Table of contents

Objekt 'Symbolkonfiguration'

Objekt 'Symbolkonfiguration'

Mit der Symbolkonfiguration können Sie Symbolbeschreibungen für Projektvariablen erstellen. Sie fügen dazu mit Befehl *Projekt ➔ Objekt hinzufügen* ein Objekt Symbolkonfiguration im Gerätebaum ein und geben dabei bestimmte Voreinstellungen mit. Siehe unten: Dialog *„Symbolkonfiguration hinzufügen“*.

Den Symbolkonfigurationseditor öffnen Sie dann durch einen Doppelklick auf das Objekt *„Symbolkonfiguration“*.

Dialog 'Symbolkonfiguration hinzufügen'

Funktion: Der Dialog dient dem Definieren von Voreinstellungen für ein Objekt *„Symbolkonfiguration“*.

Aufruf: Menü *Projekt ➔ Objekt hinzufügen ➔ Symbolkonfiguration*, Kontextmenü des Objekts Applikation.

„Kommentare in XML einschließen“	Exportiert die den Variablen zugewiesenen Kommentare mit in die Symboldatei.
„OPC UA-Funktionalitäten unterstützen“	Hinweis: Verfügbarkeit und Bearbeitbarkeit dieser Option sind geräteabhängig. ☑: Beim Download der Symbolkonfiguration werden zusätzliche Informationen auf die Steuerung geladen, die für den Betrieb des OPC UA-Servers notwendig sind: ▪ Basistypen von abgeleiteten Funktionsbausteinen ▪ Inhalte von Attributen, die über Compiler-Pragmas vergeben werden ▪ Gültigkeitsbereiche, beispielsweise VAR_INPUT, VAR_OUTPUT, VAR_IN_OUT

„Clientseitiges Datenlayout“

Sehen Sie ausführlichere Informationen und Beispiele zu den Layout-Optionen im nachfolgenden Abschnitt "Symbolkonfigurationseditor".

„Kompatibilitätslayout“	Diese Einstellung dient der Kompatibilität im Fall von alten Projekten. Das für den Client erzeugte Datenlayout wird soweit als möglich dem intern vom Compiler erzeugten Layout angeglichen.

„Optimiertes Layout“	Empfohlen für neue Projekte. Berechnet das Ausgangslayout in der optimierten Form entkoppelt vom internen Compilerlayout. Hinterlässt keine Lücken für nicht veröffentlichte Elemente und hält die Anforderungen an die Speicherausrichtung der Datentypen strikt ein. Erfordert mindestens Compilerversion 3.5.7.0.

Symbolkonfigurationseditor

Der Editor enthält eine Tabelle mit den bereits ausgewählten Variablen und eine Menüleiste für die Bearbeitung.

Menüleiste

„Ansicht“	Über diese Schaltfläche können Sie folgende Kategorien von Variablen für die Aufnahme in den Konfigurationseditor ein- und ausschalten: ▪ *„Unkonfigurierte aus Projekt“*: Variablen, die der Symbolkonfiguration noch nicht hinzugefügt sind, aber im Projekt dafür bereitstehen. ▪ *„Unkonfigurierte aus Bibliotheken“*: Variablen, die der Symbolkonfiguration noch nicht hinzugefügt sind, aber in eingebundenen Bibliotheken dafür bereitstehen. ▪ *„Über Attribute exportierte Symbole“*: Der Filter bewirkt, dass auch Variablen aufgelistet werden, die bereits über das Pragmaattribut {attribute 'symbol' := 'read'} für den Export in die Symboldatei gekennzeichnet sind. Diese Symbole sind grau dargestellt. Die Spalte *„Attribut“* zeigt jeweils, welches Zugriffsrecht durch das Pragma gesetzt ist.
„Erstellen“	Übersetzen des Projekts. Voraussetzung für eine aktuelle Bereitstellung der Variablen im Konfigurationseditor.

„Einstellungen“	▪ *„OPC-UA-Funktionalitäten unterstützen“*:
	Hinweis: Verfügbarkeit und Bearbeitbarkeit dieser Option sind geräteabhängig.
	☑: Beim Download der Symbolkonfiguration werden zusätzliche Daten auf die Steuerung geladen, die für den Betrieb des OPC UA Servers notwendig sind. Dies beinhaltet derzeit die folgenden Informationen:
	  ▪ Basistypen von abgeleiteten Funktionsbausteinen
	  ▪ Inhalte von Attributen, die über Compiler-Pragmas vergeben werden
	  ▪ Gültigkeitsbereiche beispielsweise VAR_INPUT, VAR_OUTPUT, VAR_IN_OUT
	▪ <!> *„Kommentare in XML einbinden“*
	☑: Exportiert die den Variablen zugewiesenen Kommentare mit in die

Symboldatei.

