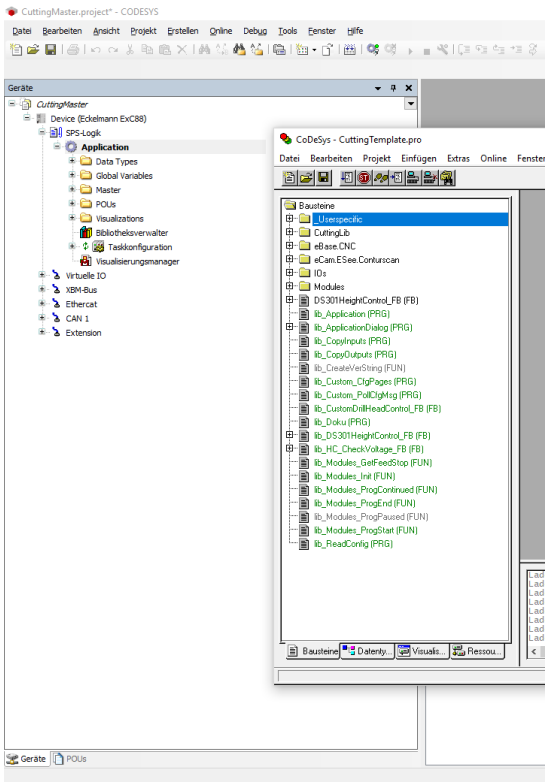
White Paper der Eckelmann AG, Wiesbaden, März 2019

Einfach umsteigen auf CODESYS V3

Der Umstieg auf CODESYS V3 ist leichter als gedacht. Die Entwicklungsumgebung bietet einem zwar auch die Möglichkeiten der objektorientierten Programmierung (OPP), dies ist jedoch kein Muss. Denn das objektorientierte Programmieren ist keine gesonderte Welt. Ich kann die neuen Möglichkeiten im gleichen Kontext gemeinsam mit der der herkömmlichen Programmierung nutzen. Anders gesagt: OPP setzt quasi auf der klassischen Programmierung auf und erweitert diese.

Das bedeutet: Man kann ein CODESYS V2 Projekt in CODESYS V3 importieren, wie gewohnt weiterentwickeln und nur dort, wo der Bedarf oder Wunsch besteht, zusätzlich auch objektorientiert programmieren. Welche Vorteile dies bringt, zeigen wir am Ende dieses White Papers anhand eines Beispiels.

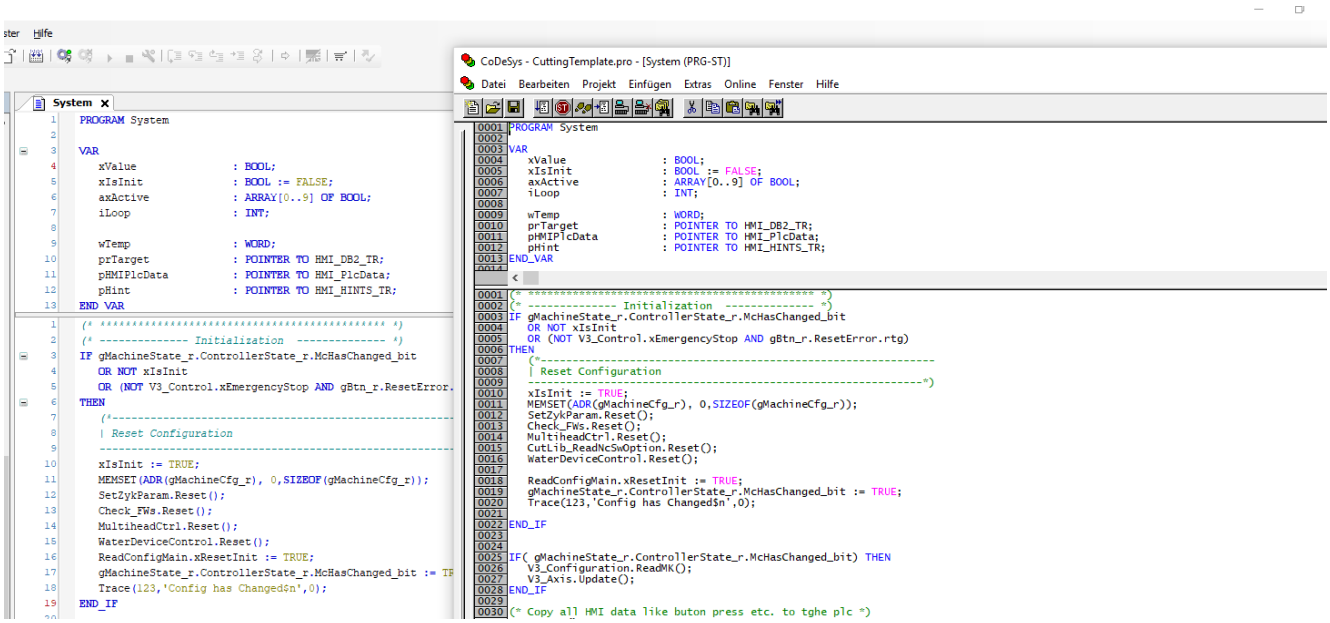
Die wahrscheinlich größere Hürde besteht für viele Anwender eher in der auf den ersten Blick etwas ungewohnten, neuen Umgebung. Wo finden sich die bekannten Menüpunkte und Funktionen? Auf die Bereiche „Bausteine“, „Datentypen“, „Visualisierung“ und „Ressourcen“, haben Sie jetzt über einen einzigen Verzeichnisbaum Zugriff, statt bisher über mehrere Reiter. Bausteine und Datentypen lassen sich nun in gemeinsamen Ordnern definieren, und in dieser Art gibt es noch einige Änderungen. Zunächst ist dies ungewohnt, aber schon nach kurzer Einarbeitung hat man sich daran gewöhnt und findet Gefallen an den Vorteilen der klaren Struktur.



