählen.
Über die Schaltfläche **Durchsuchen... (2)** kann der Windows-Zertifikatsspeicher geöffnet und der entsprechende Zertifikatsspeicher ausgewählt werden.
Damit das erstellte Webserverzertifikat vom Windows-System als vertrauenswürdig eingestuft wird, ist es im Zertifikatsspeicher **Vertrauenswürdige Stammzertifizierungsstellen (3)** abzulegen.
Über **Ok** wird das Zertifikat am Speicherort abgelegt.

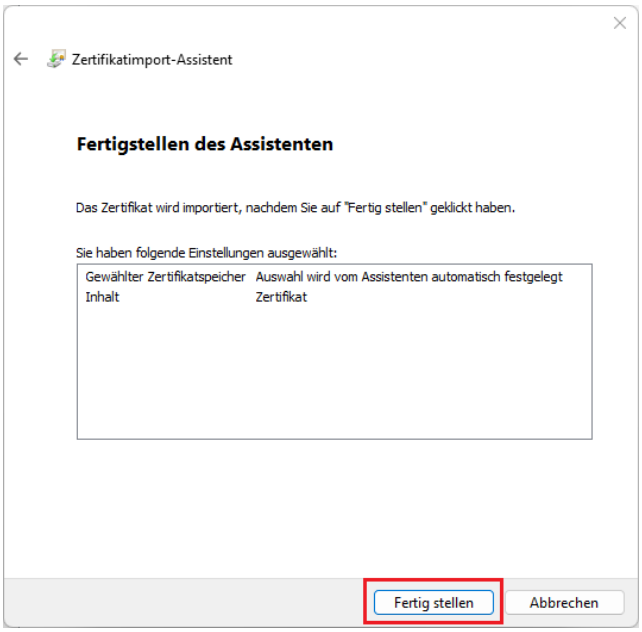


Abbildung 217: Fertige Zertifikatinstallation

Über die Schaltfläche **Fertig stellen** wird die Installation des Zertifikats auf dem Windowssystem abgeschlossen und das Zertifikat bekannt.

Alternativ kann das Zertifikat von der Steuerung heruntergeladen werden, um es auf einem anderen Windowsrechner zu installieren.

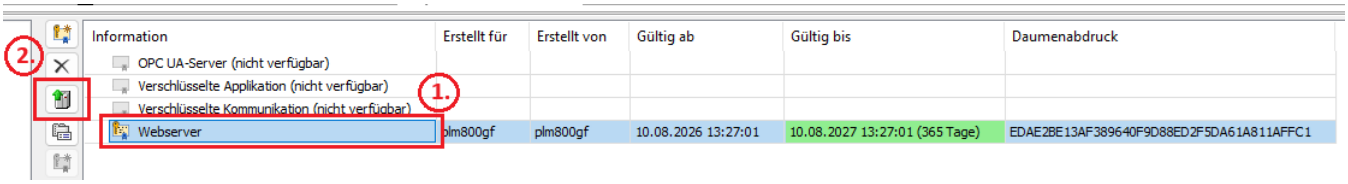


Abbildung 218: Zertifikat von der Steuerung herunterladen

Hierfür muss das Zertifikat für den **Webserver ausgewählt (1)** und über die Schaltfläche **Zertifikat von der Steuerung herunterladen (2)** als *.cer-Datei auf dem Rechner abgespeichert werden. Diese Datei kann dann über einen Doppelklick auf einem anderen Windowsrechner installiert werden, siehe dazu *Abschnitt 19.3 und Abbildung 214*.

19.4 AUFRUF DER WEBVISUALISIERUNG MIT HTTPS

Um auf die Webvisualisierung über HTTPS zugreifen zu können, muss man den Namen der Steuerung beim Aufruf angeben. Der Name der Steuerung kann in dem Reiter Kommunikation unter dem Gerätenamen abgelesen werden. In dem Bespiel ist das plm800, siehe *Abbildung 209*.

Der Gerätenamen kann über das Webconfig vorgegeben werden. Das Webconfig wird aufgerufen, indem man in einem Webbrowser die IP-Adresse der Steuerung eingibt. Dafür muss der Rechner die Steuerung über das Netzwerk erreichen können.

Der Gerätename wird unter dem Reiter Network und dem Punkt Hostname über die Schaltfläche **Change...** angepasst, *Abbildung 219*.

