

DEVELOPER'S GUIDE FOR

FULLY INTEGRATED SERVO MOTORS

CLASS 5 AND LATER SMARTMOTORS
WITH COMBITRONIC™ TECHNOLOGY



Rev. R, July 2022

DESCRIBES THE SMARTMOTOR™
COMMANDS AND PROGRAMMING FOR
CLASS 5 AND LATER

Copyright Notice

©2001-2022 Moog Inc.

Moog Animatics SmartMotor™ Developer's Guide, Rev. R, PN: SC80100003-002.

This manual, as well as the software described in it, is furnished under license and may be used or copied only in accordance with the terms of such license. The content of this manual is furnished for informational use only, is subject to change without notice and should not be construed as a commitment by Moog Inc., Animatics. Moog Inc., Animatics assumes no responsibility or liability for any errors or inaccuracies that may appear herein.

Except as permitted by such license, no part of this publication may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, electronic, mechanical, recording, or otherwise, without the prior written permission of Moog Inc., Animatics.

The programs and code samples in this manual are provided for example purposes only. It is the user's responsibility to decide if a particular code sample or program applies to the application being developed and to adjust the values to fit that application.

Moog Animatics and the Moog Animatics logo, SmartMotor and the SmartMotor logo, Combitronic and the Combitronic logo, and SMI are all trademarks of Moog Inc., Animatics. Other trademarks are the property of their respective owners.

Please let us know if you find any errors or omissions in this manual so that we can improve it for future readers. Such notifications should contain the words "Developer's Guide" in the subject line and be sent by e-mail to: animatics_marcom@moog.com. Thank you in advance for your contribution.

Contact Us:

Americas - West

Moog Animatics
2581 Leghorn Street
Mountain View, CA 94043
USA
Tel: 1 650-960-4215

Americas - East

Moog Animatics
1995 NC Hwy 141
Murphy, NC 28906
USA

Support: 1 888-356-0357

Website: www.animatics.com

Email: animatics_sales@moog.com

Table Of Contents

Introduction	28
Overview	29
Combitronic Support	29
Combitronic with the DS2020 Combitronic System	30
Communication Lockup Wizard	31
Safety Information	31
Safety Symbols	31
Other Safety Considerations	31
Motor Sizing	32
Environmental Considerations	32
Machine Safety	32
Documentation and Training	33
Additional Equipment and Considerations	33
Safety Information Resources	33
Additional Documents	35
Related Guides	35
Other Documents	35
Additional Resources	36
Part 1: Programming the SmartMotor	37
Beginning Programming	47
Understanding Firmware Versions	48
Downloading and Installing the Latest Firmware	48
Understanding the FIRMWARE VERSION Field in the Command Descriptions	48
Class 5 Firmware for D- and M-Style Motors	48
Class 6 Firmware for M-Style (MT/MT2) Motors	49
Class 6 Firmware for D-Style Motors	49
DS2020 Combitronic System Firmware	49
Setting the Motor Firmware Version in SMI	50
Setting the Default Firmware Version	50
Checking the Default Firmware Version	51
Opening the SMI Window (Program Editor)	51
Understanding the Program Requirements	52
Creating a "Hello World" Program	54
Entering the Program in the SMI Editor	54

Adding Comments to the Code	54
Checking the Program Syntax	54
Saving the Program	55
Downloading a Program to the SmartMotor	55
Syntax Checking, Compiling and Downloading the Program	55
Additional Notes on Downloaded Programs	55
Running a Downloaded Program	56
Using the Program Download Window	57
Using the Terminal Window and Run Program Button	57
Using the RUN Command in the Terminal Window	57
Creating a Simple Motion Program	59
SMI Software Features	60
Introduction	61
Menu Bar	62
Toolbar	62
Configuration Window	64
Terminal Window	67
Initiating Motion from the Terminal Window	69
Information Window	69
Program Editor	70
Motor View	72
SMI Trace Functions	73
Monitor Window	76
Serial Data Analyzer	78
Chart View	79
Chart View Example	80
Macros (Keyboard Shortcuts or Hotkeys)	83
Tuner	85
SMI Options	89
SMI Help	90
Context-Sensitive Help Using F1	90
Context-Sensitive Help Using the Mouse	90
Help Buttons	90
Hover Help	90
Table of Contents	90
Projects	91
SmartMotor Playground	92
Opening the SmartMotor Playground	93

Moving the Motor	94
Communication Details	96
Introduction	98
Connecting to a Host	99
Daisy Chaining Multiple D-Style SmartMotors over RS-232	100
ADDR=formula	102
SLEEP, SLEEP1	102
WAKE, WAKE1	102
ECHO, ECHO1	103
ECHO_OFF, ECHO_OFF1	103
Serial Commands	104
OCHN(type,channel,parity,bit rate,stop bits,data bits,mode,timeout)	104
CCHN(type,channel)	105
BAUDrate, BAUD(channel)=formula	105
PRINT(), PRINT1()	105
SILENT, SILENT1	106
TALK, TALK1	106
a=CHN(channel)	106
a=ADDR	106
Communicating over RS-485	107
Using Data Mode	107
CAN Communications	110
CADDR=formula	110
CBAUD=formula	110
=CAN, =CAN(arg)	110
CANCTL(function,value)	110
SDORD(...)	111
SDOWR(...)	111
NMT	112
RB(2,4), x=B(2,4)	112
Exceptions to NMT, SDORD and SDOWR Commands	112
I/O Device CAN Bus Controller	113
Combitronic Communications	113
Combitronic Features	114
Other Combitronic Benefits	114
Program Loops with Combitronic	115
Global Combitronic Transmissions	115
Simplify Machine Support	116

Combitronic with RS-232 Interface	116
Combitronic with the DS2020 Combitronic System	117
Other CAN Protocols	118
CANopen - CAN Bus Protocol	118
DeviceNet - CAN Bus Protocol	118
I ² C Communications (Class 5 D-Style Motors)	118
OCHN(IIC,1,N,baud,1,8,D)	120
CCHN(IIC,1)	120
PRINT1(arg1,arg2, ... ,arg_n)	120
RGETCHR1, Var=GETCHR1	120
RLEN1, Var=LEN1	120
Motion Details	121
Introduction	122
Motion Command Quick Reference	123
Basic Motion Commands	124
Target Commands	124
PT=formula	124
PRT=formula	125
ADT=formula	125
AT=formula	125
DT=formula	125
VT=formula	125
Motion Mode Commands	126
MP	126
MV	126
MT	126
Torque Commands	127
TS=formula	127
T=formula	127
Brake Commands	127
BRKRLS	127
BRKENG	127
BRKSRV	128
BRKTRJ	128
Brake Command Examples	128
EOBK(IO)	129
MTB	130
Index Capture Commands	130

DS2020 Combitronic System Index Capture	131
Other Motion Commands	132
G	132
S	132
X	132
O=formula	133
OSH=formula	133
OFF	133
SCALEA(m,d), SCALEP(m,d), SCALEV(m,d)	133
Commutation Modes	134
MDT	134
MDE	134
MDS	134
MDC	135
MDB	135
MINV(0), MINV(1)	135
Modes of Operation	136
Torque Mode	136
Torque Mode Example	136
Dynamically Change from Velocity Mode to Torque Mode	136
Velocity Mode	137
Constant Velocity Example	137
Change Commanded Speed and Acceleration	137
Absolute (Position) Mode	138
Absolute Move Example	138
Two Moves with Delay Example	138
Change Speed and Acceleration Example	138
Shift Point of Origin Example	139
Relative Position Mode	139
Relative Mode Example	139
Follow Mode with Ratio (Electronic Gearing)	140
Electronic Gearing and Camming over CANopen	140
Electronic Gearing Commands	140
SRC(enc_src)	141
MFR	141
MSR	141
MF0	141
MS0	141

MFMUL=formula, MFDIV=formula	141
MFA(distance[,m/s])	142
MFD(distance[,m/s])	142
MFSLEW(distance[,m/s])	142
Follow Internal Clock Source Example	142
Follow Incoming Encoder Signal With Ramps Example	143
Electronic Line Shaft	145
ENCDC(in_out)	145
Spooling and Winding Overview	146
Relative Position, Auto-Traverse Spool Winding	146
MFSDC(distance,mode)	147
Dedicated, Absolute Position, Winding Traverse Commands	149
MFSDC(distance,2)	150
MFLTP=formula	150
MFHTP=formula	150
MFCTP(arg1,arg2)	150
MFL(distance[,m/s])	151
MFH(distance[,m/s])	151
ECS(counts)	151
Single Trajectory Example Program	152
Chevron Wrap Example	153
Other Traverse Mode Notes	155
Traverse Mode Status Bits	156
Cam Mode (Electronic Camming)	156
Electronic Camming Details	158
Understanding the Inputs	158
Should I choose Source Counts or Intermediate Counts?	160
Should I choose Variable or Fixed cam?	160
Electronic Camming Notes and Best Practices	162
Examples	164
Electronic Gearing and Camming over CANopen	164
Electronic Camming Commands	165
CTE(table)	165
CTA(points,seglen[,location])	165
CTW(pos[,seglen][,user])	165
MCE(arg)	166
MCW(table,point)	166
RCP	166

RCTT	167
MC	167
MCMUL=formula	167
MCDIV=formula	167
O(arg)=formula	167
OSH(arg)=formula	167
Cam Example Program	168
Mode Switch Example	171
Position Counters	173
Modulo Position	174
Modulo Position Commands	174
Dual Trajectories	175
Commands That Read Trajectory Information	177
Dual Trajectory Example Program	178
Using Mixed Mode Operations After Homing	179
Synchronized Motion	179
Synchronized-Target Commands	179
PTS(), PRTS()	179
VTS=formula	180
ADTS=formula, ATS=formula, DTS=formula	180
PTSS(), PRTSS()	180
A Note About PTS and PRTS	181
Other Synchronized-Motion Commands	183
GS	183
TSWAIT	183
Program Flow Details	185
Introduction	186
Flow Commands	186
RUN	186
RUN?	187
GOTO#, GOTO(label), C#	187
GOSUB#, GOSUB(label), RETURN	188
IF, ENDIF	188
ELSE, ELSEIF	188
WHILE, LOOP	189
SWITCH, CASE, DEFAULT, BREAK, ENDS	190
TWAIT	190
WAIT=formula	191

STACK	191
END	191
Program Flow Examples	192
IF, ELSEIF, ELSE, ENDIF Examples	192
WHILE, LOOP Examples	192
GOTO(), GOSUB() Examples	193
SWITCH, CASE, BREAK, ENDS Examples	194
Interrupt Programming	195
ITR(), ITRE, ITRD, EITR(), DITR(), RETURNI	195
TMR(timer,time)	197
Variables and Math	198
Introduction	199
Variable Commands	199
EPTR=formula	199
VST(variable,number)	199
VLD(variable,number)	200
Math Expressions	200
Math Operations	200
Logical Operations	200
Integer Operations	200
Floating Point Functions	200
Math Operation Details and Examples	201
Array Variables	201
Array Variable Examples	202
Error and Fault Handling Details	203
Motion and Motor Faults	204
Overview	204
Drive Stage Indications and Faults	204
Fault Bits	204
Error Handling	205
Example Fault-Handler Code	205
PAUSE	206
RESUME	206
Limits and Fault Handling	207
Position Error Limits	207
dE/dt Limits	207
Velocity Limits	208
Hardware Limits	208

Software Limits	208
Fault Handling	209
Monitoring the SmartMotor Status	210
System Status	213
Introduction	214
Retrieving and Manipulating Status Words/Bits	214
System and Motor Status Bits	214
Reset Error Flags	217
System Status Examples	217
Timer Status Bits	218
Interrupt Status Bits	218
I/O Status	219
User Status Bits	219
Multiple Trajectory Support Status Bits	220
Cam Status Bits	221
Interpolation Status Bits	222
Motion Mode Status	222
RMODE, RMODE(arg)	222
I/O Control Details	223
I/O Port Hardware	224
I/O Connections Example (Class 5 D-Style Motors)	225
I/O Voltage Protection	225
Discrete Input and Output Commands	225
Discrete Input Commands	226
Discrete Output Commands	226
Output Condition and Fault Status Commands	227
Output Condition Commands	227
Output Fault Status Reports	227
General-Use Input Configuration	228
Multiple I/O Functions Example	228
Analog Functions of I/O Ports	230
5 Volt Push-Pull I/O Analog Functions (Class 5 D-Style Motors)	230
24 Volt I/O Analog Functions (Class 5 D-Style AD1 Option Motors, Class 5 M-Style Motors)	230
24 Volt I/O Analog Functions (Class 6 M-Style Motors)	230
24 Volt I/O Analog Functions (Class 6 D-Style Motors)	231
Special Functions of I/O Ports	232
Class 5 D-Style Motors: Special Functions of I/O Ports	233
I/O Ports 0 and 1 – External Encoder Function Commands	233

I/O Ports 2 and 3 - Travel Limit Inputs	233
I/O Ports 4 and 5 - Communications	233
I/O Port 6 - Go Command, Encoder Index Capture Input	234
Class 5 M-Style Motors: Special Functions of I/O Ports	235
COM Port Pins 4, 5, 6, and 8 - A-quad-B or Step-and-Direction Modes	235
I/O Ports 2 and 3 - Travel Limit Inputs	235
I/O Port 5 - Encoder Index Capture Input	235
I/O Port 6 - Go Command	236
Class 6 Motors: Special Functions of I/O Ports	237
A-quad-B or Step-and-Direction Modes	237
I/O Ports 2 and 3 - Travel Limit Inputs	238
I/O Port 4 and 5 - Encoder Index Capture Input	238
I/O Port 6 - Go Command	238
I/O Brake Output Commands	238
I ² C Expansion (D-Style Motors)	239
Tuning and PID Control	240
Introduction	241
Tuning and PID Control on the DS2020 Combitronic System	241
Understanding the PID Control	241
Tuning the PID Control	242
Using F	243
Setting KP	243
Setting KD	243
Setting KI and KL	244
Setting EL=formula	244
Other PID Tuning Parameters	244
KG=formula	245
KV=formula	245
KA=formula	245
Current Limit Control	246
AMPS=formula	246
Part 2: SmartMotor Command Reference	247
(Single Space Character)	248
a...z	249
aa...zz	249
aaa...zzz	249
Ra...Rz	249

Raa...Rzz	249
Raaa...Rzzz	249
ab[index]=formula	252
Rab[index]	252
ABS(value)	255
RABS(value)	255
AC	256
RAC	256
ACOS(value)	259
RACOS(value)	259
ADDR=formula	261
RADDR	261
ADT=formula	263
ADTS=formula	265
af[index]=formula	267
Raf[index]	267
Ai(enc)	270
Aij(enc)	272
Aj(enc)	274
Aji(enc)	276
al[index]=formula	278
Ral[index]	278
AMPS=formula	281
RAMPS	281
ASIN(value)	284
RASIN(value)	284
AT=formula	286
RAT	286
ATAN(value)	289
RATAN(value)	289
ATOF(index)	291
RATOF(index)	291
ATS=formula	292
aw[index]=formula	294
Raw[index]	294
B(word,bit)	297
RB(word,bit)	297
Ba	301

RBa	301
BAUD(channel)=formula	303
RBAUD(channel)	303
Be	305
RBe	305
Bh	307
RBh	307
Bi(enc)	309
RBi(enc); supports the DS2020 Combitronic system over RS-232 only	309
Bj(enc)	312
RBj(enc)	312
Bk	315
RBk	315
Bl	316
RBl	316
Bls	318
RBlS	318
Bm	320
RBm	320
Bms	322
RBms	322
Bo	324
RBo	324
Bp	325
RBp	325
Bps	327
RBps	327
Br	329
RBr	329
BREAK	331
BRKENG	333
BRKRLS	335
BRKSRV	337
BRKTRJ	339
Brs	341
RBrS	341
Bs	343
RBs	343

Bt	345
RBt	345
Bv	347
RBv	347
Bw	349
RBw	349
Bx(enc)	351
RBx(enc)	351
C{number}	353
CADDR=formula	355
RCADDR	355
CAN, CAN(arg)	357
RCAN, RCAN(arg)	357
CANCTL(function,value)	359
CASE formula	360
CBAUD=formula	363
RCBAUD	363
CCHN(type,channel)	365
CHN(channel)	367
RCHN(channel)	367
CLK=formula	369
RCLK	369
COMCTL(function,value)	370
COS(value)	372
RCOS(value)	372
CP	374
RCP	374
CTA(points,seglen[,location])	376
CTE(table)	378
CTR(enc)	380
RCTR(enc)	380
CTT	382
RCTT	382
CTW(pos[,seglen][,user])	383
DEA	386
RDEA	386
DEFAULT	388
DEL=formula	390

RDEL	390
DELM(arg)	392
DFS(value)	393
RDFS(value)	393
DITR(int)	394
DT=formula	396
RDT	396
DTS=formula	399
EA	401
REA	401
ECHO	403
ECH00	405
ECH01	406
ECHO_OFF	407
ECHO_OFF0	408
ECHO_OFF1	409
ECS(counts)	410
EIGN(...)	412
EILN	415
EILP	417
EIRE	419
EIRI	421
EISM(x)	423
EITR(int)	424
EL=formula	426
REL	426
ELSE	428
ELSEIF formula	430
ENC0	432
ENC1	433
ENCCTL(function,value)	435
ENCD(in_out)	437
END	439
ENDIF	441
ENDS	443
EOBK(IO)	445
EOFT(IO)	447
EOIDX(number)	449

EPTR=formula	450
REPTR	450
ERRC	451
RERRC	451
ERRW	453
RERRW	453
ETH(arg)	455
RETH(arg)	455
ETHCTL(function,value)	456
F	457
FAUSTS(x)	459
FD=expression	461
FABS(value)	463
RFABS(value)	463
FSA(cause,action)	465
FSAD(n,m)	467
FSQRT(value)	469
RFSQRT(value)	469
FW	471
RFW	471
G	473
GETCHR	476
RGETCHR	476
GETCHR1	478
RGETCHR1	478
GOSUB(label)	480
GOTO(label)	482
GROUP(function,value)	484
GS	487
HEX(index)	489
RHEX(index)	489
HM_ADT=formula	491
HM_MTHD=formula	492
RHM_MTHD	492
HM_OSET=formula	496
RHM_OSET	496
HM_VTS=formula	498
HM_VTZ=formula	500

I(enc)	502
RI(enc); supports the DS2020 Combitronic system over RS-232 only	502
IDENT=formula	504
RIDENT	504
IF formula	506
IN(...)	509
RIN(...)	509
INA(...)	512
RINA(...)	512
IPCTL(function,"string")	515
ITR(Int#,StatusWord,Bit#,BitState,Label#)	517
ITRD	520
ITRE	522
J(enc)	524
RJ(enc)	524
KA=formula	526
RKA	526
KD=formula	528
RKD	528
KG=formula	530
RKG	530
KI=formula	532
RKI	532
KII=formula	534
RKII	534
KL=formula	535
RKL	535
KP=formula	537
RKP	537
KPI=formula	539
RKPI	539
KS=formula	540
RKS	540
KV=formula	542
RKV	542
LEN	544
RLEN	544
LEN1	545

RLEN1	545
LFS(value)	547
RLFS(value)	547
LOAD	548
LOCKP	551
LOOP	553
MC	555
MCDIV=formula	557
RMCDIV	557
MCE(arg)	558
MCMUL=formula	560
RMCMUL	560
MCW(table,point)	562
MDB	564
MDC	566
MDE	568
MDH	570
MDHV	572
MDS	574
MDT	576
MF0	578
MFA(distance[,m/s])	580
MFCTP(arg1,arg2)	583
MFD(distance[,m/s])	585
MFDIV=formula	588
MFH(distance[,m/s])	590
MFHTP=formula	592
MFL(distance[,m/s])	594
MFLTP=formula	596
MF MUL=formula	598
MFR	600
MFSDC(distance,mode)	603
MFSLEW(distance[,m/s])	605
MH	607
MINV(arg)	608
MODE	610
RMODE	610
MP	613

MSO	616
MSR	618
MT	620
MTB	622
MV	624
NMT	626
N/A	626
O=formula, O(trj#)=formula	628
OC(...)	630
ROC(...)	630
OCHN(...)	632
OF(...)	634
ROF(...)	634
OFF	636
OR(value)	638
OS(...)	640
OSH=formula, OSH(trj#)=formula	642
OUT(...)=formula	644
PA	646
RPA	646
PAUSE	648
PC, PC(axis)	650
RPC, RPC(axis)	650
PI	653
RPI	653
PID#	654
PMA	657
RPMA	657
PML=formula	659
RPML	659
PMT=formula	661
RPMT	661
PRA	663
RPRA	663
PRC	666
RPRC	666
PRINT(...)	669
PRINTO(...)	673

PRINT1(...)	677
PRINT8(...)	680
PRT=formula	683
RPRT	683
PRTS(...)	685
PRTSS(...)	688
PT=formula	690
RPT	690
PTS(...)	692
PTSD	695
RPTSD	695
PTSS(...)	696
PTST	698
RPTST	698
RANDOM=formula	699
RRANDOM	699
RCKS	701
RES	702
RRES	702
RESUME	704
RETURN	706
RETURNI	708
RSP	710
RSP1	712
RSP5	713
RUN	714
RUN?	716
S (as command)	718
SADDR#	720
SAMP	722
RSAMP	722
SCALEA(m,d)	724
SCALEP(m,d)	726
SCALEV(m,d)	728
SDORD(...)	730
RSDORD	730
SDOWR(...)	732
SILENT	734

SILENT1	736
SIN(value)	738
RSIN(value)	738
SLD	740
SLE	742
SLEEP	744
SLEEP1	746
SLM(mode)	748
RSLM	748
SLN=formula	750
RSLN	750
SLP=formula	752
RSLP	752
SNAME("string")	754
SP2	755
RSP2	755
SP6	756
RSP6	756
SQRT(value)	757
RSQRT(value)	757
SRC(enc_src)	759
STACK	761
STDOUT=formula	764
SWITCH formula	766
T=formula	769
RT	769
TALK	771
TALK1	773
TAN(value)	775
RTAN(value)	775
TEMP, TEMP(arg)	777
RTEMP, RTEMP(arg)	777
TH=formula	779
RTH	779
TMR(timer,time)	782
RTMR(timer)	782
TRQ	784
RTRQ	784

TS=formula	786
RTS	786
TSWAIT	788
TWAIT(gen#)	789
UIA	791
RUIA	791
UJA	793
RUJA	793
UO(...)=formula	795
UP	797
UPLOAD	799
UR(...)	801
US(...)	803
USB(arg)	805
RUSB	805
VA	807
RVA	807
VAC(arg)	810
VC	815
RVC	815
VL=formula	818
RVL	818
VLD(variable,number)	820
VST(variable,number)	824
VT=formula	828
RVT	828
VTS=formula	831
W(word)	833
RW(word)	833
WAIT=formula	835
WAKE	837
WAKE1	839
WHILE formula	841
X	844
Z	846
Z(word,bit)	848
Za	850
Ze	851

Zh	852
Zl	853
Zls	854
Zr	855
Zrs	856
Zs	857
ZS	858
Zv	860
Zw	861
Part 3: Example SmartMotor Programs	862
Move Back and Forth	863
Move Back and Forth with Watch	863
Home Against a Hard Stop (Basic)	864
Home Against a Hard Stop (Advanced)	864
Home Against a Hard Stop (Two Motors)	865
Home to Index Using Different Modes	867
Maintain Velocity During Analog Drift	868
Long-Term Storage of Variables	869
Find Errors and Print Them	869
Change Speed on Digital Input	870
Pulse Output on a Given Position	870
Stop Motion if Voltage Drops	871
Camming - Variable Cam Example	872
Camming - Fixed Cam with Input Variables	873
Camming - Demo XY Circle	875
Chevron Traverse & Takeup	877
CAN Bus - Timed SDO Poll	879
CAN Bus - I/O Block with PDO Poll	880
CAN Bus - Time Sync Follow Encoder	883
Text Replacement in an SMI Program	891
Appendix	893
Motion Command Quick Reference	895
Array Variable Memory Map	897
ASCII Character Set	899
Binary Data	900
Commands Affected by SCALE	903

Command Error Codes	906
Decoding the Error	906
Finding the Error Source	907
Glossary	908
Math Operators	915
Moment of Inertia	916
Matching Motor to Load	916
Improving the Moment of Inertia Ratio	916
RCAN, RCHN and RMODE Status	917
RCAN Status Decoder	917
RCHN Status Decoder	917
Clearing Serial Port Errors	918
RMODE Status Decoder	918
Mode Status Example	918
Scale Factor Calculation	919
Sample Rates	919
PID Sample Rate Command	919
Encoder Resolution and the RES Parameter	919
Native Velocity and Acceleration Units	920
Velocity Calculations	920
Acceleration Calculations	920
Status Words - SmartMotor	921
Status Word 0: Primary Fault/Status Indicator	921
Status Word 1: Index Registration and Software Travel Limits	922
Status Word 2: Communications, Program and Memory	922
Status Word 3: PID State, Brake, Move Generation Indicators	923
Status Word 4: Interrupt Timers	923
Status Word 5: Interrupt Status Indicators	924
Status Word 6: Drive Modes	924
Status Word 7: Multiple Trajectory Support	925
Status Word 8: Cam Support	926
Status Word 9: No Bits Defined (Class 5 Only)	926
Status Word 9: SD Card and DMX Information (Class 6 Only)	926
Status Word 10: RxPDO Arrival Notification	927
Status Word 12: DMX Information (Class 5 Only)	928
Status Word 12: User Bits Word 0 (Class 6 Only)	928
Status Word 13: User Bits Word 1	929
Status Word 16: On Board Local I/O Status: D-Style Motor	929

Status Word 16: On Board Local I/O Status: M-Style Class 5 Motor	930
Status Word 16: On Board Local I/O Status - Class 6 Motor	930
Status Word 17: Expanded I/O Status - D-Style AD1 Motor	931
Fault and Status Words - DS2020 Combitronic System	932
Fault Words	932
Fault Tables	932
Fault Word 0	933
Fault Word 1	933
Fault Word 2	934
Status Words	934
Status Word 0: Primary Fault/Status Indicator	934
Status Word 1: Current CiA DS402 State	935
Status Word 2: Control and Hardware Faults	935
Status Word 3: Position/Velocity sensor and Brake Feedback Faults	935
Status Word 4: Communication Faults	936
Status Word 5: Software and Memory Faults	936
Status Word 6: I/O States	937
Torque Curves	938
Understanding Torque Curves	938
Peak Torque	938
Continuous Torque	938
Ambient Temperature Effects on Torque Curves and Motor Response:	939
Supply Voltage Effects on Torque Curves and Motor Response:	939
Example 1: Rotary Application	940
Example 2: Linear Application	940
Dyno Test Data vs. the Derated Torque Curve	940
Proper Sizing and Loading of the SmartMotor	941
SmartMotor Troubleshooting	943
Troubleshooting - First Steps	943
Commands Listed Alphabetically	946
Commands Listed by Function	954
Communications Control	955
Data Conversion	956
EEPROM (Nonvolatile Memory)	956
I/O Control	956
Math Function	957
Motion Control	957

Program Access	960
Program Execution and Flow Control	960
Reset Commands	961
System	961
Variables	962
Commands for Combitronic	963
Commands for DS2020 Combitronic	967

Introduction

This chapter provides introductory reference material.

Overview	29
Combitronic Support	29
Communication Lockup Wizard	31
Safety Information	31
Additional Documents	35
Additional Resources	36

Overview

The SmartMotor™ Developer's Guide is designed to be used by system developers and programmers when developing applications for the SmartMotor. Before using the SmartMotor™ Developer's Guide, it is strongly recommended that you first read the *SmartMotor™ Installation & Startup Guide* for your SmartMotor, which describes how to install and start up the SmartMotor, and test initial communications with the motor. After that, use this guide to learn about advanced SmartMotor features, how to develop SmartMotor applications, and the details of each command.

Part One of this guide provides information on basic to advanced programming, along with related information on key SMI software features, communications, motion control, program flow control, error and fault handling, and more.

Part Two of this guide lists all the SmartMotor commands in alphabetical order. Each command is described in detail. Code snippets and examples are provided where applicable. These are shown in a Courier font. Comments are included and separated with a single quotation mark as they would be in your own programs.

NOTE: The programs and code samples in this manual are provided for example purposes only. It is the user's responsibility to decide if a particular code sample or program applies to the application being developed and to adjust the values to fit that application.

Also, where appropriate, a Related Commands section is included, which is located at the end of the command page. It is designed to guide you to other commands that offer similar functionality, and ensure you are aware of every programming option the SmartMotor provides to address your specific application requirements.

Part Three of this guide provides a library of useful example SmartMotor programs. These can be used as "how to" examples for using a particular SmartMotor feature or solving a particular application problem, or as starting points for your application.

NOTE: The programs and code samples in this manual are provided for example purposes only. It is the user's responsibility to decide if a particular code sample or program applies to the application being developed and to adjust the values to fit that application.

The Appendix of this guide contains additional topics such as an array map, ASCII character set, command error codes, and other information that is useful to have handy during application development.

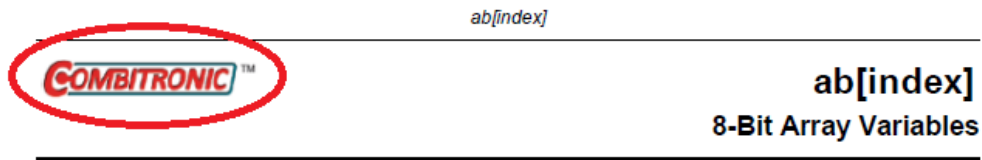
A quick-reference command list is also included at the end of this guide.

Combitronic Support

NOTE: For the Class 5 D- and M-style SmartMotors, Combitronic communication is available on models with the -CAN option. For the Class 6 D-style SmartMotor, Combitronic communication is a standard feature on all models. For the Class 6 M-style SmartMotor, Combitronic communication is currently available only on -EIP option motors. For details, see the *Class 6 SmartMotor™ EtherNet/IP Guide*.

A large number of the commands provide Combitronic™ support. Combitronic is a protocol that operates over a standard "CAN" (Controller Area Network) interface. It may coexist with either CANopen or DeviceNet protocols at the same time. Unlike these common protocols, however, Combitronic requires no single dedicated controller¹ to operate. Each Integrated Servo connected to the same network communicates on an equal footing, sharing all information, and therefore, sharing all processing resources. For more details on Combitronic features, see Combitronic Communications on page 113, and also see the overview on the Moog Animatics website at: <https://www.animatics.com/support/combitronic.html>.

For applicable commands, a table row titled "COMBITRONIC" provides the Combitronic command syntax for addressing a specific SmartMotor in the network. Those commands also display the Combitronic logo (**COMBITRONIC™**) at the top of their reference pages.



Combitronic Logo Location

FIRMWARE VERSION:	4.00 or higher
COMBITRONIC SYNTAX:	ab[0]:3=34 where "3" is the motor address; you can use the actual address or a variable.

COMBITRONIC: Table Row

Combitronic with the DS2020 Combitronic System

NOTE: DS2020 support requires: 5.0.4.55 (D), 5.98.4.55 (M); 6.4.2.x (D); ds2020_sa_1.0.0_combican (DS2020).

The Moog Animatics DS2020 Combitronic system is a cabinet mount servo drive connected to a Moog Compact Dynamic brushless servo motor. Compared to the smaller 17 to 34 frame SmartMotor products, the DS2020 Combitronic system provides access to a higher torque motor-drive combination, with torque range and power inputs to include AC mains voltages and motors above 1 KW. However, similar to other SmartMotor products, the DS2020 Combitronic system has the capability of responding to Combitronic commands.

The DS2020 Combitronic system is not fully programmable but is connected as a follower device to a SmartMotor controller. The DS2020 Combitronic system has a CAN address, which you can set through SMI along with baud rates as you would with any SmartMotor. It is then commanded by the SmartMotor through Combitronic communications using standard Combitronic syntax, e.g., ADT:3=1234, where "3" is the CAN address of the DS2020 Combitronic system.

The DS2020 Combitronic system supports a subset of the full AniBasic command set. Supported commands are primarily Combitronic type, but there are a few others, also. The DS2020 Combitronic system supported commands are flagged with "; supports the DS2020 Combitronic system" text on the command's APPLICATION line or READ/REPORT line.

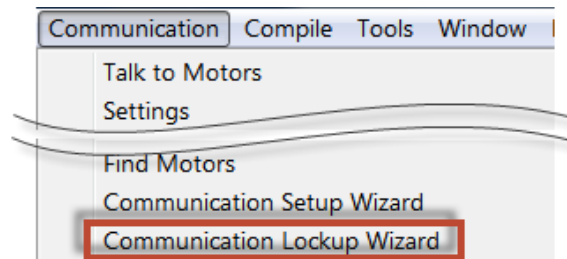
For a list of DS2020 Combitronic system supported commands, see *Commands for DS2020 Combitronic* on page 967

For details on the DS2020 Combitronic system installation and startup, see the *DS2020 Combitronic Installation and Startup Guide*.

1. Moog Animatics has replaced the terms "master" and "slave" with "controller" and "follower", respectively.

Communication Lockup Wizard

Improper use of some commands, like Z and OCHN, can lock you out of the motor and prevent further communication. If you are unable to communicate with the SmartMotor, you may be able to recover communications using the Communication Lockup Wizard, which is on the SMI software Communications menu (see the next figure). This tool sends an "E" character to the motor at startup, which prevents the motor from running its program. For more details on the Communication Lockup Wizard, see the SMI software online help, which is accessed by pressing the F1 key or selecting Help from the SMI software main menu.



Communication Menu - Communication Lockup Wizard

Safety Information

This section describes the safety symbols and other safety information.

Safety Symbols

The manual may use one or more of these safety symbols:



WARNING: This symbol indicates a potentially nonlethal mechanical hazard, where failure to comply with the instructions could result in serious injury to the operator or major damage to the equipment.



CAUTION: This symbol indicates a potentially minor hazard, where failure to comply with the instructions could result in slight injury to the operator or minor damage to the equipment.

NOTE: Notes are used to emphasize non-safety concepts or related information.

Other Safety Considerations

The Moog Animatics SmartMotors are supplied as components that are intended for use in an automated machine or system. As such, it is beyond the scope of this manual to attempt to cover all the safety standards and considerations that are part of the overall machine/system design and manufacturing safety. Therefore, this information is intended to be used only as a general guideline for the machine/system designer.

It is the responsibility of the machine/system designer to perform a thorough "Risk Assessment" and to ensure that the machine/system and its safeguards comply with the safety standards specified by the governing authority (for example, ISO, OSHA, UL, etc.) for the site where the machine is being installed and operated. For more details, see Machine Safety on page 32.

Motor Sizing

It is the responsibility of the machine/system designer to select SmartMotors that are properly sized for the specific application. Undersized motors may: perform poorly, cause excessive downtime or cause unsafe operating conditions by not being able to handle the loads placed on them. The *System Best Practices* document, which is available on the Moog Animatics website, contains information and equations that can be used for selecting the appropriate motor for the application.

Replacement motors must have the same specifications and firmware version used in the approved and validated system. Specification changes or firmware upgrades require the approval of the system designer and may require another Risk Assessment.

Environmental Considerations

It is the responsibility of the machine/system designer to evaluate the intended operating environment for dust, high-humidity or presence of water (for example, a food-processing environment that requires water or steam wash down of equipment), corrosives or chemicals that may come in contact with the machine, etc. Moog Animatics manufactures specialized IP-rated motors for operating in extreme conditions. For details, see the *Moog Animatics Product Catalog*.

Machine Safety

In order to protect personnel from any safety hazards in the machine or system, the machine/system builder must perform a "Risk Assessment", which is often based on the ISO 13849 standard. The design/implementation of barriers, emergency stop (E-stop) mechanisms and other safeguards will be driven by the Risk Assessment and the safety standards specified by the governing authority (for example, ISO, OSHA, UL, etc.) for the site where the machine is being installed and operated. The methodology and details of such an assessment are beyond the scope of this manual. However, there are various sources of Risk Assessment information available in print and on the internet.

NOTE: The next list is an example of items that would be evaluated when performing the Risk Assessment. Additional items may be required. The safeguards must ensure the safety of all personnel who may come in contact with or be in the vicinity of the machine.

In general, the machine/system safeguards must:

- Provide a barrier to prevent unauthorized entry or access to the machine or system. The barrier must be designed so that personnel cannot reach into any identified danger zones.
- Position the control panel so that it is outside the barrier area but located for an unrestricted view of the moving mechanism. The control panel must include an E-stop mechanism. Buttons that start the machine must be protected from accidental activation.
- Provide E-stop mechanisms located at the control panel and at other points around the perimeter of the barrier that will stop all machine movement when tripped.
- Provide appropriate sensors and interlocks on gates or other points of entry into the protected zone that will stop all machine movement when tripped.
- Ensure that if a portable control/programming device is supplied (for example, a hand-held operator/programmer pendant), the device is equipped with an E-stop mechanism.

NOTE: A portable operation/programming device requires *many* additional system design considerations and safeguards beyond those listed in this section. For details, see the safety standards specified by the governing authority (for example, ISO, OSHA, UL, etc.) for the site where the machine is being installed and operated.

- Prevent contact with moving mechanisms (for example, arms, gears, belts, pulleys, tooling, etc.).

- Prevent contact with a part that is thrown from the machine tooling or other part-handling equipment.
- Prevent contact with any electrical, hydraulic, pneumatic, thermal, chemical or other hazards that may be present at the machine.
- Prevent unauthorized access to wiring and power-supply cabinets, electrical boxes, etc.
- Provide a proper control system, program logic and error checking to ensure the safety of all personnel and equipment (for example, to prevent a run-away condition). The control system must be designed so that it does not automatically restart the machine/system after a power failure.
- Prevent unauthorized access or changes to the control system or software.

Documentation and Training

It is the responsibility of the machine/system designer to provide documentation on safety, operation, maintenance and programming, along with training for all machine operators, maintenance technicians, programmers, and other personnel who may have access to the machine. This documentation must include proper lockout/tagout procedures for maintenance and programming operations.

It is the responsibility of the operating company to ensure that:

- All operators, maintenance technicians, programmers and other personnel are tested and qualified before acquiring access to the machine or system.
- The above personnel perform their assigned functions in a responsible and safe manner to comply with the procedures in the supplied documentation and the company safety practices.
- The equipment is maintained as described in the documentation and training supplied by the machine/system designer.

Additional Equipment and Considerations

The Risk Assessment and the operating company's standard safety policies will dictate the need for additional equipment. In general, it is the responsibility of the operating company to ensure that:

- Unauthorized access to the machine is prevented at all times.
- The personnel are supplied with the proper equipment for the environment and their job functions, which may include: safety glasses, hearing protection, safety footwear, smocks or aprons, gloves, hard hats and other protective gear.
- The work area is equipped with proper safety equipment such as first aid equipment, fire suppression equipment, emergency eye wash and full-body wash stations, etc.
- There are no modifications made to the machine or system without proper engineering evaluation for design, safety, reliability, etc., and a Risk Assessment.

Safety Information Resources

Additional SmartMotor safety information can be found on the Moog Animatics website; open the topic "Controls - Notes and Cautions" located at:

<https://www.animatics.com/support/downloads/knowledgebase/controls---notes-and-cautions.html>

OSHA standards information can be found at:

<https://www.osha.gov/law-regs.html>

ANSI-RIA robotic safety information can be found at:

<http://www.robotics.org/robotic-content.cfm/Robotics/Safety-Compliance/id/23>

UL standards information can be found at:

<http://ulstandards.ul.com/standards-catalog/>

ISO standards information can be found at:

<http://www.iso.org/iso/home/standards.htm>

EU standards information can be found at:

http://ec.europa.eu/growth/single-market/european-standards/harmonised-standards/index_en.htm

Additional Documents

The Moog Animatics website contains additional documents that are related to the information in this manual. Please refer to these lists.

Related Guides

- Moog Animatics SmartMotor™ Installation and Startup Guides
<http://www.animatics.com/install-guides>
- *SmartMotor™ Homing Procedures and Methods Application Note*
<http://www.animatics.com/homing-application-note>
- *SmartMotor™ System Best Practices Application Note*
<http://www.animatics.com/system-best-practices-application-note>

In addition to the documents listed above, guides for fieldbus protocols and more can be found on the website: <https://www.animatics.com/support/downloads.manuals.html>

Other Documents

- SmartMotor™ Certifications
<https://www.animatics.com/certifications.html>
- *SmartMotor Developer's Worksheet*
(interactive tools to assist developer: Scale Factor Calculator, Status Words, CAN Port Status, Serial Port Status, RMODE Decoder and Syntax Error Codes)
<https://www.animatics.com/support/downloads.knowledgebase.html>
- *Moog Animatics Product Catalog*
<http://www.animatics.com/support/moog-animatics-catalog.html>

Additional Resources

The Moog Animatics website contains useful resources such as product information, documentation, product support and more. Please refer to these addresses:

- General company information:
<http://www.animatics.com>
- Product information:
<http://www.animatics.com/products.html>
- Product support (Downloads, How-to Videos, Forums and more):
<http://www.animatics.com/support.html>
- Contact information, distributor locator tool, inquiries:
<https://www.animatics.com/contact-us.html>
- Applications (Application Notes and Case Studies):
<http://www.animatics.com/applications.html>

Part 1: Programming the SmartMotor

Part 1 of this guide provides information on programming, SMI software features, communications, variables, error and fault handling, I/O control, and other details required for system and application development.

Beginning Programming	47
Understanding Firmware Versions	48
Downloading and Installing the Latest Firmware	48
Understanding the FIRMWARE VERSION Field in the Command Descriptions	48
Class 5 Firmware for D- and M-Style Motors	48
Class 6 Firmware for M-Style (MT/MT2) Motors	49
Class 6 Firmware for D-Style Motors	49
DS2020 Combitronic System Firmware	49
Setting the Motor Firmware Version in SMI	50
Setting the Default Firmware Version	50
Checking the Default Firmware Version	51
Opening the SMI Window (Program Editor)	51
Understanding the Program Requirements	52
Creating a "Hello World" Program	54
Entering the Program in the SMI Editor	54
Adding Comments to the Code	54
Checking the Program Syntax	54
Saving the Program	55
Downloading a Program to the SmartMotor	55
Syntax Checking, Compiling and Downloading the Program	55
Additional Notes on Downloaded Programs	55
Running a Downloaded Program	56
Using the Program Download Window	57
Using the Terminal Window and Run Program Button	57
Using the RUN Command in the Terminal Window	57
Creating a Simple Motion Program	59
SMI Software Features	60
Introduction	61
Menu Bar	62
Toolbar	62
Configuration Window	64
Terminal Window	67
Initiating Motion from the Terminal Window	69

Information Window	69
Program Editor	70
Motor View	72
SMI Trace Functions	73
Monitor Window	76
Serial Data Analyzer	78
Chart View	79
Chart View Example	80
Macros (Keyboard Shortcuts or Hotkeys)	83
Tuner	85
SMI Options	89
SMI Help	90
Context-Sensitive Help Using F1	90
Context-Sensitive Help Using the Mouse	90
Help Buttons	90
Hover Help	90
Table of Contents	90
Projects	91
SmartMotor Playground	92
Opening the SmartMotor Playground	93
Moving the Motor	94
Communication Details	96
Introduction	98
Connecting to a Host	99
Daisy Chaining Multiple D-Style SmartMotors over RS-232	100
ADDR=formula	102
SLEEP, SLEEP1	102
WAKE, WAKE1	102
ECHO, ECHO1	103
ECHO_OFF, ECHO_OFF1	103
Serial Commands	104
OCHN(type,channel,parity,bit rate,stop bits,data bits,mode,timeout)	104
CCHN(type,channel)	105
BAUDrate, BAUD(channel)=formula	105
PRINT(), PRINT1()	105
SILENT, SILENT1	106
TALK, TALK1	106
a=CHN(channel)	106
a=ADDR	106

Communicating over RS-485	107
Using Data Mode	107
CAN Communications	110
CADDR=formula	110
CBAUD=formula	110
=CAN, =CAN(arg)	110
CANCTL(function,value)	110
SDORD(...)	111
SDOWR(...)	111
NMT	112
RB(2,4), x=B(2,4)	112
Exceptions to NMT, SDORD and SDOWR Commands	112
I/O Device CAN Bus Controller	113
Combitronic Communications	113
Combitronic Features	114
Other Combitronic Benefits	114
Program Loops with Combitronic	115
Global Combitronic Transmissions	115
Simplify Machine Support	116
Combitronic with RS-232 Interface	116
Combitronic with the DS2020 Combitronic System	117
Other CAN Protocols	118
CANopen - CAN Bus Protocol	118
DeviceNet - CAN Bus Protocol	118
I ² C Communications (Class 5 D-Style Motors)	118
OCHN(IIC,1,N,baud,1,8,D)	120
CCHN(IIC,1)	120
PRINT1(arg1,arg2, ... ,arg_n)	120
RGETCHR1, Var=GETCHR1	120
RLEN1, Var=LEN1	120
Motion Details	121
Introduction	122
Motion Command Quick Reference	123
Basic Motion Commands	124
Target Commands	124
PT=formula	124
PRT=formula	125
ADT=formula	125
AT=formula	125

DT=formula	125
VT=formula	125
Motion Mode Commands	126
MP	126
MV	126
MT	126
Torque Commands	127
TS=formula	127
T=formula	127
Brake Commands	127
BRKRLS	127
BRKENG	127
BRKSRV	128
BRKTRJ	128
Brake Command Examples	128
EOBK(IO)	129
MTB	130
Index Capture Commands	130
DS2020 Combitronic System Index Capture	131
Other Motion Commands	132
G	132
S	132
X	132
O=formula	133
OSH=formula	133
OFF	133
SCALEA(m,d), SCALEP(m,d), SCALEV(m,d)	133
Commutation Modes	134
MDT	134
MDE	134
MDS	134
MDC	135
MDB	135
MINV(0), MINV(1)	135
Modes of Operation	136
Torque Mode	136
Torque Mode Example	136
Dynamically Change from Velocity Mode to Torque Mode	136
Velocity Mode	137
Constant Velocity Example	137

Change Commanded Speed and Acceleration	137
Absolute (Position) Mode	138
Absolute Move Example	138
Two Moves with Delay Example	138
Change Speed and Acceleration Example	138
Shift Point of Origin Example	139
Relative Position Mode	139
Relative Mode Example	139
Follow Mode with Ratio (Electronic Gearing)	140
Electronic Gearing and Camming over CANopen	140
Electronic Gearing Commands	140
SRC(enc_src)	141
MFR	141
MSR	141
MF0	141
MS0	141
MFMUL=formula, MFDIV=formula	141
MFA(distance[,m/s])	142
MFD(distance[,m/s])	142
MFSLEW(distance[,m/s])	142
Follow Internal Clock Source Example	142
Follow Incoming Encoder Signal With Ramps Example	143
Electronic Line Shaft	145
ENCD(in_out)	145
Spooling and Winding Overview	146
Relative Position, Auto-Traverse Spool Winding	146
MFSDC(distance,mode)	147
Dedicated, Absolute Position, Winding Traverse Commands	149
MFSDC(distance,2)	150
MFLTP=formula	150
MFHTP=formula	150
MFCTP(arg1,arg2)	150
MFL(distance[,m/s])	151
MFH(distance[,m/s])	151
ECS(counts)	151
Single Trajectory Example Program	152
Chevron Wrap Example	153
Other Traverse Mode Notes	155
Traverse Mode Status Bits	156
Cam Mode (Electronic Camming)	156

Electronic Camming Details	158
Understanding the Inputs	158
Should I choose Source Counts or Intermediate Counts?	160
Should I choose Variable or Fixed cam?	160
Electronic Camming Notes and Best Practices	162
Examples	164
Electronic Gearing and Camming over CANopen	164
Electronic Camming Commands	165
CTE(table)	165
CTA(points,seglen[,location])	165
CTW(pos[,seglen][,user])	165
MCE(arg)	166
MCW(table,point)	166
RCP	166
RCTT	167
MC	167
MCMUL=formula	167
MCDIV=formula	167
O(arg)=formula	167
OSH(arg)=formula	167
Cam Example Program	168
Mode Switch Example	171
Position Counters	173
Modulo Position	174
Modulo Position Commands	174
Dual Trajectories	175
Commands That Read Trajectory Information	177
Dual Trajectory Example Program	178
Using Mixed Mode Operations After Homing	179
Synchronized Motion	179
Synchronized-Target Commands	179
PTS(), PRTS()	179
VTS=formula	180
ADTS=formula, ATS=formula, DTS=formula	180
PTSS(), PRTSS()	180
A Note About PTS and PRTS	181
Other Synchronized-Motion Commands	183
GS	183
TSWAIT	183

Program Flow Details	185
Introduction	186
Flow Commands	186
RUN	186
RUN?	187
GOTO#, GOTO(label), C#	187
GOSUB#, GOSUB(label), RETURN	188
IF, ENDIF	188
ELSE, ELSEIF	188
WHILE, LOOP	189
SWITCH, CASE, DEFAULT, BREAK, ENDS	190
TWAIT	190
WAIT=formula	191
STACK	191
END	191
Program Flow Examples	192
IF, ELSEIF, ELSE, ENDIF Examples	192
WHILE, LOOP Examples	192
GOTO(), GOSUB() Examples	193
SWITCH, CASE, BREAK, ENDS Examples	194
Interrupt Programming	195
ITR(), ITRE, ITRD, EITR(), DITR(), RETURNI	195
TMR(timer,time)	197
Variables and Math	198
Introduction	199
Variable Commands	199
EPTR=formula	199
VST(variable,number)	199
VLD(variable,number)	200
Math Expressions	200
Math Operations	200
Logical Operations	200
Integer Operations	200
Floating Point Functions	200
Math Operation Details and Examples	201
Array Variables	201
Array Variable Examples	202

Error and Fault Handling Details	203
Motion and Motor Faults	204
Overview	204
Drive Stage Indications and Faults	204
Fault Bits	204
Error Handling	205
Example Fault-Handler Code	205
PAUSE	206
RESUME	206
Limits and Fault Handling	207
Position Error Limits	207
dE/dt Limits	207
Velocity Limits	208
Hardware Limits	208
Software Limits	208
Fault Handling	209
Monitoring the SmartMotor Status	210
System Status	213
Introduction	214
Retrieving and Manipulating Status Words/Bits	214
System and Motor Status Bits	214
Reset Error Flags	217
System Status Examples	217
Timer Status Bits	218
Interrupt Status Bits	218
I/O Status	219
User Status Bits	219
Multiple Trajectory Support Status Bits	220
Cam Status Bits	221
Interpolation Status Bits	222
Motion Mode Status	222
RMODE, RMODE(arg)	222
I/O Control Details	223
I/O Port Hardware	224
I/O Connections Example (Class 5 D-Style Motors)	225
I/O Voltage Protection	225
Discrete Input and Output Commands	225
Discrete Input Commands	226

Discrete Output Commands	226
Output Condition and Fault Status Commands	227
Output Condition Commands	227
Output Fault Status Reports	227
General-Use Input Configuration	228
Multiple I/O Functions Example	228
Analog Functions of I/O Ports	230
5 Volt Push-Pull I/O Analog Functions (Class 5 D-Style Motors)	230
24 Volt I/O Analog Functions (Class 5 D-Style AD1 Option Motors, Class 5 M-Style Motors)	230
24 Volt I/O Analog Functions (Class 6 M-Style Motors)	230
24 Volt I/O Analog Functions (Class 6 D-Style Motors)	231
Special Functions of I/O Ports	232
Class 5 D-Style Motors: Special Functions of I/O Ports	233
I/O Ports 0 and 1 - External Encoder Function Commands	233
I/O Ports 2 and 3 - Travel Limit Inputs	233
I/O Ports 4 and 5 - Communications	233
I/O Port 6 - Go Command, Encoder Index Capture Input	234
Class 5 M-Style Motors: Special Functions of I/O Ports	235
COM Port Pins 4, 5, 6, and 8 - A-quad-B or Step-and-Direction Modes	235
I/O Ports 2 and 3 - Travel Limit Inputs	235
I/O Port 5 - Encoder Index Capture Input	235
I/O Port 6 - Go Command	236
Class 6 Motors: Special Functions of I/O Ports	237
A-quad-B or Step-and-Direction Modes	237
I/O Ports 2 and 3 - Travel Limit Inputs	238
I/O Port 4 and 5 - Encoder Index Capture Input	238
I/O Port 6 - Go Command	238
I/O Brake Output Commands	238
I ² C Expansion (D-Style Motors)	239
Tuning and PID Control	240
Introduction	241
Tuning and PID Control on the DS2020 Combitronic System	241
Understanding the PID Control	241
Tuning the PID Control	242
Using F	243
Setting KP	243
Setting KD	243
Setting KI and KL	244
Setting EL=formula	244

Other PID Tuning Parameters	244
KG=formula	245
KV=formula	245
KA=formula	245
Current Limit Control	246
AMPS=formula	246

Beginning Programming

This chapter provides information on beginning programming with the SmartMotor. It introduces you to using the SMI™ Program Editor, understanding program requirements, creating a program, downloading the program and then running it in the SmartMotor. It concludes with a sample for creating your first motion program.

