

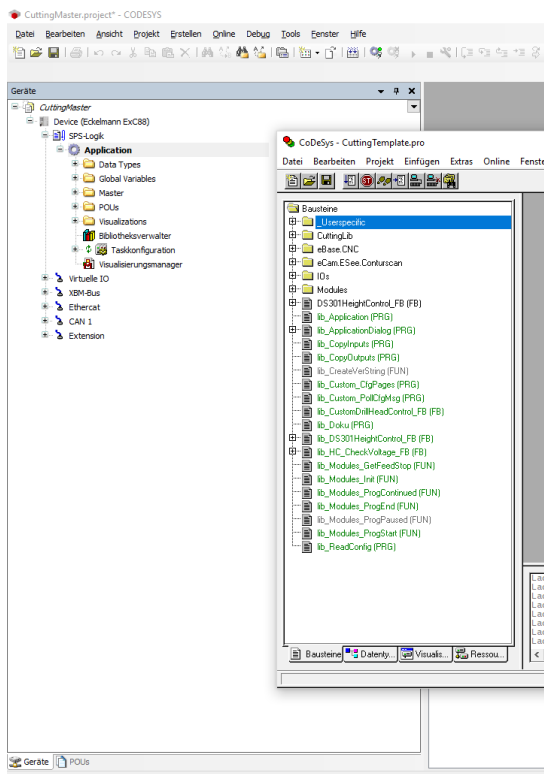
White Paper der Eckelmann AG, Wiesbaden, März 2019

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.

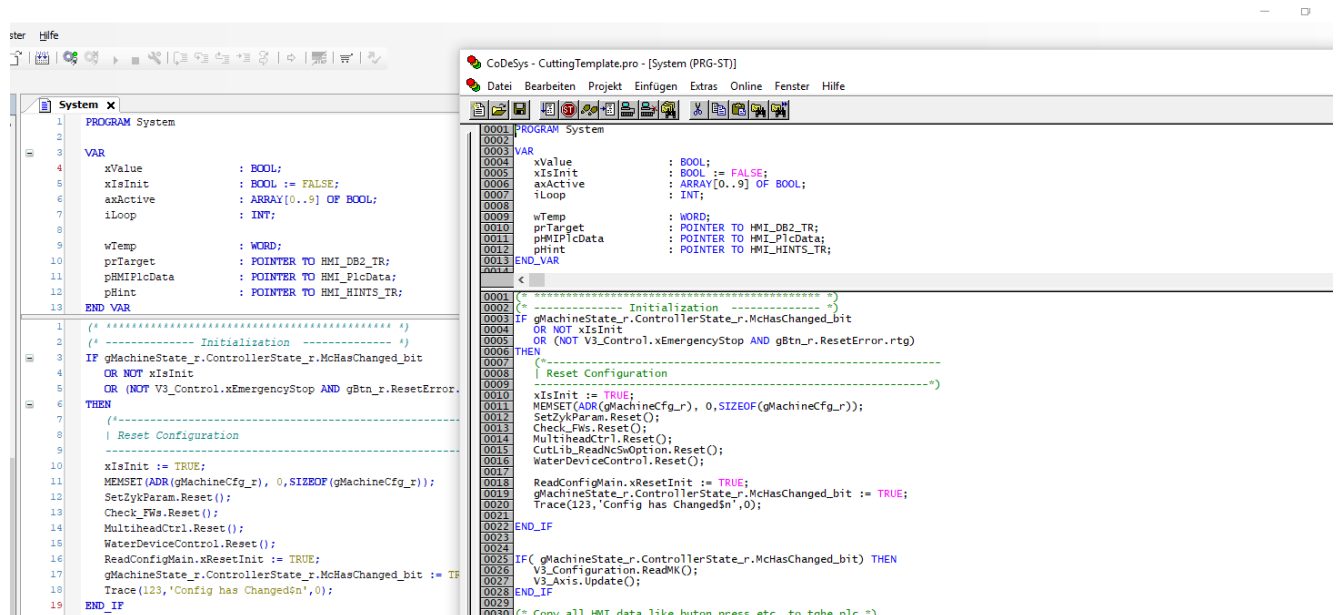




Vom „Code-Editor“ zur modernen IDE für die SPS-Programmierung

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. www.eckelmann.de



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 V3 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));