- *„Node-Flags in XML einbinden“*
  ☑: Die Namensraum-Node-Flags stellen zusätzliche Informationen über den Ursprung eines Knotens im Namensraum bereit. Die Node-Flags sind immer in der Symboltabelle, wenn OPC UA aktiviert ist. Ihre Einbindung in der XML-Datei kann jedoch deaktiviert werden, da einige fehlerhafte Parser Probleme damit haben.
- *„Kommentare und Attribute konfigurieren“*
  Öffnet den Dialog *„Kommentare und Attribute“*. Darin konfigurieren Sie im Detail, was bezüglich Kommentaren und Attributen in der Symbolkonfiguration und der XML-Datei enthalten sein soll.
- *„Synchronisierung mit IEC-Tasks konfigurieren“*:
  Öffnet den Dialog *„Eigenschaften - <device name>“*, Registerkarte *„Optionen“*.
  Diese Einstellung erlaubt den symbolischen Clients (beispielsweise Visualisierungen oder Datenbankanbindungen auf Basis des PLCHandlers) den konsistenten, mit den IEC-Tasks synchronisierten Lese- bzw. Schreibzugriff. Eine genaue Beschreibung dieser Einstellung finden Sie weiter unten im Abschnitt "Einstellung: Synchronisierung mit IEC-Tasks konfigurieren"
  Hinweis: Der Variablenzugriff, der synchron zu den IEC-Tasks ist, kann den Jitter für alle IEC-Applikationen auf diesem Gerät erhöhen! Der synchronisierte konsistente Zugriff kann die Echtzeitfähigkeit stören.
- Auswahlliste zum Festlegen des Datenlayouttyps für den Client der Symbolkonfiguration:
  Hinweis: Sehen Sie dazu auch den Abschnitt "Beispiele zu den Datenlayouttypen" am Ende dieser Hilfeseite.
  - *„Optimiertes Layout“*: Empfohlen für neue Projekte. Berechnet das Ausgangslayout in der optimierten Form entkoppelt vom internen Compilerlayout. Hinterlässt keine Lücken für nicht veröffentlichte Elemente und hält die Anforderungen an die Ausrichtung der Datentypen strikt ein. Erfordert mindestens Compilerversion 3.5.7.0.
  - *„Kompatibilitätslayout“*: Diese Einstellung dient der Kompatibilität im Fall von alten Projekten. Das für den Client erzeugte Datenlayout wird soweit als möglich dem intern vom Compiler erzeugten Layout angeglichen. Aufgrund der historisch gewachsenen Konfigurationsmöglichkeiten der Symbolkonfiguration kann es jedoch zu problematischen Verschiebungen kommen.
    Ursachen von Verschiebungen
    Speicherlücken, die durch interne Pointer oder Referenzen in Funktionsbausteinen und durch Strukturkomponenten entstehen, die nicht für die Symbolkonfiguration freigegeben werden.

Speicherlücken, die abhängig vom Datentyp wie __XINT / __XWORD in 32-Bit- und 64-Bit-Systemen unterschiedlich auftreten.
Felder, die auf ungeraden Adressen liegen. Manche Clients sind darauf nicht eingerichtet.
Unabsichtlich falsche Speicherausrichtung , die bei Verwenden der Attribute 'pack_mode' oder 'relative_offset' entsteht.