Vom „Code-Editor“ zur modernen IDE für die SPS-Programmierung

Auch CODESYS V3 besteht im Wesentlichen aus den Grundbausteinen „Programm“, „Funktion“ und „Funktionsblock“ und die Programmierung erfolgt identisch wie aus V2 gewohnt, wie der direkte Vergleich der grafischen Benutzeroberflächen zeigt. Der Editor bietet Ihnen jedoch viele nützliche Funktionen, die man von einer modernen Entwicklungsumgebung heute erwartet und den Editor von CODESYS V2 im wahrsten Sinne des Wortes „alt“ aussehen lassen.

Entwickeln in einer Umgebung mit modernem Look & Feel, fast wie man es von Integrated Development Environments (IDEs) wie Visual Studio kennt, und viele verbesserte Funktionen machen die wachsende Komplexität von SPS-Applikationen leichter beherrschbar. Mit CODESYS V3 ist die SPS-Programmierung vollständig in der PC-Welt angekommen, 3S hat dazu die klassische SPS-Entwicklung gründlich modernisiert und aufgeräumt.



Im Code-Editor von CODESYS V3 lassen sich über das Kontextmenü sehr einfach sämtliche Funktionsaufrufe (Symbol suchen > Querverweise) oder auch Variablen-Definitionen finden (Symbol suchen > Gehe zu Definition). Sucht man nach den Aufrufen einer Funktion, erhält man eine übersichtliche Liste aller Bausteine und kann direkt zu den entsprechenden Bausteinen springen. Diese Funktion stellt eine enorme Vereinfachung gegenüber der textbasierten Suche in CODESYS V2 dar. Die erweiterten, intelligenten Suchfunktionen erleichtern die Fehlersuche und Orientierung enorm.1

Systemvariablen durch Funktionsaufrufe ersetzen

Bei der Eckelmann CNC wurde bei der Weiterentwicklung auf größtmögliche Kompatibilität zwischen den Interfaces zu CODESYS V2 und CODESYS V3 geachtet. Allerdings wird bei

1 Vgl. für weitere Beispiele: Matthias Gehring: Schnelle Orientierung im CODESYS Projekt. CODESYS Blog, 15.04.2018, abgerufen am 04.03.2019.

der Kommunikation zwischen SPS und Firmware in V3 nicht mehr auf die bekannten Systemvariablen zurückgegriffen, sondern erfolgt über Funktionsaufrufe. Die Nutzung von Funktionen statt Systemvariablen ist generell weniger anfällig für Fehler, so wird das Debugging deutlich vereinfacht und beschleunigt.