Understanding Firmware Versions	48
Downloading and Installing the Latest Firmware	48
Understanding the FIRMWARE VERSION Field in the Command Descriptions	48
Class 5 Firmware for D- and M-Style Motors	48
Class 6 Firmware for M-Style (MT/MT2) Motors	49
Class 6 Firmware for D-Style Motors	49
DS2020 Combitronic System Firmware	49
Setting the Motor Firmware Version in SMI	50
Setting the Default Firmware Version	50
Checking the Default Firmware Version	51
Opening the SMI Window (Program Editor)	51
Understanding the Program Requirements	52
Creating a "Hello World" Program	54
Entering the Program in the SMI Editor	54
Adding Comments to the Code	54
Checking the Program Syntax	54
Saving the Program	55
Downloading a Program to the SmartMotor	55
Syntax Checking, Compiling and Downloading the Program	55
Additional Notes on Downloaded Programs	55
Running a Downloaded Program	56
Using the Program Download Window	57
Using the Terminal Window and Run Program Button	57
Using the RUN Command in the Terminal Window	57
Creating a Simple Motion Program	59

Understanding Firmware Versions

Before programming the SmartMotor, it is important that the correct firmware version is installed in the connected SmartMotor. This topic is intended to help you understand the differences between the firmware versions.

Downloading and Installing the Latest Firmware

It is recommended that you download and install the latest firmware for your motor. The firmware can be downloaded from the Moog Animatics website:

<https://www.animatics.com/products/smartmotor.resources.html>

The firmware files are located in the CAD File and Firmware Downloads section. In addition to the firmware files, the firmware release notes are available—these provide a succinct description of changes and enhancements for the corresponding firmware version.

When accessing the firmware downloads, note that firmware files vary depending on your motor's options (e.g., CAN, PROFIBUS, etc.). Therefore, it is important to check your motor model AND options before selecting the corresponding file.

To install the downloaded firmware in your SmartMotor, see the instructions in the SMI software online help.

Understanding the FIRMWARE VERSION Field in the Command Descriptions

The FIRMWARE VERSION field in the command description provides information about the firmware version(s) that support the command. For example, if the FIRMWARE VERSION field shows "5.x (D/M)", then the command supports any D- or M-style motor running firmware version 5 and later; if the FIRMWARE VERSION field shows "6.x (D/M)", then the command supports any D- or M-style motor running firmware version 6 and later.

In some cases, the FIRMWARE VERSION field shows a specific firmware number, for example, 5.0.4.55/5.98.4.55 (D/M), which means Class 5 D-style version 5.0.4.55 and later, or Class 5 M-style version 5.98.4.55 and later, are supported.

The next table provides more examples of FIRMWARE VERSION entries and the supported motors.

Firmware Version Examples	Supports
5.x (D/M); 6.x (D/M)	Class 5 and Class 6 D/M-style motors ¹
5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)	Class 5 and Class 6 D/M-style motors, and DS2020 Combitronic
5.x (D/M); no Class 6	Class 5 D/M-style motors only, Class 6 not supported
5.x (D/M) requires CAN option; 6.4.2.x (D)	Class 5 D/M-style motors with CAN bus, and Class 6 D-style motors ²
6.x (D/M); no Class 5	Class 6 D/M-style motors only ² , Class 5 not supported
6.x (D/M) requires EPN option; no Class 5	Class 6 D/M-style motors only ² with EPN option, Class 5 not supported
1. Class 6 D-style requires ver 6.4.2.x 2. Class 6 D-style includes CAN bus as a standard feature	

Class 5 Firmware for D- and M-Style Motors

For Class 5 SmartMotor servos, both D- and M-style, the Class 5 firmware supports most of the commands described in this guide, except those specific to only Class 6 and/or the DS2020 Combitronic system. Those exceptions are noted on the command description pages.

Class 5 firmware can be identified by the first digit "5" in the firmware version, for example, 5.0.3.2.

Noteworthy Class 5 Firmware Versions:

- 5.x.4.x - current public release series of this firmware
- 5.0.4.x series - supports D-style standard and CANopen option models
- 5.16.4.x series - supports D-style DeviceNet option models
- 5.32.4.x series - supports D-style PROFIBUS option models
- 5.97.4.x series - supports M-style DeviceNet option models
- 5.98.4.x series - supports M-style CANopen option models

For additional details, see the Class 5 D-Style Firmware Release Notes and the Class 5 M-Style Firmware Release Notes.

Class 6 Firmware for M-Style (MT/MT2) Motors

For Class 6 M-style SmartMotor servos, both MT and MT2, the Class 6 MT/MT2 firmware supports many of the commands described in this guide, except those specific to only Class 5 and/or the DS2020 Combitronic system. Additionally, there are some commands that are unique to the Class 6 motors. Those exceptions are noted on the command description pages. For more information, see the topic "Other Class 6 D-Style Changes" in the *Class 6 SmartMotor™ Installation and Startup Guide*.

Class 6 firmware can be identified by the first digit "6" in the firmware version, for example, 6.0.2.35.

Noteworthy Class 6 MT- and MT2-Series Firmware Versions:

- 6.0.2.x - current public release series of this firmware, provides support for SM23216MH, SM23166MT, SM23166MT2 and SM34166MT2 motors

For additional details, see the Class 6 – EIP/EEC/EPN Firmware Release Notes.

Class 6 Firmware for D-Style Motors

For Class 6 D-style SmartMotor servos, the firmware supports almost all of the commands described in this guide, except those specific to DeviceNet, PROFIBUS and I²C (IIC) communications. Those exceptions are noted on the command description pages. For more information on Class 6 D-style command and feature limitations, see the topic "Other Class 6 D-Style Changes" in the *Class 6 D-Style SmartMotor™ Installation and Startup Guide*.

Class 6 D-style firmware can be identified by the first digit "6" in the firmware version, for example, 6.4.2.1.

Noteworthy Class 6 D-Style Firmware Versions:

- 6.4.2.x - current public release series of this firmware

For additional details, see the Class 6 D-Style Firmware Release Notes.

DS2020 Combitronic System Firmware

For DS2020 Combitronic system, the firmware supports a limited set of SmartMotor commands. The supported commands are noted on the corresponding command description pages. Also, for a complete list of supported commands, see Commands for DS2020 Combitronic on page 967.

DS2020 Combitronic system support requires: Class 5 ver. 5.0.4.55 (D-style) or 5.98.4.55 (M-style), or Class 6 ver. 6.4.2.x (D-style only).

NOTE: The DS2020 Combitronic system is not supported on Class 6 MT/MT2 motors.

DS2020 Combitronic system firmware can be identified by the terms "ds2020" and "combican" in the firmware version, for example, **ds2020_sa_1.0.0_combican**.

Noteworthy DS2020 Combitronic System Firmware Versions:

- ds2020_sa_1.0.0_combican - current public release of this firmware

Setting the Motor Firmware Version in SMI

NOTE: In addition to the software information in this section, there is context-sensitive help available within the SMI software interface, which is accessed by pressing the F1 key or selecting Help from the SMI software main menu.

When programming the SmartMotor, it is important that the SMI software compiler's firmware version setting matches the firmware version of the connected SmartMotor.



CAUTION: The compiler's firmware version must match the firmware version of the connected motor. If it does not match, the SMI software may not catch syntax errors and may download incompatible code to the SmartMotor.

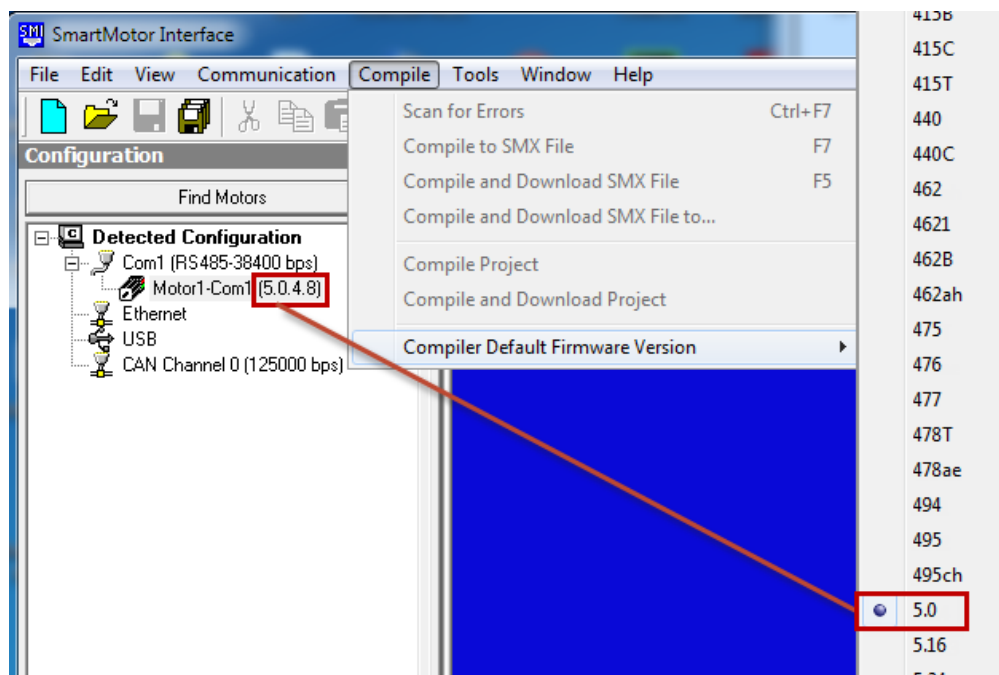
This procedure assumes that:

- The SmartMotor is connected to the computer. For details, see *Connecting the System* in the *SmartMotor Installation & Startup Guide* for your motor.
- The SmartMotor is connected to a power source. (Certain models of SmartMotors require separate control and drive power.) For details, see *Understanding the Power Requirements* in the *SmartMotor Installation & Startup Guide* for your motor.
- The SMI software has been installed and is running on the computer. For details, see *Installing the SMI Software* in the *SmartMotor Installation & Startup Guide* for your motor.

Setting the Default Firmware Version

To set the default firmware version, from the SMI software main menu, select:

Compile > Compiler default firmware version



Setting the Compiler's Default Firmware Version

From the list, select the firmware version that most closely matches the firmware version of the connected SmartMotor, as shown in the previous figure. After the default firmware version has been selected, the list closes.

Checking the Default Firmware Version

To check the default firmware version, from the SMI software main menu, select:

Compile > Compiler default firmware version

On the list, locate the blue dot to the left of the firmware version number. The dot indicates the currently-selected default firmware version.

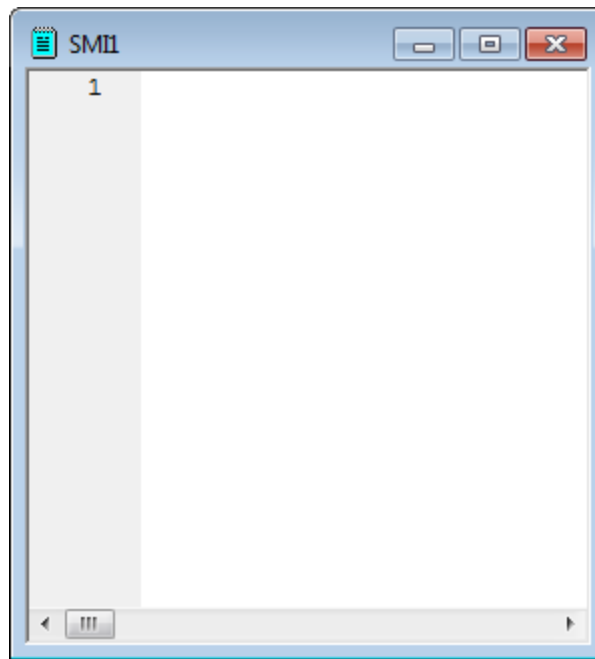
Opening the SMI Window (Program Editor)

NOTE: In addition to the software information in this section, there is context-sensitive help available within the SMI software interface, which is accessed by pressing the F1 key or selecting Help from the SMI software main menu.

In addition to taking commands over the serial interface, the SmartMotor can run programs. The SMI window is used to write and edit user programs for the SmartMotor(s). After the program has been written, it can be checked and then downloaded to the desired SmartMotor(s).

The SMI window is typically closed (default setting) when the SMI software is opened. To open the window, click the New button (📄) on the toolbar, or select:

File > New



SMI Window

After the SMI window opens, you can type your program directly into the editor, or you can copy and paste existing code from any text-based software such as Windows Notepad.

NOTE: Some word-processing software, such as Microsoft Word, has an option for "smart quotes", which use angled single (') and double (") quotation marks. The angled quotation marks are not recognized by the SMI editor. Therefore, any "smart quotes" option must be disabled before copying and pasting the program code.

Understanding the Program Requirements

SmartMotors use a simple form of code called "AniBasic", which is similar to the BASIC programming language. Various commands include means to create continuous loops, jump to different locations on given conditions and perform general math functions.

Note these AniBasic program requirements:

- The code is case sensitive:
 - All commands begin with or use all UPPER CASE letters.
 - All variables are preassigned and must use lower case.
- Command names are reserved and cannot be used as variables.
- A space is a programming element.

- Comments require an apostrophe or ASCII character 39 (') between the commands and the comment text.

NOTE: When copying and pasting code from another text editor, make sure that your text editor is not inserting "smart quotes" (angled single or double quotation marks). These are not the same as ASCII characters 39 (') and 34 ("), and the SMI program editor doesn't recognize them.
- Each program must contain at least one occurrence of the END statement.
- Each subroutine call must have a label with a RETURN statement somewhere below it.
- Each Interrupt subroutine must end with the RETURNI statement.
- The default syntax colors for the SMI editor are: commands (blue), program flow controls (pink), and comments (green). All other program text is shown in black. You can change the syntax colors through the Editor tab in the Options window. For details on the Options window, see SMI Options on page 89.
- There is no syntax checking performed until you do one of these:
 - From the main menu, select **Compile > Scan file for errors**
 - Select the Scan File for Errors button on the toolbar
 - Press Ctrl+F7
- As in BASIC, you can use the PRINT command to print to the screen, as shown in the "Hello World" example. For details, see Creating a "Hello World" Program on page 54.
- When the SmartMotor power is turned on, there is a 500 ms "pause" before any program or command is processed:
 - For all industrial networks, every node (or motor) must immediately send out a "Who am I?" info data packet when power is turned on, which tells the network host who it's talking to. This is a requirement for all industrial communications protocols (like CANopen, DeviceNet and PROFIBUS).
 - The stored program does not execute until the 500 ms pause expires. Any serial commands sent during that time are buffered and then accepted after that pause expires. Because incoming commands take priority over the internal program, any buffered commands are executed before the internal program begins.
- Commands coming in over the network have priority over the program running within the SmartMotor. For example, while a program is running, you could issue a GOSUB command from the terminal and send the program off to run the specified subroutine. When the subroutine is done, the program would resume at the point where the GOSUB command was issued.
- The RUN? command can be used at the beginning of a program to prevent it from automatically running when the SmartMotor power is turned on, as shown in the "Hello World" example. For details, see Creating a "Hello World" Program on page 54.
 - The SmartMotor will not execute any code past the RUN? line until it receives a RUN command through the serial port.
 - Using the serial port, the motor can be commanded to run subroutines even if the stored program is not running.

- User programs are stored in the SmartMotor's EEPROM memory. The maximum program size depends on the motor class you are using:
 - For Class 5 motors, the maximum program size is 32767 bytes.
 - For Class 6 motors, the maximum program size is 64150 bytes.

For details on downloading user programs to the SmartMotor, see [Downloading a Program to the SmartMotor](#) on page 55 and [LOAD](#) on page 548.

Creating a "Hello World" Program

This procedure describes how to create and save a simple "Hello World" program.

NOTE: When copying and pasting code from another text editor, make sure that your text editor is not inserting "smart quotes" (angled single or double quotation marks). These are not the same as ASCII characters 39 (') and 34 ("), and the SMI program editor doesn't recognize them.

Entering the Program in the SMI Editor

To create the program, type this code into the SMI software program editor:

```
RUN?  
PRINT("Hello World",#13)  
END
```

NOTE: The program will not run when the SmartMotor power is turned on (because of the RUN? command on the first line).

When you run this program, it outputs this text to the Terminal window:

```
Hello World
```

To run this program, you must download it to the SmartMotor and then enter the RUN command in the Terminal window. For more details on downloading the program, see [Downloading a Program to the SmartMotor](#) on page 55. For more details on running the downloaded program, see [Running a Downloaded Program](#) on page 56.

Adding Comments to the Code

You can add comments to the code by inserting a single quotation mark (') between the commands and your comment text.

NOTE: Comments do not get sent to the SmartMotor.

```
RUN?      'The program stops here until it receives a RUN command  
PRINT("Hello World",#13)  '#13 is a carriage return  
END       'The required END command
```

Checking the Program Syntax

You can syntax check the program by doing one of these:


- From the main menu, select **Compile > Scan file for errors**
- Select the Scan File for Errors button on the toolbar
- Press Ctrl+F7

If errors are found, correct them and re-check the syntax.

The program will also be syntax checked as part of the download procedure. For details, see *Downloading a Program to the SmartMotor* on page 55.

Saving the Program

After entering the program, use these steps to save it:

1. From the main menu, select: **File > Save As**, or click the Save button () on the toolbar. The Save As window opens.
2. Select a drive/folder on your PC or use the default location.
3. Assign a name, such as "HelloWorld.sms".
4. Click Save to write the program to the specified location and close the window.

If you attempt to syntax check or compile and download an unsaved program, the SMI software automatically opens the Save As window, which requires you to save the program before continuing.

Downloading a Program to the SmartMotor

NOTE: In addition to the software information in this section, there is context-sensitive help available within the SMI software interface, which is accessed by pressing the F1 key or selecting Help from the SMI software main menu.

After you've created a program, it must be downloaded to the SmartMotor. This section explains how to syntax check and download the program.


NOTE: Comments do not get sent to the SmartMotor.

Syntax Checking, Compiling and Downloading the Program

The program can be syntax checked, compiled and transmitted to the SmartMotor in one operation.

To compile the program and then transmit it to the SmartMotor:

NOTE: SMI transmits the compiled version of the program to the SmartMotor.

1. Click the Compile and Download Program button () on the toolbar or press the F5 key. The Select Motor window opens, which is used to specify which motor(s) will receive the program.
2. Select the desired motor(s) from the list. The SMI software compiles the program during this step and also checks for errors. If errors are found, make the necessary corrections and try again.
3. Click OK to close the window and transmit the program. A progress bar shows the status of the transmission.

Because the SmartMotor's EEPROM (long-term memory) is slow to write, the terminal software uses two-way communications to regulate the download of a new program.

Additional Notes on Downloaded Programs

Keep these items in mind regarding programs that have been downloaded to the SmartMotor:

- After the program has been downloaded into the SmartMotor, it remains there until replaced.
- The downloaded program executes every time power is applied to the motor.
 - There is a 500 ms timeout before the motor will accept commands on the serial port. Any commands sent during that time are buffered and then accepted once the 500 ms timeout expires. Because incoming commands take priority over the internal program, buffered commands run before the internal program begins.
 - If you do not want the program to execute every time power is applied, you must add a RUN? command as the first line/command of the program. For an example, see Creating a "Hello World" Program on page 54.
 - To get a program to operate continuously, write a loop. For details, see Program Flow Details on page 185.
- A program cannot be erased; it can only be replaced. To effectively replace a program with nothing, download a program with only one command: END.

Remember that all programs, even "empty" ones, must contain at least one END command. For more details on program requirements, see Understanding the Program Requirements on page 52.

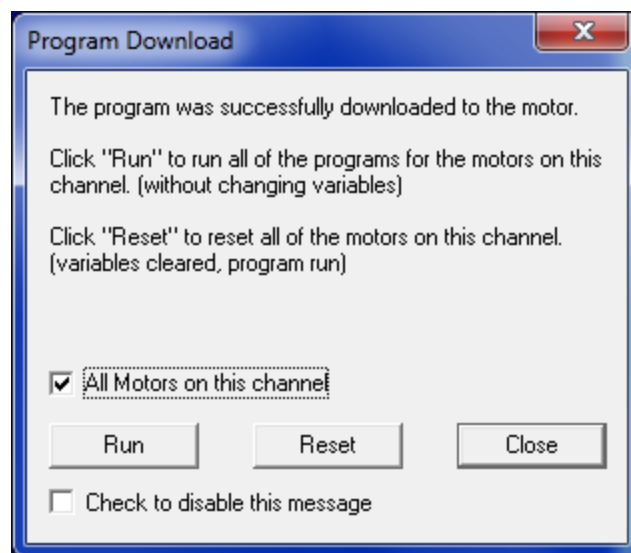
Running a Downloaded Program



WARNING: The larger SmartMotors can shake, move quickly and exert great force. Therefore, proper motor restraints must be used, and safety precautions must be considered in the workcell design (see Other Safety Considerations on page 31).

NOTE: In addition to the software information in this section, there is context-sensitive help available within the SMI software interface, which is accessed by pressing the F1 key or selecting Help from the SMI software main menu.

After the program has downloaded to the SmartMotor, the Program Download window opens, which contains options relating to running the program.



Program Download Window

Run will run the program immediately. Reset will clear all user variables and run the program as if it were power cycled. Close will close the window without running the newly-downloaded program.

"Check to disable this message" will prevent the window from being shown after a program is downloaded to the SmartMotor. Select that option if you always want to run the program using the Terminal window and the Run Program in Selected Motor button (▶), which is on the SMI software toolbar.

Using the Program Download Window

(Refer to the previous figure.)

To run the program on all motors:

1. Select the All Motors on this channel option.
2. Click Run.

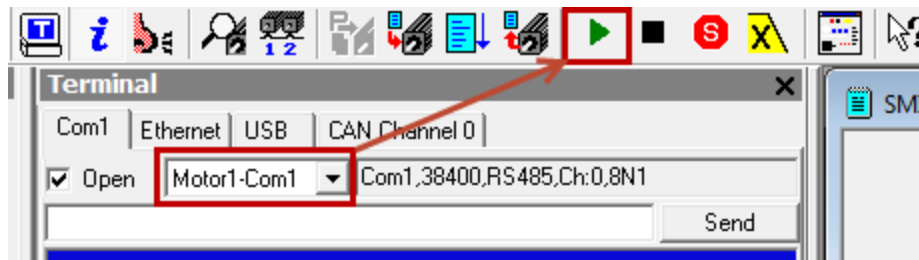
To run the program on just the selected motor:

1. Deselect the All Motors on this channel option.
2. Click Run.

Using the Terminal Window and Run Program Button

To run the program using the Terminal window and the Run Program button:

1. Use the motor selector in the Terminal window (see the next figure) to select the motor—it must be the same motor that received the program.
2. Click the Run Program in Selected Motor button (▶) to run the program in the selected motor.

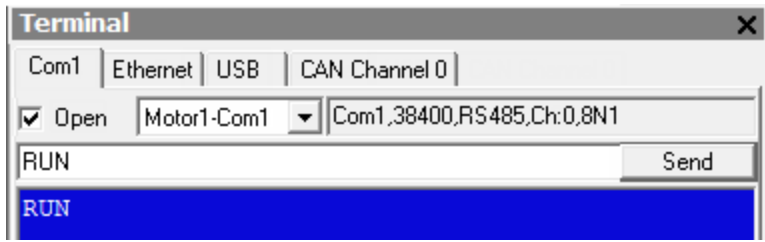


Selected Motor and Run Program Button

Using the RUN Command in the Terminal Window

To run the program using commands in the Terminal window, do one of these:

- Type RUN in the text box and click Send or press Enter
- Type RUN directly on the terminal screen (blue) area and click Send or press Enter.



RUN Command in the Terminal Window

Creating a Simple Motion Program



WARNING: The larger SmartMotors can shake, move quickly and exert great force. Therefore, proper motor restraints must be used, and safety precautions must be considered in the workcell design (see Other Safety Considerations on page 31).

Enter this motion program (see below) in the SMI editing window. Pay close attention to spaces and capitalization.

As described previously, it's only necessary to enter text on the left side of the single quote, as the text from the single quotation mark to the right end of the line is a comment and for information only. That said, it is always good programming practice to create well-commented code. Nothing is more frustrating than trying to debug or decipher code that is sparsely commented.

NOTE: Comments do not get sent to the SmartMotor.

```
EIGN (2)      'Disable left limit
EIGN (3)      'Disable right limit
ZS            'Reset errors
ADT=100       'Set target accel/decel
VT=1000000   'Set target velocity
PT=100000    'Set target position
G            'Go, starts the move
TWAIT        'Wait for move to complete
PT=0         'Set buffered move back to home
G            'Start motion
END          'End program
```

After entering the program code, you can download it to the motor and then run it. For details on downloading the program, see [Downloading a Program to the SmartMotor](#) on page 55. For details on running the downloaded program, see [Running a Downloaded Program](#) on page 56.

SMI Software Features

This chapter provides information on SMI software features.

Introduction	61
Menu Bar	62
Toolbar	62
Configuration Window	64
Terminal Window	67
Initiating Motion from the Terminal Window	69
Information Window	69
Program Editor	70
Motor View	72
SMI Trace Functions	73
Monitor Window	76
Serial Data Analyzer	78
Chart View	79
Chart View Example	80
Macros (Keyboard Shortcuts or Hotkeys)	83
Tuner	85
SMI Options	89
SMI Help	90
Context-Sensitive Help Using F1	90
Context-Sensitive Help Using the Mouse	90
Help Buttons	90
Hover Help	90
Table of Contents	90
Projects	91
SmartMotor Playground	92
Opening the SmartMotor Playground	93
Moving the Motor	94

Introduction

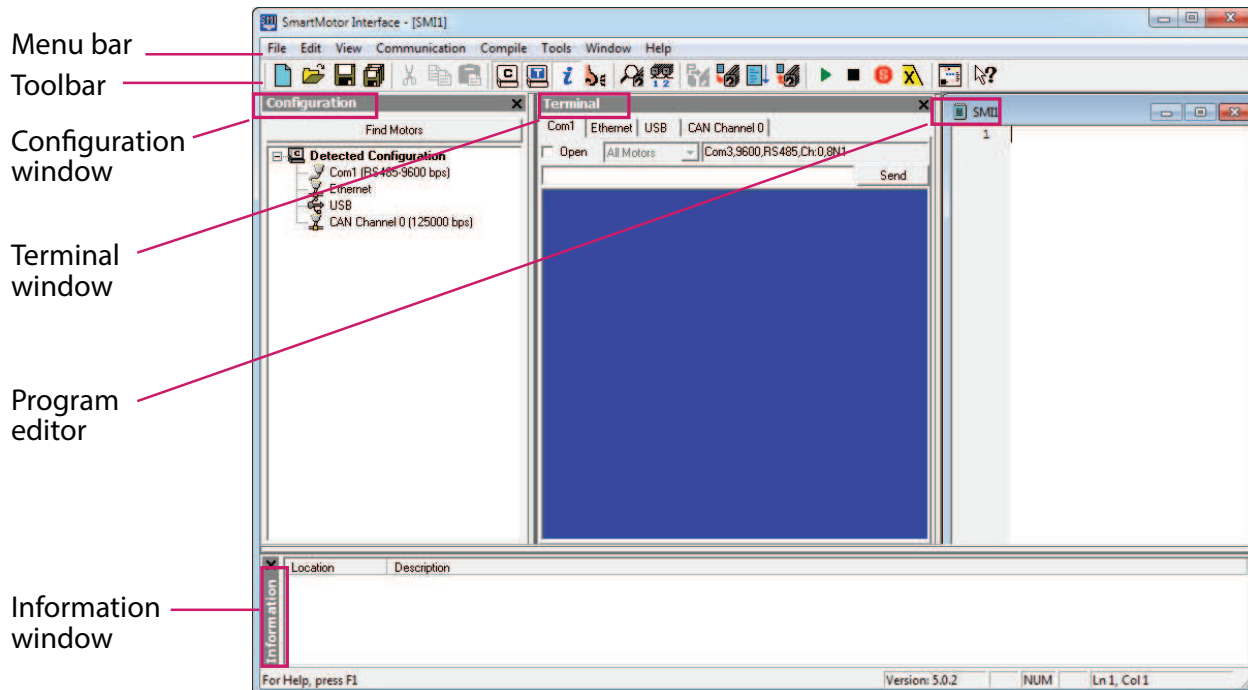
NOTE: In addition to the software information in this section, there is context-sensitive help available within the SMI software interface, which is accessed by pressing the F1 key or selecting Help from the SMI software main menu.

The SMI software interface provides access to a variety of tools that are used to communicate with, program and monitor the SmartMotor.

The SMI software also provides limited support for the DS2020 Combitronic system. These tools/features are supported:

- Tools menu items:
 - Macro
 - Motor View
 - Chart View
- Configuration tree right-click menu items:
 - Motor View
 - Set Motor Address
 - Configure DS2020

The SMI software interface can be accessed from the Windows Desktop icon or from the Windows Start menu. For details, see *Accessing the SMI Software Interface in the SmartMotor Installation & Startup Guide* for your motor.



Main Features of the SMI Software

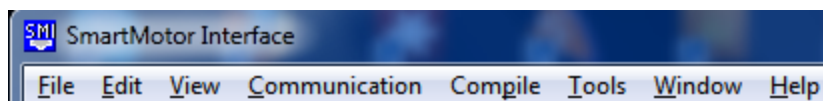
NOTE: Depending on your version of SMI software, your screens may look slightly different than those shown.

The primary software features are briefly described in the next sections. In addition to this information, there are detailed descriptions of all SMI software features in the software's online help, which can be accessed from the software's Help menu or by pressing the F1 key.

Menu Bar

NOTE: In addition to the software information in this section, there is context-sensitive help available within the SMI software interface, which is accessed by pressing the F1 key or selecting Help from the SMI software main menu.

The SMI software menu bar provides access to all SMI software features, which are grouped by functional area.



The Menu Bar

NOTE: Frequently-used features are also available from the SMI software's Toolbar. For details, see Toolbar on page 62.

Each functional area is described in the next table.

Menu	Description
File	Access standard file commands (New, Open, Close, etc.).
Edit	Edit an SMI program (Cut, Copy, Paste, etc.). Note that an SMI Program Editor window must be open to use these features.
View	Show or hide windows or items in the SMI software interface (Toolbar, Status bar, Terminal window, etc.).
Communication	Control communications with motors (Settings, Detect Motors, Upload Program, Communication Setup Wizard, etc.).
Compile	Scan a program for errors and compile SMX or project files (Scan for errors, Compile Downloadable SMX file, Compile and Transmit SMX file, Compile Project, etc.).
Tools	Access SmartMotor tools, monitoring features and options (Macro, Tuner, Motor View, Monitor View, Options, etc.).
Window	Control the appearance of the SMI software windows (Cascade, Tile Horizontally/Vertically, Arrange Icons, etc.).
Help	Access online help features of the SMI software (Contents, Index, SmartMotor Programmer's Guide, etc.).

Each menu item is described in detail in the SMI software's online help file, which can be accessed from the Help menu or by pressing the F1 key.

Toolbar

NOTE: In addition to the software information in this section, there is context-sensitive help available within the SMI software interface, which is accessed by pressing the F1 key or selecting Help from the SMI software main menu.

The SMI software toolbar provides quick access to the SMI software's frequently-used features. Each item is represented by an icon, as shown in the next figure.



The Toolbar

NOTE: The entire set of SMI software features can be accessed from the menu bar. For details, see Menu Bar on page 62.

Each icon is described in the next table.

Icon	Menu Command	Description
	New	Create a new document.
	Open	Open an existing document.
	Save	Save the active document.
	Save All	Save the Project and all open documents.
	Cut	Cut the selection and put it on the Clipboard.
	Copy	Copy the selection and put it on the Clipboard.
	Paste	Insert Clipboard contents.
	Configuration	Show or hide the Configuration window.
	Terminal	Show or hide the Terminal window.
	Information	Show or hide the Information window.
	Serial Data Analyzer	Show or hide the Serial Data Analyzer ("sniffer").
	Find Motors	Detect all available motors connected to the defined serial ports of the computer.
	Detect Motors	Detect motors connected to the currently-selected port in the Terminal window.
	Compile and Download Project	Compile and download all user programs defined in the project to their associated motors.
	Compile and Transmit SMX File	Compile and download the program in the active view to its associated motor.
	Scan for errors	Scan the program in the active view.
	Upload Program	Upload the program in a motor to an SMI file.
	Run Program	Send a RUN command to the selected motor in the Terminal window.
	Stop Running Program	Send an END command to the selected motor in the Terminal window.

Icon	Menu Command	Description
	Stop all Motors	Send an END and then an S command to all motors.
	Decelerate all Motors to a Stop	Send an END and then an X command to all motors.
	SmartMotor Playground	Opens the SmartMotor Playground, where you can monitor and jog a single motor in Position, Velocity and Torque modes.
	Context Help	Opens the context help for the selected item.

Each item is described in detail in the SMI software's online help file, which can be accessed from the Help menu or by pressing the F1 key.

Configuration Window

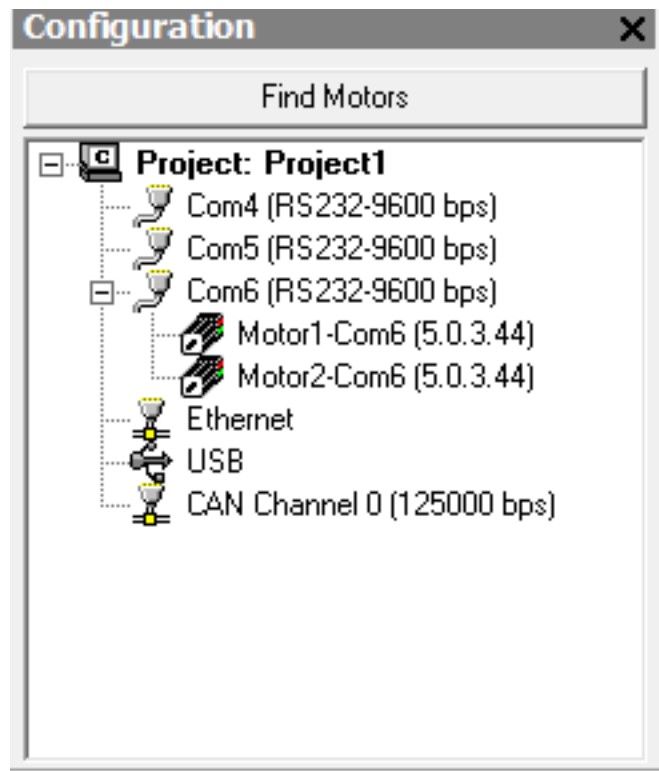
NOTE: In addition to the software information in this section, there is context-sensitive help available within the SMI software interface, which is accessed by pressing the F1 key or selecting Help from the SMI software main menu.

The Configuration window shows the current configuration and allows access to specific ports and motors. The Configuration window is essential to keeping multiple SmartMotor systems organized, especially in the context of developing multiple programs and debugging their operation.

The Configuration window is typically visible when the SMI software opens. If the window has been closed, you can open it from the SMI software main menu by selecting:

View > Configuration

NOTE: When the window is visible, the menu item will have a check mark next to it.



Configuration Window

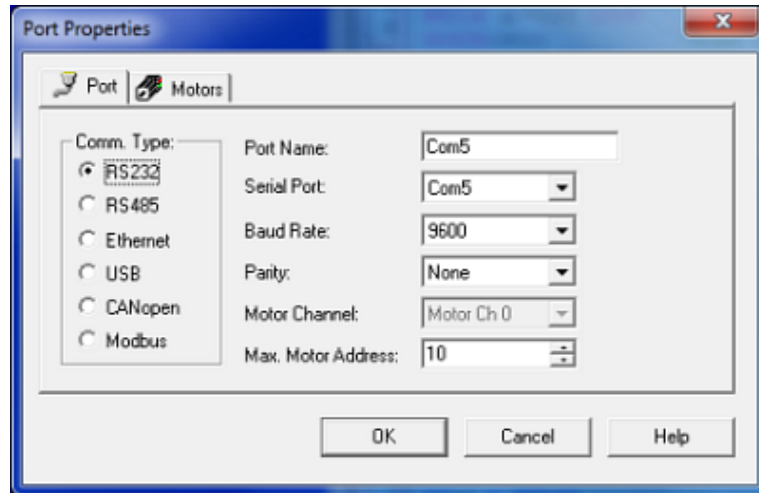
The Configuration window is essential to keeping multiple SmartMotor systems organized.

To use the Configuration window:

- Click Find Motors to analyze your system, or

Right-click on an available port to display a menu, and select either "detect motors" or "address motors" to find motors attached to that port.

- You can double-click on any port to view its properties, as shown in the next figure.



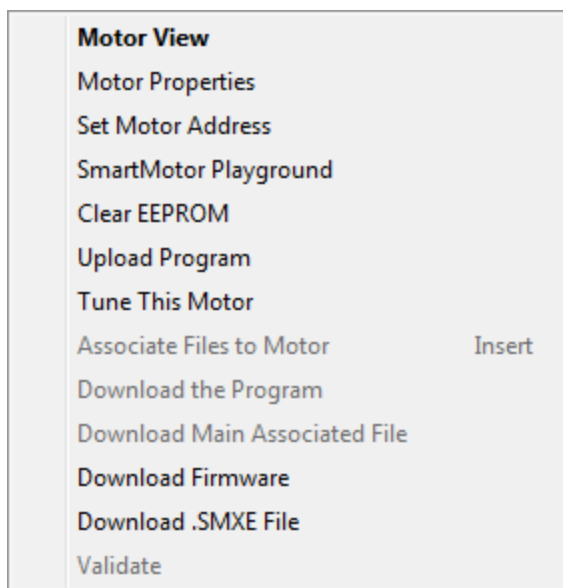
Port Properties Window

- You can also double-click on any motor to open the Motor View tool for that motor, as shown in the next figure.



Motor View Window

- By right-clicking the motor, you can access its properties along with other tools, as shown in the next figure.



Motor Tools Menu

Terminal Window

NOTE: In addition to the software information in this section, there is context-sensitive help available within the SMI software interface, which is accessed by pressing the F1 key or selecting Help from the SMI software main menu.

The Terminal window acts as a real-time portal between you and the SmartMotor. By typing commands in the Terminal window, you can set up and execute trajectories, execute subroutines of downloaded programs and report data and status information to the window.

The Terminal window is typically shown (default setting) when the SMI software is opened. However, if the Terminal window is closed, select:

View > Terminal

NOTE: When the window is visible, the menu item will have a check mark next to it.



Terminal Window

To use the Terminal window:

- Specific communication ports can be selected using the tabs.
- Commands can be entered in the white text box or directly on the blue screen. If data is flooding back from the motor, then the white text box will be more convenient, as the incoming data may cause the text to scroll out of view.
- When motor power is activated, there is a 500 ms timeout before the motor will accept commands on the serial port. Any commands sent during that time are buffered and then accepted once the 500 ms timeout expires. Because incoming commands take priority over the internal program, buffered commands run before the internal program begins.
- Because multiple SmartMotors are on a single communication port are individually addressed, commands can be routed to any or all of them by making the appropriate selection from the drop-down list, which is located just below the tabs. The SMI program automatically sends the appropriate codes to the network to route the data to the specified motor(s).
- You can double-click a previous command to resend the command (see the next figure). However,
 - If that command has a motor address in it (for example, 1RPA, where "1" = serial bus Motor 1), the command will resend to that motor.
 - If that command does not have an address, the command will be sent to the last-addressed motor. For example, if you previously sent the command 2RPA, which addresses serial bus Motor 2, an unaddressed command that you double-click (or issue) will go to serial bus Motor 2, even if it's on the list before the point where you started addressing Motor 2.



```

0RPA          -1669
-1844
-1176

1RPA          -1176
RPA           -1176

2RPA          -1844
RPA           -1844

3RPA          -1669
RPA           -1669

RPA           -1669
  
```

An example of commands sent to the last-addressed motor. Notice that double-clicking the first RPA command reports the position of motor 3 because it was the last-addressed motor.

- PRINT commands containing data can be sprinkled in programs to send data to the Terminal window as an aid in debugging.
- What is typed on the screen is not what goes to the motor. For example, 1RPA does not send a "1" to the motor — it is sending an Extended ASCII code for "1"(Hex 0x81). Then it sends ASCII "R", "P" and "A", and a SPACE (Hex 20) as the delimiter (not a carriage return). Note that the terminal window uses a space as the delimiter; the motor uses a carriage return (Hex 0x0D) as the delimiter.
- Data that has associated report commands, such as Position, which is retrieved using the RPA command, can be easily reported by simply including the report command directly in the program code.

NOTE: Be careful when using report commands within tight loops because they can bombard the Terminal window with too much data.

- If a program is sending too much data to the Terminal window, try adding a WAIT=50 command to the program, which will slow down the flow.
- Use the right-hand scroll bar to review the Terminal window history.

Initiating Motion from the Terminal Window



WARNING: The larger SmartMotors can shake, move quickly and exert great force. Therefore, proper motor restraints must be used, and safety precautions must be considered in the workcell design (see Other Safety Considerations on page 31).

To initiate motion from the terminal window, enter these commands (do not enter the comments, which are the right-hand portion of each line).

```
MP           'Initialize Position mode
ADT=100      'Set target accel/decel
VT=1000000  'Set target velocity
PT=300000   'Set target position
G           'Go, starts the move
```

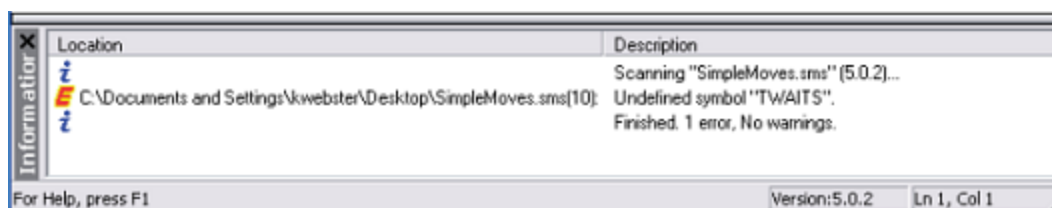
NOTE: Acceleration, velocity and position fully describe a trapezoidal-motion profile.

After the final G command has been entered, the SmartMotor accelerates to speed, slows and then decelerates to a stop at the absolute target position. The progress can be seen in the Motor View window. For details on the Motor View window, see Monitoring the SmartMotor Status on page 210.

Information Window

NOTE: In addition to the software information in this section, there is context-sensitive help available within the SMI software interface, which is accessed by pressing the F1 key or selecting Help from the SMI software main menu.

The Information window shows the program status. When a program is scanned and errors are found, they are listed in the Information window preceded by a red "E" along with the program path and line number where the error was found, as shown in the next figure.



Example Error Message

The Information window is typically visible when the SMI software opens. If the window has been closed, you can open it from the SMI software main menu by selecting:

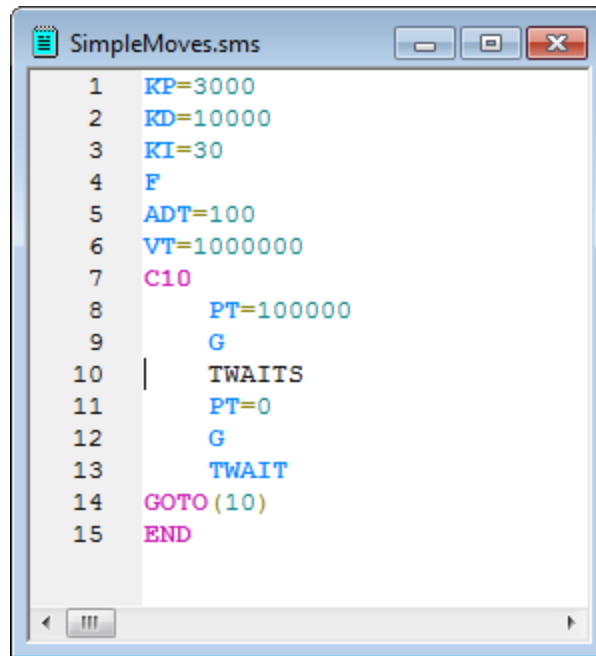
View > Information

NOTE: When the window is visible, the menu item will have a check mark next to it.

To use the Information window:

- Double-click on the error in the Information window—the specific error will be located in the Program Editor.

In the next example, the scanner does not recognize the command `TWAITS`. The correct command is `TWAIT`.



```

1  KP=3000
2  KD=10000
3  KI=30
4  F
5  ADT=100
6  VT=1000000
7  C10
8      PT=100000
9      G
10     | TWAITS
11     | PT=0
12     | G
13     | TWAIT
14 GOTO (10)
15 END

```

TWAITS Error

Correct the error and scan the program again. After all errors are corrected, the program can be downloaded to the SmartMotor.

- Warnings may appear in the Information window to alert you to potential problems. However, warnings will not prevent the program from being downloaded to the SmartMotor. It is the programmer's responsibility to determine the importance of addressing the warnings.