- *„Standardmäßig leere Namensräume verwenden (V2-kompatibel)“*: Wird benötigt bei der Verwendung einer PLC Engineering V2-kompatiblen OPC-Server-Konfiguration.
  ✅: Verhalten wie in PLC Engineering V2.3.
  - Programmvariablen werden ohne Applikationsname exportiert (Application.PLC_PRG.MyVar --> PLC_PRG.MyVar
  - Globale Variablen werden zusätzlich ohne Namen der GVL exportiert (Application.GVL.MyGlobVar --> .MyGlobVar
- *„Direkten E/A-Zugriff aktivieren“*: WARNUNG: Diese Funktionalität ist möglicherweise gefährlich und **nicht für den Produktionsbetrieb** vorgesehen. Aktivieren Sie die Einstellung nur zur Fehlersuche, für Tests, oder während der Inbetriebnahme der Maschine (beispielsweise für das Überprüfen von Kabeln und Verbindungen).
  ✅: Sie können in der Symbolkonfiguration auch Zugriffe auf direkte E/A-Adressen verwenden, die der IEC-Syntax entsprechen (beispielsweise "%IX0.0"). Der Zugriff auf Eingangsadressen (I) ist nur lesend möglich*, der Zugriff auf Ausgangsadressen (Q) und Speicheradressen (M) kann lesend und schreibend sein.
  *Information: Im Simulationsbetrieb ist auch schreibender Zugriff auf Symbole für Eingangsadressen möglich!
  Da externe Clients für Protokolle wie OPC oder OPC UA die IEC-Syntax für direkte Adressen nicht immer unterstützen, wird zusätzlich der Zugriff über eine Arraysyntax im Namensraum __MIO des impliziten Codes ermöglicht. Beispielsweise können Sie anstelle von %IX2.3 auch auf __MIO.MIO_IX[2].x3 zugreifen.
  Die Symbole für den Arrayzugriff sind beim Browsen allerdings versteckt, da einige Clients die große Anzahl der Knoten (je nach Größe der E/A-Bereiche mehrere hunderttausend) nicht handhaben können.
- *„Aufrufe von Funktionen, FBs, Methoden und Programmen unterstützen“*: Hinweis: Verfügbarkeit und Bearbeitbarkeit dieser Option sind geräteabhängig.
  ✅: Für Symbole von Bausteinen des Typs Funktion, Funktionsbaustein, Methode oder Programm kann in der Symboltabelle das Zugriffsrecht *„ausführen“* gesetzt werden. Dazu muss zusätzlich die Option *„OPC UA Funktionalitäten unterstützen“* in den *„Einstellungen“* aktiviert sein.

	▪ *„Aufrufinformation in XML einbinden“*: ☑: Die Informationen zu aufgerufenen Funktionen, Funktionsbausteinen, Methoden oder Programmen erscheinen auch in der XML-Datei der Symbolkonfiguration. Die Option ist nur aktivierbar, wenn die Option *„Aufrufe von Funktionen, FBs, Methoden und Programmen unterstützen“* vom Gerät unterstützt wird. ▪ *„Symbolgruppen aktivieren“*: ☑: Oberhalb der Symboltabelle erscheint eine Leiste mit Schaltflächen und einem Auswahlfeld. Damit können Sie Symbolgruppen für eine clientspezifische Zugriffsrechtevergabe auf der Steuerung konfigurieren. Sehen Sie dazu unten: "Werkzeugleiste zur Symbolgruppenkonfiguration".
„Download“	Wenn Sie ein Gerät verwenden, das eine eigene Applikationsdatei für die Symbolkonfiguration unterstützt, gibt es in der Symbolleiste zusätzlich diese Schaltfläche. Wenn Sie die Symbolkonfiguration im Onlinebetrieb geändert haben, können Sie damit sofort die neue <application name>._Symbols-Datei auf die Steuerung laden.
„Tools“	*„XSD-Schema-Datei speichern“*: Dieser Befehl öffnet den Standarddialog zum Speichern einer Datei im Dateisystem. Sie können damit das XSD-Schema der Symboldatei beispielsweise zur Verwendung in externen Programmen bereitstellen.

Symboltabelle

„Zugriffsrechte“	Sie können das Zugriffsrecht für ein Symbol durch einen Mausklick auf das Symbol in der Spalte *„Zugriffsrechte“* verändern. **Icons für Zugriffsrechte (in aufsteigender Reihenfolge)** ▪ : nur lesen ▪ : nur schreiben ▪ : lesen und schreiben ▪ : ausführen Dieses Recht ermöglicht ausführenden Zugriff auf Funktionen, Funktionsbausteine, Methoden und Programme. Voraussetzungen für die Zuweisung: Das Gerät bietet die Optionen *„Aufrufe von Funktionen, FBs, Methoden und Programmen unterstützen“* und *„OPC UA Funktionalitäten unterstützen“*. Beide Optionen sind in den *„Einstellungen“* aktiviert. Hinweis: Falls die Steuerung eine Benutzerverwaltung hat, können Sie mit Hilfe von Symbolgruppen ein client-spezifisches Zugriffsrecht auf dieselben Symbole definieren.
„Maximal“	Zugriffsrecht, das Sie für dieses Symbol maximal vergeben können.
„Attribut“	Wenn das Zugriffsrecht per Attribut zugewiesen wurde, wird hier ist ein entsprechendes Icon angezeigt.
„Typ“	Ab PLC Engineering V3.5 SP6 werden auch Alias-Datentypen angezeigt. Beispiel: MY_INT : INT für eine Variable, die mit Datentyp MY_INT (Typ INT) deklariert wurde.
„Member-Variablen“	Sie können Variablen eines strukturierten Datentyps ebenfalls durch Aktivieren in der Spalte *„Symbole“* zur Symbolkonfiguration hinzufügen. Dies bewirkt zunächst, dass PLC Engineering für alle "Member"-Variablen Symbole exportiert. Über die Schaltfläche [...] in der Spalte *„Member-Variablen“* können Sie jedoch gezielt nur bestimmte Komponenten der Struktur auswählen. Hinweis: Diese Auswahl gilt dann für alle Instanzen dieses Datentyps, für die Symbole exportiert werden! Wenn kein Member eines strukturierten Typs ausgewählt ist, erscheint in den Checkboxen der Members ein Sternchen ( ), um zu zeigen, dass alle exportierbaren Members des Typs exportiert werden.