Für die einfache Migration bietet die Eckelmann AG seinen Kunden aber auch eine V2-Bibliothek an, welche diese Systemvariablen für das SPS-Programm abkapselt. So kann ein V2 Projekt in V2 vorab für die Verwendung in V3 vorbereitet werden. Durch den Austausch der „Systemvariablen“ Bibliothek ist das Projekt damit sofort lauffähig in V3.

Welche minimalen Code-Anpassungen dazu nötig sind, zeigen die folgenden Screenshots im Vergleich mit Systemvariablen oder Funktionsaufrufen. Je nach Größe des Projektes erfordert diese Umstellung erfahrungsgemäß nur wenigen Stunden.

Früher mit Systemvariable: Verwendung der Systemvariablen %MWxx.

```
IF ( (DB2_MMI2SPS_MSG_CNT2_W <> DB2_SPS2MMI_QUIT_CNT_W) AND
   (DB2_MMI2SPS_MSG_CNT1_W = DB2_MMI2SPS_MSG_CNT2_W) )
THEN
   IF (stabil_bit = TRUE) THEN
      MEMCOPY(ADR(msg_r.auftrag_w), ADR(DB2_MMI2SPS_MSG_AUFTRAG_W), SIZEOF(msg_r));
      DB2_SPS2MMI_QUIT_CNT_W := DB2_MMI2SPS_MSG_CNT2_W;
      stabil_bit := FALSE;
      msg_ok_bit := TRUE;         (* Nachricht verfügbar *)
   ELSE
      stabil_bit := TRUE;
      msg_ok_bit := FALSE;     (* keine Nachricht vorhanden *)
   END_IF;
ELSE
   stabil_bit := FALSE;
   msg_ok_bit := FALSE;        (* keine Nachricht vorhanden *)
END_IF;
```

Systemvariablen als Funktionsaufrufe mit neuer „Systemvariablen“ Bibliothek: Systemvariablen ersetzt durch den V3_HMI-Baustein

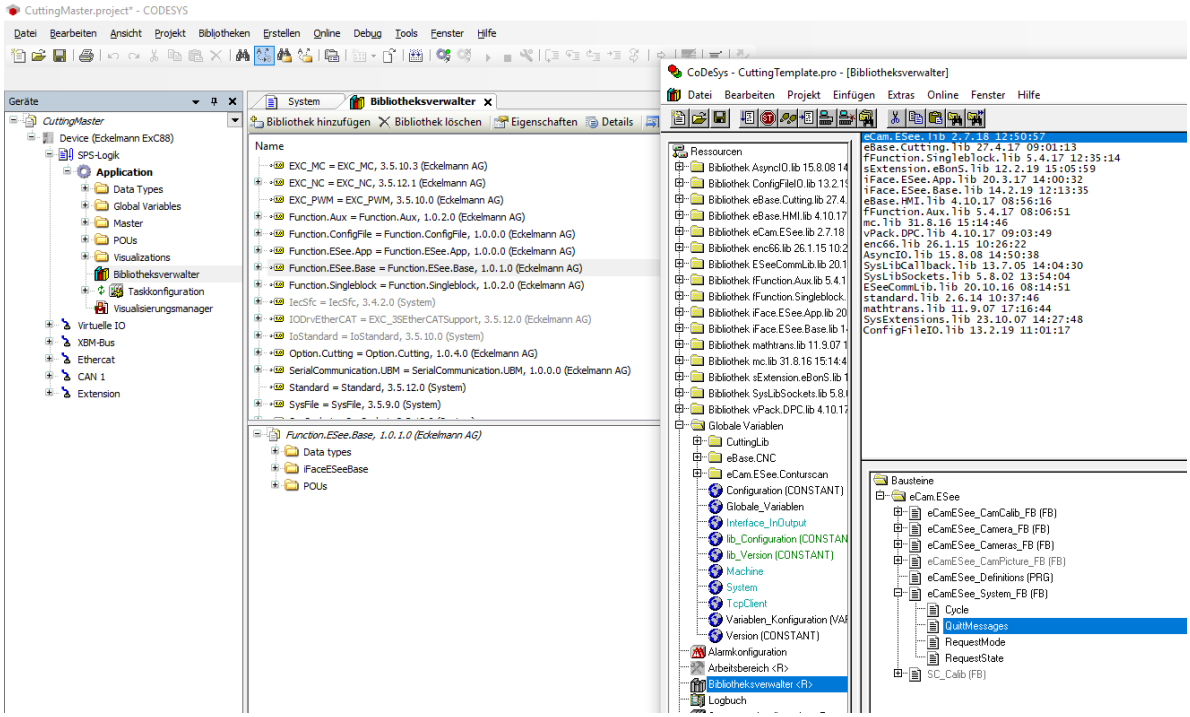
```
IF ( (V3_HMI.PlcMessages.uiHmiMsgCnt2 <> V3_HMI.uiPlcMessagegQuit) AND
      (V3_HMI.PlcMessages.uiHmiMsgCnt1 = V3_HMI.PlcMessages.uiHmiMsgCnt2) )
THEN
   IF (stabil_bit = TRUE) THEN
      MEMCOPY(ADR(msg_st.auftrag_w), V3_HMI.PlcMessages.pData, SIZEOF(msg_st));

      adiTraceData[0] := WORD_TO_INT( msg_st.auftrag_w);
      Trace( 111, 'Receive message: SBO:x%x', ADR(adiTraceData));

      V3_HMI.uiPlcMessagegQuit := V3_HMI.PlcMessages.uiHmiMsgCnt2;
      stabil_bit := FALSE;
      msg_ok_bit := TRUE;         (* Nachricht verfügbar *)
   ELSE
      stabil_bit := TRUE;
      msg_ok_bit := FALSE;        (* keine Nachricht vorhanden *)
   END_IF
ELSE
   stabil_bit := FALSE;
   msg_ok_bit := FALSE;           (* keine Nachricht vorhanden *)
END_IF
```