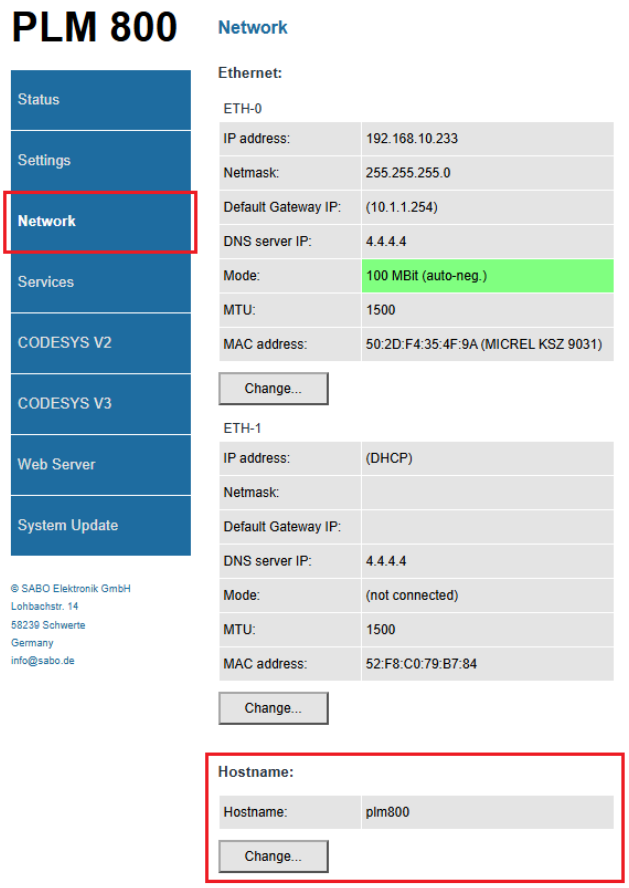


Abbildung 219: Gerätenamen der Steuerung über das Webconfig anpassen

Der Aufruf im Webbrowser für die Webvisualisierung sieht dann wie folgt aus:

```
https://plm800/webvisu.htm
```

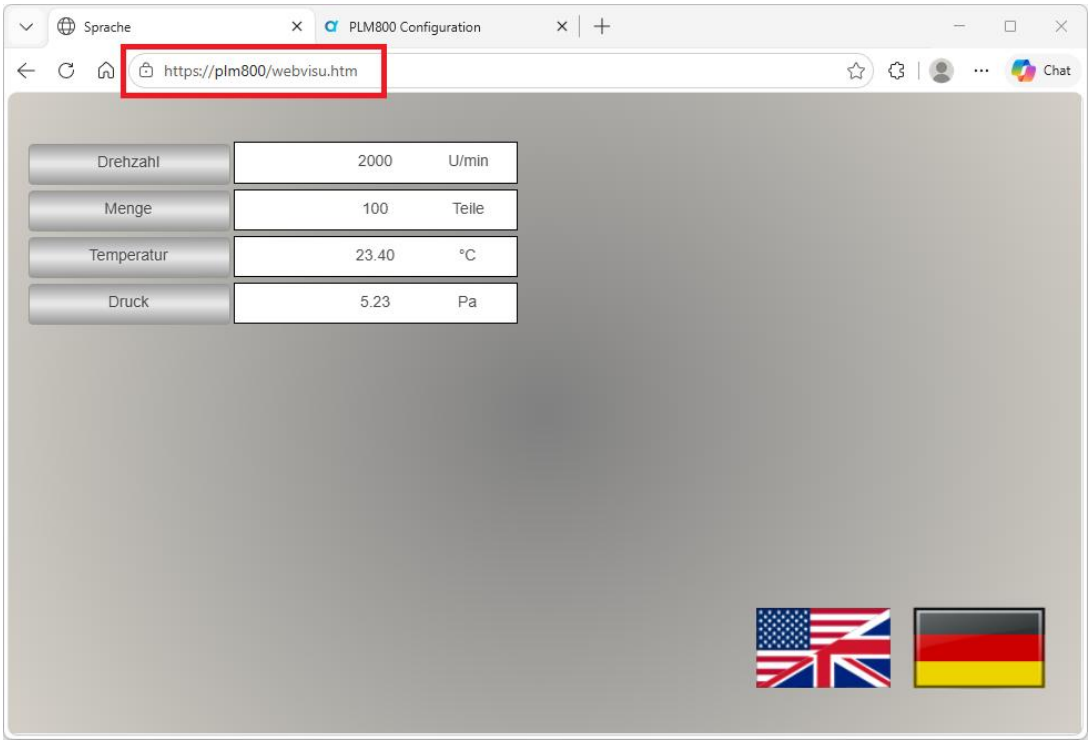


Abbildung 220: Webvisualisierung mit https-Aufruf

19.5 FEHLERSUCHE

Webvisualisierung kann nicht angezeigt werden oder sie wird nur sporadisch angezeigt.

Im Auslieferungszustand bei der Firma SABO verfügen die Steuerungen standardmäßig über den gleichen Gerätenamen **plm800**.

Werden mehrere Steuerungen mit identischem Gerätenamen gleichzeitig im selben Netzwerk betrieben, kann es zu einer Namensauflösung kommen, bei der der im Webbrowser angegebene Gerätename nicht eindeutig einer bestimmten Steuerung zugeordnet werden kann. Infolgedessen ist die Webseite der Steuerung möglicherweise nicht erreichbar und der Webbrowser zeigt eine entsprechende Fehlermeldung an.

Webbrowser zeigt eine Fehlermeldung „ERR_DNS_PROBE_FINISHED_NXDOMAIN" oder in ähnlicher Form an.

Auch hier könnte ein doppelter Gerätename das Problem sein oder aber der Windowsrechner kann den Domänennamen nicht auflösen. Netzwerkparamter des Windowsrechners und der Steuerung überprüfen.

Eventuell muss die hosts-Datei von dem Windowsrechner editiert werden, damit der Gerätename richtig aufgelöst werden kann. Der Pfad zu der hosts-Datei ist `C:\Windows\System32\drivers\etc\` und kann nur mit Adminrechten bearbeitet werden.
Um dort den Gerätenamen einer IP-Adresse zuzuweisen tragen Sie am Ende der Datei die IP-Adresse ein und direkt dahinter den Gerätenamen der Steuerung:

```
192.168.10.233        plm800
```

Speichern Sie die Datei und rufen Sie die Webvisualisierung erneut auf. Jetzt sollte der Webbrowser die Webvisualisierung anzeigen können.

Bei Fragen zu dem Thema können Sie uns gerne eine E-Mail an info@sabo.de oder support@sabo.de schreiben.