Werkzeugleiste zur Symbolgruppenkonfiguration

„Auswahlliste“	Bereits definierte Symbolgruppen
„Hinzufügen einer neuen Symbolgruppe“	Öffnet den Dialog *„Hinzufügen einer neuen Symbolgruppe“* zur Eingabe eines Namens für diese Gruppe
„ Hinzufügen einer Kopie der ausgewählten Symbolgruppe“	Öffnet den Dialog *„Hinzufügen einer Kopie der ausgewählten Symbolgruppe“*. Für die in der Auswahlliste gewählte Gruppe wird eine Kopie angelegt. Sie können den Standardnamen (<group name>_Duplizieren) ändern.
„Umbenennen der ausgewählten Symbolgruppe“	Öffnet den Dialog *„Umbenennen der ausgewählten Symbolgruppe“* zur Eingabe eines anderen Namens für die in der Auswahlliste gewählte Gruppe
„Löschen der ausgewählten Symbolgruppe“	Öffnet eine Dialogbox mit der Nachfrage, ob die in der Auswahlliste gewählte Symbolgruppe gelöscht werden soll
„Symbolrgruppenrechte konfigurieren“	Öffnet die Registerkarte *„Symbolrechte“* des Geräteeditors. Im eingeloggten Zustand können Sie dort der in der Auswahlliste gewählten Symbolgruppe pro Benutzergruppe (Client) unterschiedliche Zugriffsrechte zuweisen.

Siehe auch

- ↘ „Registerkarte 'Symbolrechte'“

Dialog 'Kommentare und Attribute'

„Inhalt Symboltabelle“

„Erweiterte OPC UA-Informationen aktivieren“	Hinweis: Verfügbarkeit und Bearbeitbarkeit dieser Option sind geräteabhängig. ☑: In die Symboltabelle werden erweiterte Informationen aufgenommen, die von OPC UA Servern ausgewertet werden können. Hierzu gehören beispielsweise die Vererbungsinformationen von nutzerdefinierten Datentypen, und die Namensraum-Node-Flags. Weitere Informationen, wie beispielsweise Kommentare und Attribute, können ebenfalls aufgenommen werden, wenn die OPC UA Einstellung aktiv ist. Bei aktivierter OPC UA-Einstellung werden Attribute gemäß folgender Regel mit in die Symboltabellen aufgenommen: ▪ Mit den Compilerversionen V3.5.5.0 bis V3.5.7.X werden immer alle Attribute entsprechend der Einstellung *„Einfache Namen auswählen“* aufgenommen. ▪ In der Compilerversion V3.5.8.X werden immer alle Attribute entsprechend der Einstellung *„Alle Attribute einbinden“* aufgenommen. ▪ Ab der Compilerversion V3.5.9.0 können Sie in diesem Dialog selbst einstellen, ob und welche Attribute aufgenommen werden.
„Kommentare einbinden“ *„Attribute einbinden“*	Voraussetzung: *„Erweiterte OPC UA-Informationen aktivieren“* ist aktiviert. ☑: Kommentare und Attribute werden mit in der Symboltabelle gespeichert.
„Auch Kommentare und Attribute für Type-Nodes einbinden“	Voraussetzung: *„Kommentare einbinden“* ist aktiviert. ☑: Die Information für Typknoten (benutzerdefinierte Typen wie STRUCT und ENUM-Elemente) werden auch eingebunden. ☐: Nur direkt exportierte Variablen verfügen über Kommentare und Attribute.