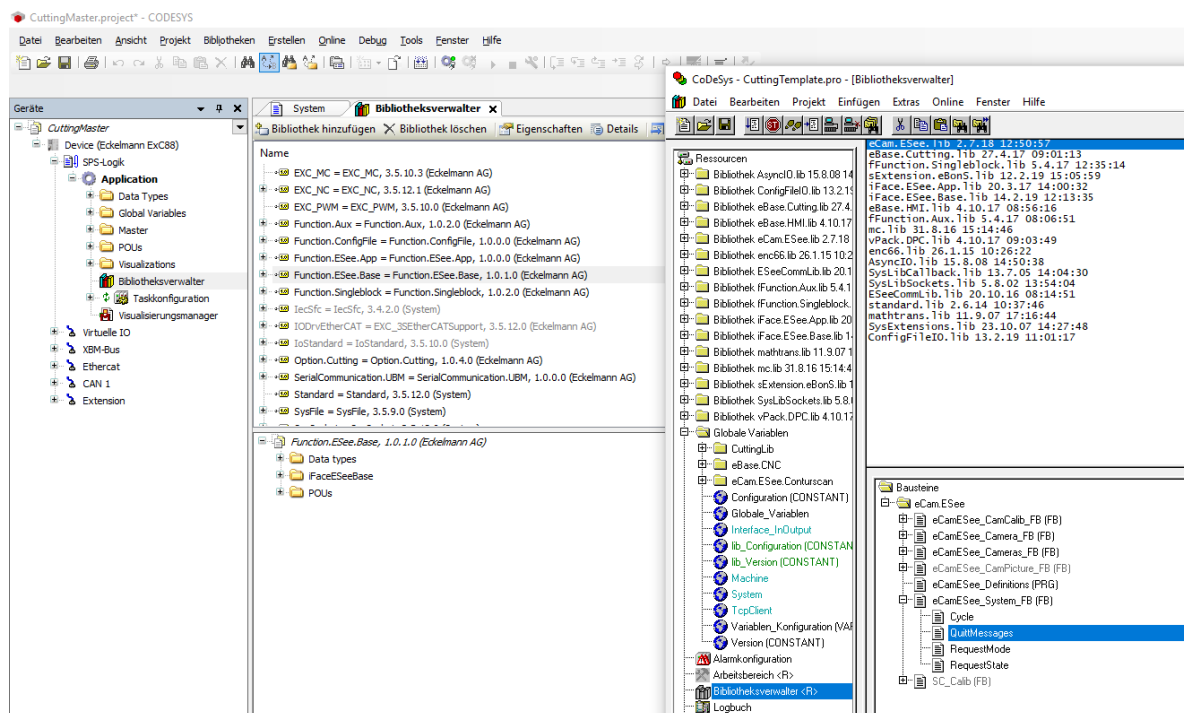
    V3_HMI.uiPlcMessagegQuit := V3_HMI.PlcMessages.uiHmiMsgCnt2;
    stabil_bit := FALSE;
    msg_ok_bit := TRUE;      (* Nachricht verfügbar *)
  ELSE
    stabil_bit := TRUE;
    msg_ok_bit := FALSE;    (* keine Nachricht vorhanden *)
  END_IF;
ELSE
  stabil_bit := FALSE;
  msg_ok_bit := FALSE;    (* keine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



Modularität und Bibliotheken

Das modulare Entwickeln und Verwenden von Bibliotheken war in der CODESYS 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



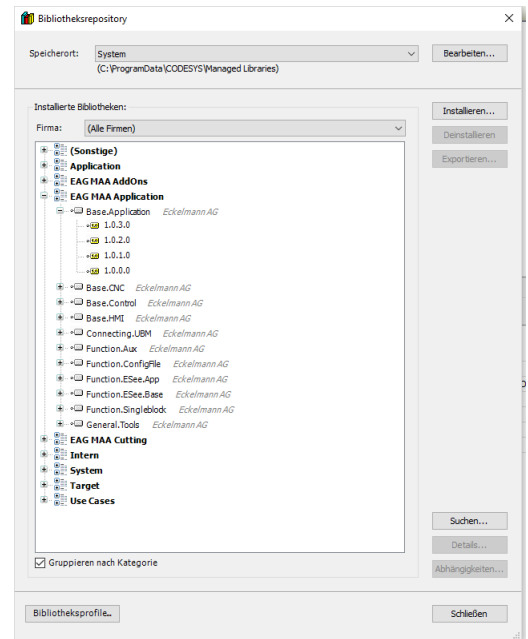
Der aus CODESYS V2 bekannte Bibliothekverwalter ist in V3 noch vorhanden, nur ist er jetzt wegen der gleichzeitigen Unterstützung verschiedener Versionen viel mächtiger und zeichnet sich durch seine hohe Praxistauglichkeit aus, was Entwicklern das Leben insgesamt leichter macht.