Program Editor

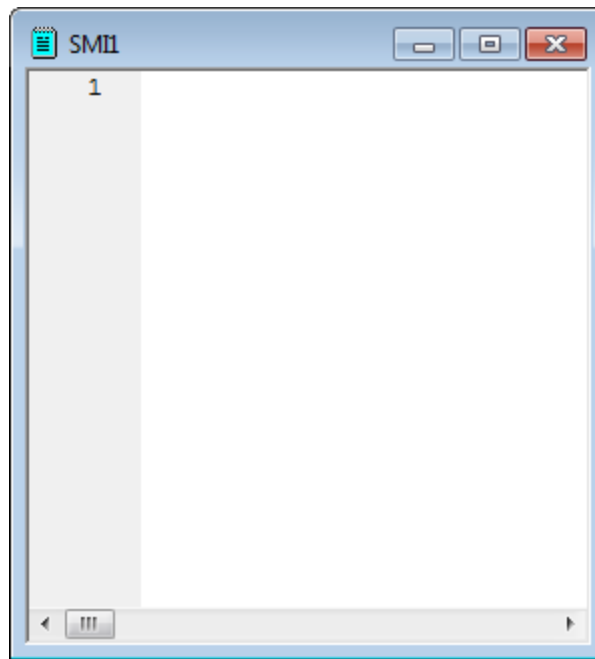
NOTE: In addition to the software information in this section, there is context-sensitive help available within the SMI software interface, which is accessed by pressing the F1 key or selecting Help from the SMI software main menu.

SmartMotor programs are written in the SMI software Program Editor before being scanned for errors and downloaded to the motor.

To open the Program Editor, from the SMI software main menu, select:

File > New

Or click the New button (📄) on the toolbar. The Program Editor opens, as shown in the next figure.



Program Editor

To use the Program Editor:

- Type the program code directly into the Program Editor. As you write the program, the editor applies syntax highlighting to the code, which makes it easier to read and debug.
- Every program requires an END command, even if the program is designed to run indefinitely and the END is never reached. For more details on program requirements, see Understanding the Program Requirements on page 52.
- The first time you write a program, you must save it before you can download it to the motor.
- *Every time a program is downloaded, it is automatically saved to that file name.* This point is important to note, as most Windows applications require a "save" action. If you want to set aside a certain revision of the program, it should be copied and renamed, or you should simply save the continued work under a new name.
- Once a program is complete, you can scan it for errors by pressing the Scan File button (🔍) on the toolbar, or scan and download it in one operation by pressing the Compile and Download Program button (🔧), which is also located on the toolbar.

If errors are found, the download will be aborted and the problems will be identified in the Information window located at the bottom of the screen.

- Programs are scanned using a language file that is related to different motor firmware versions. If Compile and Download Program is selected, the language file will be chosen based on the version read from the motor. If Scan File is selected, the default language file will be used. To change the default language file, from the SMI software main menu, select

Compile > Compiler default firmware version > [select the desired version]

For more details, see Setting the Motor Firmware Version in SMI on page 50.

Motor View

This feature supports the DS2020 Combitronic system.

NOTE: In addition to the software information in this section, there is context-sensitive help available within the SMI software interface, which is accessed by pressing the F1 key or selecting Help from the SMI software main menu.

The SMI Motor View window allows you to view multiple parameters related to the motor.

To open the Motor View window, from the SMI software main menu, select:

Tools > Motor View

and select the motor you want to view. Or, in the Configuration window, double-click the motor you want to view.



Motor View Window

NOTE: The Motor View window provides a real-time view into the inner workings of a SmartMotor.

To use the Motor View window:

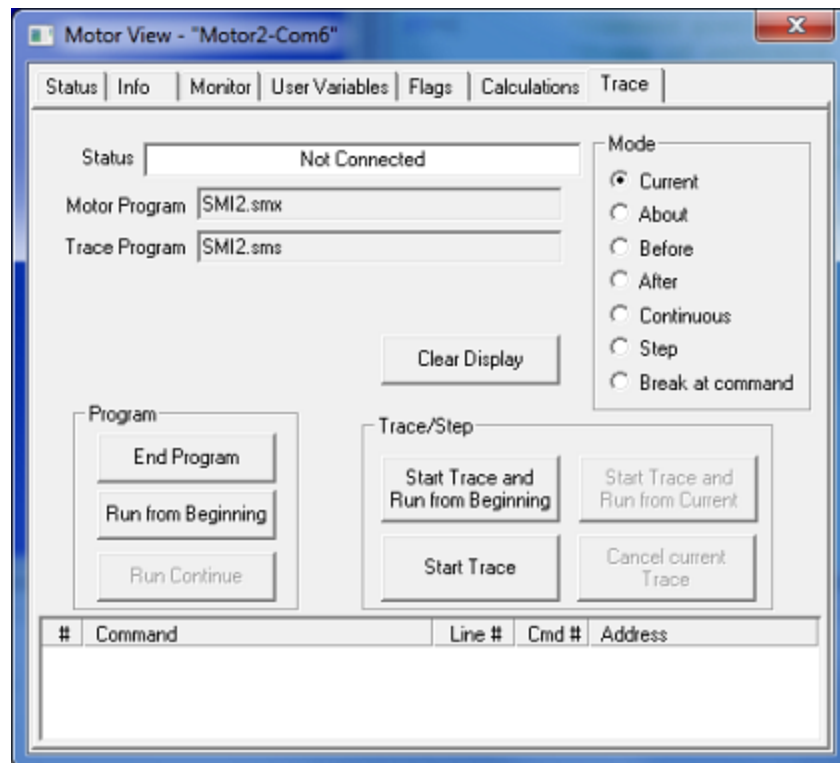
- Click Poll to initiate real-time scanning of motor parameters.
- A program can be running in the motor while the Motor View window is polling. The program must not print text to the serial channel being used for polling.
- In addition to the standard items displayed, two fields allow you to select from a list of additional parameters to display.

For example, in the previous figure, Voltage and Current are being polled. This information can be useful when setting up a system for the first time, or debugging a system in the field. Temperature is also useful to monitor in applications with demanding loads.

- All seven of the user-configurable onboard I/O points are shown. Any onboard I/O that is configured as an output can be toggled by clicking on the dot below the designating number.
- The SmartMotor has built-in provisions allowing it to be identified by the SMI software. When a motor is identified, a picture of it appears in the lower left corner of the Motor View window.
- Tabs across the top of the window provide access to additional information.

SMI Trace Functions

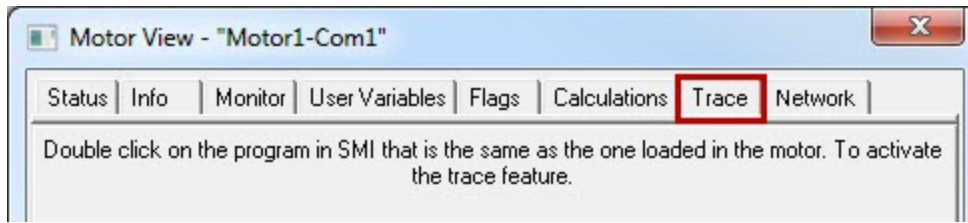
The Trace tab provides a set of functions that are useful for debugging a SmartMotor program. To access Trace functions, open the Motor View window and click the Trace tab.



Motor View Trace Functions

To use Trace functions:

1. Open the Trace window. When first opened with no program loaded, this message appears:



2. Right-click the SmartMotor in the Configuration window and select Upload Program. The program is uploaded to the SMI Editor.
3. Double-click anywhere in the program to load it into the Trace window.
4. Select the desired Mode.
5. Double-click on desired line in the Editor window, if needed.
6. Press the desired button in the Trace/Step box. The program must run before anything will happen.

The next table describes the items in the Trace window:

Item	Description
Status box	Shows the current state of the Trace program and Motor Program. This becomes active after a command is executed on the Trace tab and remains active until the Motor View is closed. Possible Status messages are: Not Connected - Not connected to motor Program Running or Program Stopped - If at a breakpoint or the program is stopped. Trace Active or Trace Inactive - If a trace is currently in progress or waiting to hit a breakpoint in progress. If a trace is active it must be canceled before selecting a new Mode. At Break Point - Program execution halted because a breakpoint was reached or a step was completed.
Motor Program box	Shows the name of the program contained in the motor.
Trace Program box	Shows the name of the program that was double-clicked.
Clear Display button	Clears the highlighted text in the editor window and removes any information in the Trace List window.
Program group	End Program - Stops program execution by writing the END command Run from Beginning - Issues a RUN command. Run Continue - Release firmware from the current breakpoint. (Only available when at a breakpoint.)
Mode group: For any trace information to be retrieved from the motor, a mode must be selected and the program must run.	
Current	Captures the first 20 points encountered.
About, Before, After	Requires the user to select a line from the program in the Editor window by double-clicking on it. The program trace responds based on the option selected in the Trace/Step group (see below). About - Captures 9 points before and 10 points after desired line. Before - captures 20 points before the desired line. After - Captures 20 points after the desired line.
Continuous	Polls the motor for commands that are executing. Because of bandwidth, not all executed lines are shown in the Trace view or highlighted in the program.
Step	Enables step mode. The program trace responds based on the option selected in the Trace/Step group (see below).
Break at Command	Requires the user to select a line in the program by double-clicking on it in the Editor window. The program trace responds based on the option selected in the Trace/Step group (see descriptions after this table).
Trace/Step group	Various options are available based on other selections (see descriptions after this table).

Trace/Step group - Options for trace selections (when Step mode is not selected):

- Start Trace and Run from Beginning button - Sets trace information in the motor and issues a RUN command.
- Start Trace button - Sets trace information in the motor.
- Start Trace and Run from Current button - Available when at a break point. The trace information is set in the motor and the program continues from the current break point.
- Cancel Trace button - Available when a trace is active to cancel the current trace.

Trace/Step group - Options for Step (when Step mode is selected):

- Step from Beginning button - Sets a breakpoint in the motor and issues a RUN command. The program executes the first line of code and then stops.
- Step from Current button - Sets a breakpoint in the motor. If the program is running, the motor stops at the next command. If the program is at a breakpoint, the motor executes the next command and then stops.

Trace/Step group - Options for Break (when Break at command mode is selected):

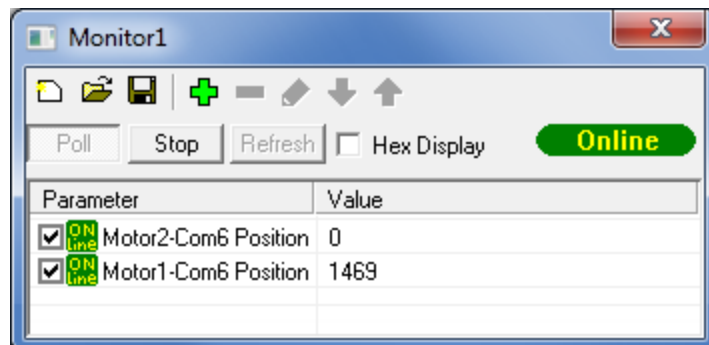
- Set Breakpoint and Run from Beginning button - Sets the breakpoint and runs the program from the beginning.
- Set Breakpoint button - Sets a breakpoint in the motor.
- Set Breakpoint and Run from Current button - If at a breakpoint, this sets the new breakpoint and runs the program from the current location.
- Remove Breakpoint button - Removes a breakpoint that was set and not reached.

Monitor Window

NOTE: In addition to the software information in this section, there is context-sensitive help available within the SMI software interface, which is accessed by pressing the F1 key or selecting Help from the SMI software main menu.

The Monitor window allows you to create your own fully-customized monitor. Because it is polling a limited set of items, it provides a more efficient monitoring method. To open the Monitor window, from the SMI software main menu, select:

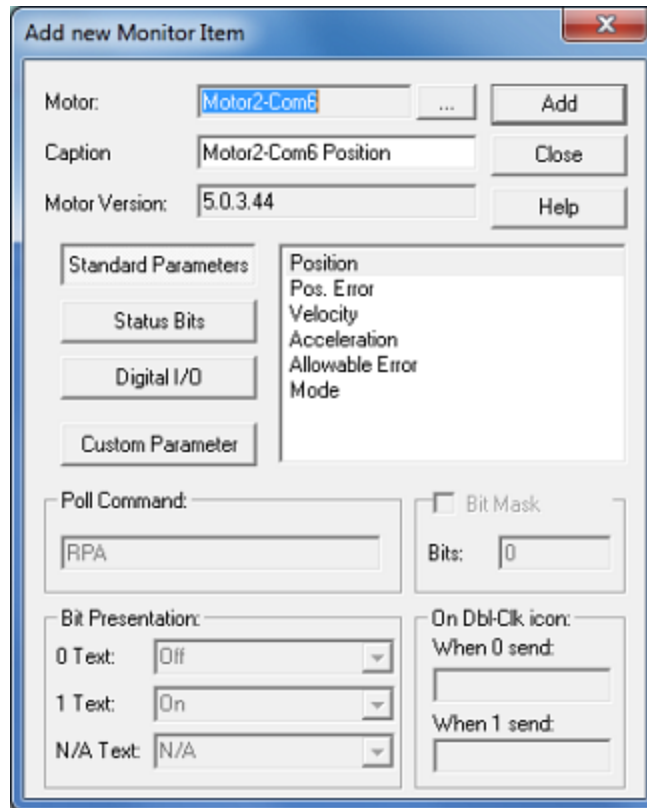
Tools > Monitor View



Monitor Window

To use the Monitor window:

- Polling items can be added or removed by pressing the + and - buttons. When adding a new item, the Add New Monitor Item window opens and provides tools for setting up the monitoring function, as shown in the next figure.



Add New Monitor Item Window

- Custom items, which do not have explicit report commands, can be added by entering the specific commands appropriate to getting the data reported (for example, make a variable equal to the desired parameter and then report that variable).

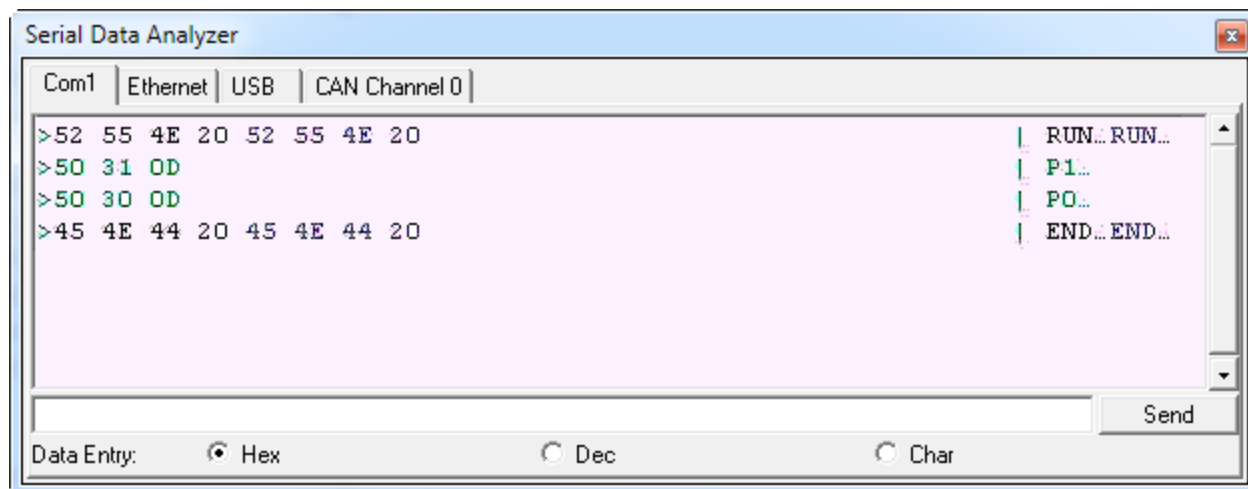
Serial Data Analyzer

NOTE: In addition to the software information in this section, there is context-sensitive help available within the SMI software interface, which is accessed by pressing the F1 key or selecting Help from the SMI software main menu.

The SMI Terminal window formats text and performs other housekeeping functions that are invisible to the user. For an exact picture of the data being traded between the PC and the SmartMotor™, use the Serial Data Analyzer (also known as the "sniffer"). To open the Serial Data Analyzer, from the SMI software main menu, select:

View > Serial Data Analyzer

Or press the Serial Data Analyzer button (🔍) on the toolbar. The Serial Data Analyzer window opens, as shown in the next figure.



Serial Data Analyzer

The Serial Data Analyzer window can display serial data in a variety of formats, and it can be a useful tool for debugging communications. For example, you can:

- View data transfer between computer and SmartMotor(s).
- View data in hexadecimal, decimal, or ASCII format in up to three columns.
- Send commands and binary data to SmartMotor(s).
- View sent and received data in different definable colors.
- Capture data transfer in different ports at the same time, and view each port using its dedicated page.

NOTE: SMI can display the precise data being sent between the host and the SmartMotor in multiple formats.

Chart View

NOTE: In addition to the software information in this section, there is context-sensitive help available within the SMI software interface, which is accessed by pressing the F1 key or selecting Help from the SMI software main menu.

In some cases, the best way to understand a data trend is by seeing it graphically. The SMI Chart View provides graphical access to any readable SmartMotor parameter.

To open the Chart View window, from the SMI software main menu, select:

Tools > Chart View

The Chart View window opens, as shown in the next figure.

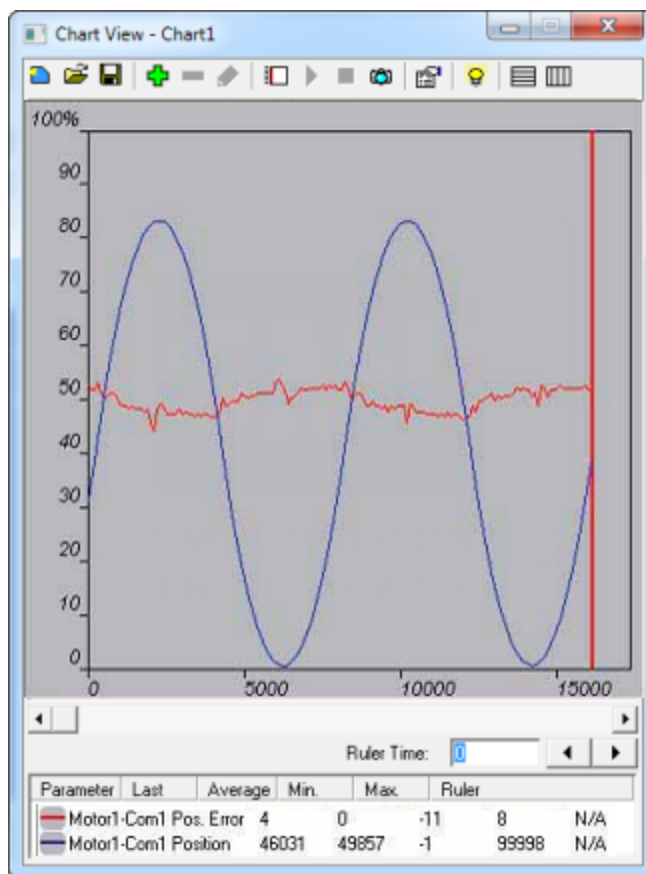


Chart View Window

To use the Chart View tool:

- Polling items are added or removed by pressing the + and - buttons.
- The fields and options are identical to those in the Monitor tool. For details on the Monitor tool, see Monitor Window on page 76.
- Adjustable upper and lower limits for each polled parameter allow them to be scaled to fit the space.

- The toolbar across the top provides additional functions such as chart editing, start/stop sampling, manual update and more.
- The Start Sampling button (▶) starts the charting action.
- While the Chart View does not include a print function, Window's standard Print Screen key can capture the chart to the clipboard, and from there, it can be pasted into other applications (like Microsoft Excel, Microsoft Word, etc.). This graphical data can be a useful addition to written system reports.

Additionally, a context menu is available by right-clicking on the Chart View window, which has selections for:

- Copying the chart data as a tab-delimited table in text format, which can then be imported into a spreadsheet, such as Microsoft® Excel®, or any text editor.
- Copying the current image of the chart to the clipboard in bitmap format, which can then be pasted in any graphic application.

Chart View Example

The SMI Chart View provides graphical access to any readable SmartMotor parameter. The next example shows how to use the Chart View tool to graphically track torque changes on the SmartMotor.

This procedure assumes that:

- The SmartMotor is connected to the computer. For details, see *Connecting the System* in the *SmartMotor Installation & Startup Guide* for your motor.
- The SmartMotor is connected to a power source. (Certain models of SmartMotors require separate control and drive power.) For details, see *Understanding the Power Requirements* in the *SmartMotor Installation & Startup Guide* for your motor.
- The SMI software has been installed and is running on the computer. For details, see *Installing the SMI Software* in the *SmartMotor Installation & Startup Guide* for your motor.
- You've completed the first-time motion example. For details, see *Moving the SmartMotor* in the *SmartMotor Installation & Startup Guide* for your motor.

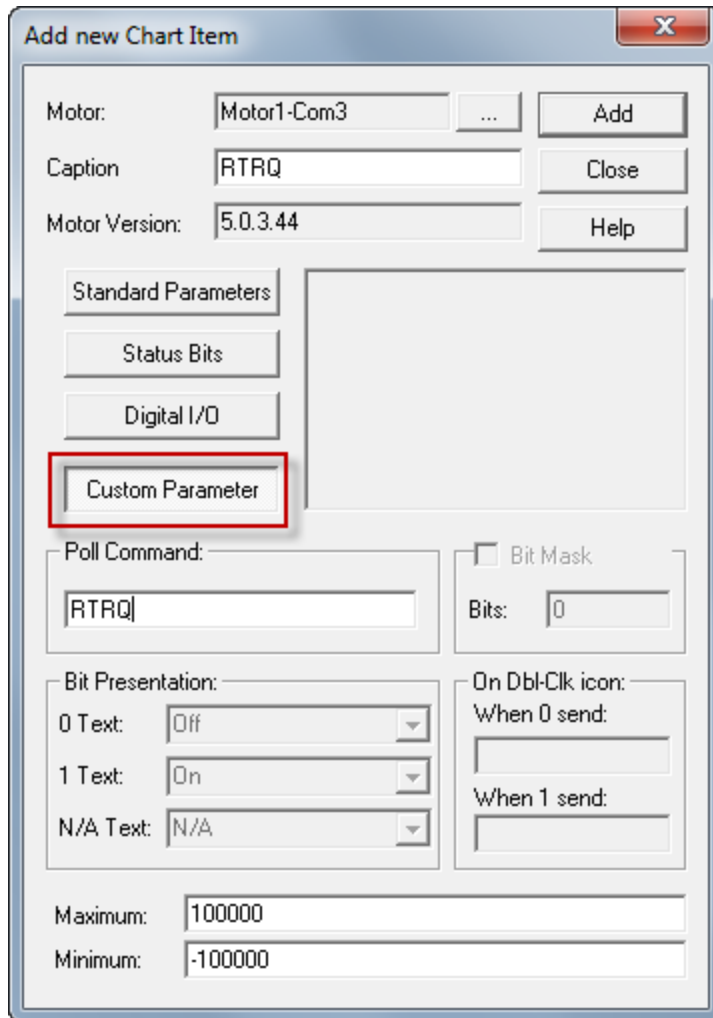
To open the Chart View window, from the SMI software main menu, select:

Tools > Chart View

The Chart View window opens. For details, see *Chart View* on page 79.

To create the example:

1. Click the Add icon (+). The Add New Chart Item window opens.
2. Click Custom Parameter to enter a nonstandard parameter for charting.
3. Fill in the text boxes as shown in the next figure.



Custom Parameter Button and Related Entries

NOTE: Be sure the Maximum and Minimum values are set to 10000 and -10000, respectively, as shown in the previous figure. They default to ten times more than those values.

4. After you've completed the entries, click Add and the custom parameter will be added to the Chart View window.
5. Click the green Play icon (▶); the chart recorder plots the RTRQ value.
6. In the SMI software Terminal window, enter these commands:

```
MT
T=0
G

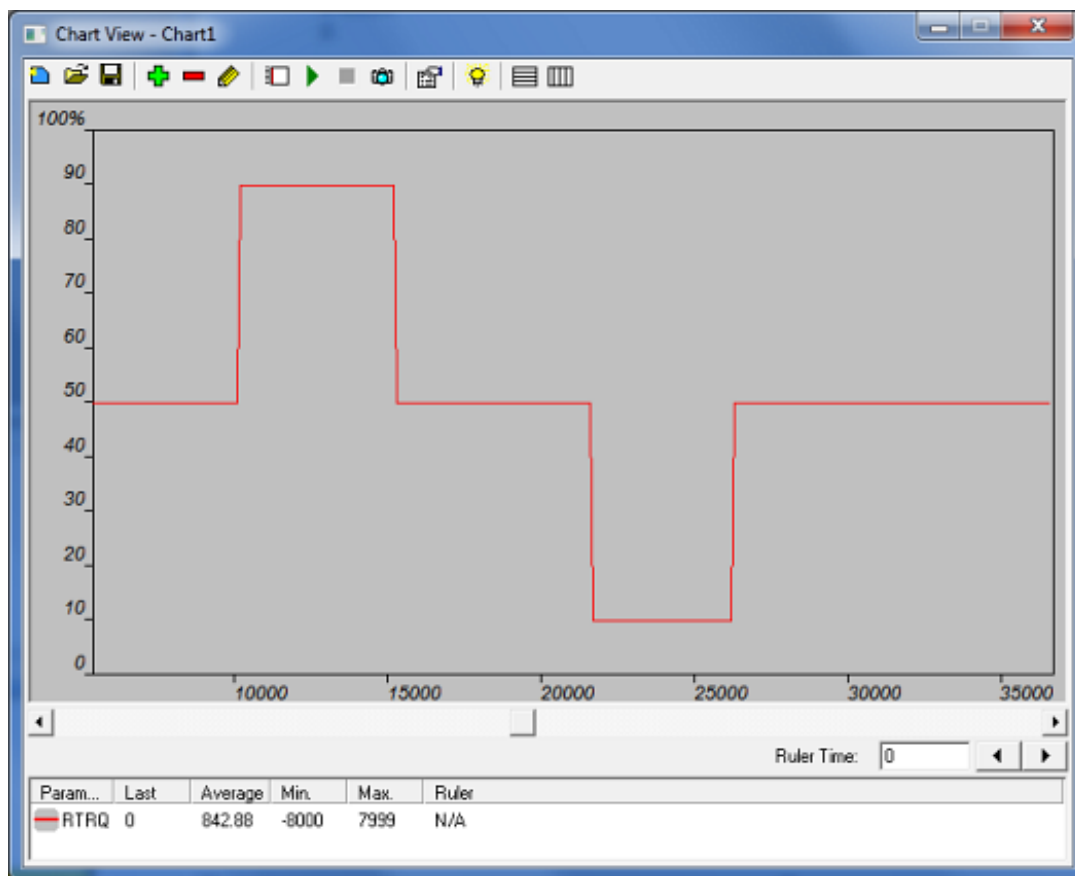
T=8000
G

T=0
G

T=-8000
G

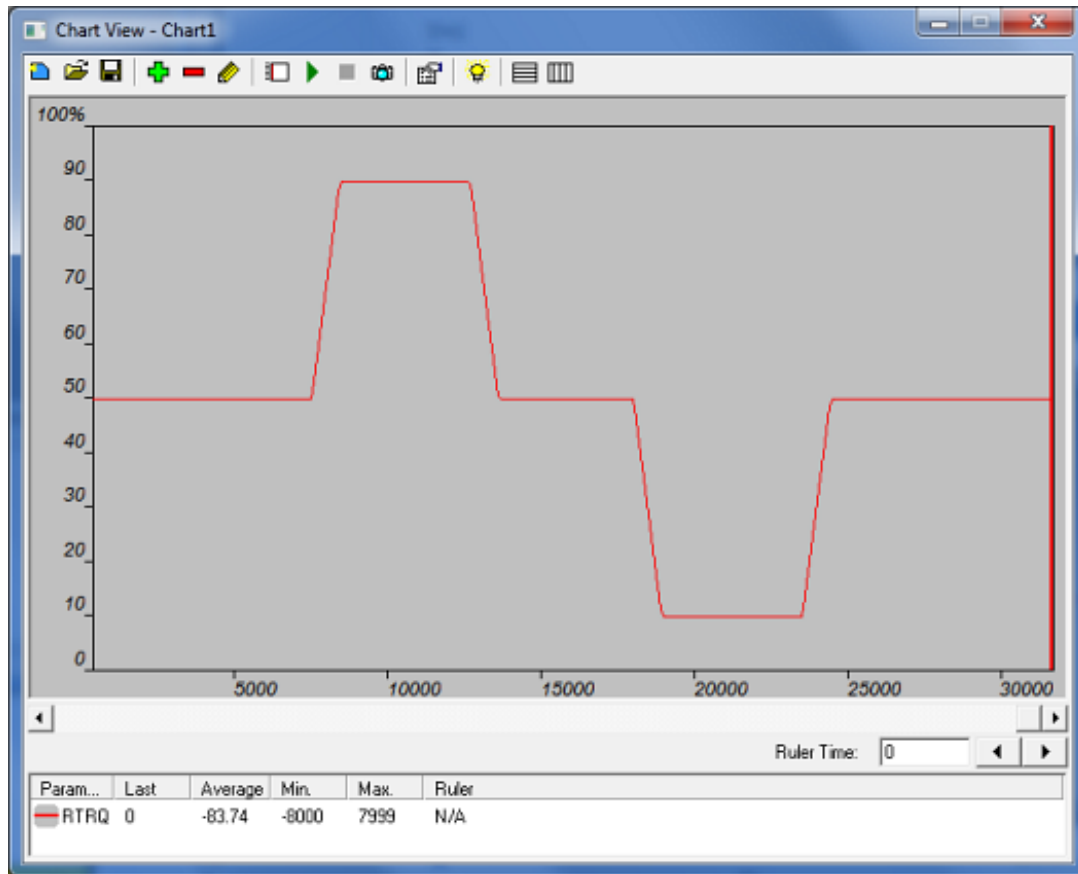
T=0
G
```

The Chart View tool plots a line similar to the one shown in the next figure.



Plotted RTRQ Values

7. In the SMI software Terminal window, enter TS=65536. This causes a one-second ramp time when T is commanded from or to zero.
8. Repeat the previous command sequence. Note the addition of "ramps" to the plot, which are caused by the TS command.



Plotted RTRQ Values With Ramps

Macros (Keyboard Shortcuts or Hotkeys)

NOTE: In addition to the software information in this section, there is context-sensitive help available within the SMI software interface, which is accessed by pressing the F1 key or selecting Help from the SMI software main menu.

The SMI software contains a Macros tool, which is useful for creating keyboard shortcuts (sometimes referred to as "hotkeys") for one command or a series of commands for use in the Terminal window. The tool allows you to optionally associate a command or series of commands with these key combinations:

- Ctrl+0 to Ctrl+9
- Ctrl+Shift+1 to Ctrl+Shift+9

NOTE: These key combinations can provide shortcuts for up to 19 macros; there is a maximum limit of 50 macros.

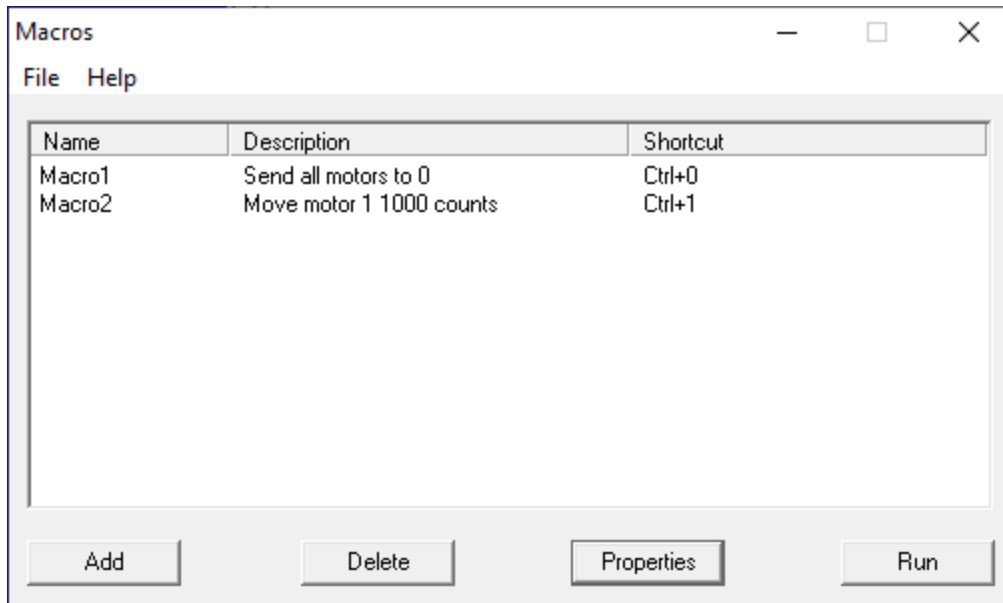
With the Macros tool, you can create multiple macros for a more efficient development process.

NOTE: In addition to these "shortcuts", SMI also provides a #define preprocessor extension command that is used to define substitutions for an SMI program. Those substitutions can then be used within that SMI program. For more details, see the topic "#define (Substitutions)" in the SMI software online help.

To open the Macros window, from the SMI software main menu, select:

Tools > Macro

The Macros window opens, as shown in the next figure.



Macros Window

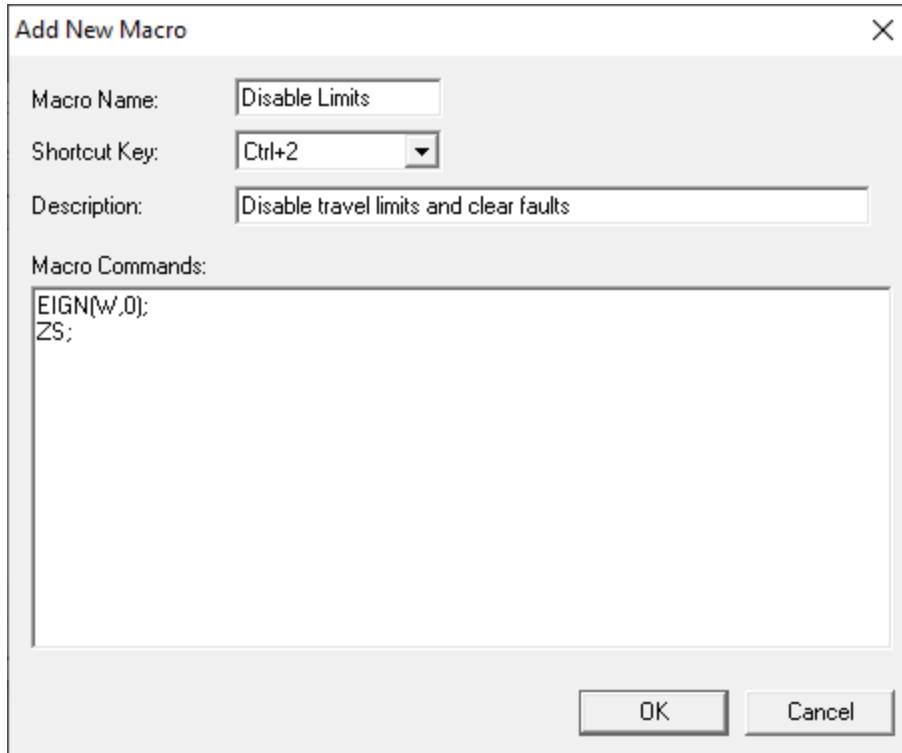
To use the Macros window:

- Add or remove macros with the Add and Delete buttons.
- Use the Properties button to view and edit the properties of an existing macro.
- The Run button allows you to test the selected macro.
- When you have finished, use the Close button to close the Macros window.

To create a macro:

In this example, you will create a macro for clearing the status bits. For details on clearing the status bits, see Checking and Clearing Status Bits in the *SmartMotor Installation & Startup Guide* for your motor.

1. Open the Macros window.
2. Click Add to open the Add New Macro window (see the next figure).
3. Fill in the information so it looks like the next figure, and then click OK to save the new macro.



Add New Macro

The Ctrl+2 shortcut key combination has now been assigned to the macro Disable Limits. When you press Ctrl+2, the SMI software issues EIGN(W,0) and ZS to the terminal screen.

Tuner

NOTE: In addition to the software information in this section, there is context-sensitive help available within the SMI software interface, which is accessed by pressing the F1 key or selecting Help from the SMI software main menu.

Tuning a SmartMotor is simpler than tuning traditional servos. However, it can be even easier when using the SMI Tuner tool to see the results of different tuning parameters.

For most applications, the default SmartMotor tuning parameters are sufficient. Viewing the position error on the Motor View tool and feeling the stiffness of the motor shaft will determine if the motor requires additional tuning.

Position:	297634
Pos. Error:	-23
Velocity:	-256566
Mode:	Position

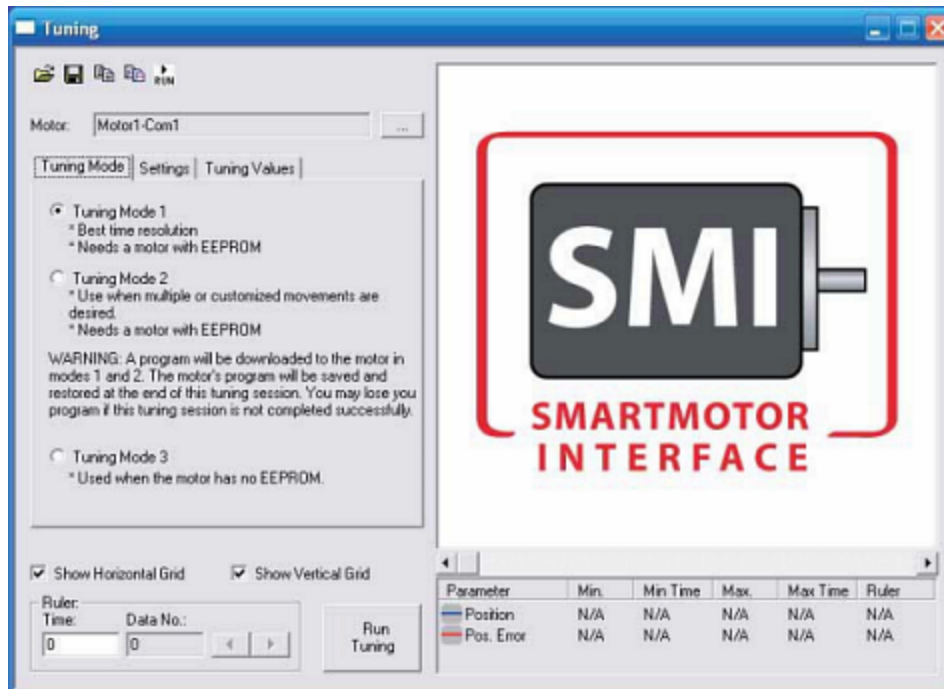
Position Error

There is a related section on tuning the PID filter later in this manual. If further tuning is required, see Tuning the PID Control on page 242.

The Tools menu has a GUI-based Tuner tool that can also be used to adjust the tuning parameters. To open the Tuner tool, from the SMI software main menu, select:

Tools > Tuner

The Tuning window opens, as shown in the next figure.



Tuning Window

The Tuner graphically shows the step response of the SmartMotor. The step response is the SmartMotor's actual reaction to a request for a small but instantaneous change in position. (Rotor inertia prevents the SmartMotor from changing its position in zero time.) The magnitude of the step response shows how well tuned the motor is.

The Tuner downloads a program that uses variables a, b, p, t, w and z. The program that was in the motor before tuning and the user variables will be restored after tuning.

Before running the Tuner:

- Be sure the motor and anything it is connected to are free to move about 1000 encoder counts or more, which is about one-quarter turn of the motor shaft.
- Be sure the device is able to safely withstand an abrupt jolt.

Click the Run Tuning button at the bottom of the Tuner window (see the previous figure).

If the SmartMotor is connected, is on and is still, you should see results similar to those in the next figure.



Sample Step Response

The upper curve with the legend on the left is the SmartMotor's actual position over time. Notice that it overshoot its target position before settling in. Adjusting the PID Tuning will stiffen the motor up and create less overshoot. For details, see [Tuning and PID Control](#) on page 240. In a real-world application, there will be an acceleration profile, not a demand for instantaneous displacement, so significant overshoot will not exist. Nevertheless, it is useful to look at the worst-case scenario of a step response.

To try a different set of tuning parameters, select the [Tuning Values](#) tab to the left of the graph area. As shown in the next figure, you will see a list of tuning parameters with two columns: the left column lists what is currently in the SmartMotor; the right column provides an area to make changes.

Tuning Mode	Settings	Tuning Values																											
		<table border="1"> <thead> <tr> <th></th> <th>Motor</th> <th>New</th> </tr> </thead> <tbody> <tr> <td>KP (Proportional coefficient):</td> <td>2000</td> <td>3000</td> </tr> <tr> <td>KI (Integral coefficient):</td> <td>240</td> <td>30</td> </tr> <tr> <td>KD (Differential coefficient):</td> <td>5000</td> <td>10000</td> </tr> <tr> <td>KL (Integral limit):</td> <td>30000</td> <td>32767</td> </tr> <tr> <td>KS (Differential sample rate):</td> <td>1</td> <td>1</td> </tr> <tr> <td>KV (Velocity feed forward):</td> <td>1500</td> <td>1500</td> </tr> <tr> <td>KA</td> <td>0</td> <td>0</td> </tr> <tr> <td>KG (Gravitational coefficient):</td> <td>0</td> <td>0</td> </tr> </tbody> </table>		Motor	New	KP (Proportional coefficient):	2000	3000	KI (Integral coefficient):	240	30	KD (Differential coefficient):	5000	10000	KL (Integral limit):	30000	32767	KS (Differential sample rate):	1	1	KV (Velocity feed forward):	1500	1500	KA	0	0	KG (Gravitational coefficient):	0	0
	Motor	New																											
KP (Proportional coefficient):	2000	3000																											
KI (Integral coefficient):	240	30																											
KD (Differential coefficient):	5000	10000																											
KL (Integral limit):	30000	32767																											
KS (Differential sample rate):	1	1																											
KV (Velocity feed forward):	1500	1500																											
KA	0	0																											
KG (Gravitational coefficient):	0	0																											
Copy Values		Apply New Values																											
Load Saved Values		Save Values																											

Apply New Values Button

To make adjustments to the tuning:

1. Change the values to those shown in the New column of the previous figure.
2. Click the "Apply New Values" button, which stores the new values in the SmartMotor.
3. Click the Run Tuning button at the bottom of the Tuning window.

The motor will jolt again and the results of the step response will overwrite the previous graph. Normally, this process involves repeated trials using the procedure outlined in the section on the PID Filter. For details, see Tuning the PID Control on page 242.

When you are satisfied with the results, the parameters producing the best results can be added to the top of your program in the SmartMotor, or in applications where there are no programs in the motors, sent by a host after each power-up. For example, the previous example's tuning parameters would be set using these tuning commands:

```

KP=3000
KI=30
KD=10000
KL=32767
F

```


SMI Options

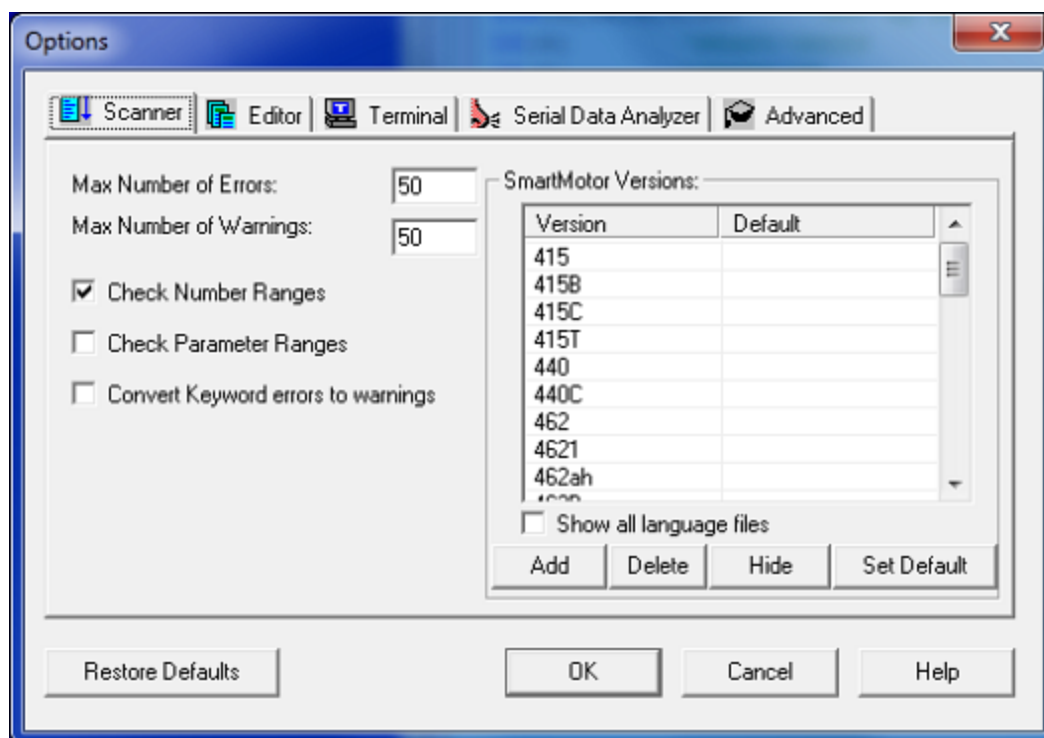
NOTE: In addition to the software information in this section, there is context-sensitive help available within the SMI software interface, which is accessed by pressing the F1 key or selecting Help from the SMI software main menu.

The SMI software has a variety of options that can be customized through the Options window. It contains tabs and selections for customizing the Scanner, Editor, Terminal and more.

To open the Options window, select:

Tools > Options

The Options window opens, as shown in the next figure.



Options Window

To use the Options window:

- Click a tab to select the options you wish to edit.
- Consider the default firmware version. Because different SmartMotor firmware versions have subtle differences, the program scanner needs to know which firmware is being used to distinguish between supported and unsupported commands.
- Other options, such as Editor syntax colors, deal with user preferences.
- After you have finished editing options, click OK to close the window and save your changes.

SMI Help

The most complete and current information available for the SMI software is available within the program's extensive Help tool. For details, see the SMI software help.

Context-Sensitive Help Using F1

- Dialog and Message box: Just press the F1 key while the box is displayed.
- Information View: Select the line and press the F1 key. The software shows a description of the selected error. For more details, see Information Window on page 69.
- Menu command: Select the menu item and press the F1 key.
- Keyword Information: In the Program Editor, select the keyword and press F1. The software shows a full description of the selected keyword.

Context-Sensitive Help Using the Mouse

There is a "context help" button on the tool bar. When you click the button (or press Shift+F1 on keyboard) the program enters the Help Mode and the cursor shape changes to context-sensitive help cursor (☞?). In Help Mode you can use the mouse or keyboard to select a menu command, a toolbar button, an error message in the Information View, or other items within SMI, and help on the item is displayed.

Help Buttons

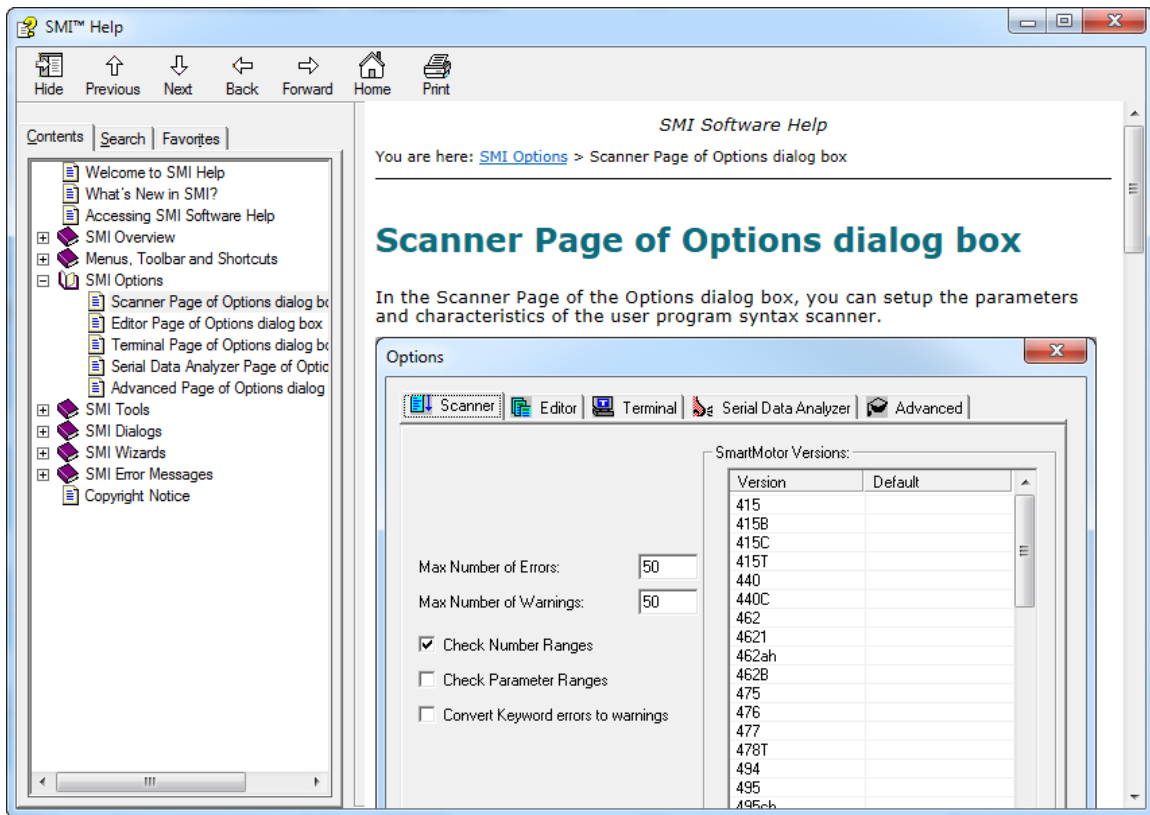
You can click the Help button, available on many dialog boxes, to get help about that dialog box.