„Inhalt XML-Symboldatei“

„Namensraum-Node-Flags einbinden“	☑: Die Namensraum-Node-Flags stellen zusätzliche Informationen über den Ursprung eines Knotens im Namensraum bereit. Die Node-Flags sind immer in der Symboltabelle, wenn OPC UA aktiviert ist. Ihre Einbindung in der XML-Datei kann jedoch deaktiviert werden, da einige fehlerhafte Parser Probleme damit haben.

„Kommentare einbinden“	☑: Kommentare können mit in die XML-Datei gespeichert werden. Für Compilerversions V3.5.5.x bis V3.5.8.0 beinhaltet dies die Einstellung *„Doku-Kommentare bevorzugen“*.
„Attribute einbinden“	☑: Attribute können mit in die Symboldatei gespeichert werden.
„Auch Kommentare und Attribute für Type-Nodes einbinden“	Voraussetzung: *„Kommentare einbinden“* ist aktiviert. ☑: Die Information für Typknoten (benutzerdefinierte Typen wie STRUCT und ENUM-Elemente) werden auch eingebunden. ☐: Nur direkt exportierte Variablen verfügen über Kommentare und Attribute.

„Kommentare auswählen“

Voraussetzung: *„Kommentare einbinden“* ist aktiviert

„Doku-Kommentare einbinden“ *„Normale Kommentare “* *„Immer beide Typen von Kommentaren einbinden“* *„Doku-Kommentare bevorzugen, ansonsten normale Kommentare“* *„Normale Kommentare bevorzugen, ansonsten Doku-Kommentare“*	Die Optionen legen fest, welche Kommentare in die Symbolkonfiguration gespeichert werden.

„Attribute filtern (Groß-/Kleinschreibung egal)“

Voraussetzung: *„Attribute einbinden“* ist aktiviert

„Alle Attribute einbinden“ *„Attribute einbinden, die starten mit“* *„Attribute mit regulären Ausdrücken filtern“*	Legt fest, welche Attribute in die Symbolkonfiguration gespeichert werden.
„Einfache Namen auswählen“	Besteht hauptsächlich aufgrund der Rückwärtskompatibilität zu älteren Versionen, um das alte Verhalten nachzubilden.

Einstellung: Synchronisierung mit IEC-Tasks konfigurieren

Für den synchronen konsistenten Zugriff wird im Laufzeitsystem bei der Bearbeitung einer Lese- oder Schreibanfrage vom symbolischen Client zunächst gewartet, bis ein Zeitpunkt gefunden wird, in dem keine IEC-Task ausgeführt wird. Sobald diese Lücke gefunden wird, wird der Neuanlauf der IEC-Tasks solange verhindert, bis alle Werte der Variablenliste kopiert wurden. Danach werden die IEC-Tasks wieder wie üblich geplant. Durch den synchronisierten konsistenten Zugriff kann es dadurch zu einem verzögerten Start von IEC-Tasks kommen, was sich als erhöhter Jitter zeigt. Da im Laufzeitsystem alle Applikationen von einem gemeinsamen Scheduler verwaltet werden, betrifft diese potentielle Verschlechterung des Echtzeitverhaltens alle Applikationen auf dem Gerät. Es sind also alle Applikationen des Geräts betroffen, unabhängig davon, ob diese eine Symbolkonfiguration enthalten oder ob diese aus einem oder mehreren PLC Engineering-Projekten auf die Steuerung geladen wurden. Daher lässt das Laufzeitsystem den synchronisierten konsistenten Zugriff nur zu, wenn dies alle Applikationen erlauben, die zum Zugriffszeitpunkt in der Steuerung geladen sind.

Die Einstellung finden Sie im Editor der Symbolkonfiguration im Menü *„Einstellungen“*. Außerdem finden Sie die Einstellung auch im Kontextmenü der Steuerung, wenn Sie den Befehl *„Eigenschaften “* und dann in dem sich öffnenden Dialog die Registerkarte *„Optionen“* wählen.

Bei Applikationen ohne Symbolkonfiguration finden Sie die Einstellung nur im Eigenschaftendialog.

HINWEIS!

Nach Ändern der Einstellung müssen alle Applikationen auf dem Gerät per Download/Online-Change neu geladen werden und alle Bootapplikationen aktualisiert werden.

In welchen Fällen wird ein synchronisierter konsistenter Zugriff benötigt?

