

CoDeSys

Objektorientierte SPS-Programmierung

OOP in einem IEC 61131-3 Tool



CoDeSys (Controller Development System) ist eine Programmiersoftware für Embedded und PC basierte Geräte nach dem internationalen Standard IEC 61131-3. Tausende von Endanwendern setzen das Software-Tool täglich zur Steuerungsprogrammierung im Maschinen- und Anlagenbau ein.

Durch eine Erweiterung der IEC 61131-3 um neue Sprachmittel kann der Anwender seine CoDeSys-Applikation objektorientiert in den Sprachen der Norm programmieren. Dabei ist die objektorientierte Programmierung (OOP) eine Option, d.h. der Anwender kann weiterhin funktional programmieren oder funktionale und objektorientierte Programmierung kombinieren.

Begriffsdefinitionen der objektorientierten Programmierung in CoDeSys

- Ein Funktionsbaustein ist eine **Klasse** mit genau einer Methode. Mit der Erweiterung auf volle Klassenfunktionalität können in einem Funktionsbaustein Methoden oder Interfaces mit deren Methoden implementiert werden.
- **Methoden** sind Routinen, die einem Funktionsbaustein bzw. einem Interface fest zugeordnet sind. Sie arbeiten auf den Daten des Funktionsbausteins, können aber wie IEC-Funktionen über Eingangs-, Ausgangs- und lokale Variablen verfügen.
- Ein **Interface** ist eine Schnittstellendefinition mit einem Satz an Methoden. Im Interface werden dabei lediglich die erforderlichen Variablen der Methoden festgelegt. Der Rumpf der Methode wird erst in der Klasse ausprogrammiert, in die das Interface implementiert wird.
- **Objekte** sind Instanzen von Funktionsbausteinen (Klassen).
- Mit dem neuen Schlüsselwort **IMPLEMENTS** implementiert ein Funktionsbaustein ein Interface. Im Funktionsbaustein müssen somit alle Methoden des Interfaces realisiert werden.
- Ein Funktionsbaustein kann eine Klasse durch das neue Schlüsselwort **EXTENDS** erweitern. Damit übernimmt der Funktionsbaustein alle Daten und Methoden der Klasse.
- Beim Aufruf von Methoden über deren Interface wird erst zur Laufzeit der Applikation entschieden, welche Implementierung der Methode tatsächlich ausgeführt wird. Diese Funktionalität nennt sich **Polymorphie**.

Typischen Anwendungsszenarien in der Automatisierungstechnik

- Programmierung von unterschiedlichen Antrieben, die aber über eine identische Funktionalität verfügen (z.B. Homing, Positionieren, Fehler auslesen) mit Interfaces und Methoden
- Programmierung unterschiedlicher Maschinenmodule mit einheitlicher oder ähnlicher Funktionalität (z.B. Handbetrieb, Automatikbetrieb, Referenzfahrt) mit Interfaces und Methoden
- Datenzugriff / Bedienung unterschiedlicher Feldbusse (z.B. Starten, Stoppen, asynchrone Nachricht senden) mit Interfaces und Methoden

Weitere Funktionen

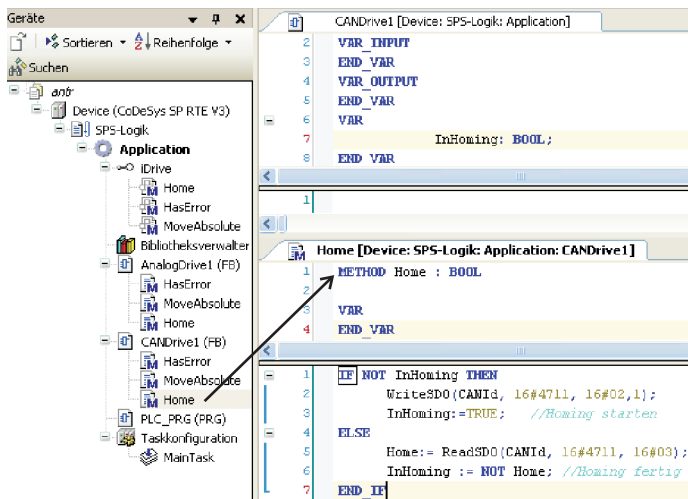
- Unterstützung von Klassenhierarchien zur Zusammenfassung von Klassen
- Constructor- und Destructor-Methoden *FB_Init* und *FB_Exit* zum Initialisieren bzw. Beenden von Objekten
- OOP ist nicht an bestimmte Programmiersprachen der IEC 61131-3 gebunden. Methoden und Klassen können in allen IEC-Editoren erstellt werden.
- Unterstützung von *Eigenschaften* für Klassen
- Die Schlüsselwörter *SUPER* und *THIS* ermöglichen den Zugriff von einer Methode auf den Vater der Methode bzw. auf die Methode selbst.
- Derzeit bewusst weggelassen: dynamische Speicherverwaltung sowie Spezialkonstrukte aus Hochsprachen (wie z.B. "Private", "protected", "internal", "public", "abstract" oder "virtual")