Hover Help

You can place (hover) the mouse pointer over an SMI software button or a Program Editor keyword to see a short description of that button or keyword.

Table of Contents

To see the list of topics within SMI software Help, use the Contents command in the Help menu.



Sample Help Page

Projects

NOTE: In addition to the software information in this section, there is context-sensitive help available within the SMI software interface, which is accessed by pressing the F1 key or selecting Help from the SMI software main menu.

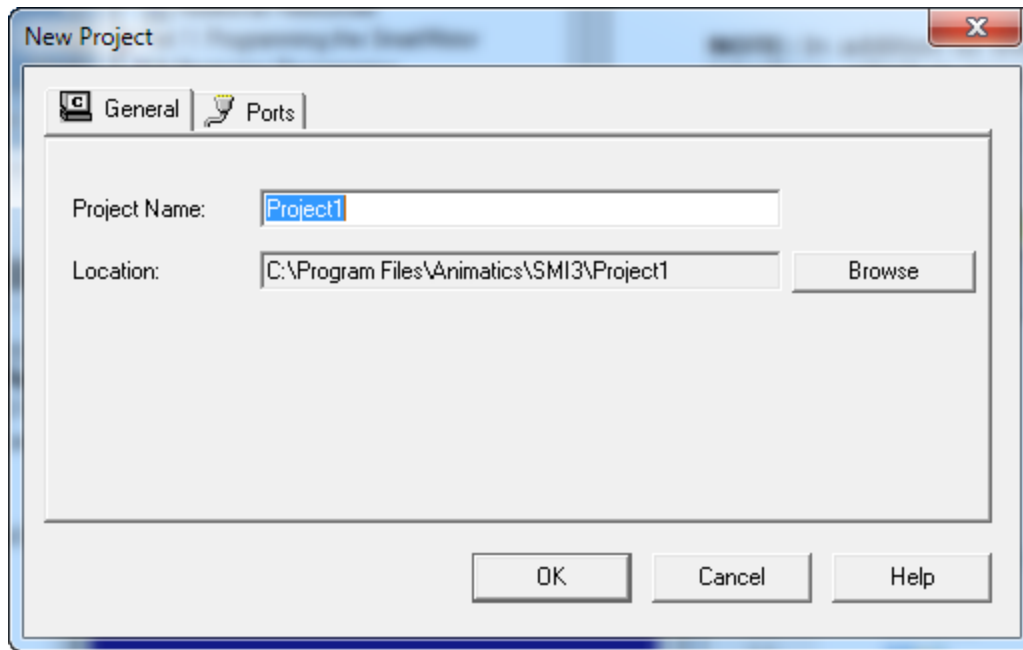
In applications with more than one SmartMotor, and possibly more than one program or communications port, it is helpful to organize all of the elements as a Project rather than deal with individual files.

NOTE: When working with multiple motors, programs or ports, a Project provides a convenient way of organizing and using all of the individual elements.

To create a project, from the SMI software main menu, select:

File > New Project

The New Project window opens.



New Project Window

To use the New Project window:

- Enter a name and location in the Project Name and Location fields to title the project and specify the location where it will be saved.
- Click OK to save the information. At this point, you have the option of letting the SMI software explore the network of motors and set up the project automatically, or of doing it manually by double-clicking on the specific communication ports or motors listed in the Information window. Unless you are a system expert and know exactly what the port and motor settings are, you should let the software detect the motors for you.
- From here, you can open one or more programs for editing in the SMI Editor.
- After the project is set up, select **File > Save Project** to save it. Projects are saved as .SPJ files.
- To open a project, select **File > Open Project**, and then select the desired project (.SPJ) file. When a project file is opened, all motor communication information, program editor windows and other elements are restored.
- Use the **File > Recent Projects** menu to view and select from the projects you've most recently edited.

SmartMotor Playground

NOTE: In addition to the software information in this section, there is context-sensitive help available within the SMI software interface, which is accessed by pressing the F1 key or selecting Help from the SMI software main menu.

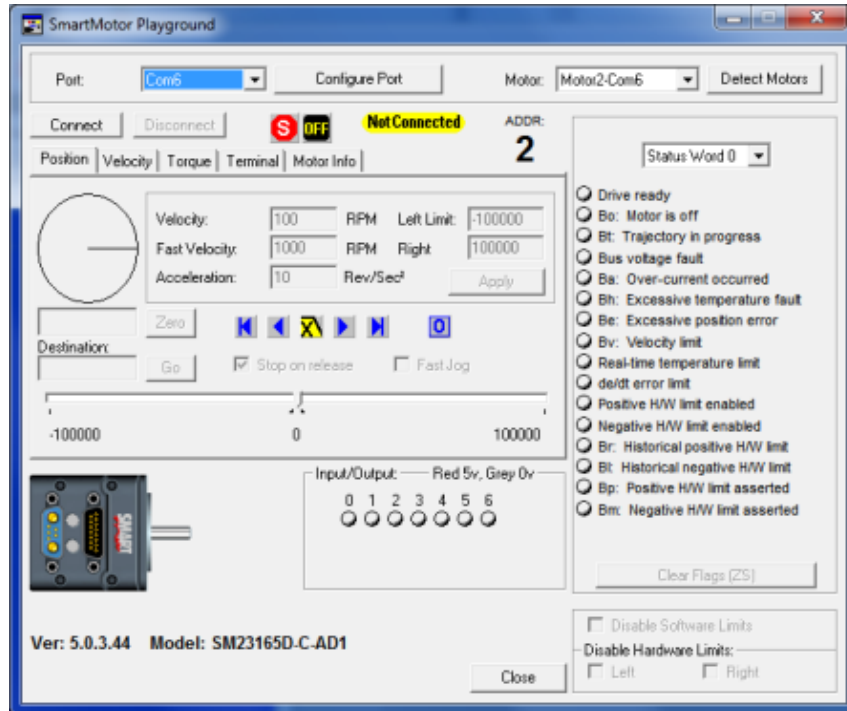
If you are a first-time user, the SmartMotor Playground contains some simple controls to help you get started with moving the motor. The SmartMotor Playground allows you to immediately move the motor without any programming.

Opening the SmartMotor Playground

There are two ways to access the SmartMotor Playground:

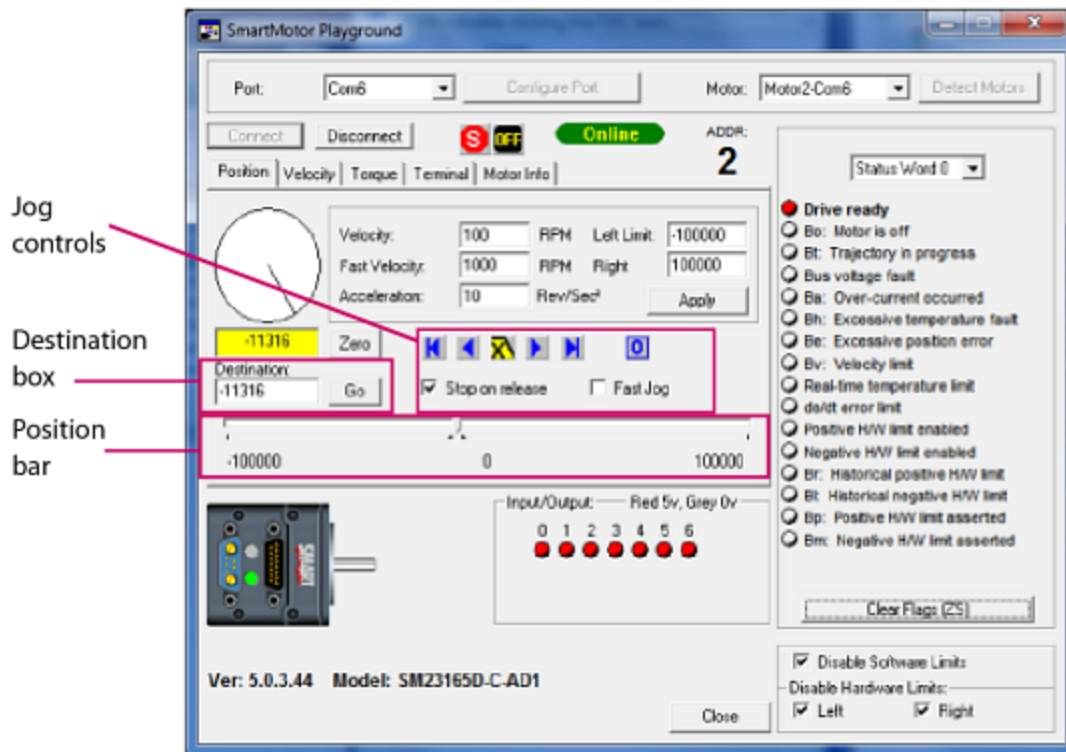
- From within the SMI software interface
- From the Windows Start menu as a stand-alone application.

To access the SmartMotor Playground from the SMI software, in the Configuration window, right-click the motor you want to move and select SmartMotor Playground from the menu.



SmartMotor Playground (Not Connected)

Click Connect (upper-left area of the window) to connect to the SmartMotor.



SmartMotor Playground (Connected)

Moving the Motor

This procedure assumes that:

- The SmartMotor is connected to the computer. For details, see Connecting the System in the *SmartMotor Installation & Startup Guide* for your motor.
- The SmartMotor is connected to a power source. (Certain models of SmartMotors require separate control and drive power.) For details, see Understanding the Power Requirements in the *SmartMotor Installation & Startup Guide* for your motor.
- The SMI software has been installed and is running on the computer. For details, see Installing the SMI Software in the *SmartMotor Installation & Startup Guide* for your motor.
- The SmartMotor has been detected and addressed. For details, see Detecting and Addressing the SmartMotors in the *SmartMotor Installation & Startup Guide* for your motor.

In addition to the above items:

- Verify that all status bits are off, except for the Drive ready bit, as shown in the previous figure. If needed, use the Clear Flags button to clear any bits that are on.
- The Drive Enable input on the M-series motor must be connected and activated.
- Verify that Disable Software Limits and Disable Hardware Limits options are set as shown in the previous figure.

NOTE: The SmartMotor's hardware limits must be grounded or disabled for motion to occur. Therefore, if your SmartMotor doesn't move when moving the slider or issuing a motion command, verify that you've either grounded the limits or selected both Disable Hardware Limits check boxes (located at the lower-right corner of the screen), as shown in the previous figure.

Within the SmartMotor Playground, you can experiment with the many different modes of operation. Try these methods (see the previous figure for the locations of these items):

- Click the left and right Jog controls and watch the motor respond.
- Move the position bar to the left or right and watch the motor respond.
- Enter a value (negative = counterclockwise; positive = clockwise) in the Destination box and click Go. Watch the motor shaft move until the position counter (yellow box) reaches that destination.

While the SmartMotor Playground is useful for moving the motor and learning about its capabilities, to develop a useful application, you will need to create a program. To learn about programming the SmartMotor, see *Beginning Programming* on page 47.

Communication Details

This chapter provides information on the communications functionality that has been designed into the SmartMotor.

Introduction	98
Connecting to a Host	99
Daisy Chaining Multiple D-Style SmartMotors over RS-232	100
ADDR=formula	102
SLEEP, SLEEP1	102
WAKE, WAKE1	102
ECHO, ECHO1	103
ECHO_OFF, ECHO_OFF1	103
Serial Commands	104
OCHN(type,channel,parity,bit rate,stop bits,data bits,mode,timeout)	104
CCHN(type,channel)	105
BAUDrate, BAUD(channel)=formula	105
PRINT(), PRINT1()	105
SILENT, SILENT1	106
TALK, TALK1	106
a=CHN(channel)	106
a=ADDR	106
Communicating over RS-485	107
Using Data Mode	107
CAN Communications	110
CADDR=formula	110
CBAUD=formula	110
=CAN, =CAN(arg)	110
CANCTL(function,value)	110
SDORD(...)	111
SDOWR(...)	111
NMT	112
RB(2,4), x=B(2,4)	112
Exceptions to NMT, SDORD and SDOWR Commands	112
I/O Device CAN Bus Controller	113
Combitronic Communications	113
Combitronic Features	114
Other Combitronic Benefits	114

Program Loops with Combitronic	115
Global Combitronic Transmissions	115
Simplify Machine Support	116
Combitronic with RS-232 Interface	116
Combitronic with the DS2020 Combitronic System	117
Other CAN Protocols	118
CANopen - CAN Bus Protocol	118
DeviceNet - CAN Bus Protocol	118
I²C Communications (Class 5 D-Style Motors)	118
OCHN(IIC,1,N,baud,1,8,D)	120
CCHN(IIC,1)	120
PRINT1(arg1,arg2, ... ,arg_n)	120
RGETCHR1, Var=GETCHR1	120
RLEN1, Var=LEN1	120

Introduction

There are various ways to communicate with a SmartMotor:

- Direct-command serial over RS-232 or RS-485 (depending on the motor)
- Data mode
- Combitronic, CANopen, DeviceNet, etc.
- I²C communications

NOTE: When using I²C, the SmartMotor is always the bus controller. You cannot communicate between SmartMotors through I²C.

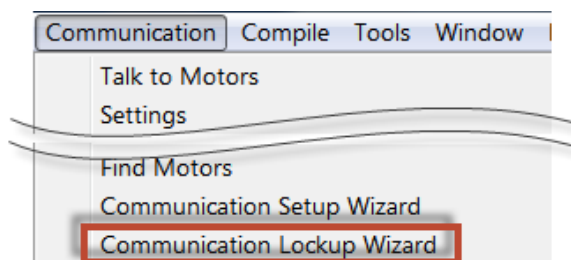
These communications methods are described in the next sections.

In applications using more than one SmartMotor, the best choice for communications is to link the SmartMotors together over their optional CAN ports, and then communicate with the group through any of the RS-232 or RS-485 ports of any of the motors on the chain. The SmartMotor's CAN-based Combitronic communications unifies all SmartMotor data and functions in a group, which makes any single motor look like a multi-axis controller from the perspective of the RS-232 or RS-485 ports. Additionally, this allows all the motors to share resources as though they were a large multi-axis controller.

Moog Animatics offers adapters for converting RS-232 to RS-485, and for converting either to USB.



NOTE: If you are unable to communicate with the SmartMotor, you may be able to recover communications using the Communication Lockup Wizard, which is on the SMI software Communications menu. For details, see the SMI software online help, which is accessed by pressing the F1 key or selecting Help from the SMI software main menu.



Communication Menu - Communication Lockup Wizard

Connecting to a Host

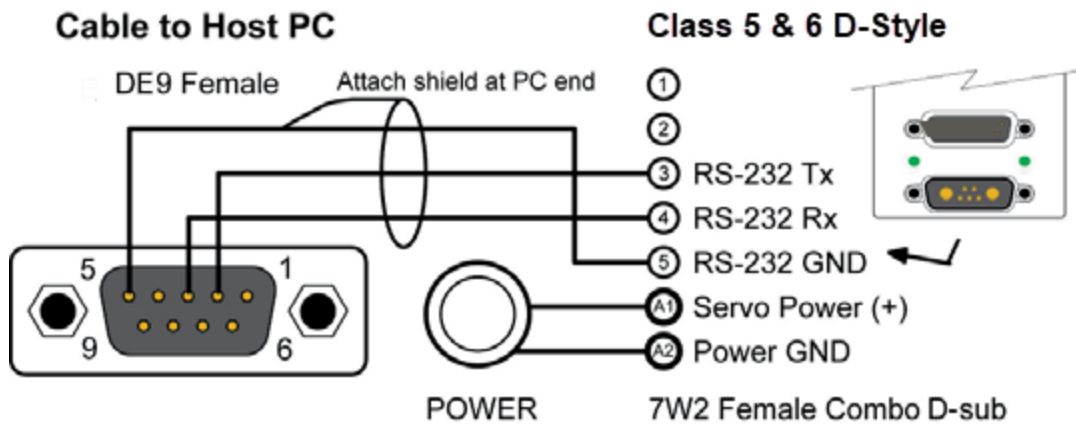
The default mode for communicating with a Class 5 or Class 6 D-style SmartMotor is serial RS-232; Class 5 and Class 6 M-style SmartMotors use serial RS-485.

NOTE: The M-style motors have one RS-485 port; they do not have an RS-232 port.

For D-style motors, the most common and cost-effective solution is through RS-232 serial communications. Under this structure, each motor is placed in an electrical serial connection such that the transmit line of one motor is connected to the receive line of the next. Each motor is set to echo incoming data to the next motor down with approximately 1 millisecond propagation delay. There is no signal integrity loss from one motor to the next, which results in highly-reliable communications.

NOTE: To maximize the flexibility of the SmartMotor, all serial ports are fully programmable with regard to bit rate and protocol.

There is a 31-byte input buffer for the RS-232 port and another for the RS-485 port. These buffers ensure that no arriving information is ever lost. However, when either port is in data mode, it is the responsibility of the user program within the SmartMotor to keep up with the incoming data.



Connection Between a Class 5 or Class 6 D-style SmartMotor and Host PC

The CBLSM1-3M cable makes quick work of connecting to your first RS-232-based SmartMotor. It combines the connections for communications and power into one cable assembly.



By default, the primary channel, which shares a connector with the incoming power in some versions, is set up as a command port with these characteristics:

	Default	Other Options
Type:	RS-232	RS-485
Parity:	None	Odd or Even
Bit Rate:	9600	2400 to 115200
Stop Bits:	1	0 or 2
Data Bits:	8	7
Mode:	Command	Data
Echo:	Off	On

Also, note that:

- If the cable used is not provided by Moog Animatics, make sure the SmartMotor's power and RS-232 connections are correct.



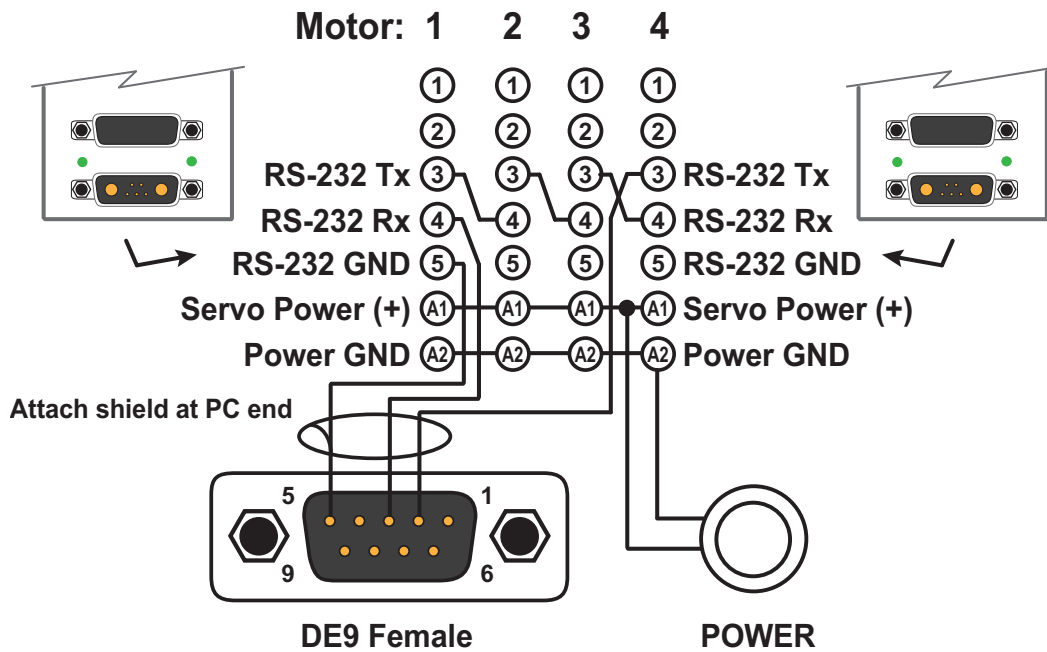
CAUTION: Be sure to use shielded cable to connect RS-232 ports, with the shield ground connected to pin 5 (ground) of the PC end only.

- Buffers on both sides mean there is no need for any handshaking protocol when commanding the SmartMotor.
- Most commands execute in less time than it takes to receive the next one. Therefore, be careful to allow processes time to complete, particularly for slower processes like printing to an LCD display or executing a full subroutine.

Daisy Chaining Multiple D-Style SmartMotors over RS-232

This section describes how to daisy chain multiple D-style SmartMotors to a single RS-232 port as shown in the next figure. Other SmartMotors can be connected together in a daisy-chain or multi-drop fashion. For details, see Connecting the System in the *SmartMotor Installation & Startup Guide* for your motor.

For low-power motors (size SM23165D and smaller), as many as 100 motors could be cascaded using the daisy-chaining technique for RS-232. To operate independently, each motor must be programmed with a unique address. In a multiple-motor system, the programmer has the choice of putting a host computer in control or having the first motor in the chain be in control of the rest.



Daisy-Chain Connection between D-Style SmartMotors and Host PC

NOTE: You can build your own RS-232 daisy-chain cable or purchase Add-A-Motor cables from Moog Animatics.

Fully-molded Add-A-Motor cables make quick work of daisy-chaining multiple motors over an RS-232 network.



CAUTION: Large (size 23 or size 34) SmartMotors draw so much power that they often require isolated communications for reliability. For such applications, consider using a DIN Rail RS-232 communication breakout device. For assistance, contact Moog Animatics.

The next sections describe related commands. For more details on these commands, see Part 2: SmartMotor Command Reference on page 247.

ADDR=formula

Set Motor to New Address

The ADDR= command causes a SmartMotor to respond exclusively to serial commands addressed to it. It is separate and independent of the motor's CAN address. The address number range is from 1 to 120.

When each motor in a chain has a unique address, an individual motor communicates normally after its address is sent over the chain one time. To send an address, add 128 to its value and output the binary result over the communication link. This puts the value above the ASCII character set, which differentiates it from all other commands or data. The address needs to be sent only once until the host computer, or motor, wants to change it to something else.

Sending out an address zero (128) causes all motors to listen and is an efficient way to send global data such as a G for starting simultaneous motion in a chain. Once set, the address features work the same for RS-232 and RS-485 communications.



RS-232 Daisy-Chained SmartMotors

Unlike the RS-485 star topology, the consecutive nature of the RS-232 daisy chain creates the opportunity for the chain to be independently addressed entirely from the host, rather than by having a uniquely-addressed program in each motor. Setting up a system this way adds simplicity because the program in each motor can be exactly the same. If the RUN? command is the first in each of the motor's programs, the programs will not start when the SmartMotor power is turned on. Addressing can then be worked out by the host before the programs are later initiated through a global RUN command.

SLEEP, SLEEP1

Assert sleep mode

WAKE, WAKE1

Deassert SLEEP

The SLEEP command causes the motor to ignore all commands except the WAKE command. This feature can often be useful, particularly when establishing unique addresses in a chain of motors. The 1 at the end of commands specifies the AniLink RS-485 port.

NOTE: The SmartMotor can be made to automatically ECHO received characters to the next SmartMotor in a daisy chain

ECHO, ECHO1*ECHO input***ECHO_OFF, ECHO_OFF1***Deassert ECHO*

The ECHO and ECHO_OFF commands toggle (turn on/off) the echoing of data input. Because the motors do not echo character input by default, consecutive commands can be presented, configuring them with unique addresses, one at a time. If the host computer or controller sent out the next command sequence, each motor would have a unique and consecutive address.

If a daisy chain of SmartMotors has been powered off and back on, the next commands can be entered into the SmartMotor Interface to address the motors (0 equals 128, 1 equals 129, etc.). Some delay should be inserted between commands when sending them from a host computer.

```

0SADDR1
1ECHO
1SLEEP
0SADDR2
2ECHO
2SLEEP
0SADDR3
3ECHO
0WAKE

```

Commanded by a user program in the first motor instead of a host, the same daisy chain could be addressed with this sequence:

```

SADDR1  'Address the first motor
ECHO    'Echo for host data
PRINT (#128, "SADDR2", #13) '0SADDR2
WAIT=10                               'Allow time
PRINT (#130, "ECHO", #13)   '2ECHO
WAIT=10
PRINT (#130, "SLEEP", #13)  '2SLEEP
WAIT=10
PRINT (#128, "SADDR3", #13) '0SADDR3
WAIT=10
PRINT (#131, "ECHO", #13)   '3ECHO
WAIT=10
PRINT (#128, "WAKE", #13)   '0WAKE
WAIT=10

```

Serial Commands

The SmartMotor allows you to communicate over the available RS-232 and/or RS-485 serial ports (depending on the style of SmartMotor you're using). There are specific serial commands used for configuring the serial communications, baud rate, printing, etc., as described below.

NOTE: D-style SmartMotors use primarily RS-232 communications, whereas all other SmartMotor use primarily RS-485 communications.

The next sections describe related commands. For more details on these commands, see Part 2: SmartMotor Command Reference on page 247.

OCHN(type,channel,parity,bit rate,stop bits,data bits,mode,timeout)

Option	Description
type: RS2 (D-style only), RS4, MB4, DMX	RS2=RS-232, RS4=RS-485, MB4=Modbus protocol over RS-485*, DMX=DMX protocol*
channel: D-style: 0, 1 M-style: 0	0=Main, 1=Secondary
parity N, O, or E	None, Odd or Even
bit rate: 2400, 4800, 9600, 19200, 38400, 57600, 115200 baud	
stop bits: 1 or 2	
data bits: 8	
mode: C or D	Command or Data
timeout: (Optional) Timeout in milliseconds between issuing a command and detecting a delimiter, e.g., RPA(space) where space is the delimiter.	

*For more details, see the documentation for the specified protocol.

NOTE: Changing the default value of any parameter other than baud rate will prevent proper command data from being received by the SmartMotor. If you are unable to communicate with the SmartMotor, you may be able to recover communications using the Communication Lockup Wizard, which is on the SMI software Communication menu. For details, see the SMI software online help, which is accessed by pressing the F1 key or selecting Help from the SMI software main menu.

Placing a communications port in Data mode will completely prevent the SmartMotor from receiving any commands and require the user program code to parse out all incoming data. Therefore, if the intent is to be able to send standard commands at any time and allow the SMI software to detect the motors, then the OCHN command could be used to change only the baud rate or the communications error timeout values — do not use it to change any other settings. The BAUD command can also be used to change the baud rate. For details, see BAUDrate, BAUD(channel)=formula on page 105.

This is an example of the OCHN command:

```
OCHN (RS4, 0, N, 38400, 1, 8, D)
```

For a D-style motor, if the primary communication channel (0) is opened as an RS-485 port, then it assumes the Moog Animatics RS485-ISO adapter is connected to it. If so, then I/O 6 is used to direct the adapter to be in transmit or receive mode according to the motor's communication activity, and I/O 6 will no longer be used as an I/O communications port. M-style motors are supplied with RS-485 on COM 0; D-style motors require an adapter for RS-485 on COM 0, but they have built-in RS-485 available on COM 1.

CCHN(type,channel)

Close a communications channel

Use the CCHN command to close a communications port when desired.

NOTE: If you are unable to communicate with the SmartMotor, you may be able to recover communications using the Communication Lockup Wizard, which is on the SMI software Communication menu. For details, see the SMI software online help, which is accessed by pressing the F1 key or selecting Help from the SMI software main menu.

BAUDrate, BAUD(channel)=formula

Set BAUD rate (RS-232 and RS-485)

The BAUD command sets the speed or baud rate of the specified serial channel. To do this, use:

- BAUDrate: sets the baud rate of the main channel
- BAUD(channel)=formula: sets the baud rate of the specified serial channel

where rate and formula are the desired baud rate, and (channel) is 0 or 1 for channel 0 or channel 1, respectively. Valid values for rate and formula are: 2400, 4800, 9600, 19200, 38400, 57600, or 115200. For additional motor-specific details, see Product-Specific Table on page 303.

PRINT(), PRINT1()

Print to RS-232 or AniLink channel

A variety of data formats can exist within the parentheses of the PRINT() command.

- A text string is marked as such by enclosing it between double quotation marks.
- Variables can be placed between the parentheses as well as two variables separated by one operator.
- To send out a specific byte value, prefix the value with the # sign and represent the value with as many as three decimal digits ranging from 0 to 255.
- Multiple types of data can be sent in a single PRINT() statement by separating the entries with commas.

NOTE: Do not use spaces outside of text strings because the SmartMotor uses spaces, carriage returns and line feeds as delimiters.

These are all valid print statements that transmit data through the main RS-232 channel:

```
PRINT("Hello World")    'text
PRINT(a*b)              'exp.
PRINT(#32)              'data
PRINT("A", a, a*b, #13) 'all
```

PRINT1 prints to the AniLink port with RS-485 protocol.

SILENT, SILENT1

Suppress PRINT() outputs

TALK, TALK1

Deassert Silent Mode

The SILENT mode causes all PRINT() output to be suppressed. This is useful when talking to a chain of motors from a host, when the chain would otherwise be talking within itself because of programs executing that contain PRINT() commands. The TALK and TALK1 commands restore print messaging.

a=CHN(channel)

Communication Error Flags

Where channel can be 0 or 1 for COM Channel 0 or 1. It holds binary coded information about historical errors on the two communications channels.

The command gives the 5-bit status of either serial port channel 0 or 1, as described in the next table.

Bit	Value	Meaning
0	1	Buffer overflow
1	2	Framing error
2	4	N/A
3	8	Parity error
4	16	Timeout occurred

The next example subroutine prints errors to an LCD display.

```

C9
  IF CHN(0)          'If CHN0 != 0
    IF CHN(0) &1
      PRINT("BUFFER OVERFLOW")
    ENDIF
    IF CHN(0) &2
      PRINT("FRAMING ERROR")
    ENDIF
    IF CHN(0) &8
      PRINT("PARITY ERROR")
    ENDIF
    IF CHN(0) &16
      PRINT("TIMEOUT OCCURRED")
    ENDIF
    Z(2,0)          'Reset CHN0 errors
  ENDIF
RETURN

```

a=ADDR

Motor's Self Address

If the motor's address (ADDR) is set by an external source, it may still be useful for the program in the motor to know to what address it is set. When a motor is set to an address, the ADDR variable reflects that address — the range is from 1 to 120.

Communicating over RS-485

Multiple SmartMotors can be connected to a single host port by connecting their RS-485 A signals together and B signals together, and then connecting them to an RS-485 port or to an RS-232 or USB adapter.

Adapters provided by Moog Animatics have built-in biasing resistors. However, extensive networks should add bias at the very last motor in the chain. The RS-485 signals of the SmartMotor share I/O functions and are not properly biased for more than just a few SmartMotors. Additionally, proper cabling would include a shielded twisted pair for transmission.

The main RS-232 ports of the D-style SmartMotors can be converted to RS-485 and isolated using Moog Animatics adapters.



The RS-232 and RS-485 ports have many configuration possibilities. To set the configuration options, use the OCHN command, which is described in the next section.

Using Data Mode

Data mode is used to retrieve data from the RS-232/RS-485 port.

If a communications port is in Command mode, then the motor responds to arriving commands it recognizes. However, if the port is opened in Data mode, then incoming data fills the 16-byte buffer until it is retrieved with the GETCHR command.

For D-style motors:

a=LEN	Number of characters in RS-232 buffer
a=LEN1	Number of characters in RS-485 buffer
a=GETCHR	Get character from RS-232 buffer
a=GETCHR1	Get character from RS-485 buffer

For M-style motors:

a=LEN	Number of characters in RS-485 buffer
a=GETCHR	Get character from RS-485 buffer

The buffer is a standard FIFO (First In First Out) buffer. This means that if the letter A is the first character the buffer receives, then it will be the first byte offered to the GETCHR command. The buffer exists to make sure that no data is lost, even if the program is not retrieving the data at just the right time.

The GETCHR buffer will stop accepting characters if the buffer overflows, and RLEN will stop incrementing. Also, the overflow bit will be set for that serial channel. When the buffer is empty, GETCHR will return a value of (negative 1.) If GETCHR is assigned to a byte ab[], then the value gets cast from the range -1 to +255 to the signed range -128 to +127. This causes -1 (empty buffer) to have the same value as char 255, since 255 gets cast to -1. It is recommended you assign GETCHR to a word or long to perform comparisons.

The LEN variable holds the number of characters in the buffer. A program must see that the LEN is greater than zero before issuing a command like a=GETCHR. Likewise, it is necessary to arrange the application so that, overall, data will be pulled out of the buffer as fast as it comes in.

The ability to configure the communication ports for any protocol as well as to both transmit and receive data allows the SmartMotor to interface with a vast array of RS-232 and RS-485 devices. Some of the typical devices that would interface with SmartMotors over the communication interface are:

- Other SmartMotors
- Bar Code Readers
- Light Curtains
- Terminals
- Printers

The next example program repeatedly transmits a message to an external device (in this case another SmartMotor) and then takes a number back from the device as a series of ASCII letter digits, each ranging from 0 to 9. A carriage return character marks the end of the received data. The program uses that data as a move position.

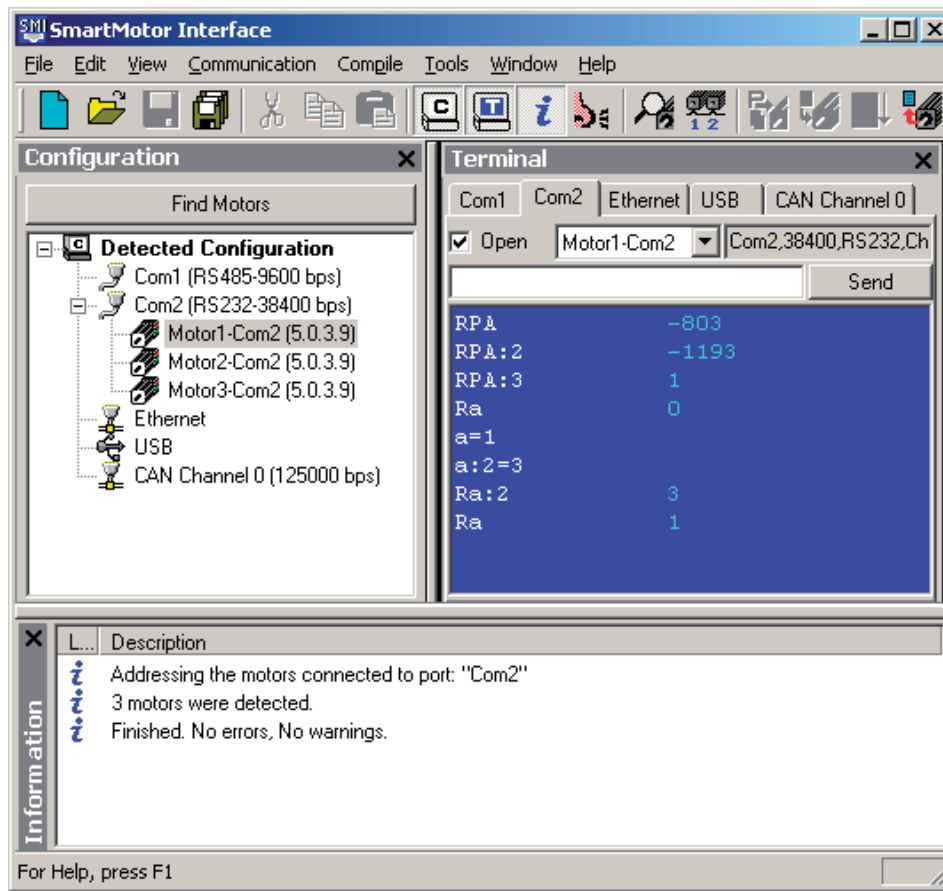
```

AT=500           'Preset acceleration.
VT=1000000      'Preset velocity
PT=0            'Zero out position.
O=0            'Declare origin
G              'Servo in place
OCHN (RS2, 0, N, 9600, 1, 8, D)
PRINT ("RPA", #13)
C0
  IF LEN                'Check for chars
    a=GETCHR           'Get char
    IF a==13           'If carriage return
      G               'Start motion
      PT=0            'Reset buffered P to zero
      PRINT ("RP", #13) 'Next
    ELSE PT=PT*10      'Shift buffered P
      a=a-48          'Adjust for ASCII
      PT=PT+a         'Build buffered P
    ENDIF
  ENDIF
GOTO (0)             'Loop forever

```

The ASCII code for zero is 48. The other nine digits count up from there so the ASCII code can be converted to a useful number by subtracting the value of 0 (ASCII 48). The example assumes that the most significant digits will be returned first. Any time it sees a new digit, it multiplies the previous quantity by 10 to shift it over and then adds the new digit as the least significant one. After a carriage return is seen (ASCII 13), motion starts. After motion starts, P (Position) is reset to zero in preparation for building up again. P is buffered, so it will not do anything until the G command is issued.

The SmartMotor has a wealth of data that can be retrieved over the Combitronic, RS-232 and RS-485 ports simply by asking. Data and status reporting commands can be tested by issuing these report commands from any hosting application. Using SMI Terminal window as the host (see the next figure), the command is shown on the left and the SmartMotor's response is shown in the middle.



SmartMotor Command with Response

The SMI host software uses these commands to implement the Motor View window and Monitor View tools. Data that does not have direct report commands can be retrieved either of two ways, by embedding the variable in a PRINT command, or by setting a variable equal to the parameter and then reporting the variable. For more details, see Part 2: SmartMotor Command Reference on page 247.

It is important to note that Combitronic reports only work if the CAN network is wired to each motor, and the CAN addresses and baud rate are configured. Keep in mind:

- Unique addresses must be assigned to each motor with the CADDR command.
- All motors on the same CAN network must be configured to the same baud rate with the CBAUD command.

CAN Communications

NOTE: DeviceNet is currently not available on the Class 6 SmartMotor.

The SmartMotor supports different protocols over the CAN port if equipped. CANopen and DeviceNet are popular industrial networks that use CAN. If a controller is communicating to a group of SmartMotors as follower devices through either of these standard protocols, the Combitronic protocol can still function without being seen by the CANopen or DeviceNet controller.

NOTE: The CAN network must have all devices set to the same baud rate to operate.

For more details about the CANopen implementation on the SmartMotor, see the CANopen fieldbus guide for your SmartMotor.

The next sections describe related commands. For more details on these commands, see Part 2: SmartMotor Command Reference on page 247.

CADDR=formula

Set CAN address

Where formula may be from 1 to 127. The setting is stored in the EEPROM. However, for it to take effect, the user must cycle power to the motor.

CBAUD=formula

Set CAN baud rate

Where formula may be one of these: 1000000, 800000, 500000, 250000, 125000, 100000, 50000, 20000. The setting is stored in the EEPROM. However, for it to take effect, the user must cycle power to the motor.

=CAN, =CAN(arg)

Get CAN error

The CAN command is used to get (read) an error or other status information about the CAN bus. For example:

RCAN(0), x=CAN(0): Report/get status bits relating to CAN.

RCAN(1), x=CAN(1): Report/get the current NMT state of this motor.

RCAN(4), x=CAN(4): Report/get the result code of the most recent SDO read or write, or NMT command as a controller.

For more details, see CAN, CAN(arg) on page 357.

Specific features are based on the fieldbus network being used. See the corresponding SmartMotor fieldbus guide for more details.

CANCTL(function,value)

Control network features

Commands execute based on the function argument to control CAN functions. For example:

function = 1: Reset the CAN MAC and all errors. Resets the CANopen stack, PROFIBUS stack or DeviceNet stack depending on firmware type. Value is ignored.

function = 5: Set timeout for Combitronic. Value is in milliseconds; the default is 30.

function = 16: Set the SDO command timeout period. In milliseconds. Range is 10 to 1000. Default is 500 (1/2 second).

function = 17: Enables the controller commands: NMT, SDORD and SDOWR. Enable simple controller: x is the value 3; disable controller: x is the value -1.

For more details, see CANCTL(function,value) on page 359.

Specific features are based on the fieldbus network being used. See the corresponding SmartMotor fieldbus guide for more details.

SDORD(...)

Read value from SDO

The SDORD command gets (reads) the value from the specified SDO on a specified device.

EXAMPLE: Read an SDO

```
x=SDORD(1, 24592,0,2)  ' Read 2 bytes from address 1,
                        ' object 0x6010, sub-index 0.
e=CAN(4)               ' Get any error information

y=SDORD(1, 24608,0,2)  ' Read 2 bytes from address 1,
                        ' object 0x6020, sub-index 0.
ee=CAN(4)              ' Get any error information

IF (e|ee)==0           ' Confirm the status of both SDO operations.
                        ' Success
    b=x                ' Set some example variable according
    c=y                ' to the data received.
    GOSUB(3)           ' Some routine to take action when this data is valid.
ELSE
    GOSUB(8)           ' Go do something to deal with error when read fails.
ENDIF
```

For more details, see SDORD(...) on page 730.

SDOWR(...)

Write value to SDO

The SDOWR command writes a value to the specified SDO on a specified device.

EXAMPLE: Write an SDO

```
a=1234
SDOWR(1,9029,0,4,a)   ' Write 4 bytes to address 1,
IF CAN(4)==0           ' Confirm the status of the most recent SDO operation.
                        ' Success
    GOSUB(4)           ' Some routine to take action when the write succeeds.
ELSE
    GOSUB(9)           ' Go do something to deal with error when write fails.
ENDIF
```

For more details, see SDOWR(...) on page 732.

NMT

Transmit NMT message to network

The NMT command transmits an NMT message to the network; it can command either a specific or all follower devices to enter the commanded state. The command uses the form:

NMT(target address, desired state)

```

NMT(0,1)    'Tell everyone to go operational.
NMT(2,128) 'Tell motor 2 to go pre-operational.
x=CAN(4)
IF x!=0
    ' NMT command failed.
ENDIF

```

For more details, see NMT on page 626.

RB(2,4), x=B(2,4)

Determine if CAN error has occurred

Report/get if an error state has occurred over CAN, CANopen or Combitronic. Further investigation through RCAN(0) will give more details. This can be cleared using the Z(2,4) or Z5 command.

For more details, see B(word,bit) on page 297.

Exceptions to NMT, SDORD and SDOWR Commands

Note these exceptions when using the NMT, SDORD, SDOWR commands:

- No Combitronic version of these commands, i.e., there is no ":" operator form of the command, for example:
`x=SDORD(...):3`
is not allowed. Refer to each command's description in Part 2 of this guide.
- No monitoring the heartbeat of other network nodes.
- No special commands for sending or receiving PDOs. PDOs must be mapped to existing objects to send or receive data as a follower device. Even the SmartMotor designated as a controller must configure its own PDO mappings.
NOTE: SmartMotors currently have 5 transmit and 5 receive PDOs.
- No capability to read EDS files. The user is responsible for writing a program with the relevant object index, sub-index and data type.
- No LSS host behavior is provided from the SmartMotor. Each follower device is expected to have the properly configured address and baud rate. Each device must have a unique address; all devices must use the same baud rate. Any need to set the baud rate or address is not the responsibility of Moog Animatics.
- Only one SmartMotor may fill the controller role. No other SmartMotors on the network may issue these commands, because this implementation does not support a multi-CANopen-controller functionality.
- No support for controller read/write of segmented or block SDO protocol. Only Expedited (32-bit or smaller) data transmission are supported by the controller functionality.

I/O Device CAN Bus Controller

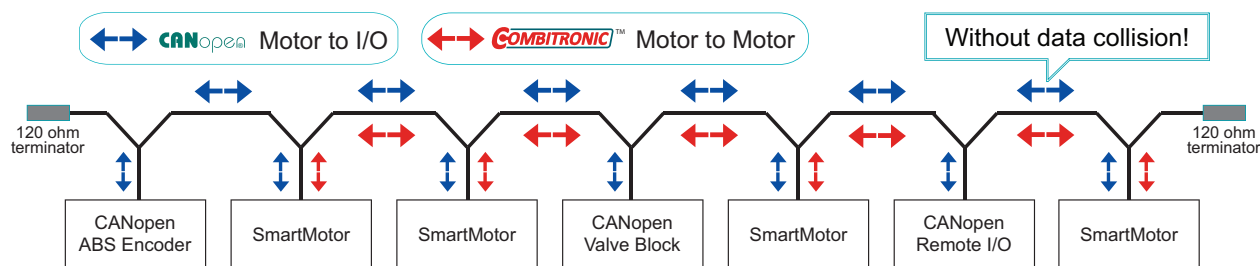
Many Moog Animatics SmartMotor servos, with appropriate firmware, can interface with standard CiA 301 CANopen devices, such as CANopen valve blocks, CANopen I/O blocks, CANopen encoders, and many other devices. This means through CAN and Combitronic communications, you now have full machine control with just a SmartMotor as the bus controller—no other external bus controller is required. This capability is enabled by the CAN communications commands (NMT, SDORD and SDOWR) described previously in this section, and new/modified objects.

NOTE: This capability is not available on all SmartMotor servos — for availability, see the *SmartMotor Installation & Startup Guide* for your motor or contact Moog Animatics.

Basic control allows 8, 16, or 32-bit sized data objects with support for both PDO and SDO protocols. The supported profiles include but are not limited to I/O profile, encoder profile, and DS4xx profile. This provides the ability to:

- Dynamically map SmartMotor PDOs, map another device's PDOs, start the NMT state
- A SmartMotor can send/receive up to 5 PDOs each or Rx (receive) and Tx (transmit)
- Read/write SDOs in expedited mode only, which works for up to 32-bit data

Multiple SmartMotors and multiple I/O devices may be on the same CAN bus. This combined with Combitronic motor-to-motor communications allows for complex, multi-axis, multi-I/O-device network control. Refer to the next figure.



Be sure to comply with the guidelines for CAN bus cabling and termination.

SmartMotor as I/O Device CAN Bus Controller

Related CANopen objects are: 2220h, 2221h and 2204h. For more details, refer to the object descriptions in the Object Reference chapter of the *SmartMotor CANopen Guide*.

Related commands are: NMT, SDORD, SDOWR, CANCTL, and B/RB. For details, see the brief descriptions in this section and the detailed descriptions in Part 2 of this guide.

Example user programs are shown in the Part 3 of this guide:

- CAN Bus - Timed SDO Poll on page 879
- CAN Bus - I/O Block with PDO Poll on page 880

Combitronic Communications

NOTE: For the Class 5 D- and M-style SmartMotors, Combitronic communication is available on models with the -CAN option. For the Class 6 D-style SmartMotor, Combitronic communication is a standard feature on all models. For the Class 6 M-style SmartMotor, Combitronic communication is currently available only on -EIP option motors. For details, see the *Class 6 SmartMotor™ EtherNet/IP Guide*.

The most unique feature of the SmartMotor is its ability to communicate with other SmartMotors and share resources using Moog Animatics' Combitronic™ technology. Combitronic is a protocol that operates over a standard "CAN" (Controller Area Network) interface. It may coexist with either CANopen or DeviceNet protocols at the same time. It requires no single dedicated controller to operate. Each SmartMotor connected to the same network communicates on an equal footing, sharing all information, and therefore, sharing all processing resources.

The optional Combitronic technology allows any motor's program to read from, write to or control any other motor simply by tagging a local variable or command with the other motor's CAN address. To do this, take any Combitronic-supported SmartMotor command, add a colon and then a number representing the address of another SmartMotor on the same CAN bus, and that parameter belongs to that SmartMotor.

For example, imagine you have three SmartMotors linked together and set with addresses 1, 2 and 3. These examples show how Combitronic communications works:

- This typical line of code, written in SmartMotor number 2, sets a target position in that same SmartMotor:

```
PT=4000      'Set Target Position in local motor
```

- This line of code, written in SmartMotor number 2, or any of the three motors, sets a target position in SmartMotor number 3:

```
PT:3=4000    'Set Target Position in motor 3
```

- The Combitronic global address for all SmartMotors is zero, so the next line of code, written in any SmartMotor, sets the target position in all SmartMotors at the same time:

```
PT:0=4000    'Set Target Position in all motors
```

- This line of code could be written in motor number 1 and set variable "a" in motor number 2 equal to an I/O of motor number 3:

```
a:2=IN(0):3  'Set variable in 2 to I/O of 3
```

For a complete list of Combitronic commands, see *Commands for Combitronic* on page 963.

Combitronic Features

- 127 addressable nodes
- 1 Mbps over the CAN bus
- No controller required
- No scan list or node list set up required
- All nodes have full read/write access to all other nodes

Other Combitronic Benefits

Combitronic technology provides a simple way to create a true parallel-processing environment.

- PLCs (Programmable Logic Controllers) can be eliminated, due to the speed of program execution within the SmartMotor, combined with the speed of the Combitronic communications and the power of the SmartMotor's programming language.
- Sensors and valves can be connected to the closest SmartMotor in the machine and be available to the program of any SmartMotor on the network.
- An HMI (Human Machine Interface) can connect to any one or more of the SmartMotor's RS-232 or RS-485 ports and provide visibility into the entire network.
- The size and complexity of the machine collapses to the point where, in many cases, there is no longer even a cabinet.

As a result, the machine builder is spared the traditional bulk, failure modes, wiring time and complexity, and costs of separate servo controllers, servo amplifiers and PLCs.

Program Loops with Combitronic

Keep in mind that while Combitronic communications are very fast, program execution is also very fast. Therefore, if a tight loop is written with a Combitronic transaction inside, you will flood the CAN bus with data, which can slow the operations of all SmartMotors on the chain.



CAUTION: Tight loops with Combitronic commands can flood the CAN bus with data and impair the function of a SmartMotor network. For the best performance, structure programs to minimize disturbance of the CAN infrastructure.