In der Regel besteht bei visualisierten Werten kein Bedarf für konsistente Werte, da es bei sich ändernden Werten meist nicht relevant ist, aus welchem IEC-Task-Zyklus diese stammen. Bei sich selten ändernden Werten sowieso nicht. Auch beim Schreiben bestehen meist keine harten Konsistenzanforderungen, da sich die Maschine, beispielsweise beim

Schreiben von Rezepturen, typischerweise in einer Art von Standby-Modus befinden muss, in dem kein direkter Zugriff auf die als Rezeptur geschriebenen Werte erfolgt.

Dagegen werden insbesondere für Datenbankanbindungen zur Speicherung von Produktionsdaten konsistente Werte benötigt. Bei getakteten Maschinen müssen diese Werte aber synchron zum Produktionstakt (also pro hergestelltem Produkt ein Wertesatz) und nicht konsistent in Bezug auf eine oder mehrere IEC-Tasks sein. Die Konsistenz in Bezug auf den Maschinentakt muss aber bereits durch die IEC-Applikation gewährleistet sein. Hierfür werden typischerweise die während eines Produktionstakts anfallenden Werte in einer globalen Variablenliste gesammelt. Am Ende des Takts wird mit einer weiteren Variablen (BOOL oder Zähler) dem symbolischen Client mitgeteilt, dass der Maschinentakt beendet und somit die Werte gültig sind. Nun hat der Client die Chance die zum Produktionstakt gehörenden Werte zur archivieren. Je nach Notwendigkeit kann das erfolgreiche Lesen auch in der Gegenrichtung mittels einer freigegebenen Variablen angezeigt werden, so dass gegebenenfalls die Produktion angehalten werden kann, wenn die Produktionsdaten nicht archiviert werden konnten. Für diesen Anwendungsfall ist der synchronisierte konsistente Zugriff nicht notwendig und hilfreich, da die Synchronisierung ja auf Applikationsebene zu erfolgen hat.

Im Gegensatz dazu liegt der typische Einsatz des synchronisierten konsistenten Zugriffs durch symbolische Clients vor allem in der Prozessindustrie mit kontinuierlich laufenden Anlagen ohne Produktionstaktung, wenn beispielsweise zyklisch in einem festen Zeitraster von 60 s Prozesswerte konsistent geschrieben werden. Dies kann entweder über die Synchronisierung auf Applikationsebene analog zu den getakteten Maschinen (siehe oben), oder über die Synchronisierung des synchronisierten konsistenten symbolischen Zugriffs erfolgen. Letzteres hat den Vorteil, dass keine Logik im IEC-Programm implementiert werden muss und der Zugriff allein durch den Client gesteuert wird.

VORSICHT!

Aufgrund des erhöhten Jitters ist das synchronisierte konsistente Monitoring nicht für Motion oder echtzeitkritische Anwendungen geeignet. Aus diesen Gründen sollte der synchronisierte konsistente Zugriff nur freigegeben und verwendet werden, wenn dies unbedingt erforderlich ist.

Wenn ein Client den durch diese Einstellung freigegebenen synchronen konsistenten Zugriff verwendet, hat dies auch Auswirkungen auf den Client. Zum einen kann hier je nach Scheduler des Laufzeitsystems die Antwortzeit auf Lese-/Schreibzugriffe mehr jittern, da gegebenenfalls noch auf eine Ausführungslücke der IEC-Tasks gewartet werden muss. Weiterhin können Lese- oder/und Schreibzugriffe scheitern, wenn IEC-Tasks sehr lange laufen (im Bereich von mehreren 100 ms) oder die CPU-Auslastung durch eine oder mehrere IEC-Tasks über einen längeren Zeitraum (im Bereich von mehreren 100 ms) bei nahezu 100% liegt. Die Verfügbarkeit der Werte hängt damit also auch von der Auslastung der Steuerung durch die IEC-Applikation ab.

Darüber hinaus kann der Client die Auswirkungen auf sich selbst und auf das Laufzeitsystem minimieren, wenn er bei der Definition der zu lesenden/schreibenden Variablenlisten Folgendes beachtet:

- Nur auf diejenigen Variablen synchronisiert konsistent zugreifen, die unbedingt konsistent benötigt werden
- Getrennte Variablenlisten für Variablen, die konsistent sein müssen und für Variablen, die inkonsistent sein dürfen
- Variablenlisten mit sehr vielen konsistenten Variablen in mehrere kleinere Listen aufteilen
- Leseintervalle für zyklisches Lesen von Werten so groß möglich wählen.