² Video: [CODESYS Webinar Bibliotheksmanagement Basic](https://www.youtube.com/watch?v=...). YouTube Kanal von CODESYS, 09.03.2015.
www.eckelmann.de



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.



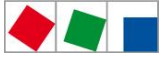
Bausteine aus Bibliotheken sind keine Blackboxen mehr

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] x
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 error
6   to send the message.
7   *)
8   VAR_INPUT
9     Msg_pr          : POINTER TO HMI_MESSAGE_TR;
10  END_VAR
11  VAR
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

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.

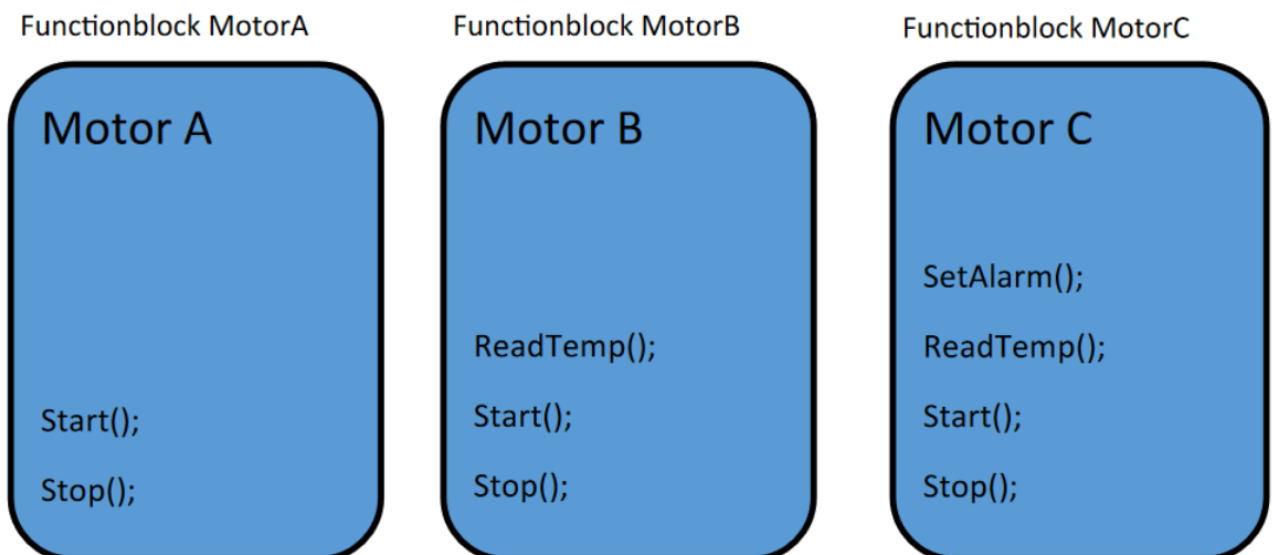


Vererbung und objektorientierte Programmierung (Beispiel)

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.



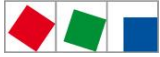
Copy & Paste (V2) versus Vererbung (V3)

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.

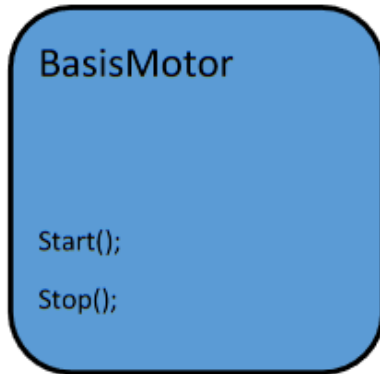
In der **V3 Welt** und mit der Möglichkeit der Vererbung würde man den Ansatz anders wählen und Bausteine erstellen, die sich schon an Grundfunktionen orientiert und nicht nur am Gesamtaufbau einer Einheit (s. [Infografik](#) auf der nächsten Seite).

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

Functionblock BasisMotor



BasisMotor.Start();
BasisMotor.Stop();



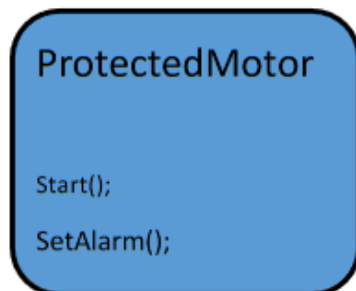
Functionblock TempMotor extends BasisMotor



TempMotor.Start();
TempMotor.Stop();
TempMotor.ReadTemp();



Functionblock ProtectedMotor extends TempMotor



ProtectedMotor.Start(); // Nutzt eine eigene Startfunktion
ProtectedMotor.Stop();
ProtectedMotor.ReadTemp();
ProtectedMotor.SetAlarm();

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](#). (Kurzutorials)
- 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