This problem can be avoided. For example, if motor 1 needs to poll the state of an input on motor 2, then instead of writing a tight loop with a Combitronic command in it:

1. Write a tight loop in motor 2 that executes a Combitronic transmission only when that input changes state.
2. Issue a Combitronic command in motor 2 that sets a variable in motor 1 in the event of the input state change.
3. Program motor 1 to poll its own internal variable.

This way, the actual polling activity is not occupying the CAN bus.

NOTE: A key to powerful programming in SmartMotors is to exploit parallel processing for throughput without unnecessary polling over the Combitronic interface, which needlessly wastes throughput.

Global Combitronic Transmissions

Global Combitronic transmissions are especially fast because they do not involve node responses at the protocol level. This fact can be leveraged to speed applications by having certain motors globally broadcast infrequent but relevant state changes. For example, if a machine had a "door" and that door could be opened or closed, the motor performing that function could set every motor's variable "d" equal to 1 when the door is opened and 0 when the door is closed, like this:

```
d:0=1  
d:0=0
```

The program in each motor can simply check its own variable "d" for the status of the door. Through this technique, the programmer has created a new type of "global" variable.

A clever way to program a network of SmartMotors is to write *one* program and download that same program to *all* motors. Then have the program first look to the motor's CAN address and execute only the portion of the controller program that pertains to that motor address. This makes supporting a

large network much easier because there is only one program. Make sure "global" variables, as created in the previous example, are all unique.

Simplify Machine Support

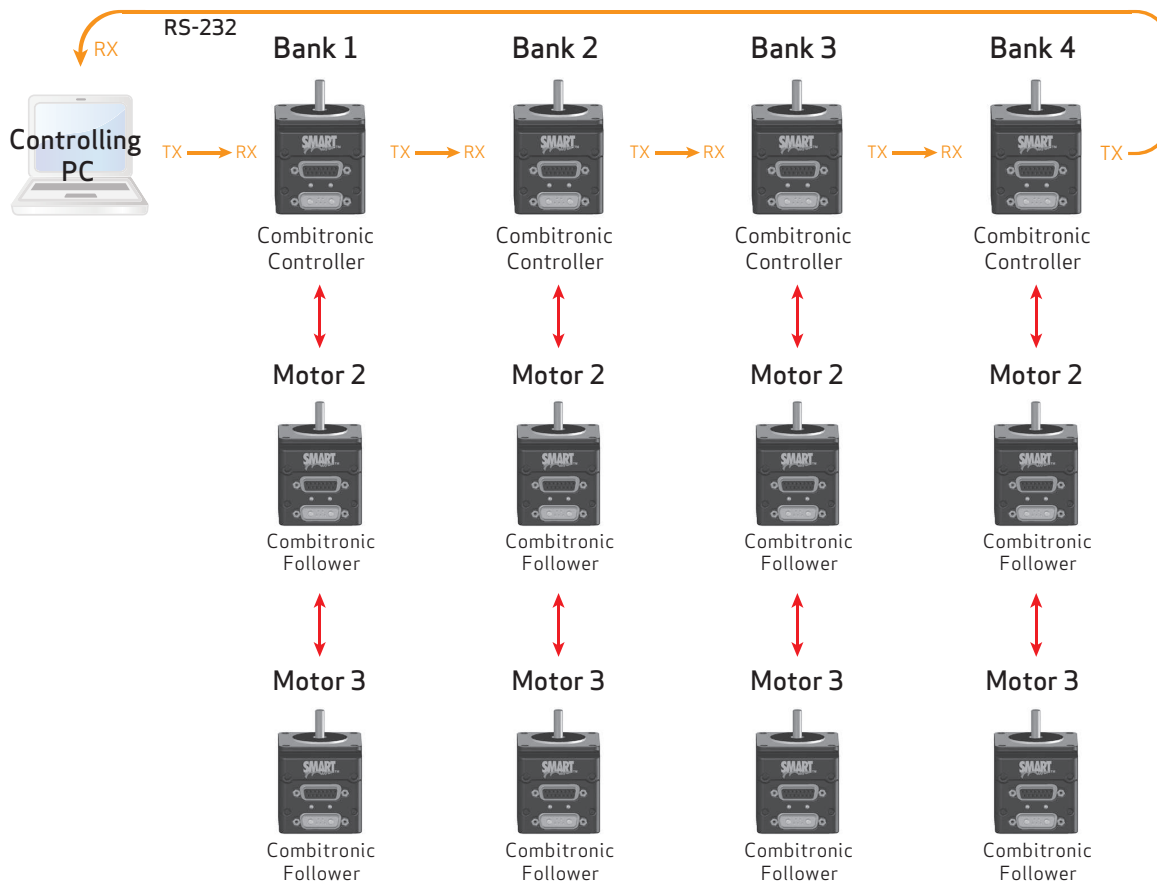
Combitronic features can also be used to simplify the support of a SmartMotor-based machine. To do this:

1. Allocate a small group of I/O, or the analog value of an input, to be unique in each motor position through the wiring leading to that motor.
2. Have the program set its CAN address in accordance with that unique input status.

With this technique, a spare SmartMotor containing the controller program could quickly replace any failed motor in the system without any special configuration. Even its own address would be automatically set.

Combitronic with RS-232 Interface

Any SmartMotor may be used as a controller access through RS-232 to all SmartMotors on its network. The next figure demonstrates 12 motors in a network where four SmartMotors are in a serial daisy chain over RS-232. Each of those four banks may have up to 119 motors on its Combitronic network.



RS-232 and Combitronic Networks

Example SMI software commands from the host PC RS-232 port for the system layout in the previous figure:

2PT:3=1234	Motor 2 sets target position of Motor 3 in its group to 1234
3PT:0=0	Motor 3 sets target position of all motors in its group to zero
4PT=345	Motor 4, only, gets its own target position set to 345
0G	Motor 1, 2, 3 and 4 receive Go command
0G:0	All motors on RS-232 and all network Combitronic motors receive Go command

Combitronic with the DS2020 Combitronic System

NOTE: DS2020 support requires: 5.0.4.55 (D), 5.98.4.55 (M); 6.4.2.x (D); ds2020_sa_1.0.0_combican (DS2020).

The Moog Animatics DS2020 Combitronic system is a cabinet mount servo drive connected to a Moog Compact Dynamic brushless servo motor. Compared to the smaller 17 to 34 frame SmartMotor products, the DS2020 Combitronic system provides access to a higher torque motor-drive combination, with torque range and power inputs to include AC mains voltages and motors above 1 KW. However, similar to other SmartMotor products, the DS2020 Combitronic system has the capability of responding to Combitronic commands.

The DS2020 Combitronic system is not fully programmable but is connected as a follower device to a SmartMotor controller. The DS2020 Combitronic system has a CAN address, which you can set through SMI along with baud rates as you would with any SmartMotor. It is then commanded by the SmartMotor through Combitronic communications using standard Combitronic syntax, e.g., ADT:3=1234, where "3" is the CAN address of the DS2020 Combitronic system.

The DS2020 Combitronic system supports a subset of the full AniBasic command set. Supported commands are primarily Combitronic type, but there are a few others, also. The DS2020 Combitronic system supported commands are flagged with "; supports the DS2020 Combitronic system" text on the command's APPLICATION line or READ/REPORT line.

For a list of DS2020 Combitronic system supported commands, see Commands for DS2020 Combitronic on page 967

For details on the DS2020 Combitronic system installation and startup, see the *DS2020 Combitronic Installation and Startup Guide*.

Other CAN Protocols

This section briefly describes two other supported CAN protocols: CANopen and DeviceNet.

NOTE: DeviceNet is currently not available on the Class 6 SmartMotor.

CANopen - CAN Bus Protocol

CANopen is an industrial CAN bus protocol supported on SmartMotors ordered with the CANopen option. The protocol supports the CiA 402 profile for drives and motion devices. The hosting controller can use an EDS file supplied by Moog Animatics that provides control of the SmartMotor over the CANopen network.

One of the more powerful features of the CIA 402 profile is Interpolation mode, which is supported by both the CANopen-enabled SmartMotor and Moog Animatics' own coordinated-motion software, SMNC and Integrated Motion DLL. By itself, the Integrated Motion DLL offers the host-application developer the means to control SmartMotors using CANopen.

DeviceNet - CAN Bus Protocol

NOTE: DeviceNet is currently not available on the Class 6 SmartMotor.

DeviceNet is an industrial CAN bus protocol supported in the SmartMotor with optional firmware. The protocol supports the Common Industrial Protocol (CIP) profile for a position controller. The hosting controller can use an Electronic Data Sheet (EDS) file supplied by Moog Animatics that allows the SmartMotor to be controlled through DeviceNet.

I²C Communications (Class 5 D-Style Motors)

The Class 5 D-style SmartMotors provide open I²C (IIC) communications capabilities, which expand the capabilities of that SmartMotor.

NOTE: I²C communications is *not* currently available on the Class 5 M-style or any Class 6 SmartMotor.

The I²C capability is comprised of two signals, SDA and SCL, on ports 4 and 5 of the 15-pin D-sub connector, respectively. These ports are most often shared with the SmartMotor's RS-485 ports. Therefore, to set up I²C communications, a choice must be made between I²C and RS-485 communications.

There are I²C devices that perform dozens of functions, such as nonvolatile memory, high resolution A-to-D and D-to-A conversion, analog and digital I/O expansion and more.

The next program example shows how to use I²C communications with a small EEPROM memory device known as the 24FC512. Only the initialization part runs at power-up. Thereafter, subroutines 100 and 200 can be called to write or read data into the EEPROM.

```

.....
' Class 5 I2C EEPROM Test 00
' Sept 10, 2009
' I2C test for 24FC512 EEPROM on Personality Module
' Address 1010 001 x
.....

SADDR1
ECHO
C0
OFF          'Turn off drive stage power
OCHN(IIC,1,N,200000,1,8,D)  'Initialize I/Os 4 and 5 as IIC port
PRINT(#13,"IIC Port Initialized",#13)
PRINT(#13)
END

C100        'Write variable a at pointer p
al[0]=a
al[1]=p
PRINT(#13)
PRINT("Load ",al[0]," at pos ",p,#13)
PRINT1(IIS,#160,#ab[5],#ab[4],#ab[3],#ab[2],#ab[1],#ab[0],IIP)
PRINT("Load bytes: ",ab[3]," ",ab[2]," ",ab[1]," ",ab[0],#13)
PRINT(#13)
RETURN

C200        'Read into variable a at pointer p
al[1]=p
PRINT1(IIS,#160,#ab[5],#ab[4],IIP)      'Write memory pointer
WAIT=1      'Must have small wait to give the write time it needs
PRINT1(IIS,#161,IIG4,IIP)              'Setup to read four bytes
WAIT=1      'Must have small wait to give the write time it needs
ab[3]=GETCHR1
ab[2]=GETCHR1
ab[1]=GETCHR1
ab[0]=GETCHR1
a=al[0]
PRINT(#13)
PRINT("Read bytes: ",ab[3]," ",ab[2]," ",ab[1]," ",ab[0],#13)
PRINT("Read ",a," at pos ",p,#13)
PRINT(#13)
RETURN

```

The next sections describe related commands. For more details on these commands, see Part 2: SmartMotor Command Reference on page 247.

OCHN(IIC,1,N,baud,1,8,D)

The OCHN command is used to set the I²C communication parameters:

IIC	Literal syntax IIC to tell what kind of communication this is
1	Literal value 1, since this is the location of that port
N	Literal, not relevant to IIC
baud	Bit rate for communication with the IIC device
1	Literal, not relevant to IIC
8	Literal, not relevant to IIC
D	Literal, always in data mode for IIC communication

CCHN(IIC,1)

The CCHN(IIC,1) command is simply used to close the I²C communications channel.

PRINT1(arg1,arg2, ... ,arg_n)

Where arg is:

IIS	Start or restart an IIC command. For IIC devices that require a restart, simply call the IIS command a second time within a PRINT command.
IIP	Stop an IIC command.
IIGn	Get n bytes from the IIC channel (requires the previous commands to have provided whatever addressing or command is required for the device to start sending). The G argument provides the right number of clock intervals to acquire the data from the IIC device.

RGETCHR1, Var=GETCHR1

Returns data from the IIC device (if available). The data is always in unsigned byte values, so assign the data to a 16 or 32-bit register first in order to test for special cases.

For example, the value is 0-255 for normal data, which represents all possible values for the byte. If the value from the GETCHR1 command is -1, it means the buffer is empty.

RLEN1, Var=LEN1

Gets the number of bytes in the receive buffer.

Motion Details

This chapter provides details on making motion with the SmartMotor.

Introduction	122
Motion Command Quick Reference	123
Basic Motion Commands	124
Target Commands	124
Motion Mode Commands	126
Torque Commands	127
Brake Commands	127
Index Capture Commands	130
Other Motion Commands	132
Commutation Modes	134
MDT	134
MDE	134
MDS	134
MDC	135
MDB	135
MINV(0), MINV(1)	135
Modes of Operation	136
Torque Mode	136
Velocity Mode	137
Absolute (Position) Mode	138
Relative Position Mode	139
Follow Mode with Ratio (Electronic Gearing)	140
Cam Mode (Electronic Camming)	156
Mode Switch Example	171
Position Counters	173
Modulo Position	174
Modulo Position Commands	174
Dual Trajectories	175
Commands That Read Trajectory Information	177
Dual Trajectory Example Program	178
Using Mixed Mode Operations After Homing	179
Synchronized Motion	179
Synchronized-Target Commands	179
Other Synchronized-Motion Commands	183

Introduction

All SmartMotor™ commands are grouped by function with these notations:

#	Numerical integer value, constrained by command. For example, 0, 1,..22.
formula (or frm)	Formula or number. For example, 123 or a=1 or a=(2*3)-1.
expression (or exp)	Simple expression or number. For example, a+3 or al[1] or 5.
W	(Capital W letter by itself.) Refers to " <u>W</u> ord", or 16 bits of information. Option for addressing I/O 16-bit status words.
mask (or msk)	The mask value of the bits that will be affected when working with integers can typically be passed into a command as an expression. Mainly used for I/O and status word bit manipulations. For details on binary data, see Binary Data on page 900.

NOTE: In the command syntax, when optional bracketed arguments are shown, the comma within the brackets is only used with the optional argument. For example, the comma is used with the optional "m/s" argument in the command MFSLEW(distance[,m/s]).

Enter these commands in the Terminal window to move the SmartMotor:

```
EIGN (2)      'Disable left limit
EIGN (3)      'Disable right limit
ZS            'Reset errors
ADT=100       'Set target accel/decel
VT=100000     'Set target velocity
PT=300000    'Set target position
G             'Go, starts the move
```

NOTE: As shown in the example, a complete move requires: a position, a velocity and an acceleration, and then a G (Go) command to start the move.

On power-up the motor defaults to position mode. Once Acceleration-Deceleration Target (ADT) and Velocity Target (VT) are set, simply issue new Position Target (PT) commands, and then a Go (G) command to execute moves to new absolute locations. The motor does not instantly go to the programmed position but uses a trajectory to get there. The trajectory is bound by the maximum target velocity and target acceleration parameters. The result is a trapezoidal velocity profile, or a triangular profile if the maximum velocity is never met.

NOTE: Position, velocity and acceleration can be changed at any time during or between moves. However, the new parameters only apply when a new G command is sent.

NOTE: Many motion commands and related report commands are affected by the scaling commands (SCALEA, SCALEP and SCALEV). For details, see SCALEA(m,d) on page 724, SCALEP(m,d) on page 726, and SCALEV(m,d) on page 728. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

Motion Command Quick Reference

The next table provides a quick reference for the primary Class 5 motion commands. For the complete list of motion control commands and links to their descriptions, see Motion Control on page 957.

		Absolute Position	Relative Position	Velocity	Accel and Decel Together	Accel	Decel	Following Error	DE/Dt Derivative Error Limit	Over Speed Limit
Report	Actual	RPA	RPRA	RVA	N/A	N/A		REA	RDEA	
Report	End Target	RPT	RPRT	RVT	RAT	RAT	RDT	REL	RDEL	RVL
Report	Commanded	RPC	RPRC	RVC	RAC	RAC				
Assign	End Target	PT=	PRT=	VT=	ADT=	AT=	DT=			
Assign	Command	N/A	N/A	N/A	N/A	N/A	N/A	EL=	DEL=	VL=

In the chart above, you will notice Actual, End Target, and Commanded:

- Actual: The value of the parameter as the processor sees it in real time at the shaft, regardless of anything commanded by the trajectory generator
- Target: The requested trajectory target to reach and/or maintain at any given time
- Commanded: The compensated value of the trajectory generator at any time in its attempt to reach the target

For example, in terms of the position commands:

- Position Target (PT): The desired target position you are shooting for; what you have specified as a target position value
- Position Actual (PA): The current position in real time (right now), regardless of target or where it is being told to go
- Position Commanded (PC): The position the controller processor is actually commanding it to go to at the time

NOTE: Any difference between Position Commanded (PC) and Position Actual (PA) is due to position error.

There are two position types:

- Absolute: The finite position value in reference to position zero
- Relative: A relative distance from the present position at the time

All commands shown above are associated with both Mode Position (MP) and Mode Velocity (MV). They may also be used in dual trajectory mode when running either of those modes on top of gearing or camming.

All distance parameters are in encoder counts. Encoder resolution may be obtained and used in a program through the RES command. The RRES command will report encoder resolution. You can also use the RES command directly in math formulas.

EXAMPLE:

If you want it the axis to move to location 1234, then you would issue:

PT=1234

While moving there:

- RPC would report the commanded position from the processor.
- RPA would report actual position of the encoder or motor shaft.
- $x=PC-PA$ would calculate position error at that moment.
- REA would report actual position error at that moment.
- RBt would report a 1 (while moving) because the trajectory is active.

After the move has completed, RBt would report a 0 (to indicate the trajectory is no longer active).

Basic Motion Commands

The basic motion commands described in this section are used to set the operating mode, control acceleration/deceleration, velocity, torque, origin and position, and to start and stop the motion. Use the Motion Command Quick Reference on page 895 to understand the relationship between basic motion commands and the terms Actual, Commanded and Target.

Target Commands

The section describes target-related commands. For more details on these commands, see Part 2: SmartMotor Command Reference on page 247.

PT=formula

Set Target Position (Absolute)

The PT command sets an absolute end position to move to when the motor is in Position mode. The units are encoder counts and can be positive or negative in the range -2147483648 to +2147483647. It is not advisable to attempt to use absolute moves that would cross the rollover point of the most positive and most negative values. Also, absolute moves should not attempt to specify a move with a relative distance of more than 2147483647. The end position can be set or changed at any time during or at the end of previous moves. SmartMotor™ sizes 17 and 23 resolve 4000 increments per revolution, while SmartMotor size 34 resolves 8000 increments per revolution.

The next program illustrates how variables can be used to set motion values to real-world units and have the working values scaled in motor units for a size 17 or 23 SmartMotor.

```

a=100           'Acceleration in rev/sec*sec
v=1            'Velocity in rev/sec
p=100         'Position in revs
GOSUB (10)     'Initiate motion
END           'End program
C10          'Motion routine
  ADT=a*4.096 'Set target accel/decel
  VT=v*32768  'Set target velocity
  PT=p*4000   'Set target position
  G          'Start move
RETURN       'Return to call

```

NOTE: If any errors exist, they must be cleared before the G command will work. All errors can be cleared with the ZS command.

PRT=formula***Set Relative Target Position***

The PRT command allows a relative-distance move to be specified when the motor is in position mode. The value indicates the encoder counts to travel; it must be within the range of -2147483648 to +2147483647. This relative distance is added to the current trajectory position and not the actual position either during or after a move. If a previous move is still in progress, then the current trajectory position is added to when G is commanded. If the total distance traveled needs to directly correspond to the number of moves made, make sure a move has finished before issuing another G command.

ADT=formula***Set Target Acceleration/Deceleration***

Target Acceleration/Deceleration must be a positive integer within the range of 0 to 2147483647. The default is zero, so a nonzero number must be entered to initiate motion. A typical value is 100. This command sets acceleration and deceleration of the motion profile to the value specified. This value can be changed at any time. The value set does not take effect until the next G command is executed. Native acceleration units are (counts/sample/sample)*65536. The default sample rate for Class 5 is 8.0 kHz; the default sample rate for Class 6 is 16.0 kHz.

AT=formula***Set Target Acceleration Only*****DT=formula*****Set Target Deceleration Only***

The AT and DT commands allow setting different values for the acceleration and deceleration of the motion profile, respectively. Standard practice should be to use the ADT command instead unless separate values are needed. There is an override that automatically sets DT equal to AT if the motor power is turned on and only AT is set. However, this should be avoided by using the ADT command when DT is not used.

To convert acceleration in revolutions per second² to units of ADT, AT or DT, use this formula:

$$\text{ADT} = \text{Acceleration} * ((\text{enc. counts per rev.})/(\text{sample rate}^2)) * 65536$$

If the motor has a 4000 count encoder (sizes 17 and 23), multiply the desired acceleration, in rev/sec², by 4.096 to arrive at the appropriate setting for ADT. With an 8000 count encoder (size 34), the multiplier is 8.192. These factors assume a PID rate of 8.0 kHz, which is the default.

Note that ADT, AT and DT allow only even numbers. When odd numbers are used, they are rounded up. The default values are zero.

VT=formula***Set Target Velocity***

The VT command specifies a target velocity (speed and direction) for velocity moves, or a slew speed for position moves. The value must be in the range -2147483647 to 2147483647. Note that in position moves, this value is the unsigned speed of the move and does not imply direction. The value set by the VT command only governs the calculated trajectory of MP and MV modes (position and velocity). In either of these modes, the PID compensator may need to "catch up" if the actual position falls behind the trajectory position. In this case, the actual speed exceeds the target speed. The value defaults to

zero, so it must be set before any motion can take place. The new value does not take effect until the next G command is issued.

To convert velocity in revolutions per second to units of VT, use this formula:

$$VT = \text{Velocity} * ((\text{enc. counts per rev.})/(\text{sample rate})) * 65536$$

If the motor has a 4000 count encoder (sizes 17 and 23), multiply the desired velocity in rev/sec by 32768 to arrive at the setting for VT. With an 8000 count encoder (size 34), the multiplier is 65536. These factors assume a PID rate of 8.0 kHz, which is the default.

Motion Mode Commands

The section describes motion-mode commands. For more details on these commands, see Part 2: SmartMotor Command Reference on page 247.

MP

Position Mode

Issuing the Mode Position (MP) command puts the SmartMotor in Position mode. Position mode is the default mode of operation for the SmartMotor on power-up. In Position mode, the PT, PRT, VT, ADT, AT and DT commands can be used to govern motion. At a minimum, ADT, VT and (PT or PRT) must be issued.

MV

Velocity Mode

The Mode Velocity (MV) command allows continuous rotation of the motor shaft. In Velocity mode, the programmed position using the PT or the PRT commands is ignored. Acceleration and velocity need to be specified using the ADT and the VT commands. After a G command is issued, the motor accelerates up to the programmed velocity and continues at that velocity indefinitely. Similar to Position mode, in Velocity mode, velocity and acceleration are changeable on the fly, at any time — simply specify new values and enter another G command to trigger the change. In Velocity mode, the velocity can be entered as a negative number, unlike in Position mode where the location of the target position determines velocity direction or sign. If the 32-bit register that holds position rolls over in Velocity mode, it will have no effect on the motion.

Velocity mode calculates its trajectory as an ideal position over time and corrects the resulting measured position error instead of measuring velocity error. This is significant in that this mode will "catch up" lost position, just as Position mode will if a disturbance causes a lagging position error.

MT

Torque Mode

The Mode Torque (MT) command puts the SmartMotor in Torque mode. In Torque mode, the motor applies a PWM commutation effort to the motor proportional to the T command and independent of position. If the motor model has a current-control commutation mode, then torque is controlled in proportion to the T command. Otherwise, torque depends on the actual motor speed and bus voltage, eventually reaching an equilibrium speed. Nevertheless, for a locked rotor, the torque will be largely proportional to the T value and bus voltage.

To run the motor in Torque mode, use the T command and issue a G command for the new torque value to take effect.

NOTE: You must issue a G command for a new torque value to take effect in Torque mode.

Internal encoder tracking still takes place and can be read by a host or program. However, the value will be ignored for motion because the PID loop is inactive.

Torque Commands

These commands set the torque slope and value. For more details on these commands, see Part 2: SmartMotor Command Reference on page 247.

TS=formula

Set Torque Slope

The TS= command causes new torque settings to be reached gradually, rather than instantly. Values may be from -1 to +2147483647. -1 disables the slope feature and causes new torque values to be reached immediately. A TS setting of 65536 increases the output torque by one unit per PID sample period.

T=formula

Set Torque Value, -32767 to 32767

In Torque mode, activated by the MT command, the drive duty cycle can be set with the T= command. The value (number or variable) must fall in the range from -32767 to 32767. The full-scale value relates to full-scale or maximum-duty cycle. At a given speed, there will be reasonable correlation between drive duty cycle and torque. With nothing loading the shaft, the T= command will dictate open-loop speed. A G command must be entered after the T= command for the new value to take effect.

The next example increases torque, one unit every PID sample period, up to 8000 units.

```
MT           'Select torque mode.
T=8000      'Final torque after the TS ramp that we want.
TS=65536    'Increase the torque by 1 unit of T per PID sample.
G           'Begin move
```

Brake Commands

These commands control the brake functions for the motion. For more details on these commands, see Part 2: SmartMotor Command Reference on page 247.

BRKRLS

Brake Release - manual override command

Mechanically disengages brake (regardless of the brake operational mode)

BRKENG

Brake Engage - manual override command

Mechanically engages brake (regardless of the brake operational mode)

NOTE: When BRKSRV or BRKTRJ is issued *after* a manual override command has been issued, the brake will respond to the state of automatic control of the mode chosen.

BRKSRV***Automatically Release Brake Only When Servo Active***

(Default mode of operation) causes brake to mechanically engage when: the motor faults (for any reason), OFF is issued, or the drive is already in OFF state.

BRKTRJ***Automatically Release Brake Only When in Trajectory***

(Optional mode of operation) causes brake to mechanically engage any time the Trajectory bit is off and brake to release any time the trajectory bit turns on.

The SmartMotor is available with power-loss brakes. These brakes apply a force to keep the shaft from rotating should the SmartMotor lose power. Issuing the BRKRLS command releases the brake and BRKENG engages it. There are two other commands that initiate automated operating modes for the brake. The command BRKSRV engages the brake automatically, should the motor stop servoing and no longer hold position for any reason. This event might be due to loss of power or just a position error, limit fault or overtemperature fault.

Finally, the BRKTRJ command engages the brake in response to all of the previously-mentioned events, including any time the motor is not performing a trajectory. In this mode the motor is off and the brake holds it in position rather than the motor servoing when it is at rest. As soon as another trajectory is started, the brake releases. The time it takes for the brake to engage and release is only a few milliseconds.

The brakes used in the SmartMotor are zero-backlash devices with extremely long life spans. It is well within their capabilities to operate interactively within an application. However, take care to avoid a situation where the brake sets repeatedly during motion, which will reduce the brake life.

Where a SmartMotor is not equipped with a physical brake, it simulates braking with its Mode Torque Brake (MTB) feature, which causes a faulted motor to still experience strong resistance to shaft motion. Note that MTB only works when power is applied to the SmartMotor. Therefore, it is not a substitute for an actual brake when safety is an issue.



WARNING: The MTB feature only works when power is applied to the SmartMotor. Therefore, DO NOT use it as a substitute for a physical brake when operator or equipment safety is an issue.

Brake Command Examples***Example 1***

Motor drive is in the ON state and not moving — it may be in Position mode and holding position, or in Velocity mode with zero velocity (same as holding position):

- If BRKENG is issued, the brake will engage (even if not already engaged for whatever reason) and you then issue BRKTRJ, the brake will STAY engaged
- If BRKRLS is issued, the brake will release (even if not already released for whatever reason), and then you then issue BRKTRJ, the brake will engage.

Example 2

The motor is faulted due to any typical fault, such as travel limits, overcurrent, overtemp, etc. The brake should already be mechanically engaged regardless of the mode (BRKTRJ or BRKSRV), and

provided no manual override commands were issued since the fault occurred.

- If BRKENG is issued, the brake will stay engaged; then issue BRKTRJ, the brake will stay engaged.
- If BRKRLS is issued, the brake will release (even though the motor is faulted because a manual override command was just issued); you then issue BRKTRJ, the brake will engage because the trajectory bit is off due to the fact that the motor is faulted.
- If the motor was in BRKTRJ and BRKSRV is issued, the brake will remain mechanically engaged.
- If the motor was in BRKSRV and BRKTRJ is issued, the brake will remain mechanically engaged.

Example 3:

If the motor is moving or holding position, *and* the Trajectory bit is ON, *and* no manual override commands have been issued — regardless of modes BRKSRV or BRKTRJ, the brake will be mechanically disengaged.

- If BRKENG is issued and the motor is NOT moving, the brake will engage.
- If BRKENG is issued and the motor IS moving, the brake will engage causing the motor to mechanically be loaded to the point of stopping and faulting out.

Example 4:

Because BRKSRV requires 3 to 5 milliseconds to fully engage, there may be certain cases where it isn't fast enough to hold the position, e.g., a vertical load where the user wants to put the machine to bed for the night without any position slippage.

Use this procedure:

1. The program / host checks for zero motion and it knows there are no further motion commands
2. Then issue BRKENG
3. Wait for mechanical brake engage time (3 to 5 milliseconds)
4. Turn motor OFF
5. Remove power from the machine

EOBK(IO)

Reroute Brake Signal to I/O

NOTE: When using the EOBK and MFR commands in the same program, there is interaction that must be considered in the code. For details, see the Programming Note in EOBK(IO) on page 445 or MFR on page 600.

When the automated brake functions are desired for an external brake, this command can be used to choose a specified I/O port. This corresponds to the same I/O pin numbering used by other I/O commands. These commands re-route the internal brake signal to the respective I/O pins. The brake signal is active high to engage the brake to the shaft on the pulled-up 5 Volt I/O. On the 24 Volt I/O, the default state is off (0 Volts), so the brake engages the shaft when the 24 Volt signal is low. The EOBK(-1) command removes the brake function from any external I/O. Only one pin can be used as the brake pin at any one time. Therefore, each command supersedes the other.

For the M-style SmartMotor, only output 8 works for that motor. Therefore, the values are:

- EOBK(8) to enable
- EOBK(-1) to disable

MTB

Mode Torque Brake

Mode Torque Brake is the default state on power-up. It causes the motor control circuits to tie the three phases of the motor together as a form of dynamic braking. For a fault or the OFF command, instead of the motor coasting to a stop, it abruptly stops. This is not done by servoing the motor to a stop, but by simply shorting all of the coils to ground. If there is a constant torque on the motor, it allows only very slow movement of the shaft.



WARNING: The MTB feature only works when power is applied to the SmartMotor. Therefore, DO NOT use it as a substitute for a physical brake when operator or equipment safety is an issue.

The MTB command immediately activates dynamic braking independently of the Brake mode. Issuing MTB while the motor is running turns off the motor drive and enables dynamic braking, even if BRKRLS has been issued.

To remove the effect of the MTB command, either issue a motion command, or manually "freewheel" the motor by issuing a BRKRLS command and then an OFF command. Those two commands do not need to be in immediate sequence—i.e., other commands, except MTB, can be between them.

BRKENG can engage dynamic braking unconditionally, as well. (The opposite of that command is BRKRLS.) Note that OFF will not remove the effect of BRKENG.

To ensure that the MTB command is not active, command BRKRLS and the dynamic braking will release. Finally, because faults can also activate dynamic braking, clear the faults or choose a fault action of freewheel (refer to the next NOTE).

NOTE: The FSA command's default cause/action enables MTB on all faults, even if previously disabled as described in the previous paragraphs. Therefore, to prevent that action, you must issue FSA(cause,action), where "cause" is the fault type 0, 1 or 2, and the "action" is 1, which specifies servo off (freewheel). For more details, see FSA(cause,action) on page 465.

Status Word 6, Bit 11 reports if dynamic braking is active or not, including as a result of the MTB command, the BRKENG command or a fault action.

Index Capture Commands

The SmartMotor's encoder capture mechanism has many capabilities. Both the internal and external encoders can be triggered by certain events to capture their positions.

The DS2020 Combitronic system also provides index capture capability; for details, see DS2020 Combitronic System Index Capture on page 131.

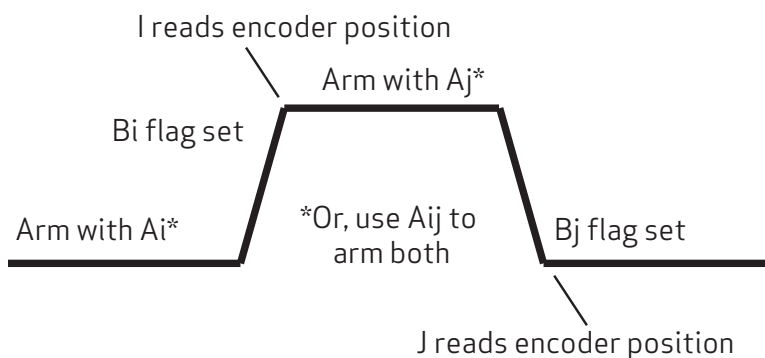
For a capture to occur, one of the arming commands must be issued (see the next list). These commands allow you to select a rising or falling edge of the source event and specify the encoder to be armed.

- Ai(arg) Arm the rising edge only; encoder selected by arg
- Aj(arg) Arm the falling edge only; encoder selected by arg
- Aij(arg) Arm the rising edge, wait for that event, then arm the falling edge; encoder selected by arg
- Aji(arg) Arm the falling edge, wait for that event, then arm the rising edge; encoder selected by arg

Arg is 0 to arm the event currently assigned to the internal encoder; arg is 1 to arm the event currently assigned to the external encoder.

Status Word 1 contains bits (Bi, Bj, etc.) that indicate when a particular arming sequence is active and when the capture has taken place. For details, see Motor Index/Capture Directly-Addressed Status Bits on page 216. Also, see Status Word 1: Index Registration and Software Travel Limits on page 922.

After the capture has occurred, the corresponding rising or falling edge can be read using the commands I(arg) and J(arg), respectively. That allows the rising and falling edges to be recorded separately. Again, arg is 0 for the internal encoder's position at the time of the event; arg is 1 for the external encoder's position at the time of the event.



Rising and Falling Edge Index Capture

By default, the internal encoder will be triggered by the internal encoder's index mark. However, it can be reconfigured to use an external signal to trigger the internal encoder capture. Refer to the next commands.

- EIRE (Default) Use the internal encoder's index to capture that encoder's position. The I/O signal is used to capture the external encoder.
- EIRI Use a predefined I/O signal to capture the internal encoder. This displaces that I/O from being used to capture the external encoder. Class 5 D-style motors use I/O logical input 6 (pin 7 on the DA-15 connector); Class 5 M-style motors use I/O logical input 5 (pin 4 of the 12-pin I/O connector). Class 6 D-style motors use I/O logical input 5 (pin 6 of the HD26-pin connector).

DS2020 Combitronic System Index Capture

DS2020 with resolver motors: Reading position value and physical position

In resolver motors, the physical angular position always corresponds to the read position on a single turn. This is different from SmartMotors with incremental encoder — its "absolute" position is unknown until the index impulse (i.e., the zero mark on the encoder) is found. This means that a DS2020 Combitronic system with resolver motor behaves in this manner:

- Assume feed FD=65536;
- At the startup, a position value in the range [-32768 32767] is always returned by RPA, and this corresponds to the physical angle of the motor shaft, read by resolver;

- For example, RPA 16384 (that is $\frac{1}{4}$ of feed) means that shaft angle is 90° with respect to the zero position; this is always true, even if the system is restarted with a different initial position of the shaft;
- If many complete turns have been done (position is outside the range [-32768 32767]), when the system is restarted, the read position will be reported in the range [-32768 32767] and its value corresponds to the angular position of the shaft.

DS2020 with resolver motors: Index Capture function

The index capture procedure uses the commands $A_i(0)$, $R_{Bi}(0)$ and $R_I(0)$. When the procedure is started with $A_i(0)$, the closest position that corresponds to a zero-angle of motor shaft is captured. $R_{Bi}(0)$ initially returns 0 (meaning the procedure has not completed). After the zero-angle position is reached, it is sampled: $R_{Bi}(0)$ returns 1 and $R_I(0)$ returns the corresponding value. $R_{Bi}(0)$ remains 1 until a new $A_i(0)$ command is issued (this resets $R_{Bi}(0)$ to 0 and restarts the procedure). Here are some examples:

1. Assume $FD=8000$, initial position RPA 1000, $A_i(0)$ command issued and motor moved in the negative direction; when zero-angle is reached, $R_{Bi}(0)$ returns 1 and $R_I(0)$ returns 0.
2. Assume $FD=8000$, initial position RPA 1000, $A_i(0)$ command issued and motor moved in the positive direction; when zero-angle is reached, $R_{Bi}(0)$ returns 1 and $R_I(0)$ returns 8000.
3. Assume $FD=8000$, initial position RPA 9000, $A_i(0)$ command issued and motor moved in the positive direction; when zero-angle is reached, $R_{Bi}(0)$ returns 1 and $R_I(0)$ returns 16000.

Other Motion Commands

These commands are used to start, stop or decelerate motion, reset or shift the origin, and turn the motor servo off. For more details on these commands, see Part 2: SmartMotor Command Reference on page 247.

G

Go, Start Motion

The G command does more than just start motion. It can be used dynamically during motion to create elaborate profiles. Because the SmartMotor allows position, velocity and acceleration to change during motion, the G command can be used to replace the current move with a new one. All faults must be cleared before the G command will work, as indicated by the "drive ready" status bit. Faults can be cleared by correcting the fault situation and then issuing the ZS command.

S

Abruptly Stop Motion in Progress

If the S command is issued while a move is in progress, it causes an immediate and abrupt stop with all the force the motor has to offer. After the stop, assuming there is no position error, the motor will still be servoing. The S command works in all modes.

X

Decelerate to Stop

If the X command is issued while a move is in progress, it causes the motor to decelerate to a stop at the last entered deceleration value according to the ADT, DT and AT commands. When the motor comes to rest, it will servo in place until commanded to move again. The X command works in Position, Velocity and Torque modes. It also applies to Follow and Cam modes.

O=formula***Set/Reset Origin to Any Position***

The O= command (using the letter O, not the number zero) allows the host or program to declare the current position to a specific value, positive or negative, or 0 in the range -2147483648 to +2147483647. This command sets the commanded trajectory position to the value specified at that point in time and the actual position is adjusted similarly. The O= command directly changes the motor's position register and can be used as a tool to avoid ± 31 -bit rollover Position mode problems. If the SmartMotor runs in one direction for a very long time, it will reach position -2147483648 or +2147483647, which causes the position counter to change sign. While that is not an issue with Velocity mode, it can create problems in absolute position moves or create confusing results when reading position.

OSH=formula***Shift the Origin by Any Distance***

The OSH= command shifts the origin by the amount described, which may be from -2147483648 to +2147483647. This command is similar to O=, except that it specifies a relative shift. This can be useful in applications where the origin needs to be shifted during motion without losing any position counts.

OFF***Turn Motor Servo Off***

The OFF command turns off the motor's drive. When the drive is turned off, the PWR/SERVO status LEDs revert to flashing green. The motor will not freewheel by default in the OFF state because each SmartMotor has a safety feature that engages dynamic braking equivalent to the MTB command. This has the effect of causing a resistance to motion. To make a SmartMotor truly freewheel when off, issue BRKRLS and be sure any faults are cleared.

SCALEA(m,d), SCALEP(m,d), SCALEV(m,d)***Scale (Acceleration/Deceleration, Position, Velocity)***

The SCALE commands are used to scale the values of various acceleration/deceleration, position and velocity commands, and related reporting commands. This is done by issuing SCALEA, SCALEP and SCALEV, respectively. For example, SCALEA affects the ADT command; issuing SCALEA(10,1) applies a factor of 10x to any set, or 1/10x to any reported, acceleration/deceleration. Once set, the SCALE commands are in effect for all subsequent applicable commands until they are deactivated (all three commands are deactivated by default).

For more details, see SCALEA(m,d) on page 724, SCALEP(m,d) on page 726, and SCALEV(m,d) on page 728. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903. Also, see the Motor Scaling tool in the SMI software help.

Commutation Modes

Because the SmartMotor uses a brushless motor, it does not have the mechanical commutator that a brushed motor has to switch the current to the next optimal coil as the rotor swings around. To cause shaft rotation in a brushless motor, the control electronics have to see where the shaft is, and then decide which coils to deliver the current to next.

The most typical way to determine the orientation of the rotor is with small magnetic-sensing devices called Hall sensors. The process of shifting the current to the proper coils based on shaft rotation is called commutation. There are many methods for commutating a motor; the best commutation method depends on the application. As a general rule, sine mode commutation provides very smooth low torque ripple performance, and trapezoidal commutation provides the highest torque and fastest speeds.

These commands allow selection of different commutation modes. For more details, see Part 2: SmartMotor Command Reference on page 247.

NOTE: MDE, MDS and MDC require angle match (the first sighting of the encoder index) before they will take effect. This means the SmartMotor's factory calibration is valid and the index mark of the internal encoder has been seen since startup. Until then, the SmartMotor will operate in default MDT.

MDT

Mode Drive Trapezoidal

Trapezoidal commutation uses only the Hall sensors (default). It is the most simple commutation method, and it is always ready on boot up. MDT is effective despite the minor inaccuracies typically found in the mechanical placement of the sensors.

NOTE: M-style motors boot up in MDC mode (see MDC on page 135).

MDE

Mode Drive Enhanced

This driving method is exactly the same as basic trapezoidal commutation using Hall sensors, except that it also uses the internal encoder to add accuracy to the commutation trigger points. This idealized trapezoidal commutation mode offers the greatest motor torque and speed, but it can exhibit minor ticking sounds at low rates because the current shifts abruptly from one coil to the next. Because MDE uses the encoder, it requires angle match (the first sighting of the encoder index) before it will engage.

MDS

Mode Drive Sine

This is sinusoidal (sine) commutation, voltage mode. It provides smoother commutation compared to trapezoidal modes by shifting current gradually from one coil to the next. Because MDS uses the encoder, for motors with incremental encoders, it requires angle match (the first sighting of the encoder index) before it will engage. MDS is not as efficient as a trap commutation mode and has less torque available, especially at higher speeds (for more details, see MDS on page 574). However, for applications that require extremely smooth and quiet low-speed operation, MDS is the best choice.

MDC

Mode Drive Current

Available only for M-style Class 5 SmartMotors, this sinusoidal (sine) commutation method, augmented with digital current control, offers the best possible performance without sacrificing quiet operation.

Status Word 6 contains bits that indicate what commutation mode is currently active. Note that a command for a mode may not take effect until the angle match is indicated by bit 8 in status word 6. The angle match may not take effect until the motor rotates past the index mark of the internal encoder. Test for this using these status bits.

Status Word 6:

- Bit 0 Trap-Hall mode
- Bit 1 Trap-Encoder (enhanced) mode
- Bit 2 Sine Voltage mode
- Bit 3 Sine Current (vector) mode

MDB

Trajectory Overshoot Braking (TOB) Option

This command should be used after entering MDT or MDE to enable TOB action. This option reverts to off when one of the previous commutation choices is made. This option is off by default. Status Word 6, Bit 9 indicates if this mode is active.

MINV(0), MINV(1)

Invert Motion Direction

The MINV(1) command inverts the direction convention of the SmartMotor.

The MINV(0) command restores the default.

Modes of Operation

SmartMotors can be operated in several different modes. You can switch to and from almost any mode freely at any time. The next sections provide details on each operation mode.

NOTE: For details on any command, see Part 2: SmartMotor Command Reference on page 247.

Torque Mode

NOTE: Torque mode is an immediate response mode.

In Torque mode, also referred to as Mode Torque (MT), the SmartMotor shaft applies a torque independent of position. The internal encoder tracking still takes place, and can be read by a host or in a program. However, the value is ignored for motion because the PID loop is inactive. A torque-mode move does not mean the motor applies a constant torque regardless of speed; rather, the motor is powered at a fixed duty cycle of PWM to the motor windings in a manner similar to increasing and decreasing voltage to a traditional DC motor. To specify the value of the torque move, use the T= command with a number between -32767 and 32767. Remember that:

- Positive numbers apply a clockwise torque
- Negative numbers apply a counter-clockwise torque
- The default value for T is zero
- Speed is proportional to counter-torque or load on the shaft when in torque mode
- The larger the load, the slower the motor turns for a given torque value

The next list details the minimum requirements for a move to occur in Torque mode:

- Initiate the mode with the MT command
- Issue G

Torque Mode Example

The next example shows a basic torque move. Note that T is set before MT and G, which provides a known commanded torque before issuing an MT or G command.

```
T=2000      ' set torque to 2000
MT          ' set motor to Torque mode
G          ' start moving, open loop
```

Dynamically Change from Velocity Mode to Torque Mode

The next example dynamically changes from Velocity mode to Torque mode through torque transfer (TRQ command). In the example, about two seconds after going into Velocity mode, the motor is switched to Torque mode. Then, two seconds later, the motor is turned off.


```

MV           ' set motor to Velocity mode
VT=100000   ' set velocity to 100000
ADT=1000    ' set accel/decel to 1000
G           ' Go (Start moving)
WAIT=2000   ' wait about 2 seconds
T=TRQ       ' set torque to the value the PID filter was commanding in MV
MT G        ' set motor to Torque mode
WAIT=2000   ' wait about 2 seconds
OFF         ' turn the motor off

```

Velocity Mode

Velocity mode allows the SmartMotor to run at a constant commanded speed. SmartMotors close the speed loop on position, not encoder counts per unit time. As a result, moving to and from Position mode to Velocity mode is simple.

The next list details the minimum requirements for a move to occur in Velocity mode:

- | | | |
|---------------------------------|------------|---------------------------------|
| • Initiate Velocity mode | MV command | if not already in Velocity mode |
| • Nonzero value of Velocity | VT=### | set velocity equal to ### |
| • Nonzero value of Acceleration | ADT=### | set accel/decel equal to ### |
| • Go command to initiate move | G | start move immediately |

Constant Velocity Example

The next example shows a basic constant-velocity move. In the example, the motor starts moving when the G command is issued. It accelerates up to a velocity of 100000 at a rate of 1000 samples/sec/sec. It then remains at that speed until told to do otherwise.

```

MV           ' set motor to Velocity mode
VT=100000   ' set velocity to 100000
ADT=1000    ' set accel/decel to 1000
G           ' Go (Start moving)

```

Change Commanded Speed and Acceleration

In this example, the command speed and acceleration are changed while the program is in progress. The motor's move parameters are changed about two seconds after the initial commanded move begins.

```

O=0         ' set current position to zero
MV          ' set motor to Velocity mode
VT=100000   ' set velocity to 100000
ADT=1000    ' set accel/decel to 1000
G           ' Go (Start moving)
WAIT=2000   ' wait 2 seconds
VT=800000   ' set new velocity of 800000
ADT=500     ' set new accel/decel of 500
G           ' initiate change in speed and acceleration

```

Absolute (Position) Mode

Absolute (Position) mode is the default power-up mode of operation for the SmartMotor. In Position mode, the SmartMotor operates on absolute position commands, which use encoder counts.

The next list details the minimum requirements for a move to occur in Position mode:

- | | | |
|---------------------------------|------------|---------------------------------|
| • Initiate Position mode | MP command | if not already in Position mode |
| • Nonzero value of Velocity | VT=### | set velocity equal to ### |
| • Nonzero value of Acceleration | ADT=### | set accel/decel equal to ### |
| • Absolute commanded position | PT=### | set target position to ### |
| • Go command to initiate move | G | start move immediately |

NOTE: Commanded position must be different than present position to cause a move. If acceleration or velocity are at zero, the motor will not move.

Absolute Move Example

In this example, the motor starts moving when the G (Go) command is received and stops at an absolute position of 20000 encoder counts.

```
MP          ' set to position mode (required if currently in another mode)
VT=100000  ' set velocity to 100000
ADT=1000   ' set accel/decel to 1000
PT=20000   ' set commanded absolute position to 20000
G          ' Go (Start moving)
```

Two Moves with Delay Example

The next example shows two position moves with a delay in between.

```
O=0        ' set current position to zero
MP         ' set to position mode (required if currently in another mode)
VT=100000 ' set velocity to 100000
ADT=1000  ' set accel/decel to 1000
PT=20000  ' set commanded absolute position to 20000
G         ' Go (Start moving)
TWAIT     ' wait here until the motor has reached 20000
WAIT=1000 ' wait 1 second
PT=-500   ' Set commanded position of -500
G         ' start moving to new commanded position.
```

NOTE: The move is made at the previously-commanded speed and acceleration.