Unterstützung bei der aktuellen Konfiguration und mögliche Korrekturmaßnahmen

Rot markierte Einträge in der Symboltabelle zeigen Variablen an, die für den Export in die Symboldatei konfiguriert sind, aber gerade in der Applikation nicht gültig sind. Ursache kann beispielsweise sein, dass die Deklaration im Baustein entfernt wurde.

Ab Version 3.5.8.0 erhalten Sie einen Warnhinweis im Editor, wenn Variablen, für die Symbole konfiguriert sind, im IEC-Code nicht verwendet oder im Falle von E/A-Variablen nicht gemappt werden. Der Compiler weist außerdem auf

Variablen hin, die in der Symbolkonfiguration aus veralteten Bibliotheksversionen referenziert werden.

HINWEIS!

Variablen von Objekten, die im Programmcode nicht verwendet werden, werden standardmäßig nicht kompiliert und sind somit nicht in der Symbolkonfiguration verfügbar.

Aber: Variablen von nicht-kompilierten Objekten werden von PLC Engineering dennoch in der Symbolkonfiguration bereitgestellt, wenn eine der folgenden Bedingungen erfüllt ist:

- Die Bausteineigenschaft *„Immer binden“* ist aktiviert.
- Das Pragmaattribut {attribute 'linkalways'} wird verwendet.

Siehe auch

- ↘ „Arbeiten mit Steuerungsnetzwerken“
- ↘ „Dialog 'Eigenschaften' - 'Optionen'“
- ↘ „Dialog 'Eigenschaften' - 'Build'“
- ↘ „Attribut 'linkalways'“
- ↘ „Auswirkungen der Pragmas auf Symbole “

Beispiele zu den Datenlayouttypen

Beispiele zu den Layouttypen

Im Folgenden sehen Sie Beispiele aus einer IEC-Applikation, bei denen es durch nicht veröffentlichte Symbole, durch interne "unsichtbare" Pointer, oder durch eine "pack mode"-Definition über die Gerätebeschreibung zu Lücken im clientseitigen Speicherlayout kommen kann. Mit der Einstellung *„Optimiertes Layout“* werden die Lücken vermieden, die Symboldatei enthält also je nach ausgewählter Layout-Einstellung unterschiedliche Angaben für Größe und Offset von Speicherstellen.

Beispiel: Große Struktur

```
// Beispiel einer großen Struktur, von der nicht alle Members veröffentlicht werden :
// Example of a big structure, where not all members get published :
STRUCT
    {attribute 'symbol':='readwrite'}
    PublicNumber : INT;

    {attribute 'symbol':='none'}
    InternalData : ARRAY[0..100] OF BYTE;

    {attribute 'symbol':='readwrite'}
    SecondNumber : INT;

    {attribute 'symbol':='none'}
    MoreData : ARRAY[0..100] OF BYTE;
END_STRUCT
END_TYPE
```

Resultierende Einträge in der Symboldatei, beachten Sie "size" und "byteoffset":

Symboldatei, Große Struktur, Option Kompatibilitätslayout

```
<TypeUserDef name="T_GrosseStruktur" size="208" nativesize="208" typeclass="Userdef" pouclass="STRUCTURE" i
ecname="GrosseStruktur">

<UserDefElement iecname="PublicNumber" type="T_INT" byteoffset="0" vartype="VAR" />

<UserDefElement iecname="SecondNumber" type="T_INT" byteoffset="104" vartype="VAR" />

</TypeUserDef>
```

Symboldatei, Große Struktur, Option Optimiertes Layout

```
<TypeUserDef name="T_GrosseStruktur" size="4" nativesize="208" typeclass="Userdef" pouclass="STRUCTURE" iec
name="GrosseStruktur">

<UserDefElement iecname="PublicNumber" type="T_INT" byteoffset="0" vartype="VAR" />

<UserDefElement iecname="SecondNumber" type="T_INT" byteoffset="2" vartype="VAR" />

</TypeUserDef>
```

Beispiel: Struktur mit ungeraden Adressen

```
// Die folgenden Mechanismen können zu Komponenten mit nicht korrekter Speicherausrichtung führen:
// The following mechanisms can cause misalignment:
// - {attribute 'relative_offset':='...'} an einem Member / at a member
// - {attribute 'pack_mode':='...'} in einer Strukturdeklaration / at a structure declaration
// - target setting 'memory-layout\pack-mode' in der Gerätebeschreibung / in the device description

{attribute 'pack_mode':='1'}
TYPE UngeradeAdressen :
STRUCT
    {attribute 'relative_offset':='3'}
    {attribute 'symbol':='readwrite'}
    PublicNumber : INT;

    {attribute 'symbol':='readwrite'}
    PublicValue : LREAL;
END_STRUCT
END_TYPE
```