Modularität und Bibliotheken

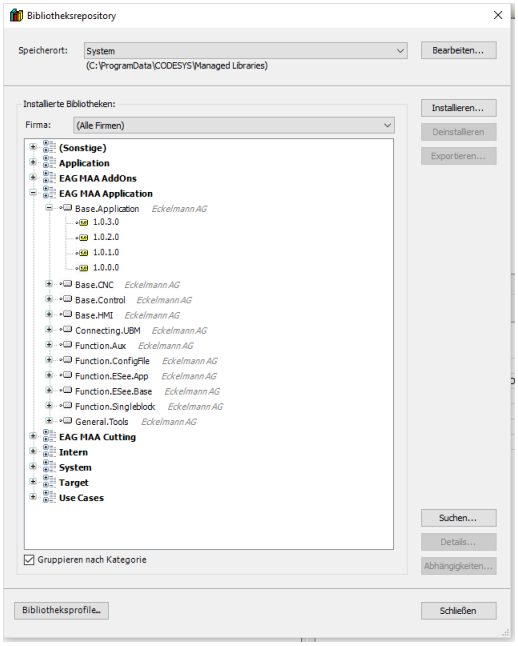
Das modulare Entwickeln und Verwenden von Bibliotheken war in der CODSYS V2 Welt kaum möglich. Dies liegt in erster Linie an der sehr eingeschränkten Bibliotheksverwaltung. Bibliotheken konnten nicht projektbezogen angegeben werden, eine Bibliothek galt damit für alle Projekte und man musste sich sicher sein, dass eine Änderung an einer Bibliothek keine Auswirkungen auf andere Projekte hatte, egal wie alt das Projekt war.
CODESYS V3 bietet nun eine moderne Bibliotheksverwaltung, bei der man im sogenannten Bibliotheksrepository verschiedene Versionen einer Bibliothek verwalten kann. Für jedes Projekt lässt sich die gewünschte Bibliothek und deren Version auswählen. Öffnet man ein altes Kundenprojekt, werden automatisch die dazu gehörenden Bibliotheken geladen. Zeitgleich kann man auch eine zweite CODESYS-Instanz ausführen, in der man ein anderes Projekt mit denselben Bibliotheken, aber einer anderen Version geöffnet hat. Dies erleichtert nicht nur die Entwicklung und Pflege von Projekten enorm, eine moderne Bibliotheksverwaltung ist auch der Schlüssel für mehr modularen und damit leichter wiederverwendbaren Code. Ob Maschinenvarianten oder lange Produktlebenszyklen mit langfristigem Maschinen-Support, eine flexible Bibliotheksverwaltung macht sich somit schnell bezahlt. Wer sich einmal mit der Bibliotheksverwaltung vertraut gemacht hat, wird sie schon bald nicht mehr missen wollen.2



Der aus CODESYS V2 bekannte Bibliothekverwalter ist in V3 noch vorhanden, nur ist er jetzt wegen der gleichzeitigen Unterstützung verschiedener Versionen viel mächtiger und zeichnet sich durch seine hohe Praxistauglichkeit aus, was Entwicklern das Leben insgesamt leichter macht.