Change Speed and Acceleration Example

In this example, the commanded speed and acceleration are changed while the motor is executing the absolute position move.

```

O=0           ' set current position to zero
MP           ' set to position mode (required if currently in another mode)
VT=100000    ' set velocity to 100000
ADT=1000     ' set accel/decel to 1000
PT=1000000   ' set commanded absolute position to 1000000
G           ' Go (Start moving)
WAIT=8000    ' wait about 8 seconds
VT=800000    ' set new velocity of 800000
ADT=500      ' set new accel/decel of 500
G           ' initiate change in speed and acceleration

```

Shift Point of Origin Example

The next example demonstrates how to change (shift) the point of origin between moves. This is accomplished through the OSH command. The Origin command O={value} may also be used and can be set to any absolute number.

```

O=0           ' set current position to zero
MP           ' set to position mode (required if currently in another mode)
VT=100000    ' set velocity to 100000
ADT=1000     ' set accel/decel to 1000
PT=2000      ' set commanded absolute position to 2000
G           ' Go (Start moving)
TWAIT       ' wait until move is complete
OSH=-2000    ' shift current position back 2000 counts
WAIT=8000    ' wait 8 seconds
PT=2000      ' set commanded absolute position to 2000
G           ' Go (Start moving)
TWAIT       ' wait until move is complete

```

NOTE: The motor moved a total of 4000 counts, but its current position is only 2000 because it was reset to zero between moves.

Relative Position Mode

In Relative Position mode the SmartMotor moves relative to its current position by the use of the PRT (Position Relative Target) command.

The next list details the minimum requirements for a move to occur in Relative mode:

- | | | |
|---------------------------------|------------|---------------------------------|
| • Initiate Position mode | MP command | if not already in Position mode |
| • Nonzero value of Velocity | VT=### | set velocity equal to ### |
| • Nonzero value of Acceleration | ADT=### | set accel/decel equal to ### |
| • Relative commanded position | PRT=### | set relative position to ### |
| • Go command to initiate move | G | start move immediately |

Relative Mode Example

The next example illustrates the use of Relative mode. The example moves the motor through three 2000-count moves or a total of 6000 counts.

```

MP          ' set to position mode (required if currently in another mode)
VT=100000  ' set velocity to 100000
ADT=1000   ' set accel/decel to 1000
PRT=2000   ' set commanded relative position move to 2000
G          ' Go (Start moving 2000 counts)
TWAIT     ' wait until move is complete
G         ' Go (move 2000 counts again)
TWAIT     ' wait until move is complete
G         ' Go (One more time)
    
```

Follow Mode with Ratio (Electronic Gearing)

Follow Mode with Ratio (MFR) allows a motor to follow a standard TTL quadrature external encoder input signal, or internal clock, at a user-defined ratio.

By default, Follow mode runs continuously at a ratio of 1:1 in terms of input counts to distance moved.

The user can freely select either the external encoder or fixed rate internal clock as the input source. The fixed rate internal clock runs at 8000 counts per second by default, but can be influenced by the PID commands. The SRC command defines whether to follow the internal counter or external encoder.

NOTE: Changed MFR values do not take effect until after the next G command.

The next list details the minimum requirements for a move to occur in Follow mode:

- Set Incoming counts multiplier MFMUL=### may be negative or positive
- Set Incoming counts divisor MFDIV=### may be negative or positive
- Calculate above ratio and mode MFR
- Go command to initiate the move G start following the encoder

NOTE: If the external encoder is not moving, no motion will occur. Commanded position must be different than present position to cause a move.



CAUTION: Do not switch between gear modes while in operation. When a transition is made, the profile must be stopped or the motor must be turned off.

Electronic Gearing and Camming over CANopen

Beginning with firmware 5.x.4.30 and later, the SmartMotor provides precise time synchronization over CANopen between motors for electronic gearing and camming applications (for example, traverse and take-up spooling). The CANopen objects related to this are: 1005h, 1006h, 2207h, 2208h, 2209h, 220Ah-220Dh. For details on these objects refer to the *SmartMotor CANopen Guide*. For a sample user program, see CAN Bus - Time Sync Follow Encoder on page 883.

NOTE: This capability is currently available on Class 5 SmartMotors only.

Electronic Gearing Commands

The next sections describe related commands. For more details on these commands, see Part 2: SmartMotor Command Reference on page 247.

SRC(enc_src)**Select the input source used in Follow and Cam modes**

The SRC() command can allow the SmartMotor to use the many advanced following and camming functions even without an external encoder input. Values for enc_src:

- 0 Null (pauses controller)
- 1 External encoder (-1 inverts direction)
- 2 Time-base at PID rate (-2 inverts direction)

NOTE: SRC() can be updated while the motor is running. However, sudden speed changes may occur.

MFR

NOTE: When using the EOBK and MFR commands in the same program, there is interaction that must be considered in the code. For details, see the Programming Note in EOBK(IO) on page 445 or MFR on page 600.

Configure A & B inputs to Quadrature mode and select Follow mode

The Mode Follow Ratio (MFR) command configures the A and B inputs of the motor to be read as standard quadrature inputs and puts the SmartMotor in Follow mode.

For a figure showing use examples of this command, see MFSDC Modes on page 148.

MSR**Configure A & B inputs to Step/Direction mode and select Follow mode**

The Mode Step Ratio (MSR) command configures the A and B inputs of the motor to be read as standard Step and Direction inputs and puts the SmartMotor in Follow mode.

MFO**Reset external quadrature encoder****MSO****Exit Step/Direction Follow mode**

When using the ENC1 command, be careful that you do not inadvertently change the operation of the encoder with the MFO or MSO command. If you must use an alternate encoder source for Follow mode and at the same time use ENC1, then choose the version of the above commands to match your encoder type for the ENC1 command.

MFMUL=formula, MFDIV=formula**Set Follow mode ratio**

The internal mathematics work best by describing the Follow mode ratio in terms of a fraction of two integers. Choose MFMUL and MFDIV to create the Follow mode ratio if it is not 1:1 (the default). Any change to the ratio will be enabled by the G command.

```
MFMUL=formula      'Multiplier applied to Follow mode ratio
MFDIV=formula      'Divisor applied to Follow mode ratio
```

MFA(distance[,m/s])**Ascend ramp to sync. ratio from ratio of 0**

- distance Setting from 0 to 2147483647. Set to 0 to disable. By default, it is disabled.
- [,m/s] Is optional and specifies the meaning of distance. Values of [,m/s]: 0 for designating input units (controller units) and 1 for designating distance traveled (follower units).

For a figure showing use examples of this command, see MFSDC Modes on page 148.

MFD(distance[,m/s])**Descend ramp from ratio to ratio of 0**

- distance Setting from 0 to 2147483647. Set to 0 (default) to disable.
- [,m/s] Is optional and specifies the meaning of distance. Values of [,m/s]: 0 for designating input units (controller units) and 1 for designating distance traveled (follower units).

For a figure showing use examples of this command, see MFSDC Modes on page 148.

MFSLEW(distance[,m/s])**Slew at ratio for a fixed distance**

- distance Setting from -1 to 2147483647. Set to -1 (default) to disable. When disabled, Follow mode runs at ratio continuously.
- [,m/s] Is optional and specifies the meaning of distance. Values of [,m/s]: 0 for designating input units (controller units) and 1 for designating distance traveled (follower units).

For a figure showing use examples of this command, see MFSDC Modes on page 148.



CAUTION: The next gearing examples are relative to the motor shaft position at the time the G command is issued.

All distances are relative.

Follow Internal Clock Source Example

In this example, the motor follows the internal source at a 1:1 ratio and moves at a rate of 8000 counts per second. For a NEMA 23 frame motor, that results in 2 RPS or 120 RPM motion.

NOTE: The trajectory and drive status LEDs will be continuously green. This is because the motor is constantly calculating a trajectory from the gearing source signal.

- SRC(-2) Results in inverting controller signal direction and changing motor direction. Changing the sign of either MFMUL or MFDIV also inverts direction.
- SRC(-1) Inverts the external encoder signal; provides the easiest way to correct for

inverted encoder signal direction.

- SRC(0) Null (pauses controller counts).
- SRC(1) (Default) Sets motor to follow the external encoder.
- SRC(2) Use one count per PID cycle (default is 8.0 kHz).

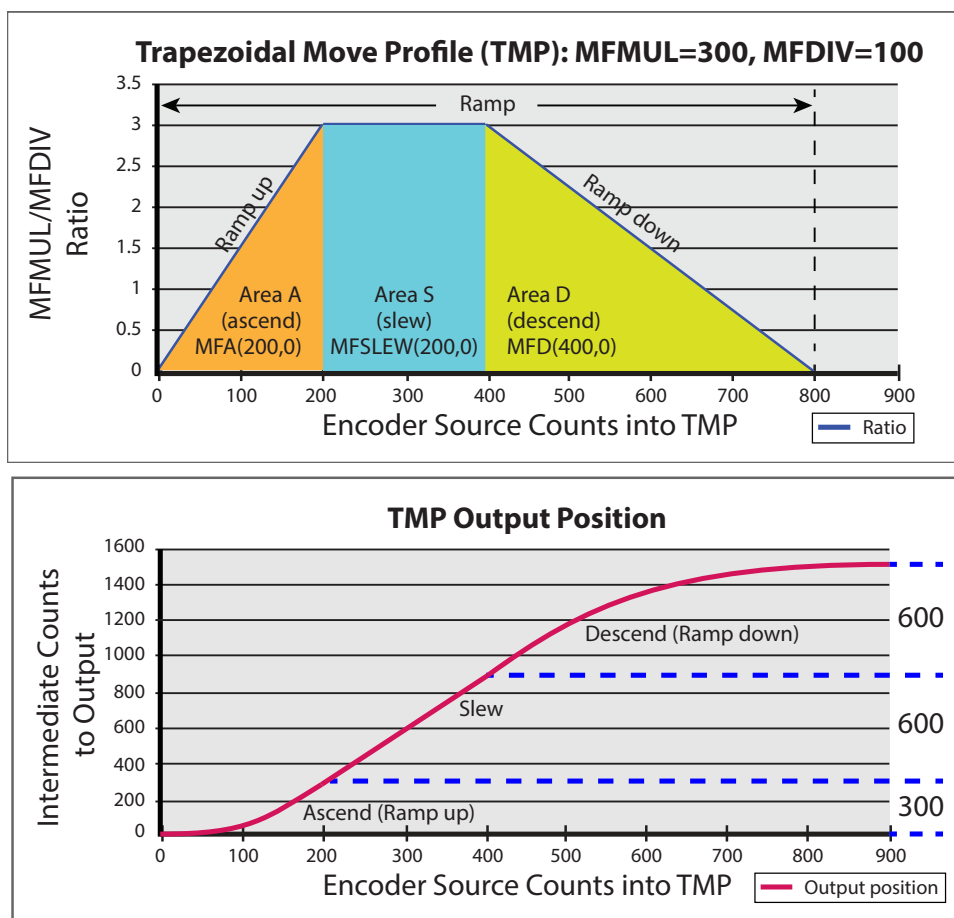


CAUTION: MFMUL must not be set excessively higher than MFDIV. Doing so will result in small changes in controller counts and large changes in follower-gearing counts. This may cause motor harmonic distortion or following errors.

```
SRC (2)           'Set signal source to internal 8K counts/sec
MFMUL=100        'Default is 1
MFDIV=100        'Default is 1
MFR              'Enable Follow mode at specified ratio
G
```

Follow Incoming Encoder Signal With Ramps Example

This example shows a profile driven by an incoming encoder signal. In addition to following the incoming encoder, the SmartMotor performs an acceleration (ascend or ramp up) into the following relationship (slew), and after a prescribed distance, performs a deceleration (descend or ramp down) back to rest. Refer to the next figures.



Trapezoidal Move Profile (TMP) and Output Position Diagrams

In the first graph, the 'controller' (encoder source counts into TMP) is along the horizontal axis of the graph, and the gear ratio (MFMUL/MFDIV) is along the vertical axis of the graph. This demonstrates that "area under the curve" is the 'follower' position.

The second graphs shows the follower position as a function of controller encoder source counts to intermediate counts (the TMP output). In this example, MFA, MFD and MFSLEW are commanded in controller units (source counts). These three commands can accept either controller (source counts) or follower units (intermediate counts) according to the second argument as a 0 or 1, respectively. The firmware automatically calculates the move accordingly. The next example uses the command(x,0) form to specify 'controller' or source counts.

```

MFMUL=300
MFDIV=100
MFA(200,0)      'Move 200 controller counts over ascend (area "A")
MFD(400,0)      'Move 400 controller counts over descend (area "D")
MFSLEW(200,0)   'Maintain sync ratio for 200 controller counts (area "S")
MFR             'Enable Follow mode at specified ratio
G
    
```

Each time a G (Go) is received, the motor follows the Trapezoidal Move Profile (TMP).

For labeling applications, it may be beneficial to feed out over a preset distance profile, also known as a "one-shot" gearing trapezoidal profile. This is where the MFSLEW() command comes in, as shown in the code and figures.

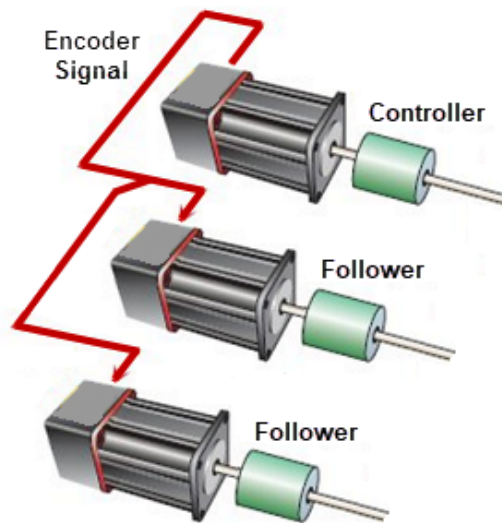
Each parameter distance is subject to follower counts, *not* controller counts.

The total distance traveled by the follower will be MFA distance + MFD distance + MFSLEW distance. In this case, 300+600+600 or 1500 total follower counts moved.

Electronic Line Shaft

NOTE: This section only applies to M-style motors.

For some applications, it is useful to create a controller/follower motor relationship known as an "electronic line shaft" (see the next figure). This setup is used for machines such as printing presses, where everything must run at proportional speeds to the main (controller) axis.



Electronic Line Shaft Diagram

On the M-style SmartMotors, there is a bidirectional Encoder Bus port. Using this port, along with Moog Animatics encoder bus cables, you can daisy chain a series of M-style motors:

- One motor will have ENCD(1) issued; this will be the controller.
- All other motors will have ENCD(0) (default) issued; these will be the follower devices.

ENCD(in_out)

Sets the Encoder Bus port as an input or an output — only applies to motors with Encoder Bus connectors, such as M-style SmartMotors.

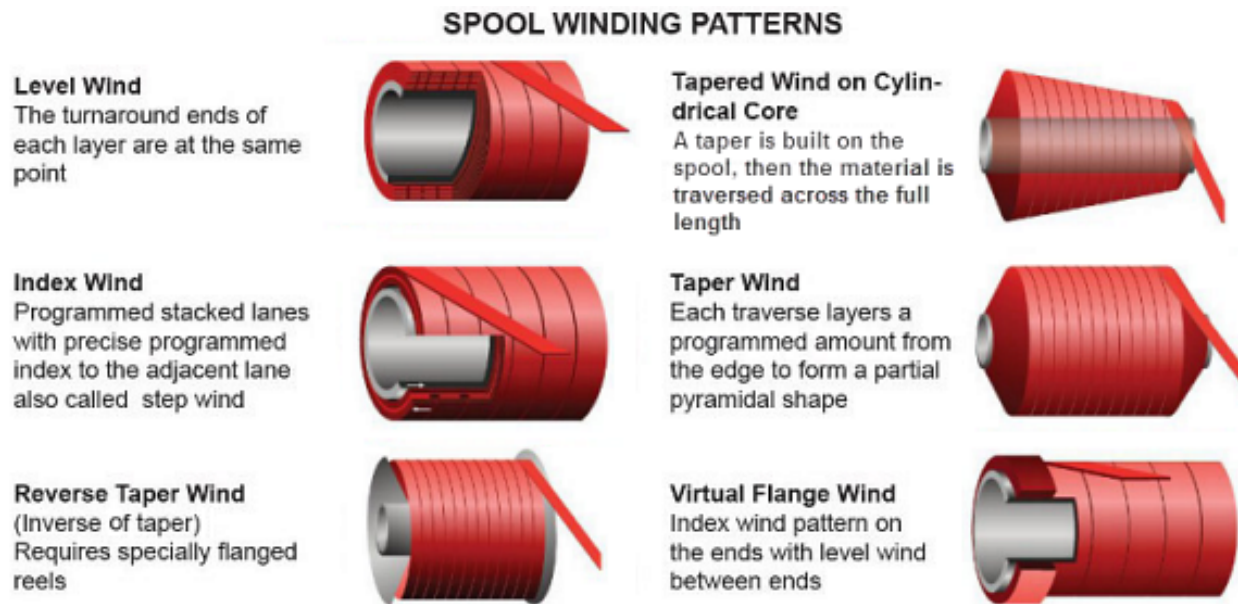
The ENCD() command allows the M-style SmartMotor to use the Encoder Bus port as either an input or an output. This allows the motor to operate as a controller (output) or follower (input) when daisy chained to other M-series motors through the Encoder Bus ports. The value for in_out can be hard-coded or a variable:

- 0 (Default) Encoder Bus port is an input.
- 1 Encoder Bus port is an output.

The ENCD() command can be used over the Combitronic network.

Spooling and Winding Overview

Spooling or winding provides a cost-effective way to package materials of very long length, such as thread, film, labels, cable and thermoplastics. The material is fed from a large spool at a certain rate onto another spool, with a traversing mechanism between the two spools to create the desired pattern or evenly wind onto an flanged spool or cylindrical core despite the core shape. The integrity of the spool is often based on precise patterns and proper tension control throughout the winding process. The next figure provides examples of common spool-winding patterns:

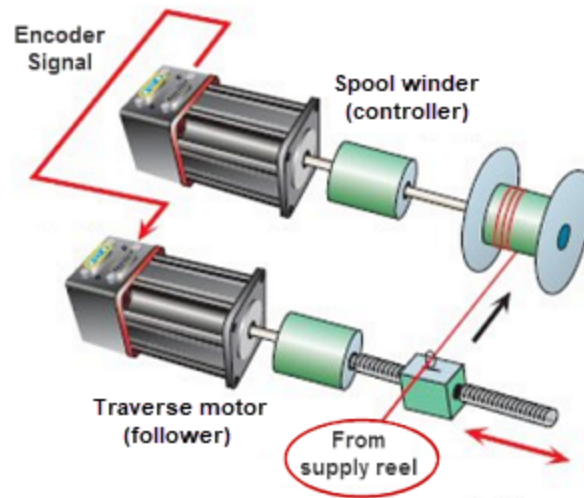


There are various problems associated with winding and spooling applications, such as: material tension control, setting proper dwell points, over/under-travel (which results in a "dog bone" shape) and tapered patterns with low-friction materials or wound onto cylindrical cores.

The next sections provide commands and example programs designed to help you handle the challenges of spooling/winding applications.

Relative Position, Auto-Traverse Spool Winding

The next figure provides a simple representation of an auto-traversing spool winding application.



Auto-Traversing Spool Winding Mechanism

The MFSDC command is used to initiate the dwell and reverse direction (traverse) needed to perform spooling/winding operations.

MFSDC(distance,mode)

Dwell at 0 ratio for input distance

- distance** Set from 0 to 2147483647 through a variable or hard-coded value to specify the number of controller counts the follower dwells at zero ratio. Set to -1 (default) to disable (see the first row of the next table). When disabled, Follow mode runs at ratio continuously.
- mode** Specifies the gearing profile application in firmware, as shown in the next table. These follow a predefined trapezoidal profile (see the next figure).

For example, a setting of 0 for mode is typical for feeding labels in label applications; a setting of 1 is typical for traverse-and-takeup spool winding applications.

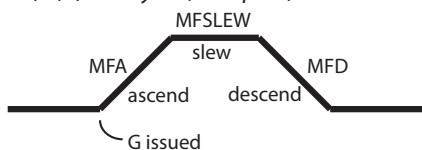
dist	mode	Motion	Repeat	Run state; Initiated by
-1	0	relative	one cycle, no repeat	once; a G or G(2) command
x	0	relative	repeat, one direction	continuous; after initial G or G(2)
x	1	relative	traverse back/forth	continuous; after initial G or G(2)
x	2	absolute	traverse back/forth	continuous; after initial G or G(2)



CAUTION: Any value other than -1 for the MFSDC distance command causes the motion profile to continuously dwell and repeat. Reissue the command with distance equal to -1 to stop the repetitive motion.

NOTE: The MFMUL and MFDIV commands do not have an effect on dwell time or distance. Dwell is strictly based on raw controller encoder counts selected by the SRC() command specifying internal virtual or external controller count source.

MFSDC(-1,0) one cycle (no repeat)



MFSDC(x,0) x=dwell distance; motor repeats in one direction

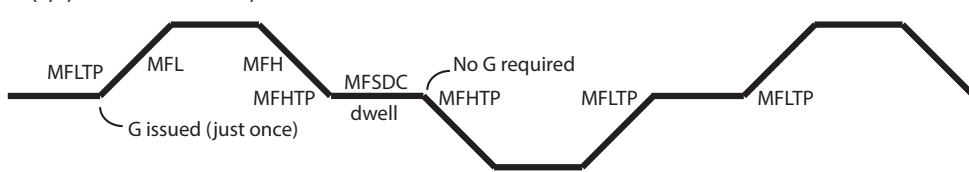


Can be used with Camming operation

MFSDC(x,1) x=dwell distance; motor traverses



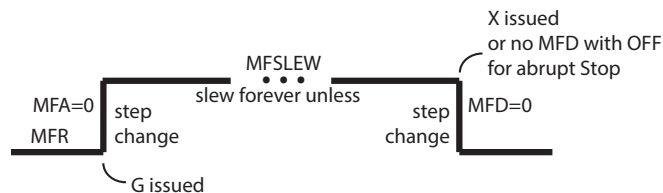
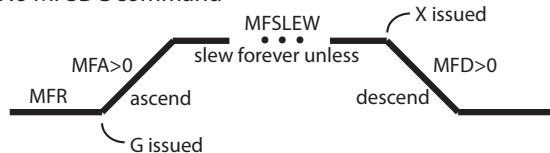
MFSDC(x,2) x=dwell distance; motor traverses between abs. values of MFHTP & MFLTP



DO NOT USE with Camming operation

NOTE: The examples above work when MFSLEW(x,y) has been set. By default, x is -1, which means "slew forever" and prevents the repeating cycles shown above. Therefore, to disable any of these modes and go back to forever-run at slew, set MFSLEW(-1,0); to get one of the cycles above, ensure that MFSLEW(x,y) where x >= 0.

No MFSDC command



MFSDC Modes

The next example demonstrates the use of the MFSDC command. It is a spool-winding program that performs a following profile across the spool, a dwell at the end for a specific span of input distance and then reverses the profile back to the original end of the spool for another dwell. The motion repeats until another MFSDC command is issued with Exp1 equal to -1 and then a G command, or an X or S command, is issued.

```

a=1000      'Ascend and descend distance in follower counts
b=200000   'Spool width in follower counts
c=4000     'One rev of spool in controller counts
s=b- (a*2)  'Calculate MFSLEW distance
m=1000     'Gear ratio multiplier
d=1000     'Gear ratio divisor
MFMUL=m    'Set ratios for gearing
MFDIV=d
MFA(a,1)   'Set ascend into ratio distance
MFD(a,1)   'Set descend out of ratio distance
MFSLEW(s,1) 'Set slew dis. between the accel and decel points
MFSDC(c,1) 'Set dwell for "c" counts, auto rev. after dwell
MFR        'Enable Follow mode at specified ratio
G          'Start following the external controller encoder
    
```

NOTE: The G command assumes the cycle starts at the same end of the spool each time.

A shift back and forth to the oscillation can be achieved by running in dual-trajectory mode. For more information on dual-trajectory mode, see Dual Trajectories on page 175.

```

SRC(2)      'Set signal source to internal 8K counts/sec
MFMUL=100   'Default is 1
MFDIV=100   'Default is 1
MFA(500,1)  'Set ascend ratio distance of 500 follower counts
MFD(500,1)  'Set descend ratio distance of 500 follower counts
MFR(2)      'Enable Follow mode for SECOND TRAJECTORY at specified ratio
MFSLEW(8000,1) 'Stay at slew ratio for 8000 counts of the follower
MFSDC(100,1) 'Dwell for 100 counts, auto repeat in reverse direction
G(2)        'Begin to follow controller signal in SECOND trajectory
MP(1)       'Set FIRST TRAJECTORY mode to Position mode
VT=100000   'Set velocity to run over top of gearing
ADT=100     'Set accel/decel to run over gearing
PRT=1000    'Set relative move
G(1)        'Shift all motion 1000 counts in positive direction
    
```



CAUTION: In the above example, repeating G(1) continuously shifts oscillation in the positive direction by 1000 counts or the value in PRT.

NOTE: A velocity MV(1) or position MP(1) mode may be used over gearing. All distances are relative to gearing. The command X(2) stops gearing; the command X(1) stops position or velocity moves.

Dedicated, Absolute Position, Winding Traverse Commands

This section applies to absolute positioning in electronic gearing. It is specifically tailored to traverse and take-up winders.

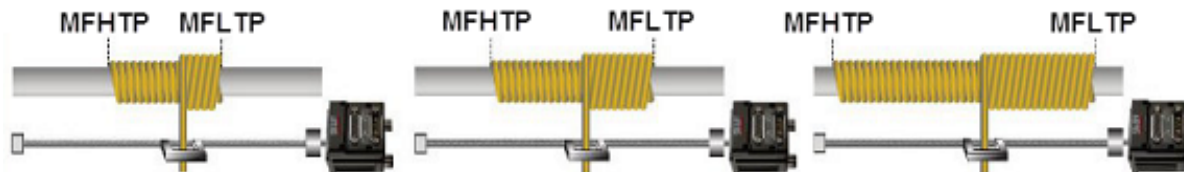
Refer to the next figure. The MFMUL and MFDIV commands, which were previously described in this chapter, are used to set the ratio of controller to follower motion as a maximum when slew is reached. The sign of MFMUL/MFDIV is ignored in this mode of operation — only the absolute value is used. The initial direction of motion is not affected by the sign of MFMUL/MFDIV.

NOTE: If MFMUL=0, then the traverse process will end when the next endpoint is reached.



Refer to the next figure. The MFSDC command enables the absolute traverse mode of operation. The MFLTP and MFHTP commands are then used to set the low and high traverse points, respectively, which control the spool width and position.

NOTE: The value of MFHTP must be greater than or equal to the value of MFLTP.



The next sections describe related commands. For more details on these commands, see Part 2: SmartMotor Command Reference on page 247. They are not intended to be combined with Cam mode.

MFSDC(distance,2)

Absolute traverse mode for input distance

- distance Length of the dwell at both ends of the move in controller units. Use -1 to disable; range of distance is 0 to 2147483647.
- 2 Activates the absolute traverse mode of operation.

For additional details and figures, see MFSDC(distance,mode) on page 147.

MFLTP=formula

Mode follow lower traverse point

- formula Specifies the lower traverse point. Range is any 32-bit signed value. It must be less than or equal to MFHTP; MFHTP-MFLTP must be less than 2^{31} .

For a figure showing use examples of this command, see MFSDC Modes on page 148.

MFHTP=formula

Mode follow higher traverse point

- formula Specifies the higher traverse point. Range is any 32-bit signed value. It must be greater than or equal to MFHTP; MFHTP-MFLTP must be less than 2^{31} .

For a figure showing use examples of this command, see MFSDC Modes on page 148.

MFCTP(arg1,arg2)

Sets control information for traverse mode

- arg1 Sets initial direction motor will move upon receiving G.
arg1=-1: (When G is issued) Traverse toward most recent direction when previous traverse move ended. This is most likely required in all winders.

- This direction is indicated by Status Word 7, Bit 13.
- This state is not reset by an X or an OFF.

arg1=0: (Power-up default value) Initially traverse toward higher bound when G is issued.

arg1=1: Initially traverse toward lower bound when G is issued.

arg2 Special bits:

arg2=1: The RPC(2) frame of reference is updated with shaft motion when the servo is off (OFF, MTB, MT). This is a special setting to ensure backward compatibility with existing applications that may use the RPC(2) frame of reference.

arg2=0: (Power-up default value) The RPC(2) frame of reference is frozen when the servo is off (through OFF, MTB, MT).

MFL(distance[,m/s])

Ramp at the lower end of traverse; designate controller or follower

distance Specifies the ramp distance at the lower end of the traverse. Distance range:
0 to 2147483647

[,m/s] 0=controller, 1=follower; distance range: 0 to 2147483647

For a figure showing use examples of this command, see MFSDC Modes on page 148.

MFH(distance[,m/s])

Ramp at the higher end of traverse; designate controller or follower

distance Specifies the ramp distance at the higher end of the traverse. Distance range:
0 to 2147483647

[,m/s] 0=controller, 1=follower; distance range: 0 to 2147483647

For a figure showing use examples of this command, see MFSDC Modes on page 148.

ECS(counts)

Encoder count shift — immediately compensates for variation in material width

counts Specifies the counts to be added to (or subtracted from) incoming controller counts as if they had an immediate change in value. For example, if the external encoder count is 4000, and ECS(1234) is issued, the count would immediately shift to 5234. Note that when the MFMUL:MFDIV ratio is other than 1:1, the ratio is multiplied by the ECS value.

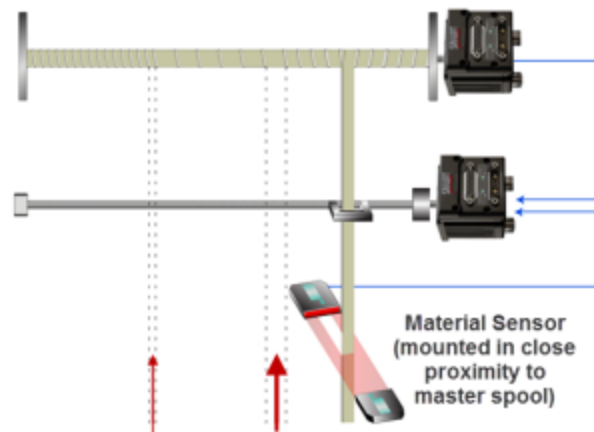


CAUTION: Large values may cause jerks in motion or following errors.

The ECS command is dynamic and immediate (not buffered), and it does not require a G command. Further, it works on top of *any* gearing or camming mode.

Proper use of the ECS command will allow full packing of material onto a spool regardless of variance in material width.

1. A sensor reads the material width.
2. Through programming, the user will scale value to encoder counts.
3. Based on encoder counts, the traversing SmartMotor will have ECS(encoder counts) issued, which results in a change to incoming controller counts by that value.
4. The SmartMotor adjusts its gearing.



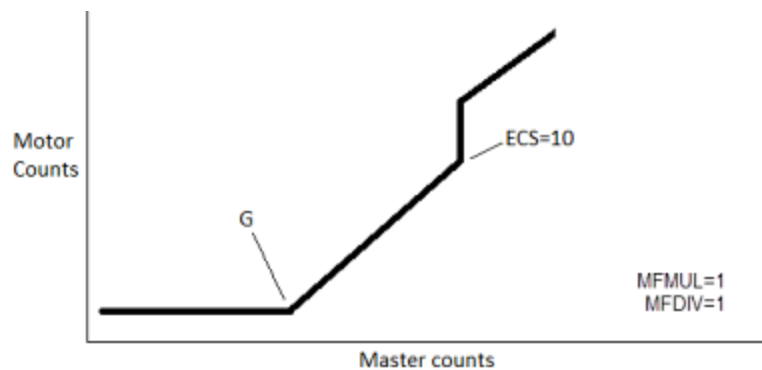
The ECS command is for tiny continuous corrections, where changing MFMUL or MFDIV is not desired because the basic ratio needs to remain fixed, but changes in demand for correction may need to be adjusted over time. For example, assuming you have a program with normal gearing:

```
MFMUL=1      'Ratio (default is 1)
MFDIV=1      'Ratio (default is 1)
MFR          'Enable Follow mode at specified ratio
G            'Begin move.
```

In this case, the motor begins spinning at a 1:1 ratio of external encoder input. Then you issue:

```
ECS (10)      'Encoder count shift of 10 counts
```

The motor will lunge forward by 10 encoder counts, as shown in the next figure.



Change Caused by ECS Command

However, if MFMUL=100 and MFDIV=1 (100:1 ratio), the motor would lunge forward by 1000 counts because the ratio is multiplied by the ECS value. In this case, if EL (Error Limit) was set to 1000 or less, the change from ECS would cause an instantaneous following error. It could also cause peak overcurrent errors.

Single Trajectory Example Program

The next example shows a single-trajectory traverse winding application. It uses the commands for high/low ramps and traverse points, which were discussed previously. For details, see Dedicated,

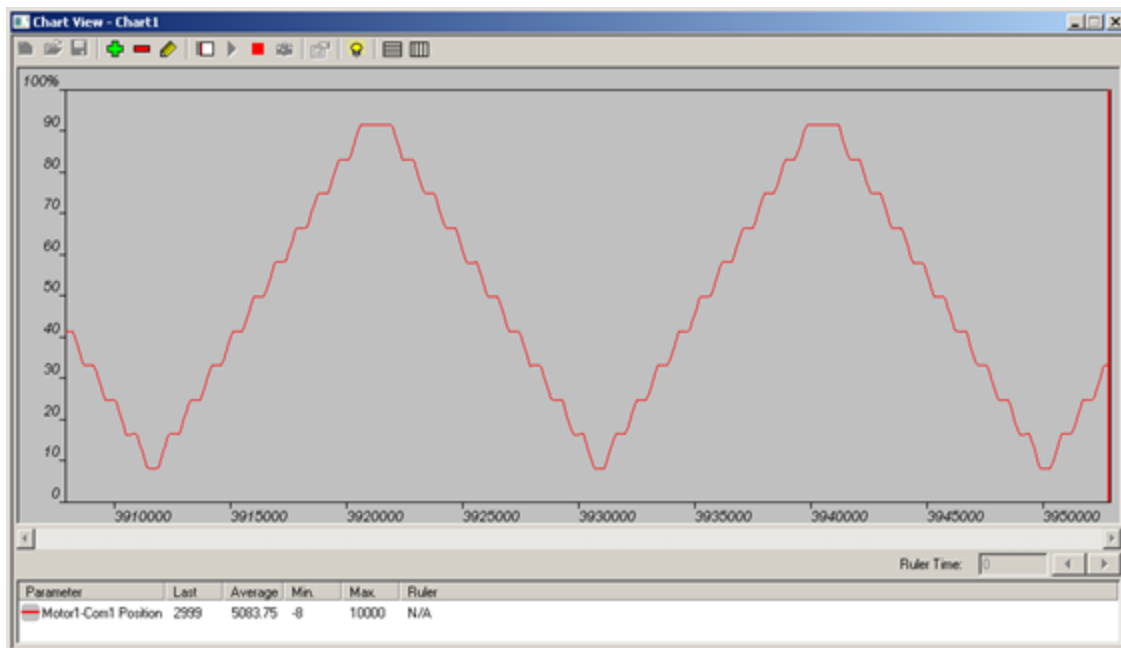
Absolute Position, Winding Traverse Commands on page 149.

```
' *** User does some type of homing before this. ***
SRC(2)      '*** For Demo controller signal ***
'Typical applications would use SRC(1) for encoder input.
MFCTP(0,1)  'Start traverse state in "normal" direction
            'Activate update of RCP(2) when servo is off
MFL(1000,1) 'Lower-end ramp
MFH(1000,1) 'Higher-end ramp
MFLTP=-1000 'Lower traverse point
MFHTP=1000  'Higher traverse point
MFMUL=1     'Ratio (default is 1)
MFDIV=1     'Ratio (default is 1)
MFSDC(4000,2) 'Dwell for 4000 counts, 2 is active traverse mode
MFR         'Enable Follow mode at specified ratio
G          'Begin move
```

Chevron Wrap Example

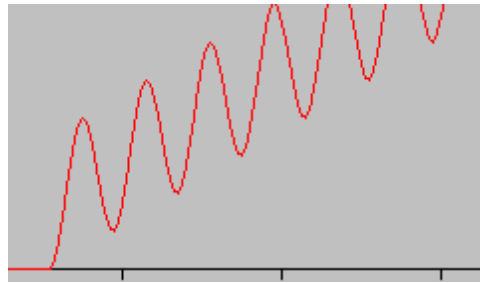
This example uses a more complex winding method, where camming (high-frequency oscillation) occurs on top of gearing (low-frequency traverse), to create a custom "chevron" wrap. For electronic camming details, see Cam Mode (Electronic Camming) on page 156.

The frequency plot for this winding method is shown in the next figure.



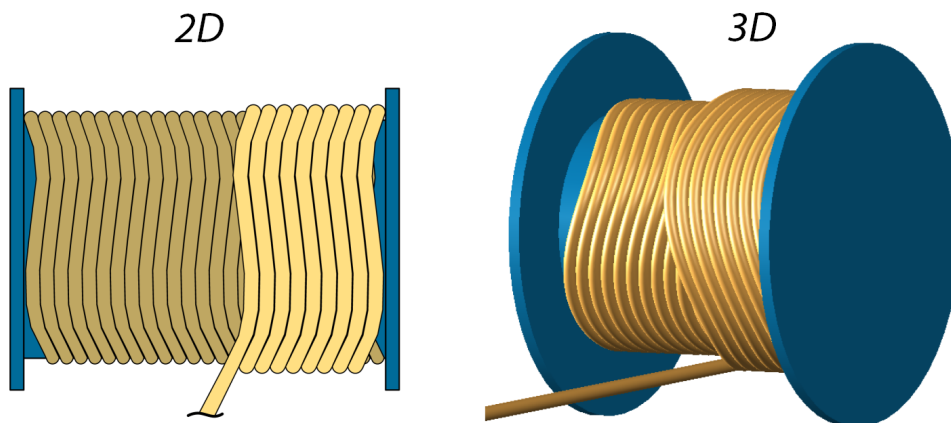
"Chevron" Winding Frequency Plot

Note that camming can be linear, or cubic spline, which permits a smooth, high-frequency oscillation on top of the low-frequency traverse. Refer to the next figure.



Smooth High-Frequency Oscillation

The overlapping "chevron" wraps are advantageous because they prevent the material from becoming trapped in the windings of the underlying layer, which can cause it to be pinched/kinked or break during removal. Either of those conditions would cause the spool to be defective. Refer to the next figures.



To create this solution, the SmartMotor requires only four parameters:

```
'System parameters:
c=8000      'Controller (External) Encoder resolution
            '(counts per 360 deg turn of spool)
w=10000    'Spool width distance in encoder counts of traversing follower motor

'Chevron shape parameters:
n=1000     'Follower counts per full (360 deg) turn of controller spool (pitch)
nn=1000    'Follower counts per half (180 deg) turn of controller spool
            '(amplitude of chevron)
```

The complete code example is available in Chevron Traverse & Takeup on page 877. For more information on electronic camming, see Cam Mode (Electronic Camming) on page 156. Also, see the Fixed Segment Cam Simulator (available on the Moog Animatics website at <https://www.animatics.com/support/downloads.knowledgebase.html>), which is a gearing/camming training aid.

Other Traverse Mode Notes

These are other notes related to operation in Traverse mode.

- MFA() is not used at this time in Traverse mode.
- MFD() should only be used for a stop from the X command while in Traverse mode.
- The traverse points are in the context of move generator 2: RPC(2). If multiple trajectories are commanded after the start of the move, then they are not specifically in the context of the motor shaft's actual position (RPA).
- RPC is automatically copied to RPC(2) at the start of the Traverse move only during single-trajectory moves. This is done to accommodate the typical use of a single trajectory where the user is homing the machine based on shaft position.
- If multiple trajectories are active, then the user is responsible for setting RPC(2) with the O(2)= command or shifting with the OSH(2)= command to establish the desired frame of reference.



CAUTION: Do not issue a change to RPC(2) during a traverse move using either the O(2)= or OSH(2)= command. This will produce unpredictable behavior that is undefined at this time. Those commands should be issued before issuing a G or G (2) when in dual-trajectory mode.

- MFMUL and MFDIV determine the ratio of controller to follower motion as a maximum when a slew is reached.
 - The sign of MFMUL/MFDIV is irrelevant in this mode of operation. Only the absolute value is used.
 - The initial direction of motion is not affected by the sign of MFMUL/MFDIV.
 - If MFMUL=0, then the traverse process ends when the next endpoint is reached.
- Traverse mode of operation is initiated using the G command. G may be issued from drive off or other modes of operation.



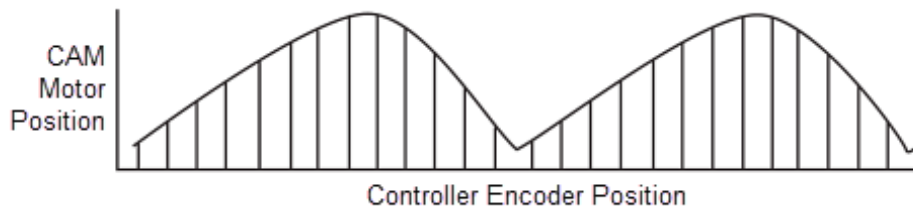
CAUTION: Do not repeat the G command while in traverse mode. Doing so will produce unpredictable behavior that is undefined at this time.

Traverse Mode Status Bits

Status		
Word	Bit	Description
7	8	Trajectory is in progress. Indicates that trajectory is being generated and continuous motion is in progress, even if in a dwell state. Bit is cleared when X, S, fault, OFF or MFMUL=0 ends the trajectory.
7	9	Ascend: In Traverse mode, this indicates the lower traverse point ramp is in progress.
7	10	Slew: in Traverse mode, this indicates the slew segment is in progress.
7	11	Descend: In Traverse mode, this indicates the higher traverse point ramp is in progress.
7	12	Dwell: In Traverse mode, this indicates the higher dwell state in progress.
7	13	State: The most recent Traverse mode state. =0 Motion profile is set to/moving in the forward direction. This is the power-up default state. =1 Motion profile is set to/moving in the reverse direction. The use of status word 7, bit 13 is undefined in other modes of operation with trajectory 2.
7	14	Dwell: In Traverse mode, this indicates the lower dwell state in progress.

Cam Mode (Electronic Camming)

Electronic camming is similar to mechanical cams — for a given controller rotating device, a follower device tracks the speed and moves through a fixed profile of positions. In electronic camming, the profile is a look-up table of data stored in the follower motor.



Example Cam Profile

The SmartMotor supports motion profiles based on data stored in a Cam table. The Cam table can reside in EEPROM memory or in the user array.

NOTE: Cam tables can be written to EEPROM memory, which retains its contents when power is removed; or to the variable data space, which provides more flexibility but is cleared when power is removed.

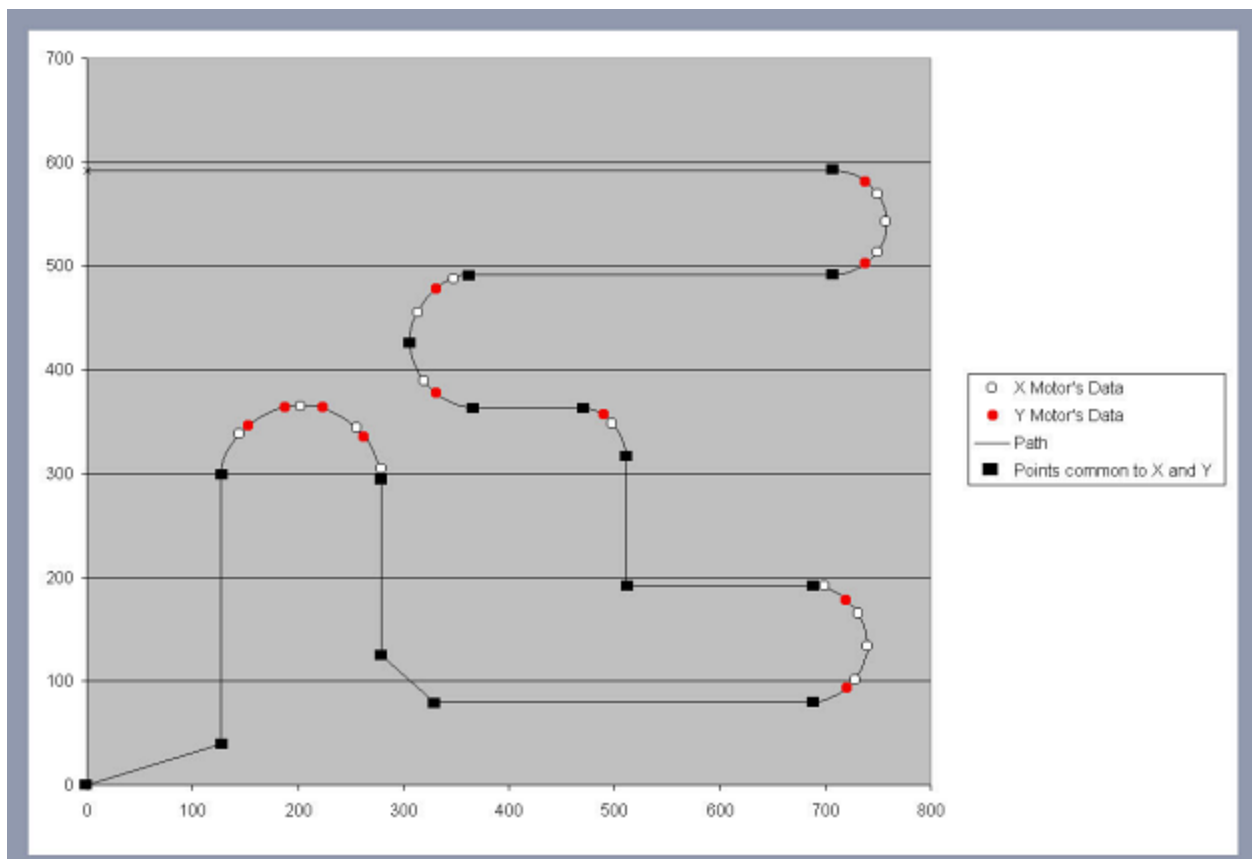
These are the available storage locations:

- RAM storage: 1 Cam table
 - 52 fixed length data points, 35 variable length data points
- Flash storage: 9 Cam tables
 - 750 fixed length data points, 500 variable length data points
- EEPROM: Up to 8000 points total may be stored and moved to flash or RAM

Cam table data may be directly imported from a tab delimited text file or spreadsheet.

- Data imported to the SMI software can be written into a program, copied to the clipboard or written directly (live) into a motor
- Import function allows for optimizing data points for cubic spline interpolation

The motor position is interpolated between each data point. This interpolation can be specified as linear, spline that is not periodic and spline that is periodic. Spline mode allows motion paths to be created from a reduced number of points. For example, the next figure shows an X-Y plot of Cam tables running on two motors. While the original data contained over 700 data points, Spline mode reduced the data set to approximately 30 points in each motor.



Example of Spline Mode Points and Motion Path

Cam mode has the ability to apply sophisticated shaping and selection of the encoder input source using Follow mode. Cam mode uses MFMUL and MFDIV to set the follow ratio for incoming controller

counts. Through the use of the SRC command, either the external encoder or a fixed-rate "virtual encoder" can be used as the input source to the cam. This fixed-rate (virtual) encoder also works through Follow mode, so the actual rate into the cam can be set. The speed of the virtual encoder is equivalent to the PID rate. In other words, for a Class 5 motor at its default PID rate of 8000 Hz, the virtual encoder sees 8000 encoder counts per second.

One example of this is a complex "chevron" pattern winding application where camming (high-frequency oscillation) occurs on top of gearing (low-frequency traverse). This is used to spool material in a way that prevents it from getting pinched or trapped in the underlying layers. For more details, see Chevron Wrap Example on page 153.

Electronic Camming Details

For a brief description of the Cam mode (electronic camming) commands in this section, see Electronic Camming Commands on page 165. Follow mode (electronic gearing) commands were previously discussed in Electronic Gearing Commands on page 140. Also, refer to the detailed command descriptions in Part 2: SmartMotor Command Reference on page 247.

Understanding the Inputs

There are two modes of operation (and associated commands) that involve the Trapezoidal Move Profile (TMP):

- MFR (Mode Follow Ratio) / MSR (Mode Step Ratio): TMP function only.

MFR uses external encoder input if it is in quadrature mode; MSR uses step/direction mode. When using SRC(2) — internal time base at PID rate — there isn't a distinction.

- MC (Mode Cam): TMP function with Cam function

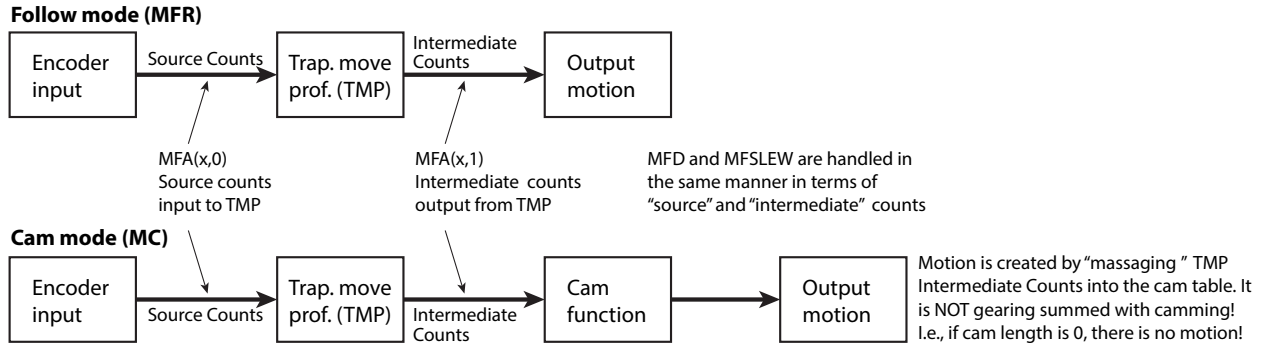
The TMP function's output (intermediate counts) feeds into the cam's input.