Resultierende Einträge in der Symboldatei, beachten Sie "size" und "byteoffset":

Symboldatei, Struktur mit ungeraden Adressen, Option Kompatibilitätslayout

```
<TypeUserDef name="T_UngeradeAdressen" size="13" nativesize="13" typeclass="Userdef" pouclass="STRUCTURE"
iecname="UngeradeAdressen">

<UserDefElement iecname="PublicNumber" type="T_INT" byteoffset="3" vartype="VAR" />

<UserDefElement iecname="PublicValue" type="T_LREAL" byteoffset="5" vartype="VAR" />

</TypeUserDef>
```

Symboldatei, Struktur mit ungeraden Adressen, Option Optimiertes Layout

```
<TypeUserDef name="T_UngeradeAdressen" size="16" nativesize="13" typeclass="Userdef" pouclass="STRUCTURE"
iecname="UngeradeAdressen">

<UserDefElement iecname="PublicNumber" type="T_INT" byteoffset="0" vartype="VAR" />

<UserDefElement iecname="PublicValue" type="T_LREAL" byteoffset="8" vartype="VAR" />

</TypeUserDef>
```

Beispiel Funktionsbaustein

```
// Each POU contains some implicit variables, which do not get published. Depending on the data type these might cau
se memory gaps of different sizes.
FUNCTION_BLOCK Baustein IMPLEMENTS SomeInterface
VAR_INPUT
    in : INT;
END_VAR
VAR_OUTPUT
    out : INT;
END_VAR
VAR
END_VAR
```

Jeder Baustein enthält interne Variablen, die nicht veröffentlicht werden. Wenn es sich dabei um Datentypen wie beispielsweise __XWORD handelt, entstehen dadurch im clientseitigen Datenlayout unterschiedlich große Speicherlücken, je nachdem, ob auf einem 64-Bit- oder 32-Bit-System gearbeitet wird.

Resultierende Einträge in der Symboldatei für 64 Bit und 32 Bit beachten Sie "size" und "byteoffset":

Symboldatei, Funktionsbaustein, Option Kompatibilitätslayout, 64 Bit

```
<TypeUserDef name="T_Baustein" size="24" nativesize="24" typeclass="Userdef" pouclass="FUNCTION_BLOCK" iec
name="Baustein">

<UserDefElement iecname="in" type="T_INT" byteoffset="16" vartype="VAR_INPUT" />

<UserDefElement iecname="out" type="T_INT" byteoffset="18" vartype="VAR_OUTPUT" />

</TypeUserDef>
```

Symboldatei, Funktionsbaustein, Option Optimiertes Layout, 64 Bit

```
<TypeUserDef name="T_Baustein" size="4" nativesize="24" typeclass="Userdef" pouclass="FUNCTION_BLOCK" iecn
ame="Baustein">

<UserDefElement iecname="in" type="T_INT" byteoffset="0" vartype="VAR_INPUT" />

<UserDefElement iecname="out" type="T_INT" byteoffset="2" vartype="VAR_OUTPUT" />

</TypeUserDef>
```

Symboldatei, Funktionsbaustein, Option Kompatibilitätslayout, 32 Bit

```
<TypeUserDef name="T_Baustein" size="12" nativesize="12" typeclass="Userdef" pouclass="FUNCTION_BLOCK" iec
name="Baustein">

<UserDefElement iecname="in" type="T_INT" byteoffset="8" vartype="VAR_INPUT" />

<UserDefElement iecname="out" type="T_INT" byteoffset="10" vartype="VAR_OUTPUT" />

</TypeUserDef>
```

Symboldatei, Funktionsbaustein, Option Optimiertes Layout, 32 Bit

```
<TypeUserDef name="T_Baustein" size="4" nativesize="12" typeclass="Userdef" pouclass="FUNCTION_BLOCK" iecn
ame="Baustein">

<UserDefElement iecname="in" type="T_INT" byteoffset="0" vartype="VAR_INPUT" />

<UserDefElement iecname="out" type="T_INT" byteoffset="2" vartype="VAR_OUTPUT" />

</TypeUserDef>
```

Siehe auch

- ↘ „Symbolkonfiguration“