2 Video: CODESYS Webinar Bibliotheksmanagement Basic. YouTube Kanal von CODESYS, 09.03.2015.

Im sog. Bibliotheksrepository sieht man nicht nur die installierten Bibliotheken, sondern eben auch die vorhandenen Versionen einer Bibliothek. Der Screenshot zeigt beispielsweise die Bibliothek „Base.Application“ und deren Versionen 1.0.0.0 bis 1.0.3.0.
Das konsequente Denken in Bibliotheken ermöglicht insgesamt eine einfachere Auslagerung und Wiederverwendung von allgemeinen Bausteinen. Durch das einfache Handling unterschiedlicher Versionsstände fällt der Umstieg auf eine neuere Version auch einfacher, so können Bestandsmaschinen schneller von Patches und Performance-Verbesserungen profitieren.



Bausteine aus Bibliotheken sind keine Blackboxen mehr

Eine wesentliche Weiterentwicklung unter V3 und der Verwendung von Bibliotheken ist die Sichtbarkeit des beinhalteten Codes. In V2 war ein Bibliothekbaustein immer eine Blackbox: Man sieht, was rein und raus geht, nicht aber was darin passiert.
In V3 hat man nun die Möglichkeit, den Quellcode in Bibliotheken offen zu lassen, d.h. man sieht auch den Quellcode von Bausteinen in Bibliotheken und kann beim Debuggen in Bausteine springen und wie den eigenen Codes analysieren und nach Fehlern suchen. Eine Änderung des Quellcodes in einer Bibliothek ist zwar nicht direkt möglich, man kann jedoch bei eigenen Bibliotheken diese parallel in einer weiteren CODESYS-Instanz öffnen, dort Korrekturen vornehmen und eine neue Bibliotheksversion installieren.

Library Manager | SendHmiMsg [from Base.HMI]

```
1  FUNCTION SendHmiMsg : BOOL
2  {library private}
3  (*
4  Function:
5  This block is used to send a message to the HMI by using the DB2 Message Channel. An err
6  to send the message.
7  *)
8  VAR_INPUT
9     Msg_pr         :POINTER TO HMI_MESSAGE_TR;
10 END_VAR
11 VAR
```

```
1  (* send message *)
2  SendHmiMsg := PutMsg(Msg_pr, ADR(gHmiMsgQueue_r), FALSE);
3  IF (NOT SendHmiMsg) THEN
4     dw := Msg_pr^.Order_w;
5     PlcError( eBHMI_HMI_MSG_QUEUE_OVERRUN_e, eBHMI_CLASS_LOW_E, ADR('%d'), ADR(dw));
6  END_IF
```