Motion is created by "massaging" TMP intermediate counts into the cam table. It is NOT gearing summed with camming. If the cam length is 0, there is no motion!

Use the Fixed Segment Cam Simulator (available on the Moog Animatics website at <https://www.animatics.com/support/downloads/knowledgebase.html>) to learn how to properly select the appropriate settings that perform the number of cam cycles desired. For example, you can use it to determine the settings if the application intends to perform a single-shot of the whole cam (refer to Camming - Demo XY Circle on page 875 for a single-shot program example).

NOTE: The Fixed Segment Cam Simulator is intended as a gearing/camming training aid only. It is not designed as an all-inclusive means for creating camming applications.

Refer to the next figure.



NOTE: The relationship between TMP source counts and TMP intermediate counts is affected by the ratio of MFMUL/MFDIV. For gearing, MFMUL/MFDIV determines how far; for camming, MFMUL/MFDIV determines how fast.

Follow Mode and Cam Mode Functional Diagrams

NOTE: Before programming an electronic camming application, it is *strongly recommended* to first evaluate your application in terms of source counts or intermediate counts, and variable or fixed cam. Refer to the next two sections.

Should I choose Source Counts or Intermediate Counts?

NOTE: In order to simplify programming and math calculations, it is *strongly recommended* that all MFA, MFSLEW and MFD second parameters be the same, either a 0 or a 1.

Refer to the previous figure. The choice of MFA(x,0) vs. MFA(x,1), 'encoder input to TMP' (or "source counts") versus 'motor output from TMP' (or "intermediate counts") is application-dependent. You should use whichever represents the values used in your application. In other words, it should be values that you want to keep the same effect even if MFMUL/MFDIV is changed.

NOTE: The MFA(x,1) format of the command is based on the output of the TMP function (intermediate counts); it IS NOT the motor/shaft total output!

For example:

- If you know that you need to follow a conveyor (not driven by the SmartMotor) and accelerate over 1000 counts distance on that conveyor as the input value, then choose MFA(x,0) for source counts.
- If you know that you need Follow mode, and the output distance of the acceleration needs to be a certain distance of the conveyor being driven by a SmartMotor, then choose MFA(x,1) for intermediate counts.

When in Cam mode, *the output of the TMP function goes into the cam*. Therefore, you will likely want to select MFA(x,1) to ensure the cam input is a known amount for the TMP (i.e., you will execute a known portion of the cam regardless of the MFMUL/MFDIV settings). If you have a cam application and you need values in terms of source counts distances, then some additional calculations will be needed in the program (based on the examples given). Refer to the figure Source Counts into Cam versus Intermediate Counts into Cam on page 161.

Should I choose Variable or Fixed cam?

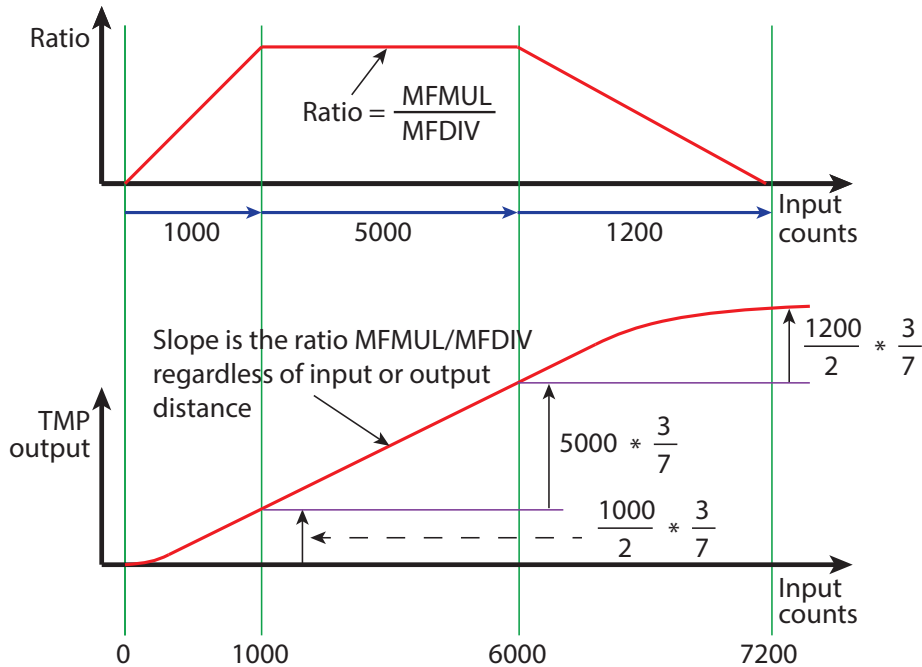
The choice of fixed length segments vs. variable length segments in a cam is another choice to make.

Generally speaking, fixed-length segments are simpler and allow more overall points because less storage space is needed. This works well if you have a large amount of uniformly sampled data points, for example, the output of a CAD drawing.

Variable segment lengths allow for some special applications. You can more carefully craft a set of data points that exactly coincide with certain events in the cam input. If you have a set of data points where you want to reach a specific position for a specific distance of the input (but that doesn't line up with a regularly-sampled rate), then the variable-length segments can be used to exactly line up with those input values. This can produce a smoother, and more predictable, result if spline interpolation is enabled in Cam mode.

Further, with variable segments, there is a possibility that certain types of applications could significantly reduce the number of cam points required. For example, there is a tool in SMI that can take data points and apply a "data pruning" to reduce the number of cam points required. However, the tradeoff is that each remaining point requires more storage space but the overall table possibly uses vastly fewer points. In addition, variable-segment cams make it easier to have specific linear and spline interpolated moves in a reduced cam table, allowing sharp corners mixed with smooth curves in a single path.

Source Counts into Cam - MFA, MFD, MFSLEW use command (x,0)

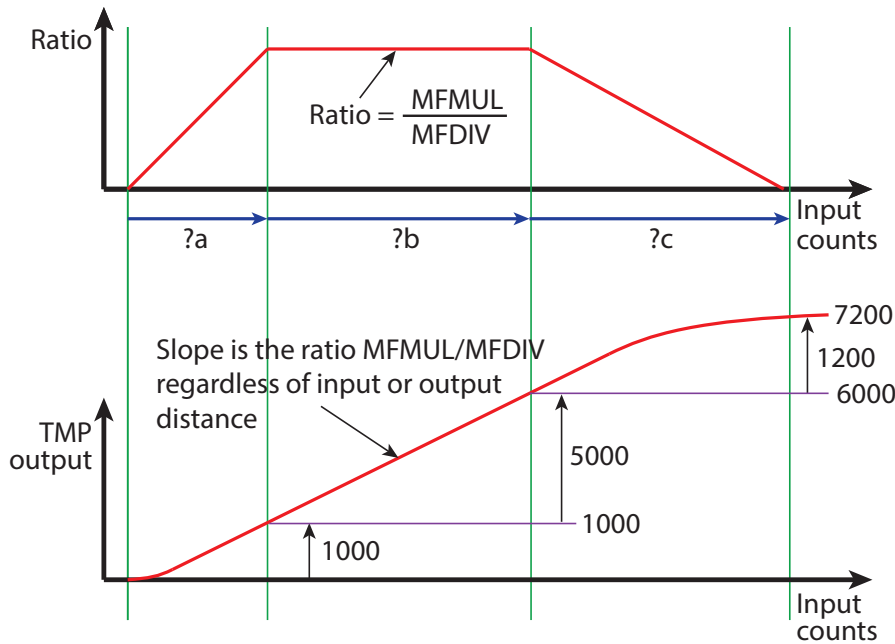


MFA(x,0), MFD(x,0), MFSLEW(x,0)

If MFMUL or MFDIV are changed, the new move will respect the output targets and adjust output accordingly.

MFA(1000,0) MFMUL=3
 MFSLEW(5000,0) MFDIV=7
 MFD(1200,0)

Intermediate Counts into Cam Diagram - MFA, MFD, MFSLEW use command (x,1)



When MFA(x,1), MFD(x,1), MFSLEW(x,1), the firmware back-figures which input values are needed to achieve this:

$$?a = \frac{7}{3} * 2 * 1000$$

$$?b = \frac{7}{3} * 5000$$

$$?c = \frac{7}{3} * 2 * 1200$$

The ratio of MFMUL/MFDIV is still respected.

If MFMUL or MFDIV are changed, the new move will respect the output target and adjust inputs accordingly.

MFA(1000,1) MFMUL=3
 MFSLEW(5000,1) MFDIV=7
 MFD(1200,1)

Source Counts into Cam versus Intermediate Counts into Cam

Electronic Camming Notes and Best Practices

These are important notes and best practices for electronic camming applications:

- The first cam point should be CTW(0,0) or CTW(0) variable length or fixed length cam segment, respectively.
- It is up to the programmer to pick a repeating cycle of the TMP function (MFA + MFSLEW + MFD) that also matches the controller length of the table (CTA). Note that:
 - The output of MFA, MFSLEW and MFD are physically executed in sequence. They DO NOT overlap in any way (see the next bullets). Therefore, it may be helpful to place them in sequence in your program.
 - For simplicity, use MFA(x,1), MFSLEW(y,1), and MFD(z,1), output from TMP counts, because then: $x+y+z =$ the total distance as input into the cam.

For example, if your cam repeats every 800 input counts, then you must distribute that 800 counts over the ascend, slew, and descend parts of the TMP. For an illustration of the parts of the TMP, refer to the Gearing Profile figure on page 144.

NOTE: DO NOT assume that MFSLEW (slew) alone is the total cam length; it is not!

- If the application needs to function in terms of *input encoder counts* (see the previous figure), then these transforms must be considered:

TMP ascend and descend relationship is: $\text{Output} = (\text{Input}/2) * (\text{MFMUL}/\text{MFDIV})$

Slew section is: $\text{Output} = \text{Input} * (\text{MFMUL}/\text{MFDIV})$

- As shown in the previous two bullets, MFMUL and MFDIV are imposed on command(x,0) values. However, note that MFMUL and MFDIV still have an effect in *both* modes (i.e., the ratio of input to output counts is still MFMUL/MFDIV). The difference is the *final* output value used as the trigger to go to the next phase is not affected in (x,1) mode.
- MFSDC takes two arguments. To disable any "repeating" of the TMP function, the command is MFSDC(-1,0).

NOTE: MFSDC(-1) is seen as an error and won't change the existing setting.

For diagrams of MFSDC settings and their results, see MFSDC Modes on page 148. However, note that the last mode shown *is not* for use with Cam mode.

- If no MFSLEW is set (default operation), the ramp function will execute MFA and then run forever (until X command). Use the command MFSLEW(-1), MFSLEW(-1,0) or MFSLEW(-1,1) to disable the slew distance (i.e., any of those three forms of MFSLEW will disable the slew distance).
- Camming with gearing fed into it is the default operation, i.e., camming with gearing of MFMUL=1, MFDIV=1, the MFA and MFD commands are at 0 distance, and MFSLEW is ignored. In this default operation, if you issue MC, G commands, the motor starts camming off of the source SRC(1, or 2) immediately. When a set single shot or repeated cycle is desired, that is where MFSDC and the other commands come into play, especially for label feeding, traverse cutting, and similar applications. For an example, refer to the sample program Camming - Demo XY Circle on page 875.

- Always structure the program to minimize writing of the cam table to the EEPROM.



CAUTION: When writing a cam table to EEPROM, structure the program so that the cam table is not frequently rewritten or written from a loop. Repeated erasing and rewriting can burn bits and corrupt data.

There are various ways to achieve this—refer to the next code snippet for one example:

```
C123          'Example of using EEPROM to flag code that has been run
EPTR=100     'set pointer to EEPROM
VLD(a,1)     'load value
IF a!=123    'if value does not equal 123

'do something here that you want to do only once
'such as write a cam table to nonvolatile memory

          EPTR=100      'set EEPROM pointer
          a=123        '
          VST(a,1)     'write value to EEPROM
ELSE
'something was already done
ENDIF
RETURN
```

Examples

Fixed cam example with both controller or follower counts as inputs to the cam:

```

CTE(1)
CTA(5,8000)
CTW(0)      'CP=0 {cam pointer or cam index pointer}
CTW(500)    'CP=1
CTW(4000)   'CP=2
CTW(500)    'Will turn off at this point
CTW(0)
MFMUL=1
MFDIV=2
MCMUL=3
MCDIV=4

```

'Cam input values in terms of "controller" (encoder input to ramp) counts:

```

MFA(a,0)
MFSLEW(s,0)
MFD(d,0)
'(a/2 + s + d/2) * MFMUL/MFDIV = 8000*4 = 32000 counts

```

'OR, cam input values in terms of "follower" (motor output from ramp) counts:

```

MFA(a,1)
MFSLEW(s,1)
MFD(d,1)
'a+s+d = 8000*4 = 32000 counts

```

Variable cam example with both controller or follower counts as inputs to the cam:

```

CTE(1)
CTA(5,0)
CTW(0,0)      'CP=0 {cam pointer or cam index pointer}
CTW(500,8000) 'CP=1
CTW(4000,16000) 'CP=2
CTW(500,24000) 'Will turn off at this point
CTW(0,32000)
MFMUL=1
MFDIV=2
MCMUL=3
MCDIV=4

```

'Cam input values in terms of "controller" (encoder input to ramp) counts:

```

MFA(a,0)
MFSLEW(s,0)
MFD(d,0)
'(a/2 + s + d/2 ) * MFMUL/MFDIV = 32000

```

'OR, cam input values in terms of "follower" (motor output from ramp) counts:

```

MFA(a,1)
MFSLEW(s,1)
MFD(d,1)
'a+s+d = 32000

```

Electronic Gearing and Camming over CANopen

Beginning with firmware 5.x.4.30 and later, the SmartMotor provides precise time synchronization over CANopen between motors for electronic gearing and camming applications (for example, traverse and

take-up spooling). The CANopen objects related to this are: 1005h, 1006h, 2207h, 2208h, 2209h, 220Ah-220Dh. For details on these objects refer to the *SmartMotor CANopen Guide*. For a sample user program, see CAN Bus - Time Sync Follow Encoder on page 883.

NOTE: This capability is currently available on Class 5 SmartMotors only.

Electronic Camming Commands

The next sections describe related commands. For more details on these commands, see Part 2: SmartMotor Command Reference on page 247.

CTE(table)

Erase tables in EEPROM memory starting at the value specified

To erase all EEPROM tables, choose CTE(1). By choosing a number higher than 1, lower table numbers can be preserved. If, for example, there were three tables stored, CTE(2) would erase table 2 and 3, but not table 1. CTE(0) is not defined.

CTA(points,seglen[,location])

Add a Cam table

The CTA command configures a table to use either EEPROM memory (default) or the data variable space (optional) in preparation for writing the table with the CTW command.

points	Specifies the number of points in the table.
seglen	Specifies the controller encoder distance between each point. If seglen is set to 0, then the distance is specified per data record through the CTW command.
[,location]	Is optional and specifies if this is a table in user variables or EEPROM. By default, if [,location] is omitted, then EEPROM is chosen. If [,location] is 0, then the user array location is chosen (al[0] through al[50].) Only one table can exist in the user variables. Up to 10 tables (numbered 1 through 10) can exist in EEPROM location.

CTW(pos[,seglen][,user])

Write a Cam table

The CTW command writes to the table addressed by the most recent CTA command. CTW writes to either the EEPROM-stored tables or the user-array-stored tables.

NOTE: Typically, the actual Cam table would not be part of the program that executes the mode. SMI tools are available to facilitate Cam table generation.

pos	The position coordinate of the motor for that data point. The first point in the table should be set to 0 to avoid confusion. When the table is run, the currently commanded motor position seamlessly becomes the starting point of the table. By keeping the first point of the table at 0, it is easier to realize that all of the data points are relative to that starting point.
[,seglen]	If this Cam table was specified as variable length in the CTA command, then [,seglen] is required for each data point. It is optional when using a fixed-length Cam table (specified in the CTA command). [,seglen] represents the absolute distance of the encoder source beginning from the start of the table. For

reasons similar to pos, [,seglen] should also be 0 for the first data point specified.

If you wish to use the optional [,user] parameter, then the [,seglen] parameter must be used (set to the default: 0).

[,user] Optional. Defines Cam user bits and Spline mode override. It is an 8-bit binary weighted value where:

Bit 0-5: User may apply as desired to Cam status bits 0-5 of Status word 8.

Bit 6: Factory Reserved — leave as 0.

Bit 7: When set to 0, no special override of Spline mode. When set to 1, the segment between the previous point and this point are forced into linear interpolation. Bit 7 has no effect when MCE has chosen linear mode.

When loading Cam tables, it is important to be aware of the table capacity. As mentioned previously:

- When a Cam table is stored in user array memory (a[0]-a[50]), 52 points can be stored as fixed-length segments; 35 points are possible when variable-length segments are used.
- When Cam tables are written to EEPROM memory, significantly more data can be written. For fixed-length segments, there is space for at least 750 points. For variable-length segments, at least 500 points can be written.

MCE(arg)

Cam table interpolation mode

The MCE(arg) command sets up the Cam function and defines the behavior based on these arguments:

- | | |
|---|---|
| 0 | Force linear motion for all sections |
| 1 | Spline mode with non-periodic data at ends of table |
| 2 | Spline mode with periodic data wrapped at ends of table |

MCW(table,point)

Cam table starting point

The MCW() command determines where to start the Cam function.

table Defines the Cam table number

point Defines the starting point in the table

RCP

Read Cam pointer

The RCP command reports the Cam pointer, and the CP variable can be used by the user program.

RCTT

Read number of Cam tables

The RCTT command reports the number of Cam tables, and the CTT variable can be used by the user program.

MC

Enter Cam mode

The MC command enters Cam mode and must be issued before the G command.

MCMUL=formula

Cam table value multiplier

This value is multiplied by the Cam table value and fed as a commanded value to the trajectory of the camming motor.

MCDIV=formula

Cam table value divisor

This value is divided into the Cam table value and fed as a commanded value to the trajectory of the camming motor

O(arg)=formula

Set move generator origin to value

The O()₌ command sets the move generator origin based on these arguments:

- 0 Set the origin of the global move generator (sets value of PA)
- 1 Set the origin of move generator 1 (sets value of PC(1))
- 2 Set the origin of move generator 2 (sets value of PC(2))

OSH(arg)=formula

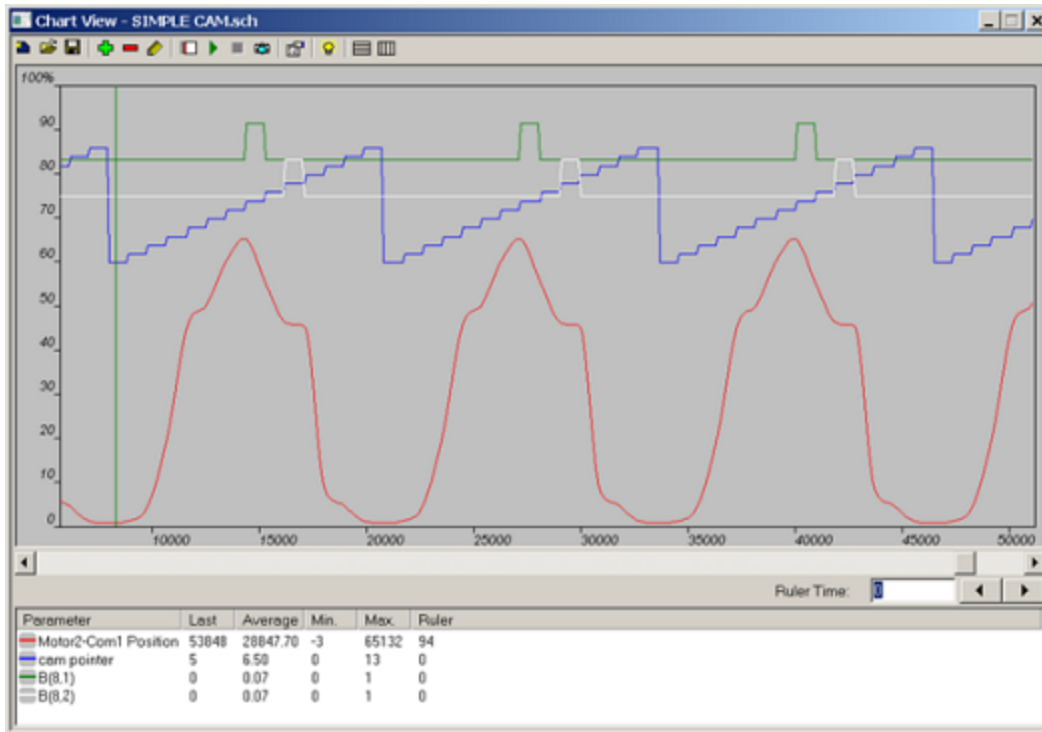
Shift move generator origin to value

The OSH()₌ command shifts the move generator origin based on these arguments:

- 0 Shift the origin of the global move generator (sets value of PA)
- 1 Shift the origin of move generator 1 (sets value of PC(1))
- 2 Shift the origin of move generator 2 (sets value of PC(2))

Cam Example Program

The next chart shows a plot of from the example code below it. This shows the effects of certain status bits, I/O points and the resulting motion profile. For additional cam example programs, see Part 3: Example SmartMotor Programs on page 862.



Sample Plot from Example Program

Note that:

- Changes to the SRC sign results in moving the opposite direction through the Cam table.
- Changes to MFMUL and/or MFDIV result in changes to the frequency or speed at which the Cam table is processed. The MFMUL and MFDIV commands do *not* have an effect on dwell time or distance. Dwell is strictly based on raw controller encoder counts selected by the SRC() command specifying internal virtual or external controller count source.
- Changes to MCMUL and MCDIV affect the amplitude of the wave form.
- In all cases, updates automatically take effect when passing from the last point of the Cam table into the first point (or from the first point into the last point, if traveling in the opposite direction).

```

ECHO           'ECHO on to allow auto addressing downstream
a=1           'Set default variable for address 1
WAIT=2000     'Wait for boot up time differences
PRINT(#128,"a=a+1",#13)      'each motor prints downstream a=a+1
WAIT=2000     'Wait for response time variations
ADDR=a       'Set motor address
WAIT=2000
EIGN(W,0) ZS

```



```

=====
'Set up parameters
rr=-1           'Home direction
vv=100000      'Home speed
aa=1000        'Home accel
ee=100         'Home error limit
tt=1500        'Home torque limit
hh=4000        'Home offset
mm=90000       'Max stroke with room
=====
GOSUB5         'Home to hard stop
GOSUB40        'Write cam table one time
GOSUB41        'Run cam operation
END
C40  ' Write cam table one time
    IF q==123 RETURN ENDIF
    CTE(1)
    CTA(15,8000)
    CTW(0)           'CP=0 {cam pointer or cam index pointer}
    CTW(500)         'CP=1
    CTW(4000)        'CP=2
    CTW(20000)
    CTW(45000)
    CTW(50000)
    CTW(60000)
    CTW(65000)
    CTW(55000,0,1)  'Turn on Bit 0 Status Word 8
    CTW(46000)      'Will turn off at this point
    CTW(45000,0,2)  'Turn on Bit 1 Status Word 8
    CTW(8000)       'Will turn off at this point
    CTW(4000)
    CTW(500)
    CTW(0)           'CP=14
    q=123
RETURN
=====
C41  ' Run cam operation
    MP PT=0 G TWAIT
    SRC(2)
    MCE(1)           'Spline
    MCW(1,0)
    MFA(0,1)
    MFD(0,1)
    MFMUL=1
    MFDIV=1
    MCMUL=1
    MCDIV=1
    MFSLEW(112000,1)
    MFSDC(100,0)    'Set dwell for "c" counts, auto rev. after dwell
    MC
    G
RETURN

```

```
'=====
C5 'Home routine (Home to hard stop)
  PRINT("HOME MOTOR",#13)
  VT=vv*rr      'Set home velocity
  ADT=aa        'Set home accel/decel
  MV            'Set to Velocity mode
  ZS            'Clear any prior errors
  T=tt*rr      'Preset torque values
  G             'Begin move toward hard stop
  MT
  WHILE ABS(EA)<ee LOOP      'Loop, while position error within limit
  PRINT("HIT HARD STOP",#13)
  G WAIT=50                'Wait 50 milliseconds
  O=hh*rr                  'Set origin to home offset
  PRINT("MOVING TO ZERO",#13)
  MP PT=0 G TWAIT          'Set motor to zero
RETURN
'=====
```



CAUTION: When writing a cam table to EEPROM, structure the program so that the cam table is not frequently rewritten or written from a loop. Repeated erasing and rewriting can burn bits and corrupt data. For details and sample code, refer to Electronic Camming Notes and Best Practices on page 162.

Mode Switch Example

Each SmartMotor™ can move freely between modes of operations including:

- Velocity Mode
- Torque Mode
- Relative Position Mode
- Absolute Position Mode
- Electronic Gearing
- Electronic Camming

This example shows how to use the SMI software to switch from Velocity mode to Torque mode while the SmartMotor is running.

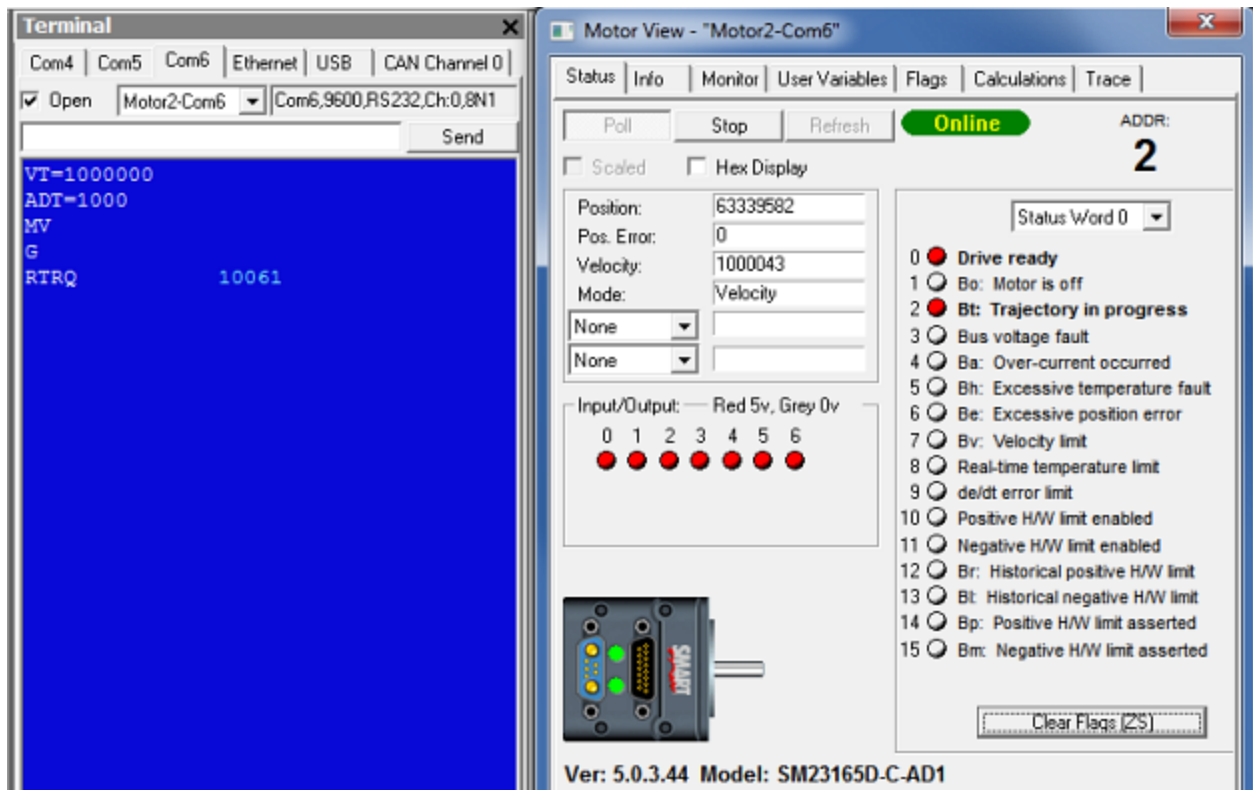
This procedure assumes that:

- The SmartMotor is connected to the computer. For details, see *Connecting the System* in the *SmartMotor Installation & Startup Guide* for your motor.
- The SmartMotor is connected to a power source. (Certain models of SmartMotors require separate control and drive power.) For details, see *Understanding the Power Requirements* in the *SmartMotor Installation & Startup Guide* for your motor.
- The SMI software has been installed and is running on the computer. For details, see *Installing the SMI Software* in the *SmartMotor Installation & Startup Guide* for your motor.
- You've completed the first-time motion example. For details, see *Moving the SmartMotor* in the *SmartMotor Installation & Startup Guide* for your motor.

To create the example:

1. Open the Motor View window. For details, see *Motor View* on page 72.
2. In the SMI software Terminal window, enter these commands:

```
VT=100000  
ADT=1000  
MV  
G  
RTRQ
```

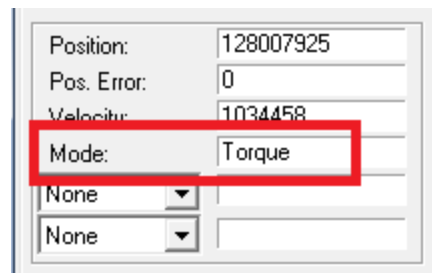


Commands for Velocity Mode

3. In the SMI software Terminal window, enter these commands:

```
T=TRQ
MT
G
```

The motor switches to Torque mode. The change is visible in the Mode box of the Motor View window.



Motor Switched to Torque Mode

Position Counters

The SmartMotor's processor contains various position counters. These are used to keep track of shaft position and the position of trajectories, or commanded counts within those trajectory generators. Some of the counters are virtual, while others directly track hardware. All counters may be set to zero or changed to a specific value as shown in the next table.

Counter	Description	Cleared to 0	Set to any value	Begins counting	Notes
CTR(0), RCTR(0)	Internal encoder	O=0	(When ENCO: O=value, OSH=relative) ¹	Always (from internal encoder)	Internal Encoder Hardware level counter that can be set or changed by user
CTR(1), RCTR(1)	External encoder input	MF0, MS0	(When ENC1: O=value, OSH=relative) ²	Always (from encoder ext inputs)	External Encoder Hardware level counter that can be set or changed by user
PA, RPA	Points to CTR(0) when ENCO (default), CTR(1) when ENC1	O=0	O=value, OSH=relative value	Always	Position Actual — Where the shaft is at any given time with regard to the encoder being used for PID
PMA, RPMA	Modulo actual per PML limit	O=0	(O=value, OSH=relative value) mod PML	Always	Position Modulo Actual is a modulo wrapped counter based off of PA but limited by PML (Position Modulo Limit)
PC, RPC	Trajectory into PID (EA=RPC-RPA)	O=0		Always (updates to RPA when drive is off)	Position Commanded by the PID trajectory
PC(0), RPC(0)	Identical to PC, RPC	O=0			Same as above, (PC)
PC(1), RPC(1)	Relative position of trajectory 1	O(1)=0	O(1)=value, OSH(1)=relative value	Accumulates delta position from RPA (but may have a con- stant offset from RPA)	PC(1)=PC(0) when in Single Tra- jectory mode
PC(2), RPC(2)	Relative position of trajectory 2	O(2)=0	O(2)=value, OSH(2)=relative value	Accumulates rel- ative targets (but don't assume PT = PA if relative moves were interrupted)	PC(2) can be used to track gearing or camming offsets from point where G(2) was issued
PT, RPT	Target position	PT=0	PT=value		Position Target is user spe- cified desired absolute target position; when G or G(1) is issued, this will be the Target
PRT, RPRT	Relative Target Pos- ition	PRT=0	PRT=value		PRT is always a relative dis- tance offset to where PC is; similar to PT, G or G(1) is required
PRA, RPRA	Relative Move Actual Position	Start new move		Start of MV or MP move (absolute or relative)	This tracks the distance that a trajectory 1 move MV or MP has traveled since the G com- mand, offset by the position error of the PID
PRC, RPRC	Relative Move Com- manded Position	Start new move		Start of MV or MP move (absolute or relative)	This tracks the distance that a trajectory 1 move MV or MP has traveled since the G com- mand

1. When ENCO is commanded, CTR(0) tracks PA.
2. When ENC1 is commanded, CTR(1) tracks PA.

Modulo Position

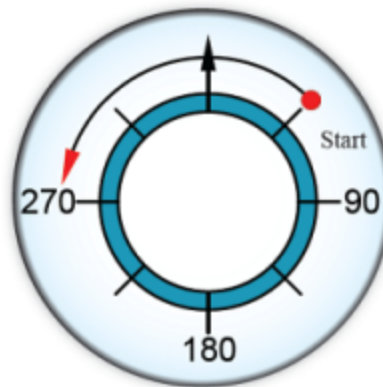
Modulo Position mode allows the user to define maximum position counter rollover.

A separate position counter (RPMA) is provided which can be programmed to report a restricted range of position from 0 to a settable amount. The PML command (position modulo limit) will set the range from 0 to PML-1. For example PML=10000 will roll over from 9999 to 0 and count upward in the forward direction. In the reverse physical direction of the motor, the numbers will roll back from 0 to 9999 and count downward.

NOTE: The modulo (RPMA) count will never go negative; it will always be $0 \leq \text{modulo value} \leq (\text{PML}-1)$.

In contrast, the typical RPA counter counts in the range -2147483648 to 2147483647, and will roll over when that range is exceeded; for example, 2147483647 rolls over to -2147483648.

Modulo Position mode is especially useful in rotary pan or azimuth controls for targeting systems, radar, and camera bases. Combined with the Combitronic™ interface, multi-camera surveillance systems can pass subject tracking from one pan & tilt to the next.



Modulo Position Example

For the previous figure:

- PML= 360 (Position Modulo Limit) maintains counts between 0 and 359 degrees.
- PMT= 270 (Position Modulo Target) takes the shortest path to the specified target position (270 degrees).

Modulo Position Commands

These commands are used to set and read the modulo position. For more details, see Part 2: SmartMotor Command Reference on page 247.

PML=formula	Set the Modulo Limit. The modulo counter reports between 0 and this value minus one. By default, this is 1000. The minimum value is 1000. This command resets the modulo count to 0.
x=PML	Assign to a variable the value set as the Modulo Limit.
RPML	Report the value set as the Modulo Limit.
PMT=formula	Set a Modulo Position Target for a position move. The motor takes the shortest path to reach this value. This means that the motor may move in either a clockwise or

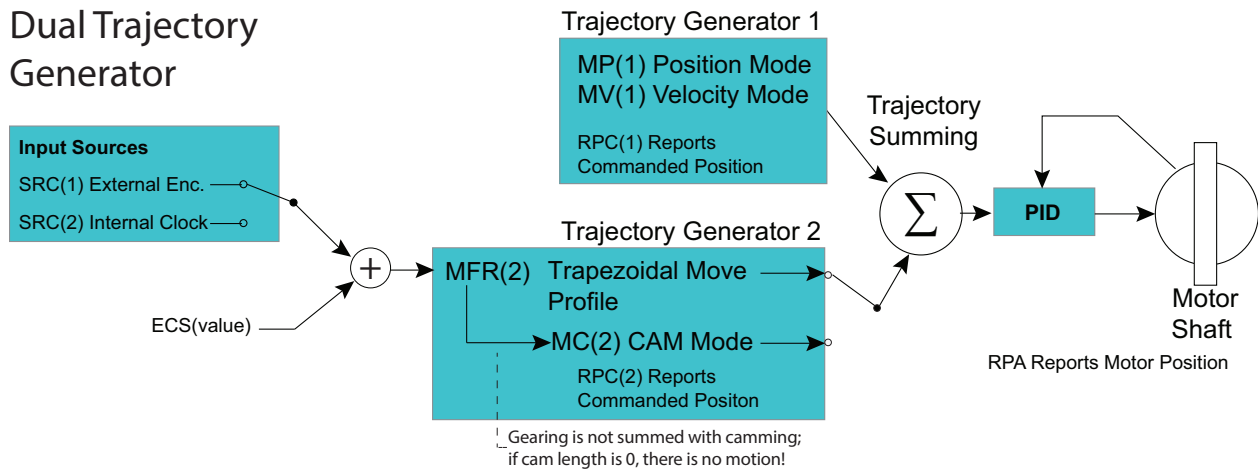
x=PMT	counter-clockwise direction, depending on which one produces the shortest motion in modulo terms.
RPMT	Assign to a variable the value set as the Modulo Target.
x=PMA	Report the value set as the Modulo Target.
RPMA	Assign to a variable the value of the Actual Modulo Position. NOTE: This counter is affected by the O= and OSH= commands.
ENC0	Report the value of the Actual Modulo Position (see NOTE above).
ENC1	Uses internal Encoder for commanded motion, actual position reporting and modulo position reporting.
	Uses external Encoder for commanded motion, actual position reporting and modulo position reporting. Be sure that the correct encoder type is selected with the MF0 (for quadrature) command, or the MS0 (step and direction) command.

Dual Trajectories

NOTE: In addition to this information, refer to Motion Command Quick Reference on page 895 for details on the primary motion commands and the differences between Actual, Target and Commanded, etc.

It is possible to create two trajectories that run concurrently. The next figure shows a flow diagram for a dual trajectory generator.

Dual Trajectory Generator



Dual Trajectory Generator Flow Diagram

There are restrictions on which combinations of moves are possible. In the next table, a combined move consists of a selection from column 1 and a selection from column 2.

NOTE: Torque mode cannot be combined with any other mode; selecting Torque mode replaces any other mode currently running.

Trajectory 1 - PC(1)	Trajectory 2 - PC(2)
Position - MP(1)	

Trajectory 1 - PC(1)	Trajectory 2 - PC(2)
Velocity - MV(1)	
CANopen Interpolation	
	Follow - MFR(2)
	Cam - MC(2)
Torque mode overrides all other modes	

Some commands may be directed at a specific trajectory generator. The next list shows these commands and which trajectory they can act on. For more details, see Part 2: SmartMotor Command Reference on page 247.

MP(trj#)	trj# = 1 only
MV(trj#)	trj# = 1 only
MFR(trj#)	trj# = 2 only
MSR(trj#)	trj# = 2 only
MC(trj#)	trj# = 2 only
G(trj#)	trj# = 1 or 2
X(trj#)	trj# = 1 or 2
S(trj#)	trj# = 1 or 2
TWAIT(trj#)	trj# = 1 or 2
x=PC(trj#)	trj# = 1 or 2
RPC(trj#)	trj# = 1 or 2
O(trj#)=	trj# = 1 or 2
OSH(trj#)=	trj# = 1 or 2
RMODE(trj#)	trj# = 1 or 2

Commands That Read Trajectory Information

The next list shows the commands that can be used to read trajectory information. For more details, see Part 2: SmartMotor Command Reference on page 247.

x=VT	Assign to a variable the value set as the Velocity Target.
RVT	Report the value set as the Velocity Target.
x=PT	Assign to a variable the value set as the Position Target.
RPT	Report the value set as the Position Target.
x=PRT	Assign to a variable the value set as the Position Relative Target.
RPRT	Report the value set as the Position Relative Target.
x=AT	Assign to a variable the value set as the Acceleration Target.
RAT	Report the value set as the Acceleration Target.
x=DT	Assign to a variable the value set as the Deceleration Target.
RDT	Report the value set as the Deceleration Target.
x=PC	Assign to a variable the value of the Commanded Position of the motor shaft as a result of motion trajectory generation. NOTE: This may include a sum of concurrent moves such as a Follow mode move combined with a position move.
RPC	Report the value of the Commanded Position of the motor shaft as a result of motion trajectory generation (see NOTE above).
x=PC(0)	Equivalent to x=PC.
RPC(0)	Equivalent to RPC.
x=PC(1)	Assign to a variable the value of the Commanded Position in trajectory generator 1's frame of reference.
RPC(1)	Report the value of the Commanded Position in trajectory generator 1's frame of reference.
x=PC(2)	Assign to a variable the value of the Commanded Position in trajectory generator 2's frame of reference.
RPC(2)	Report the value of the Commanded Position in trajectory generator 2's frame of reference.
x=VC	Assign to a variable the value of the real-time Commanded Velocity from all trajectory generators.
RVC	Report the value of the real-time Commanded Velocity from all trajectory generators.
x=AC	Assign to a variable the value of the real-time Commanded Acceleration from all trajectory generators (negative indicates deceleration).
RAC	Report the value of the real-time Commanded Acceleration from all trajectory generators (negative indicates deceleration).
x=VA	Assign to a variable the value of the Actual Velocity of the motor shaft. NOTE: The units are encoder counts per PID sample times 65536. The factor of 65536 allows for finer resolution of slower speeds. This finer-resolution information below 65536 is calculated through a filter because the only direct

	measurement is whole units of encoder counts per sample.
RVA	Report the value of the Actual Velocity of the motor shaft (see NOTE above).
VAC(arg)	Controls the filter used to measure speed. Default value is 65000. Higher values provide a smoother filter, at the cost of a longer settling time. The maximum value is 65535. A value of 0 turns off this filtering.
x=PA	Assign to a variable the value of the Actual Position of the motor shaft based on the encoder chosen by ENC1, ENC0 commands.
RPA	Report the value of the Actual Position of the motor shaft based on the encoder chosen by ENC1, ENC0 commands.
x=CTR(0)	Assign to a variable the value of the internal encoder Counter. NOTE: Unaffected by ENC1, ENC0 commands.
RCTR(0)	Report the value of the internal encoder Counter (see NOTE above).
x=CTR(1)	Assign to a variable the value of the external encoder Counter. NOTE: Unaffected by ENC1, ENC0 commands.
RCTR(1)	Report the value of the external encoder Counter (see NOTE above).

Dual Trajectory Example Program

In this example program, the SmartMotor moves to its origin and then instantly begins gearing to an external encoder. It then performs a relative move on top of the gearing relationship, with the relative move governed by the VT= and ADT= limits.

```

MP (1)      'Choose position mode from column 1
MFR (2)    'Choose Follow mode from column 2 at same time
PT=0       'This command only relevant to position move
VT=100000 'This command only relevant to position move
ADT=10     'This command only relevant to position move
MFMUL=1    'This command only relevant to Follow mode
MFDIV=1    'This command only relevant to Follow mode
G (1)      'Position move starts
TWAIT (1)  'Wait for position move only
G (2)      'Start Follow mode
WAIT=1000  'Wait for one second
PRT=1000   'Prepare for a relative move on top of
           'the Follow mode
G (1)      'Execute the relative position move
TWAIT (1)  'Wait for position move to finish
PT=0       'Command position 0 of the position mode
           'frame of reference
G (1)
S          'Stop all moves (follow and position)
MP        'Position mode exclusively. Note there is no
           'parentheses to make only position mode
PT=0
G         'Motor returns to position 0 in terms of actual
           'position because this is only position mode
    
```

Using Mixed Mode Operations After Homing

There are applications where you may wish to use mixed modes of operation after homing. For example, you may want to use dual trajectory and gearing. In these cases, you need to ensure that the motor is at a known starting location. To accomplish that, issue the next command sequence after the homing operation.

```
' Issue these commands immediately after homing, while at position 0.
O=0      'Set the origin to 0
O(1)=0   'Set trajectory 1 to 0
PRT=0    'Set the relative target position to 0
```

That command sequence sets the applicable internal counters to zero, which ensures that counter values are as expected for mixed mode operations.

Synchronized Motion

All SmartMotors equipped with the CAN port option come with Combitronic™ capability, which is basically the unification of all SmartMotors on a CAN network. With Combitronic technology comes the ability to perform multiple-axis, synchronized motion. The next sections describe the command set that enables multi-axis synchronized linear moves.

Synchronized-Target Commands

This section describes the synchronized-target commands. For more details on these commands, see Part 2: SmartMotor Command Reference on page 247.

PTS(), PRTS()

Position Target Synchronized Abs. and Rel.

These commands allow the user to identify two or three axis positions (posn) and associated axis CAN addresses (axisn) to cause a synchronized multi-axis move where the combined path velocity is controlled. For multiple-axis machines that are not using two motors to drive an axis, use this syntax:

```
PTS(pos1;axis1,pos2;axis2[,pos3;axis3])
```

In addition to the three-axis limitation, keep in mind the overall limit of 64 characters per line of code in the SmartMotor. Using variables in place of explicit positions is more space efficient. The PTS() command processes the positions as absolute, whereas the PRTS() command treats them as relative. After a PTS() or PRTS() command, the combined distance is stored in the PTSD variable and the combined axis move time is stored in the PTST variable, in (ms), in the event these may be useful to the programmer. PTSD and PTST can be used in a program or read over the serial channel by the RPTSD and RPTST commands. The PTS() command first goes out to the Combitronic network and gathers the last target positions in order to calculate the relative motion necessary to get to the next absolute position. It is extremely important that prior to a synchronized move being calculated with the PTS() or PRTS() commands, the previous target positions are accurate and uncorrupted by origin shifts. It is equally important that the synchronized move *not* be initiated before each axis reaches its previous target positions.

Some gantry-type, multiple-axis machines have two motors operating the same axis of motion (see the next figure). Below is the full syntax for the PTS command, which shows additional/optional parameters (enclosed in braces "{ }") for support of two motors operating the same axis. The optional parameter contains the motor address for the second motor of the axis. (For the PRTS command, replace PTS with PRTS.)

```
PTS(pos1;addr1{;addr1'},pos2;addr2{;addr2'}[,pos3;addr3{;axis3'}])
```

This is illustrated in the next example. (If you are using the PRTS command, substitute PRTS in place of PTS below.)

Position target X = 2000

Position target Y = 1000

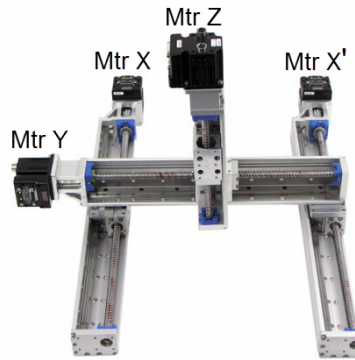
Position target Z = 500

Motor address X = 5

Motor address X' = 6

Motor address Y = 7

Motor address Z = 8



`PTS (2000;5;6,1000;7) 'Two-motor X axis (X, X'), plus Y axis`

`PTS (2000;5;6,1000;7,500;8) 'Two-motor X axis (X, X'), plus Y & Z axes`

In these cases, the same position, velocity and acceleration data sent to motor address 5 is also sent to motor address 6, with both motors driving the gantry's X axis.

For additional information on the PTS()command, see PTS(...) on page 692. For additional information on the PRTS()command, see PRTS(...) on page 685. Also, refer to A Note About PTS and PRTS on page 181.

VTS=formula

Velocity Target for Synchronized Move

The motion along a synchronized move is defined along the path. The VTS command is specific to defining the combined velocity of all contributing axes. If the move were to occur in an X-Y plane, for example, the velocity set by VTS would not pertain to the X axis or the Y axis, but rather to the combined motion, in the direction of motion.

ADTS=formula, ATS=formula, DTS=formula

Accel Targets for Synchronized Move

Like the velocity parameter, ADTS pertains to the combined path motion. The PTS() command scales the path velocity and accelerations set by the VTS= and ADTS= commands so that each axis reaches its constant velocity portions at exactly the same time, creating combined, straight-line motion. The ADTS= command sets both acceleration and deceleration, whereas ATS= and DTS= allow you to set separate acceleration and deceleration where desired.

PTSS(), PRTSS()

Position Target Sync, Absolute and Relative, Supplemental

The PTSS() and PRTSS() commands allow supplemental axis moves to be added and synchronized with the previous PTS() or PRTS() commanded motion. Issue these additional axis commands after a PTS() or PRTS() command, but before the next GS. These commands allow the user to identify one axis position (posn) and associated axis CAN addresses (axisn) at a time.

`PTSS(posn;axisn)`

The supplemental axis motions will start at exactly the same time as the main PTS() or PRTS() motion. With the next GS, they will transition from their accelerations to their slew velocities at exactly the same time, and decelerate and stop at exactly the same time.

Moves too short to ever reach the VTS= velocity will execute a triangular rather than trapezoidal profile, but the moves will still be synchronized.

The difference between a PTSS() move and a PTS() member is that the supplemental axis moves do not reduce the primary profile velocity in an effort to hold the total motion to a total combined velocity set by VTS. The same applies for acceleration. The combined motion of the PTS() move will be controlled to the VTS limit, and then PTSS() moves will simply align with that combined motion.

```
. . .
x=1000 y=2000 z=3500 a=100 b=200
PTS (x;1, y;2, z;3)   'Set next positions, axes 1, 2 & 3
PTSS (a;4)           'Set supplemental position, axes 4
PTSS (b;5)           'Set supplemental position, axes 5
GS                   'Go, starts the synchronized move
```

NOTE: If the supplemental axis move is longer than the PTS() move, the supplemental axis velocity will exceed the limit set by VTS=.