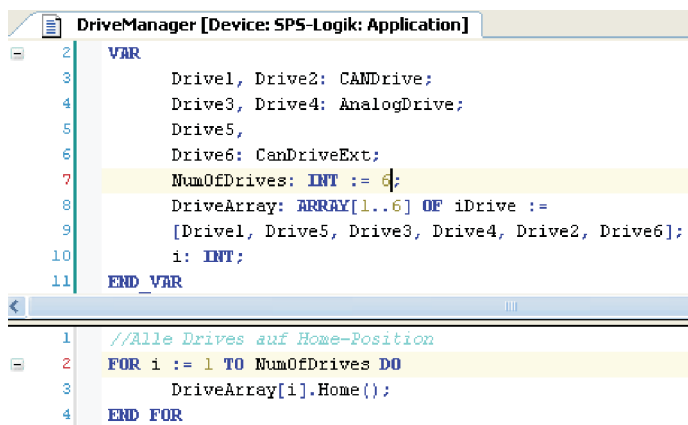
Beispiel

Unterschiedliche Antriebe (CAN-Antriebe, CAN-Antriebe mit erweiterter Funktionalität, Analog-Antriebe) werden in einer Steuerungsapplikation programmiert. **Alle Antriebe verfügen über die Funktionen Home, MoveAbsolute und HasError.**

Aufgrund dieser einheitlichen Funktionalität wird im CoDeSys-Projekt ein **Interface** mit den entsprechenden **Methoden** angelegt. Die Methoden enthalten im Interface keinen Code, sondern lediglich ihre Aufruf- und Rückgabevariablen.

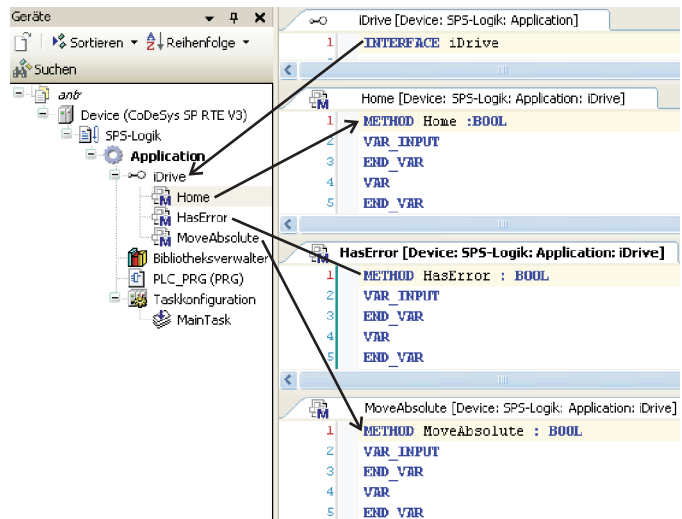


Der Funktionsbaustein *CANDriveExt* für den Antrieb mit erweiterter Funktionalität erweitert die Klasse *CANDrive1*. Ihm werden automatisch alle Methoden der Basisklasse *CANDrive1* vererbt. Die Methode *Home* wird neu angelegt und überschreibt damit die geerbte Methode.

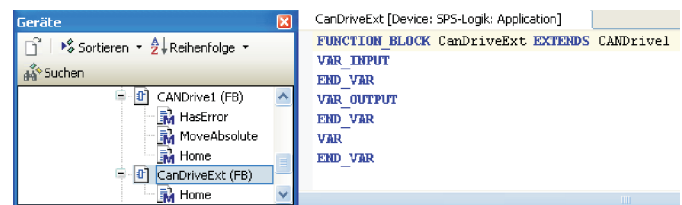


Vorteile der objektorientierten Programmierung

- Der erzeugte Programmcode ist leichter zu verändern bzw. zu erweitern und damit besser wartbar
- Die Wiederverwendung von Code ist wesentlich einfacher und die Kapselung deutlich verbessert
- Die Performance der Steuerungsapplikation ist verbessert, außer bei sehr kleinen Applikationen



Die neuen Funktionsbausteine *CANDrive1* und *AnalogDrive1* implementieren das Interface *iDrive* und erhalten damit sofort die Methoden des implementierten Interfaces. In diesen Methoden wird nun der geräte-spezifische Aufruf programmiert.



Im übergeordneten Programm *DriveManager* werden alle Antriebe instanziiert und damit zu den Objekten *Drive1*, *Drive2* etc. Damit alle Antriebe bequem in einer Schleife z.B. auf Home-Position gefahren werden können, werden sie in ein Array abgelegt. Der Datentyp des Arrays ist dabei das Interface *iDrive*. Beim Aufruf der Methode *Home* in der Schleife weiß CoDeSys nur durch die Zuordnung der Objekte, um welche Methode es sich handelt.

Gibt es Veränderungen in der Applikation, weil z.B. *Drive2* nun kein *CANDrive* mehr ist, sondern ein *AnalogDrive*, oder nunmehr zwei Antriebe zusätzlich verfügbar sind, so muss lediglich die Instanziierung im Deklarationsteil des *DriveManager* erfolgen.

- Objektorientierte Programmierung ist heute gängige Lehrmethode in der Hochsprachenprogrammierung (in der Ausbildung von Steuerungsprogrammierern hat sie sich bislang noch nicht etabliert)
- In der Office-Welt hat sich die objektorientierte Programmierung millionenfach bewährt