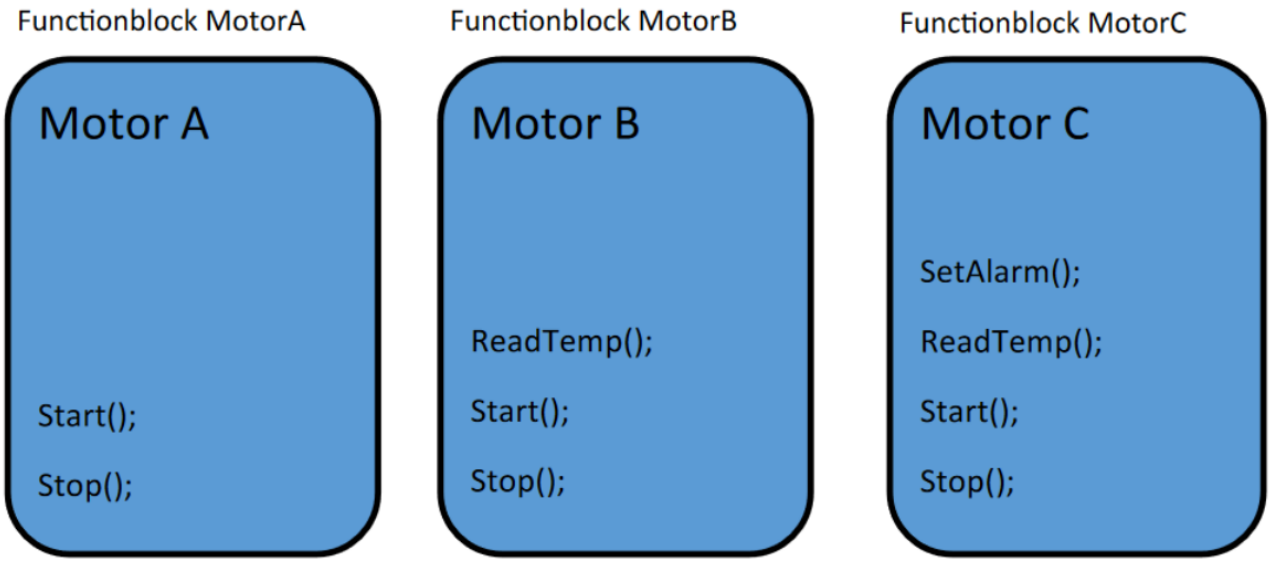
Lediglich der Hinweis im Titel des Reiters weist darauf hin, dass der Baustein „SendHmiMsg“ Teil der Bibliothek „Base.HMI“ ist, ansonsten lässt er sich genauso wie ein lokaler Baustein lesen und debuggen.

Vererbung und objektorientierte Programmierung (Beispiel)

Ein großer Vorteil von CODESYS V3 gegenüber V2 ist, wie eingangs bereits gesagt, die Möglichkeit, dass man erstmals auch objektorientiert programmieren kann (aber nicht muss). Dies erlaubt eine feinere Modularisierung der Applikation durch eine Kapselung von Funktionalitäten. Code lässt sich dadurch leichter wiederverwenden und durch Vererbung können einfacher Varianten realisiert werden. Die vielfältigen Vorteile gegenüber der bekannten „Copy&Paste“-Methode in V2 liegen auf der Hand:

- Weniger Lines of Code und damit effizientere Programmierung
- Geringere Fehleranfälligkeit
- Leichtere Orientierung im Projekt
- Geringerer Testaufwand, da Basisfunktionen nur einmal programmiert werden müssen
- Leichtere Erweiterbarkeit
- Einfache Reproduzierbarkeit von Applikationen auch bei Maschinen mit vielen Varianten
- Vereinfachte Pflege von Altmaschinen, da Patches nicht für jeden Maschine einzeln nachprogrammiert werden müssen
- Leichteres Upgrade / Retrofit von Altmaschinen
- Schneller Inbetriebnahme

Das Prinzip der Vererbung soll im Folgenden an einem Beispiel mit 3 Arten von Motoren kurz aufgezeigt werden:



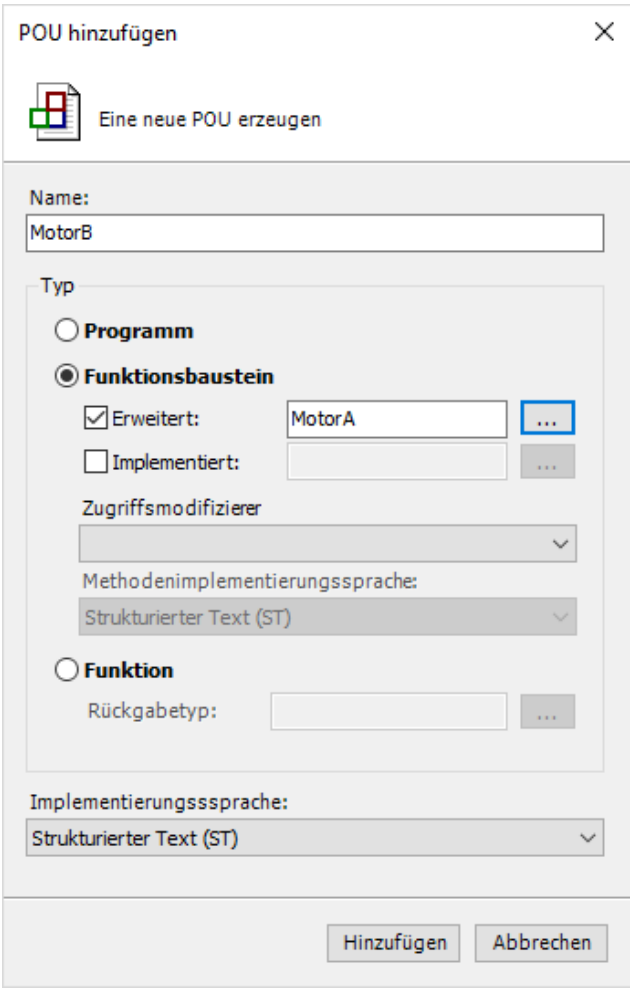
a. Ein einfacher Motor, den man nur ein- und ausschalten kann.
b. Ein Motor, der zusätzlich eine Motortemperatur liefert.
c. Ein Motor, bei dem neben der Temperaturüberwachung eine Sicherheitsabschaltung aktiviert werden kann.

Copy & Paste (V2) versus Vererbung (V3)

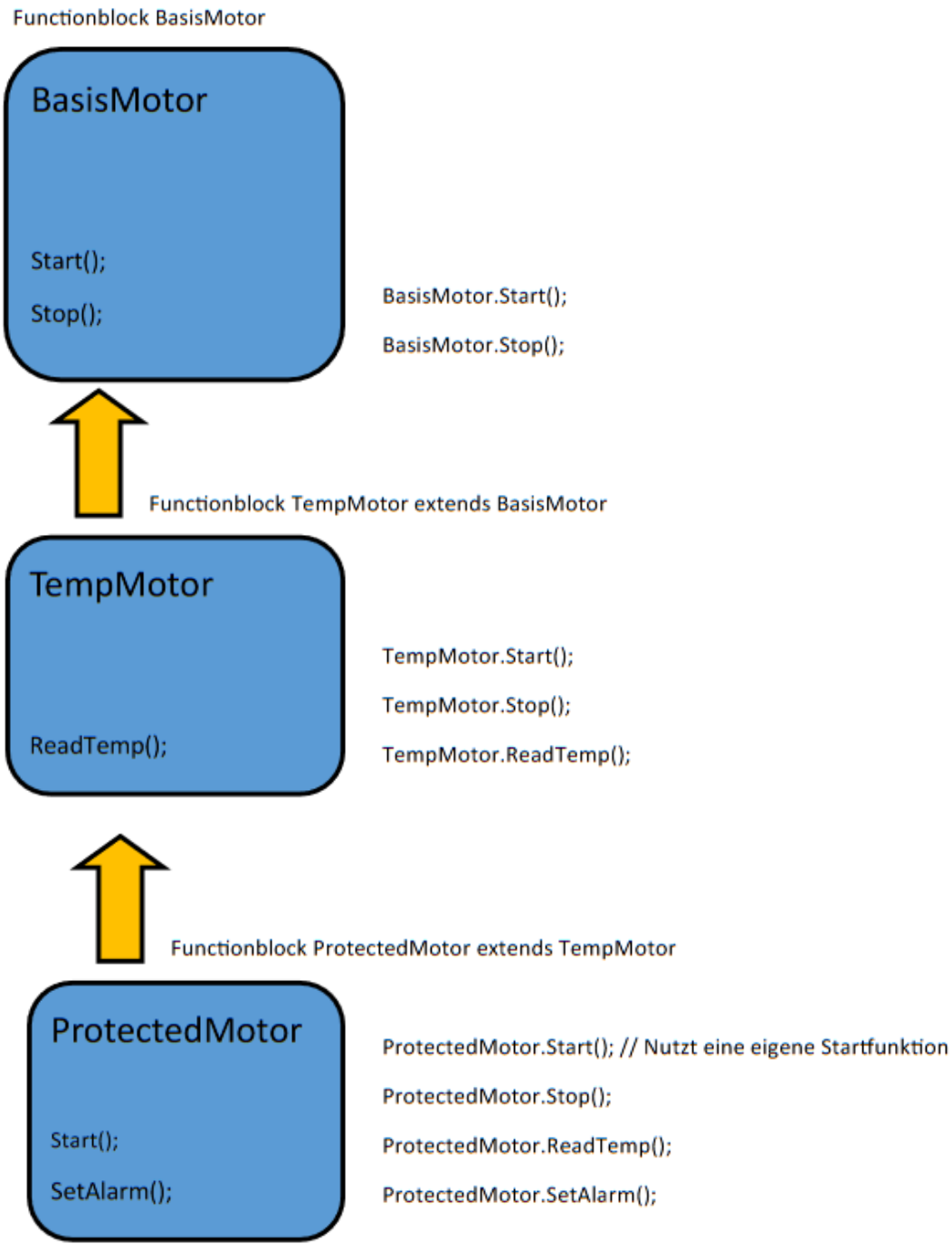
In der **V2 Welt** muss man für jede Art eines Motors einen eigenen Baustein schreiben, obwohl sich u.U. die Funktionen zum Beispiel für Start() nicht unterscheiden, d.h.:

- Jede Funktion muss trotz Gleichheit mehrfach programmiert werden.
- Eine Fehlerbehebung ist erst mal nur für einen Typ umgesetzt, ein Fehler in einer Funktion muss mehrfach behoben werden und mehrfach getestet werden.
- Ein neuer Motortyp D wird oft durch Kopieren eines bekannten Bausteins, Löschen vermeintlich nicht benötigter Stellen und Programmieren neuer Funktionen implementiert. Damit entsteht ein neuer Baustein mit unnötig viel Code, teils nicht benötigte Codefragmente und der Baustein muss komplett neu getestet werden.
- Der gesamte Code „wuchert“ und ist schlecht pflegbar.

In der **V3 Welt** und mit der Möglichkeit der Vererbung würde man den Ansatz anders wählen und Bausteine erstellen, die sich schon an Grundfunktionen orientiert und nicht nur am Gesamtaufbau einer Einheit (s. Infografik auf der nächsten Seite).
So ist in diesem Beispiel auch ein temperaturüberwachter Motor erst mal ein Motor mit den Funktionen Start und Stop und unterscheidet sich nur durch die zusätzliche Funktion ReadTemp. Daher programmiert man einen Basisbaustein „BasisMotor“ mit den Funktionen Start und Stop. Dieser Baustein kann für alle Motoren verwendet werden.
Beim temperaturüberwachten Motor programmiert man einen neuen Baustein, der vom BasisMotor abgeleitet ist. Er erbt dessen Funktionen und enthält selbst nur die Funktion ReadTemp. Die Funktionen Start und Stop sind automatisch aufgrund der Vererbung vorhanden. Die Vererbung kann beliebig weitergeführt werden.
Die Vererbung bietet natürlich auch die Möglichkeit des Überschreibens einer Funktion, d.h. wenn ein Motor z.B. eine ganz spezielle Startfunktion hat, kann er die Startfunktion der Basisklasse überschreiben.



Prinzip der Vererbung



Sie möchten auch auf CODESYS V3 umsteigen, um Ihre CNC/SPS-Applikationen in einer modernen IDE zu entwickeln oder bestehende Projekte auf CODESYS V3 portieren. Sprechen Sie gerne unseren technischen Vertrieb an, wie wir Sie beim Umstieg auf CODESYS V3 unterstützen können.

Literatur-Tipps:

- 3S-Smart Software Solutions: Vergleich CODESYS V2 zu CODESYS V3. Kurzübersicht ausgewählter Eigenschaften. 2014. (PDF)
- CODESYS Development System V3 (Downloadseite für die aktuelle Version von CODESYS V3, kostenlos)
- Matthias Gehring: Schnelle Orientierung im CODESYS Projekt. CODESYS Blog, 15.04.2018, abgerufen am 04.03.2019.
- CODESYS Group: Gut zu wissen. (Kurztutorials)
- CODESYS Group: CODESYS Training Standard (Schulungsprogramm)

Autor

Dipl.-Ing. Frank Ebert, Entwicklungsingenieur bei der Eckelmann AG

Kontakt

Eckelmann AG
Berliner Straße 161
65205 Wiesbaden
Deutschland

Tel.: +49 611 7103-0
Fax: +49 611 7103-133
E-Mail: info@eckelmann.de
Internet: www.eckelmann.de