A Note About PTS and PRTS

The PTS and PRTS commands require motors to be stationary (not moving), not faulted, and in position mode before the commands are issued. These prerequisites are needed because if the previous move was in gearing, camming, or torque mode, or in a drive fault state as well, target trajectories and commanded trajectories may not align, which could lead to math errors when calculating synchronized move parameters.

NOTE: In preparation for a synchronized move, the motors must be stationary (not moving), in position mode and with no faults.

This applies to both PTS and PRTS commands before they are issued. Also, because PTSS and PRTSS only work after the above commands are issued and before a GS command, the following examples will still work if they are included.

Example subroutine of 2-axis synchronized relative move to position x:y for motors 1 and 2

This sample code may be executed by any motor sharing the same CAN bus network with the motors being commanded to move.

NOTE: Ensure no motor drive faults exist prior to calling this subroutine.

```
C20
OFF:0 MP:0 PRT:0=0 G TWAIT 'Initialize to stationary in position mode
PT:1=PC:1 PT:2=PC:2      'Set target and commanded positions equal
WAIT=50
VTS=v                    'Set target path velocity
ADTS=a                   'Set target path accel/decel
PRTS (x;1, y;2)          'Use Position Target Synchronized moves
IF PTSD!=0               'Prevent 0-length (divide by zero) move
  GS                      'Go Synchronized
  TSWAIT                  'Wait until path move time is complete
ENDIF
RETURN
```

Example subroutine of 3-axis synchronized relative move to position x:y:z for motors 1, 2 and 3

This sample code may be executed by any motor sharing the same CAN bus network with the motors being commanded to move.

NOTE: Ensure no motor drive faults exist prior to calling this subroutine.

C20

```

OFF:0 MP:0 PRT:0=0 G TWAIT      'Initialize to stationary in position mode
PT:1=PC:1 PT:2=PC:2 PT:3=PC:3  'Set target and commanded positions equal
WAIT=50
VTS=v                          'Set target path velocity
ADTS=a                          'Set target path accel/decel
PRTS (x;1,y;2,z;3)             'Use Position Target Synchronized moves
IF PTSD!=0                     'Prevent 0-length (divide by zero) move
    GS                          'Go Synchronized
    TSWAIT                      'Wait until path move time is complete
ENDIF
RETURN

```

Example subroutine of 2-axis synchronized absolute move to position x:y for motors 1 and 2

This sample code may be executed by any motor sharing the same CAN bus network with the motors being commanded to move.

NOTE: Ensure no motor drive faults exist prior to calling this subroutine.

C20

```

OFF:0 MP:0 PRT:0=0 G TWAIT      'Initialize to stationary in position mode
PT:1=PC:1 PT:2=PC:2            'Set target and commanded positions equal
WAIT=50
VTS=v                          'Set target path velocity
ADTS=a                          'Set target path accel/decel
PTS (x;1,y;2)                 'Use Position Target Synchronized moves
IF PTSD!=0                     'Prevent 0-length (divide by zero) move
    GS                          'Go Synchronized
    TSWAIT                      'Wait until path move time is complete
ENDIF
RETURN

```

Example subroutine of 3-axis synchronized absolute move to position x:y:z for motors 1, 2 and 3

This sample code may be executed by any motor sharing the same CAN bus network with the motors being commanded to move.

NOTE: Ensure no motor drive faults exist prior to calling this subroutine.

C20

```

OFF:0 MP:0 PRT:0=0 G TWAIT      'Initialize to stationary in position mode
PT:1=PC:1 PT:2=PC:2 PT:3=PC:3  'Set target and commanded positions equal
WAIT=50
VTS=v                          'Set target path velocity
ADTS=a                          'Set target path accel/decel
PTS (x;1,y;2,z;3)             'Use Position Target Synchronized moves
IF PTSD!=0                     'Prevent 0-length move (prevent divide by zero)
    GS                          'Go Synchronized
    TSWAIT                      'Wait until path move time is complete
ENDIF
RETURN

```

Other Synchronized-Motion Commands

These commands are used to start a synchronized move and wait for a synchronized move to complete. For more details on these commands, see Part 2: SmartMotor Command Reference on page 247.

GS

Start Synchronized Move

To start a synchronized motion profile, use the GS command. It acts behind the scenes and issues G commands to all axes involved in the previous PTS() or PRTS() command. It is important to be sure all motors are at their previous targets before issuing the GS command. Otherwise, the motion will not be synchronized.

TSWAIT

Wait for Synchronized Move to Complete

After a GS command has been issued to start a synchronized move, the TSWAIT command can be used to pause program execution until the move has been completed. A standard TWAIT command would not work where the motor issuing the PTS() and GS commands had a zero length contribution to the total move. The TSWAIT command was created for this reason.

```
. . .
ADTS=100           'Set target synchronized accel/decel
VTS=100000        'Set target synchronized velocity
PTS (30000;1,40000;2) 'Set target positions, axes 1 & 2
GS                'Go, starts the synchronized move
TSWAIT           'Optional wait for sync. move to complete
```

The previous example is a synchronized move in its simplest form. The code could be written in either motor 1 or 2 and it would work the same.

The TSWAIT command merely pauses program execution (except for interrupt routines). It may be desirable to continue running the program while waiting. In that event, the program can loop around the Synchronized Move Status Bit, which is status word 7, bit 15, accessible by variable B(7,15). So, the next While Loop code example is equivalent to the TSWAIT command, except that more code can be added within the loop for execution during the wait.

```
WHILE B(7,15)==1    'While synchronized move in process
. . .
LOOP                'Loop back
```

The next code example adds subroutine efficiency, the efficiency of setting up the next move while the existing move is ongoing, and adds an error check before continuing to issue synchronized move commands.

```

. . .
ADTS=100          'Set target synchronized accel/decel
VTS=100000       'Set target synchronized velocity
WHILE B(0,2):1|B(0,2):2|B(0,2):3  'Loop while motion in any axis
    WAIT=10      'Allow time for other CAN communications
LOOP             'Loop back
x=1000 y=2000 z=3500 GOSUB10 'Put positions into variables
x=2200 y=1800 z=1200 GOSUB10 'Put positions into variables
x=1500 y=2600 z=2500 GOSUB10 'Put positions into variables
x=-120 y=1000 z=1500 GOSUB10 'Put positions into variables
x=0 y=0 z=0 GOSUB10  'Put positions into variables
END             'End Program
C10            'Place label
    PTS(x;1,y;2,z;3) 'Set next positions, axes 1, 2 & 3
                    'and do this while the previous move
                    'is in progress
WHILE B(7,15)==1 'While synchronized move in process
'If one motor faults, stop all and end program -
    IF B(0,0):1==0 MTB:0 ENDFIF '*note
    IF B(0,0):2==0 MTB:0 ENDFIF '*note
    IF B(0,0):3==0 MTB:0 ENDFIF '*note
LOOP             'Loop back
GS              'Go, starts the synchronized move
RETURN         'Return to call
'*note: Managing faults is better done by using interrupts
'in other motors, taught later in this guide.

```

There is a note in the preceding example program stating that a better job can be done of detecting and reacting to errors by using interrupts. This is true because the example, as written, causes a considerable amount of unnecessary communications over the Combitronic interface. By loading interrupt routines in each SmartMotor that constantly monitor for drive status, each motor can be made responsible for reporting a local error. By this means, it is no longer necessary to poll each motor. The motor controlling the synchronized motion can simply do a quick check for reports right before issuing the next GS command. For more details, see Interrupt Programming on page 195.

Program Flow Details

This chapter provides information on using program flow commands with the SmartMotor.

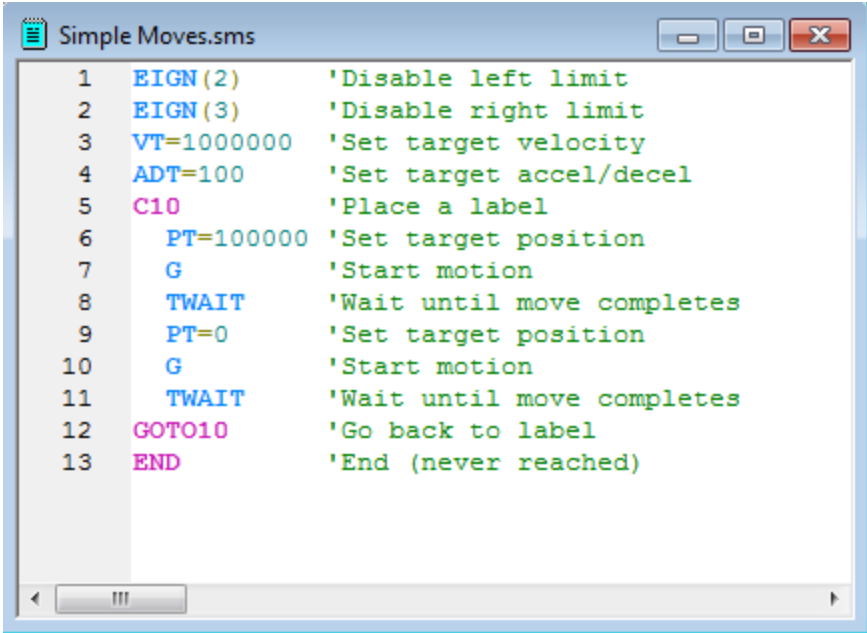
Introduction	186
Flow Commands	186
RUN	186
RUN?	187
GOTO#, GOTO(label), C#	187
GOSUB#, GOSUB(label), RETURN	188
IF, ENDIF	188
ELSE, ELSEIF	188
WHILE, LOOP	189
SWITCH, CASE, DEFAULT, BREAK, ENDS	190
TWAIT	190
WAIT=formula	191
STACK	191
END	191
Program Flow Examples	192
IF, ELSEIF, ELSE, ENDIF Examples	192
WHILE, LOOP Examples	192
GOTO(), GOSUB() Examples	193
SWITCH, CASE, BREAK, ENDS Examples	194
Interrupt Programming	195
ITR(), ITRE, ITRD, EITR(), DITR(), RETURNI	195
TMR(timer,time)	197

Introduction

Program commands are like tasks. Whether the task is to turn on an output, set a velocity or start a move, a program is a list of these tasks. When a programmed SmartMotor is powered-up or is reset with the Z command, it executes its program (list of tasks) from top to bottom, with or without a host PC connected. This section covers the commands that control the flow of the program.

SmartMotor programs are written in the SMI software editor, which is opened by selecting **File > New** from the SMI software toolbar. For details, see *Opening the SMI Window (Program Editor)* on page 51.

The next program example shows an infinite loop. It causes the motor to move back and forth forever.



```

Simple Moves.sms
1  EIGN(2)    'Disable left limit
2  EIGN(3)    'Disable right limit
3  VT=1000000 'Set target velocity
4  ADT=100    'Set target accel/decel
5  C10        'Place a label
6  PT=100000  'Set target position
7  G          'Start motion
8  TWAIT      'Wait until move completes
9  PT=0       'Set target position
10 G          'Start motion
11 TWAIT      'Wait until move completes
12 GOTO10     'Go back to label
13 END        'End (never reached)

```

Simple Move Program Example

NOTE: Programs execute at rates of thousands of lines per second.

Flow Commands

After a program starts, there are a variety of commands that can redirect program flow, and most of those can do so based on certain conditions. How these conditional decisions are set up determines what the programmed SmartMotor will do and how "smart" it will actually be.

Flow commands are described in the next sections. A later section describes Interrupt commands, which are used to execute subroutines based on the change in a status bit. For more details, see *Part 2: SmartMotor Command Reference* on page 247.

RUN

Execute Stored User Program

If the SmartMotor is reset with a Z command or at power-up, all previous variables and mode changes are erased for a fresh start, and the program begins execution from the top. Alternatively, the RUN command can be used to start the program, in which case the state of the motor is unchanged and its program will be invoked.

RUN?

Halt Program If No RUN Issued

The RUN? command prevents further execution of code until the RUN command is received over the serial channel. On power up, program code executes to the point of reaching the RUN? command. When RUN is issued through the serial port, the CPU executes all code— it starts at the top and moves down through the program, jumps over the RUN? command to the next line of code and then continues execution to the end of the program.

```
PRINT ("Boot-Up", #13)      'Message always prints
RUN?                       'Program stops here on reset or power up
PRINT ("Run Issued", #13)  'This runs if RUN received
END
```

The above code prints the first message only after a Z command or on power-up, but it prints *both* messages when a RUN command is received over the serial line.

During development, the RUN? command placed at the top of your program can protect you from accidentally locking up your SmartMotor with a bad program.

GOTO#, GOTO(label), C#

Redirect Program Flow, Place a Label

The most basic command for redirecting program flow, without inherent conditions, is GOTO# or GOTO (label), used in conjunction with the label C#. A label consists of the letter C followed by a number (#) from 0 to 999, and it is inserted in the program as a placeholder. If a label like C1 is placed in a program and that same number is placed at the end of a GOTO command like GOTO1, the program flow will be redirected to label C1, and the program will proceed from there.

```
C10           'Place label
  IF IN(0)==0 'Check Input 0
    GOSUB20   'If Input 0 low, call Subroutine 20
  ENDIF      'End check Input 0
  IF IN(1)==0 'Check Input 1
    a=30     'as example for below
    GOSUB(a) 'If Input 1 low, call Subroutine 30
  ENDIF      'End check Input 1
GOTO(10)     'Will loop back to C10
```

As many as a thousand labels (0 - 999) can be used in a program. However, the program will become increasingly difficult to read or debug as more GOTO commands are used.

Therefore, try using only one GOTO command, and use it to create the infinite loop necessary to keep the program running indefinitely as some embedded programs do. For example, put a C10 label near the beginning of the program but after the initialization code, and then place a GOTO10 at the end. Then, every time the GOTO10 is reached, the program will loop back to label C10 and start over from that point until the GOTO10 is reached again, which will start the process at C10 again, and so on. This will make the program run continuously.

Any program can be written with only one GOTO. It might be a little harder, but it will force better program organization. If you organize your program using the GOSUB command instead of multiple GOTO commands, it will be much easier to read and support.

GOSUB#, GOSUB(label), RETURN

Execute a Subroutine and Return

Just like the GOTO# command, the GOSUB# command, used in conjunction with a C# label, redirects program execution to the location of the label. However, unlike the GOTO# command, the C# label needs to eventually require a RETURN command. This returns the program execution to the location of the original GOSUB# command that initiated the redirection.

There may be many sections of a program that need to perform the same basic group of commands. By encapsulating these commands between a C# label and a RETURN, they become a subroutine that can be called anytime from anywhere with a GOSUB# or GOSUB(label), rather than being repeated. There can be as many as one thousand (0 - 999) different subroutines, and they can be accessed as many times as the application requires.



CAUTION: Calling subroutines from the host can crash the stack if not done carefully.

By pulling sections of code out of a main loop and encapsulating them into subroutines, the main code can also be easier to read. Therefore, organizing code into multiple subroutines is a good practice.

```
C10           'Place label
  IF IN(0)==0 'Check Input 0
    GOSUB20   'If Input 0 low, call Subroutine 20
  ENDIF      'End check Input 0
  IF IN(1)==0 'Check Input 1
    a=30      'as example for below
    GOSUB(a)  'If Input 1 low, call Subroutine 30
  ENDIF      'End check Input 1
GOTO(10)     'Will loop back to C10
```

IF, ENDIF

Conditional Test

When the execution of the code reaches the IF command, the code between that IF and the next ENDIF executes only when the condition directly after the IF command is true. For example:

```
a=IN(0)      'Variable "a" set 0,1
a=a+IN(1)    'Variable "a" 0,1,2
IF a==1      'Use double "=" to test
  b=1        'Set "b" to one
ENDIF        'End IF
```

Variable "b" gets set to one only when variable "a" is equal to one. If a is not equal to one, then the program continues to execute using the command after the ENDIF command.

Also, notice that the SmartMotor language uses a single equal sign (=) to make an assignment, such as where variable a is set to equal the logical state of input 0. Alternatively, a double equal sign (==) is used as a test, to query whether variable a is equal to 1 without making any change to it. These are two different functions. Having two different syntaxes has other benefits.

ELSE, ELSEIF

The ELSE and ELSEIF commands can be used to add flexibility to the IF statement. If it were necessary to execute different code for each possible state of variable "a", the program could be written as:

```

a=IN(0)          'Variable "a" set 0,1
a=a+IN(1)        'Variable "a" 0,1,2
IF a==0          'Use double "=" test
    b=1          'Set "b" to one
ELSEIF a==1      '
    c=1          'Set "c" to one
ELSEIF a==2      '
    c=2          'Set "c" to two
ELSE             'If not 0 or 1
    d=1          'Set "d" to one
ENDIF           'End IF

```

There can be many ELSEIF statements but only one ELSE. If ELSE is used, it needs to be the last statement in the structure before the ENDIF. There can also be IF structures inside IF structures — that's called "nesting" and there is no practical limit to the number of IF structures that can nest within one another.

The commands that can conditionally direct program flow based on a test, such as the IF, where the test may be `a==1`, can have far more elaborate tests inclusive of virtually any number of operators and operands. The result of a comparison test is zero if "false", and one if "true". For example:

```

IF ABS(EA-5)>x   'A numeric test
    'placing further commands here
ENDIF
IF (a<b)&(c<d)   'A logical test using bit-wise AND
    'placing further commands here
ENDIF
IF (a==b)|(c!=d)'A logical test using bit-wise OR
    'placing further commands here
ENDIF

```

Complex logical tests involving bit-wise AND, OR and exclusive OR depend on whether the result of an operation is zero or one. Any test for zero or not zero must be made explicitly.

```

IF (a<b)&c       'This should be avoided and replaced by
IF (a<b)&(c!=0) 'an explicit test of c not zero

```

WHILE, LOOP

The most basic looping function is a WHILE command. The WHILE requires an expression that determines whether the code between the WHILE and the next LOOP command will execute or be passed over. While the expression is true, the code executes. An expression is true when it is nonzero. If the expression results in a "zero" then it is false. These are valid WHILE structures:

```

WHILE 1 '1 is always true
    OS(0) 'Set output to 1
    OR(0) 'Set output to 0
LOOP 'Will loop forever
a=1 'Initialize variable "a"
WHILE a 'Starts out true
    a=0 'Set "a" to 0
LOOP 'This never loops back
a=0 'Initialize variable "a"
WHILE a<10 'a starts less
    a=a+1 'a grows by 1
LOOP 'Will loop back 10 times

```

The task or tasks within the WHILE loop executes as long as the loop condition remains true.

The BREAK command can be used to break out of a WHILE loop, although that somewhat compromises the elegance of a WHILE statement's single test point and makes the code harder to read/debug. The BREAK command should be used sparingly or, preferably, not at all in the context of a WHILE loop.

If it's necessary for a portion of code to execute only once based on a certain condition, then use the IF command.

SWITCH, CASE, DEFAULT, BREAK, ENDS

Long, drawn out IF structures can be cumbersome, and burden the program visually. In these instances it can be better to use the SWITCH structure.

This code would accomplish the same thing as the ELSEIF program example:

```

a=IN(0) 'Variable "a" set 0,1
a=a+IN(1) 'Variable "a" 0,1,2
SWITCH a 'Begin SWITCH
    CASE 0
        b=1 'Set "b" to one
        BREAK
    CASE 1
        c=1 'Set "c" to one
        BREAK
    CASE 2
        c=2 'Set "c" to two
        BREAK
    DEFAULT 'If not 0 or 1
        d=1 'Set "d" to one
        BREAK
ENDS 'End SWITCH

```

Just as a rotary switch directs electricity, the SWITCH structure directs the flow of the program. The BREAK statement then jumps the code execution to the code after the associated ENDS command. The DEFAULT command covers every condition other than those listed. Its use is optional.

TWAIT

Wait for Trajectory to Finish

The TWAIT command pauses program execution while the motor is moving. The pause is terminated by either the controlled end of a trajectory or the abrupt end of a trajectory due to an error. If there are a

succession of move commands without this command or similar waiting code between them, the commands will overtake each other because the program advances even while moves are taking place.

The next program example has the same effect as the TWAIT command, but it allows other things to be programmed during the wait instead of just waiting — such things would be inserted between the two commands.

```
WHILE Bt          'While trajectory
...
LOOP             'Loop back
```

WAIT=formula

Wait, Pause Program Execution for Time in Milliseconds

There will probably be circumstances where the program execution needs to be paused for a specific period of time. The WAIT command pauses the program for the specified number of milliseconds. WAIT=1000, for example, would wait one second. The next code example would be the same as WAIT=1000, only it would allow code to execute during the wait if it was placed between the WHILE and the LOOP.

```
CLK=0            'Reset CLK to 0
WHILE CLK<1000  'CLK will grow
...
LOOP            'Loop back
```

STACK

Reset the GOSUB Return Stack

Information about the nesting of subroutines is held in the STACK ("nesting" is when one or more subroutines exist within others). In the event program flow is directed out of one or more nested subroutines without executing the included RETURN commands, the stack will be corrupted. The STACK command resets the stack with zero recorded nesting. Use it with care and try to build the program without requiring the STACK command.

One possible use of the STACK command might be if the program used one or more nested subroutines and an emergency occurred. In this case, the program or operator could issue the STACK command and then a GOTO command, which would send the program back to a label at the beginning. Using this method instead of the RESET command would retain the states of the variables and allow further specific action to resolve the emergency.

```
C1              'Subroutine C1
  STACK         'Clear the nesting stack
  RUN           'Begin the program, retaining variables
RETURN         'Never reached, but necessary for comp.
```

END

End Program Execution

To compile properly, every program needs an END command somewhere, even if it is never reached. If the program needs to run continuously, the END statement has to be outside the main loop.

If it is necessary to stop a program, issue an END command and execution stops at that point. An END command can also be sent by the host to intervene and stop a program running within the motor.



WARNING: An END command will not stop motion of a motor.

The SmartMotor program is never erased until a new program is downloaded. To erase the program in a SmartMotor, download only the END command as if it were a new program. That will be the only command that is left in the SmartMotor until a new program is downloaded.

Program Flow Examples

This section describes techniques that can be used for flow control in your SmartMotor program. All sample code shows advanced use of program flow syntax. Additionally, there are references to lesser-used math functions and system parameters to enforce learning of new programming techniques.

NOTE: The Variables and Math section should be referenced for a better understanding of the next example programs. For details, see Variables and Math on page 198.

IF, ELSEIF, ELSE, ENDIF Examples

The next example shows the proper use of the IF, ELSE and ENDIF commands along with nested conditions and math capabilities. For more details on the IF, ELSE and ENDIF commands, see IF, ENDIF on page 188.

```
'Find shortest dist. to Top Dead Center Shaft position from present position
IF (PA%RES)>(RES/2)      'Check shortest distance using Modulo Math function
    PRT=RES-(PA%RES)    'Set Relative Position to Modulo Remainder
ELSE
    PRT=- (PA%RES)      'Otherwise Set to RES - Modulo remainder
ENDIF
```

The next example uses #define to associate values and I/O points for use in code.

```
#define UpperLimit      3000      'Set high voltage threshold
#define LowerLimit      1500      'Set low Voltage threshold
#define MyVoltage      INA(V1,3)  'Set input Port to read
C124
    IF (UpperLimit>MyVoltage) & MyVoltage>LowerLimit
        PRINT("Voltage is in range",#13)
    ELSEIF MyVoltage>=UpperLimit
        PRINT("Voltage is too high",#13)
    ELSE
        PRINT("Voltage is toolow",#13)
    ENDIF
RETURN
```

WHILE, LOOP Examples

The next example shows the proper use of the WHILE and LOOP commands. For more details on these commands, see WHILE, LOOP on page 189.


```

#define GoSelSwitch      INA(V1,6)
#define Go                200
#define Sel              4000
C345 'Detecting switch on SmartBox
IF (GoSelSwitch<Sel) & (GoSelSwitch>Go)
    PRINT("Switch Released",#13)
ENDIF
WHILE 1
    IF GoSelSwitch>=Sel
        PRINT("Sel",#13)
    WHILE GoSelSwitch>=Sel LOOP
        PRINT("Switch Released",#13)
    ENDIF
    IF GoSelSwitch<=Go
        PRINT("Go",#13)
    WHILE GoSelSwitch<=Go LOOP
        PRINT("Switch Released",#13)
    ENDIF
LOOP
RETURN

```

GOTO(), GOSUB() Examples

The Class 5 software allows passing of values into GOTO and GOSUB commands. For details, see GOTO#, GOTO(label), C# on page 187 and GOSUB#, GOSUB(label), RETURN on page 188.

There are two theories on writing code: One says uses all GOTO commands; the other says use all GOSUB commands. There are pros and cons to both methods.

- GOTO is good for conditional code bypassing
- GOSUB ensures a return to where you came from

Pay attention to either command when you run into a RETURN that gets ignored by a GOTO or when you never reach a RETURN for a previous GOSUB due to a GOTO.

```

i=400      'Motor Current  to check for
WHILE 1    'While forever
    IF UIA>i          'If motor current in mAmps is > "i"
        GOSUB(100)
        WHILE UIA>i LOOP 'prevent double trigger
    ENDIF
LOOP
C100
    IF UIA>(i*2)      'If current is twice as much
        GOTO200      'bypass PRINT line below
    ENDIF
    PRINT("Current is above ",i,"mAmps",#13)
C200
    PRINT("Current twice as high as it should be!",#13)
RETURN

```

GOTO(label) and GOSUB(label) may be used where label can be a variable, a specific number or a single operand such as a+b, x-5, etc.

NOTE: Nested parenthesis are not permitted—for example, GOTO(IN(3)).

```

WHILE 1
  x=IN(W,0,15)      'precalculate to prevent parenthesis in GOSUB()
  IF y!=x
    y=x
    GOSUB(15-x)    'simple single operand math only
  ENDIF
LOOP

```

SWITCH, CASE, BREAK, ENDS Examples

The next code example shows the proper use of the SWITCH, CASE, BREAK and ENDS commands. For more details on these commands, see SWITCH, CASE, DEFAULT, BREAK, ENDS on page 190.

```

C500
#define FiveValues      INA(V1,3)/1000
#define CheckPot        501
WHILE 1
  IF x!=FiveValues
    GOSUB(CheckPot)
  ENDIF
LOOP
RETURN
C501      'CheckPot (Check Potentiometer)
y=FiveValues
SWITCH FiveValues
CASE 0 PRINT("Value ",y,#13) BREAK      'Note: Defines not allowed in PRINT
CASE 1 PRINT("Value ",y,#13) BREAK
CASE 2 PRINT("Value ",y,#13) BREAK
CASE 3 PRINT("Value ",y,#13) BREAK
CASE 4 PRINT("Value ",y,#13) BREAK
ENDS
x=FiveValues
RETURN

```

The next code shows an example of printing menu selections for a terminal screen using SWITCH. This example comes from the SmartBox demo program.

```

IF INA (V1,6)>3800           'If SEL pressed
    m=m+1                   'increment menu item
    IF m>9 m=1 ENDIF        'Limit menu items between 0 and 9
        GOSUB102           'PRINT MENU OPTION
    WHILE INA (V1,6)>3500 LOOP 'don't double trigger
ENDIF
'=====
C102
    SWITCH m
        CASE 1 PRINT("Electronic Gearing 1:1 ",#13) BREAK
        CASE 2 PRINT("Absolute Position Mode ",#13) BREAK
        CASE 3 PRINT("Velocity Mode ",#13) BREAK
        CASE 4 PRINT("Torque Mode ",#13) BREAK
        CASE 5 PRINT("Relative Position Mode ",#13) BREAK
        CASE 6 PRINT("High Speed Indexing ",#13) BREAK
        CASE 7 PRINT("CAM Mode <Gearing> ",#13) BREAK
        CASE 8 PRINT("Variable Gearing ",#13) BREAK
        CASE 9 PRINT("Preset Moves ",#13) BREAK
    ENDS
RETURN

```

Interrupt Programming

The section describes interrupt commands that can be used in your SmartMotor programs. For more details, see Part 2: SmartMotor Command Reference on page 247.

ITR(), ITRE, ITRD, EITR(), DITR(), RETURNI

Interrupt Commands

The interrupt ITR() function can be used to configure a SmartMotor to execute a routine based on the change of a status bit. There are dozens of different bits of information available in the SmartMotor, which are held in groups of 16 status bits called Status Words. ITR() can tell the SmartMotor to execute a subroutine after the change of any one of these status bits in any Status Word. When the status bit changes, that subroutine executes at that instant from wherever the normal program happens to be. A program of some sort must be running for the interrupt routine to execute.

Interrupt subroutines end with the RETURNI command to distinguish them from ordinary subroutines. After the interrupt code execution reaches the RETURNI command, it goes back to the program exactly where it was interrupted. An interrupt subroutine must not be called directly with a GOSUB command.

The ITR() function has five parameters:

ITR(Int #, Status Word, Bit #, Bit State, Label #)	
Int #:	Can be eight, numbered 0 to 7
Status Word:	0-8,12,13,16 and 17
Bit #:	0 to 15
Bit State:	State that causes the interrupt, 0 or 1
Label #:	Subroutine label number to be executed, 0 to 999

For an interrupt to work it must be enabled at two levels:

- Individually enable an interrupt with the EITR() command with the interrupt number, 0 to 7, in the parentheses.
- Enable all interrupts with the ITRE command.

Similarly, individual interrupts can be disabled with the DITR() command, and all interrupts can be disabled with the ITRD command.

The STACK and END commands clear the tracking of subroutine nesting, and disable all interrupts. For details on these commands, see Flow Commands on page 186.

In the next program example, interrupt number zero is set to look at Status Word 3, Bit 15, which is Velocity Target Reached. When this status bit switches to 1, subroutine 20 executes, which issues an X command and stops the motor. Every time the motor reaches its target velocity, it immediately decelerates to a stop, causing it to forever accelerate and decelerate without ever spending any time at rest or at the target velocity.

NOTE: The STACK and END commands disable all interrupts.

```

EIGN (2)           'Disable left limit
EIGN (3)           'Disable right limit
ZS                 'Clear faults
VT=700000         'Set target velocity
ADT=100           'Set target accel/decel
MV                 'Set Velocity mode
ITR(0,3,15,1,20) 'Set interrupt
EITR(0)           'Enable interrupt zero
ITRE               'Enable all interrupts
G                 'Start motion
C10               'Place a label
  GOTO10          'Loop, req. for int. operation
END               'End (never reached)
C20               'Interrupt subroutine
  X               'Decelerate to stop
  TWAIT          'Hold program until motor reaches stop
  G               'Restart velocity motion
RETURNI           'Return to infinite loop

```

TMR(timer,time)*Timers*

The TMR() function controls four timers. Their states are found in the first four bits of Status Word 4. The TMR() function can be used to execute an interrupt routine after a certain period of time.

The TMR() function has two parameters where:

- timer Specifies the timer #. There are four timers: 0 to 3.
- time Specifies the time (in milliseconds) to count down to zero.

While the timer is counting down, the corresponding status bit in Status Word 4 will be one. When it reaches zero, the status bit reverts to zero. This bit change can be made to trigger a subroutine using the ITR() function.

```

EIGN (2)           'Disable left limit
EIGN (3)           'Disable right limit
ZS                'Clear faults
MP                'Set Position mode
VT=500000         'Set target velocity.
AT=300            'Set target acceleration.
DT=100            'Set target deceleration.
TMR (0,1000)      'Set Timer 0 to 1s
ITR (0,4,0,0,20) 'Set interrupt
EITR (0)          'Enable interrupt
ITRE              'Enable all interrupts
p=0               'Initialize variable p
O=0               'Set commanded and actual pos. to zero
C10               'Place a label
  IF PA>47000     'Just before 12 moves
    DITR (0)      'Disable interrupt
    TWAIT         'Wait till reaches 48000
    p=0           'Reset variable p
    PT=p          'Set target position
    G             'Start motion
    TWAIT         'Wait for move to complete
    EITR (0)      'Re-enable interrupt
    TMR (0,1000) 'Re-start timer
  ENDIF
GOTO10            'Go back to label
END               'End (never reached)

C20               'Interrupt subroutine label
  TMR (0,1000)    'Re-start timer
  p=p+4000        'Increment variable p
  PT=p           'Set target position
  G              'Start motion
RETURNI           'Return to main loop

```

Variables and Math

This chapter provides information on using variables and math functions with the SmartMotor.

Introduction	199
Variable Commands	199
EPTR=formula	199
VST(variable,number)	199
VLD(variable,number)	200
Math Expressions	200
Math Operations	200
Logical Operations	200
Integer Operations	200
Floating Point Functions	200
Math Operation Details and Examples	201
Array Variables	201
Array Variable Examples	202

Introduction

Variables are data holders that can be set and changed within the program or over one of the communication channels. Although most of the variables are 32-bit signed integers, there are also eight floating-point variables. All variables are represented by lower-case text.

Variables are stored in volatile memory, meaning that they are lost when power is removed, and they default to zero on power-up. If they need to be saved, you must store them in the EEPROM (nonvolatile memory) using the VST (Variable Store) command. For more details, see Variable Commands on page 199.

There are three sets of integer variables, each containing twenty-six, 32-bit signed integers and referenced by:

- a,b,c,...,x,y,z
- aa,bb,cc,...,xx,yy,zz
- aaa,bbb,ccc,...,xxx,yyy,zzz

There is an additional set of fifty-one, 32-bit signed integers in array form, $al[i], i=0\dots 50$.

The eight floating-point variables are also in array form and referenced by $af[i], i=0\dots 7$.

- a = # Set variable a to a numerical value
- a = formula Set variable a to value of a variable or formula

For more details on array variables, see Array Variables on page 201.

Variable Commands

These commands are used to load and store variables. For more details, see Part 2: SmartMotor Command Reference on page 247.

EPTR=formula

Set EEPROM Pointer in Bytes, 0-32767

To read or write into this memory space, it is necessary to properly locate the pointer. This is accomplished by setting EPTR equal to the offset in bytes. EEPROM locations above EPTR equal to 32767 contain important motor information and are read-only.

VST(variable,number)

Store Variables

Use the VST command to store a series of variables starting at the pointer. In the "variable" space of the command, put the name of the variable; in the "number" space, put the total number of sequential variables that need to be stored. Enter a one if just the variable specified needs to be stored. The actual sizes of the variables are recognized automatically. Do not put the VST command in a tight program loop or you will likely exceed the 1M write-cycle limit, which will damage the EEPROM.

NOTE: Keep the VST command out of tight loops to avoid exceeding the 1M write-cycle limit of the EEPROM.

VLD(variable,number)

Load Variables

Use the VLD command to load a series of variables starting at the pointer. In the "variable" space of the command, put the name of the variable; in the "number" space, put the number of sequential variables to be loaded. Enter a one if just the variable specified needs to be loaded. Again, the actual sizes of the variables are recognized automatically.

Math Expressions

Variables can be used in mathematical expressions with: math operations, logical operations and integer operations, as described in the next sections.

Math Operations

All variables can be used in mathematical expressions assuming standard hierarchical rules and using any of the mathematical operations:

+	Addition
-	Subtraction
*	Multiplication
/	Division

Logical Operations

The previous mathematical operations can be combined with these logical operations:

>	Less than
<	Greater than
==	Equal to
!=	Not equal to
<=	Less than or equal to
>=	Greater than or equal to

Integer Operations

These integer operations are also supported:

^	Raise to an integer power <=4
&	Bit wise AND (see ASCII Character Set on page 899)
	Bit wise OR (see ASCII Character Set on page 899)
!	Bit wise exclusive OR (see ASCII Character Set on page 899)
%	Modulo
SQRT (x)	Integer Square Root (x = integer, where x>=0)
ABS (x)	Integer Absolute Value (x = integer)

Floating Point Functions

These floating point functions are also supported:

FSQRT(x)	Floating Point Square Root (x = float, where x >= 0)
FABS(x)	Floating Point Absolute Value (x = float)
SIN(x)	Floating Point Sine (x = float in degrees)
COS(x)	Floating Point Cosine (x = float in degrees)
TAN(x)	Floating Point Tangent (x = float in degrees)
ASIN(x)	Floating Point Arc Sine in degrees on [-90,90], (x = float on the interval [-1,1])
ACOS(x)	Floating Point Arc Cosine in degrees on [0,180], (x = float on the interval [-1,1])
ATAN(x)	Floating Point Arc Tangent in degrees on [-90,90], (x = float)
PI	Floating Point representation of PI = 3.14159265359...

Math Operation Details and Examples

In any operation, if the input is an integer then the result remains an integer. A result is promoted to a float once the operation includes a float. Functions that require a floating-point argument implicitly promote integer arguments to float. In converting a floating-point number to an integer result, the floating-point number is truncated toward zero. Although the floating point variables and their standard binary operations conform to IEEE-754 double-precision results, the floating-point square root and trigonometric functions only produce IEEE-754 single-precision results. Here are some examples:

```

a= (b+c/d) *e      'Standard hierarchical rules apply
a=2^3              'a=8
c=123%12           'Modulo, c=3, remainder of 123/12
b= (-10<a) & (a<10) 'b=0 if "a" not in range, b=1 otherwise
x=ABS(EA)          'Set x to the abs value of pos error
r=SQRT(a)          'if a=64, r=8,; if a=63, r=7

```

Array Variables

An array variable has a numeric index component that specifies the variable a program is to access. This memory space is flexible because it can hold fifty-one 32-bit integers, or one-hundred-two 16-bit integers, or two-hundred-four 8-bit integers (all signed). The array variables use the form:

ab[i]=formula	Set variable to a signed 8-bit value where index i=0...203
aw[i]=formula	Set variable to a signed 16-bit value where index i=0...101
al[i]=formula	Set variable to a signed 32-bit value where index i=0...50

NOTE: The index i may be a number, a variable or an expression.

The same array space can be accessed with any combination of variable types and can be viewed simply as the union of the data type arrays. Keep in mind how much space each variable takes. Also, note that one type of variable can be written and another read from the same space. For example, if the first four eight bit integers are assigned as:

```

ab[0]=0
ab[1]=0
ab[2]=1
ab[3]=0

```

they would occupy the same memory space as the first single 32-bit number or the first pair of 16-bit numbers. The order is from least significant to most significant with ab[3] being the most significant.

Because of the way binary numbers work, this would make the 32-bit variable `al[0]` equal to 65,536, as well as the 16-bit variables `aw[0]` equal to 0 and `aw[1]` equal to 1.

A common use of the array variable type is to set up a buffer. In many applications, the SmartMotor is tasked with inputting data about an array of objects and to do processing on that data in the same order but not necessarily at the same time. Therefore, it may be necessary to "buffer" or "store" that data until it is time for the SmartMotor to process it.

To set up a buffer, the programmer allocates a block of memory to it, assigns one variable to an input pointer and another variable to an output pointer. Both pointers start out as zero and increment:

- Every time data goes into the buffer — the input pointer increments.
- Every time data is used — the output buffer likewise increments.
- Every time a pointer increments — it is checked for exceeding the allocated memory space and rolled back to zero in that event. It then continues to increment as data comes in.

This is a first-in, first-out or FIFO circular buffer. There should be enough memory allocated so that the input pointer never overruns the output pointer.

NOTE: Every SmartMotor has a small solid-state disk drive for long term storage of data, which is based on EEPROM technology. It can be written to and read from more than one million times.

Array Variable Examples

These are examples of different uses for array variables:

```

b=af[0]
af[0]=SIN(57.3)
af[7]=ATAN(af[6])*180/PI
af[4]=af[3]*(af[1]/af[2]-1)
af[0]=(a+b)/2+3.0
af[0]=(a+b)/2.0+3.0
af[5]=FSQRT(a)

```

```

'if af[0]=1.8, b=1; if af[0]=-1.8, b=-1
'Set float var af[0] to sine 57.3 degrees
'Set af[7] to arctan result converted to radians
'Standard hierarchical rules apply
'if a=8 and b=1, af[0]=7.0
'if a=8 and b=1, af[0]=7.5
'if a=63, af[5]=7.937253952

```

Error and Fault Handling Details

This chapter provides information on the error and fault handling functionality that has been designed into the SmartMotor.

Motion and Motor Faults	204
Overview	204
Drive Stage Indications and Faults	204
Fault Bits	204
Error Handling	205
Example Fault-Handler Code	205
PAUSE	206
RESUME	206
Limits and Fault Handling	207
Position Error Limits	207
dE/dt Limits	207
Velocity Limits	208
Hardware Limits	208
Software Limits	208
Fault Handling	209
Monitoring the SmartMotor Status	210

Motion and Motor Faults

This section provides information on motion faults and motor faults.

Overview

Status bits and LED indicators are used to keep the programmer or operator aware of present or past fault conditions of the SmartMotor. Keep these points in mind when viewing the status bits or LEDs.

- Red LEDs do not necessarily mean the motor has faulted.
- Faults do not mean the motor is "broken" or not working properly.
- A motor fault typically means a user or design limit has been reached.
- Status bits are not always fault bits.
- Status bits are used to indicate present conditions or past (historical) conditions.

For more details on LED functions, see Understanding the Status LEDs in the SmartMotor Installation and Startup Guide for your SmartMotor.

Drive Stage Indications and Faults

These are fault bits that stop motion and turn off the Drive OK bit:

- | | | | |
|----|---|--|---|
| 1. | B(0,3) Voltage Fault | Either high or low | Historical indication |
| | a. | B(6,14) High bus voltage | Voltage depends on motor model |
| | b. | B(6,13) Low bus voltage | Voltage depends on motor model |
| 2. | B(0,5) Excessive Temperature | | Historical indication $\geq 85^{\circ}\text{C}$ |
| | NOTE: The only fault bit with hysteresis, 80°C prior to clearing | | |
| 3. | B(0,6) Excessive Position Error | | Historical indication |
| | a. | EA (Actual Position Error) exceeded EL (Error Limit) | |
| | b. | EL=1000 encoder counts by default | |
| 4. | B(0,7) Velocity Limit | | Historical indication |
| | VL sets Velocity Limit and defaults to 10400 RPM for most motors | | |
| 5. | dE/dt Error Limit Rate of Change for Position Error | | |
| | Defaults to $+2^{31}$ and is in same scaled units as velocity | | |
| 6. | Travel Limits, both hardware and programmable software limits | | |

NOTE: Peak Over Current limit is NOT a fault-causing limit! It is an indication that the drive stage is working as hard as it can to keep up with demand.

For additional details, see Status Word 0: Primary Fault/Status Indicator on page 921, and Status Word 1: Index Registration and Software Travel Limits on page 922.

Fault Bits

The Z5 command typically clears all fault bits. However, issuing the Z5 command when you get a fault on the peak overcurrent bit means you will potentially mask other fault conditions and, therefore, not know why there is a fault on the peak overcurrent bit.

Mass Moment of Inertia mismatches and high acceleration/deceleration are the primary reasons for getting faults on the peak overcurrent limit. However, this does not mean you will get a continuous condition for: overcurrent, position error or overtemperature. Those are typically in the RMS range of load and are caused by general overload conditions, excessive friction or ambient temperature rise. For more details, see Moment of Inertia on page 916.

Proper load to motor sizing is crucial in preventing most of these fault conditions from occurring. Please consult the *Moog Animatics Product Catalog* for more information. Also, see Torque Curves on page 938.

Error Handling

This section describes techniques and commands that can be used for error handling in your SmartMotor program.

Example Fault-Handler Code

In many multiple-axis applications, if there is a fault in one axis, it is desirable to stop all motion in the machine. An effective way to accomplish this is to place the next example code into every motor.

When any axis experiences a drive-disable condition, interrupt routine C0 executes. The C0 routine immediately broadcasts a global Mode Torque Brake (MTB) to stop all axes. After that, the motor calling for the shutdown places its address in the user-accessible mode bits of Status Word 0. For additional details, see Status Word 0: Primary Fault/Status Indicator on page 921.

```
EIGN(W,0,12)    'Another way to disable Travel Limits
ZS              'Clear faults
ITR(0,0,0,0,0) 'Set Int 0 for: stat word 0, bit 0,
                'shift to 0, to call C0
EITR(0)        'Enable Interrupt 0
ITRE           'Global Interrupt Enable
PAUSE          'Pause to prevent "END" from disabling
                'Interrupt, no change to stack

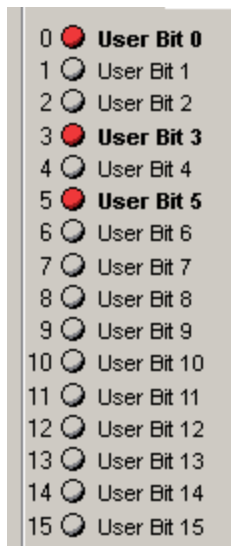
END

C0              'Fault handler
  MTB:0         'Motor will turn off with Dynamic
                'breaking, tell other motors to stop.
  US(0):0       'Set User Status Bit 0 to 1 (Status
                'Word 12 bit zero)
  US(ADDR):0    'Set User Status Bit "address" to 1
                '(Status Word 12 Bit "address")

RETURNI
```

After all motors are stopped, appropriate recovery actions can be taken.

Monitor Status Word 12 in SMI Motor View to see the results of an axis fault. For more details on Motor View, see Monitoring the SmartMotor Status on page 210.



The next sections describe related commands. For more details on these commands, see Part 2: SmartMotor Command Reference on page 247.

PAUSE

Suspend Program Execution

The PAUSE command suspends program execution until the RESUME command is received. It will not affect the present state of the Interrupt Handler. If the Interrupt Handler is enabled, it will still be enabled after a PAUSE, and its execution has no effect on the interrupt/subroutine stack.

NOTE: There is a separate stack for PAUSE that can go up to ten "resumes" deep. It allows for PAUSES over communications and within user program interrupt routines.

RESUME

Resume from a PAUSE

The RESUME command restarts program execution from the location after the PAUSE command. It is intended to be issued externally over communications and cannot be compiled within a program.

The RESUME command does not differentiate where the PAUSE came from. If you have a PAUSE in the main program and a PAUSE in an interrupt, the PAUSE that is currently active will be the one that is resumed.

Limits and Fault Handling

There are commands available for interacting with these types of limits:

- Position error limits
- Velocity limits
- Hardware limits
- Software limits

These are described in the next sections. Additionally, this section describes the FSA (fault stop action) command.

For more details, see Part 2: SmartMotor Command Reference on page 247.

Position Error Limits

These commands are used to read position error, and to set and read position error limits:

EL=formula	Set the position Error Limit.
x=EL	Assign to a variable the value set as the position Error Limit.
REL	Report the value set as the position Error Limit.
x=EA	Assign to a variable the value of the position Actual Error (current position error in real time).
REA	Report the value of the position Actual Error.

dE/dt Limits

These commands are used to set and read dE/dt limits:

x=DEA	Assign to a variable the value of the actual rate of change of the PID position error (Actual Derivative Error). This value is averaged over four consecutive PID cycles. It is in units of position error per PID cycle *65,536.
RDEA	Report the value of the PID position error (Actual Derivative Error).
DEL=formula	Set the position error rate limit (Derivative Error Limit). This is useful for detecting obstructions to the motor's path and faulting the motor sooner than the position error limit alone would. This is in the same units as the =DEA command.
x=DEL	Assign to a variable the value set as the position error rate limit (Derivative Error Limit).
RDEL	Report the value set as the position error rate limit (Derivative Error Limit).

Velocity Limits

These commands are used to set and read velocity limits:

VL=formula	Set the velocity fault limit (Velocity Limit) in revolutions per minute. When the motor exceeds this speed (traveling clockwise or counter clockwise), then the motor will fault.
x=VL	Assign to a variable the value set as the velocity fault limit (Velocity Limit) in revolutions per minute.
RVL	Report the value set as the velocity fault limit (Velocity Limit) in revolutions per minute.

Hardware Limits

These commands are used to enable the positive and negative hardware limits (external stimulus to limit motion; causes a motion fault if exceeded):

EILP	Enable Positive Limit switch on I/O port.
EILN	Enable Negative Limit switch on I/O port.

NOTE: The SmartMotor's hardware limits must be connected (and properly tied low or high, depending on the motor type) or disabled for motion to occur. For Class 5 D-style motors, which have *sinking* inputs, the connected limits must be tied low; for Class 5 M-style and Class 6 motors, which have *sourcing* inputs, the connected limits must be tied high.¹

Therefore, if your SmartMotor doesn't move when adjusting the SmartMotor Playground's slider or issuing a motion command, verify that you've either connected the limits (and properly tied them low or high, depending on your motor type) or selected both Disable Hardware Limits check boxes (located at the lower-right corner of the SmartMotor Playground screen).

Software Limits

Software limits offer distinct advantages over hardware limits connected to the limit inputs of the SmartMotor. Software limits are "virtual" (non-hardware) limit switches that can interrupt motion with a limit fault in the event the actual position of the motor strays beyond the desired region of operation. The limit fault is directionally sensitive, so it will cause a fault if motion is commanded further in the direction of a limit once that limit has been exceeded.

SLE	Software Limits Enable.
SLD	Software Limits Disable.
SLN=formula	Sets (left) Negative Limit.
SLP=formula	Sets (right) Positive Limit.
SLM(mode)	Software Limit Mode. Determines if the software limits result in a fault or not. 0 no fault, 1 causes fault when soft limit history asserted.

¹Allen-Bradley / Rockwell uses opposite meanings for the terms *sourcing* and *sinking*.

Fault Handling

When a limit is exceeded, motion is interrupted with a fault. The FSA (fault stop action) command is used in fault handling.

FSA(cause,action) Fault Stop Action.

where:

cause: the type of fault to set a mode on:

0 - All types of fault.

1 - Hardware travel limits.

2 - Reserved.

action: action to take:

0 - Default action (MTB).

1 - Servo off.

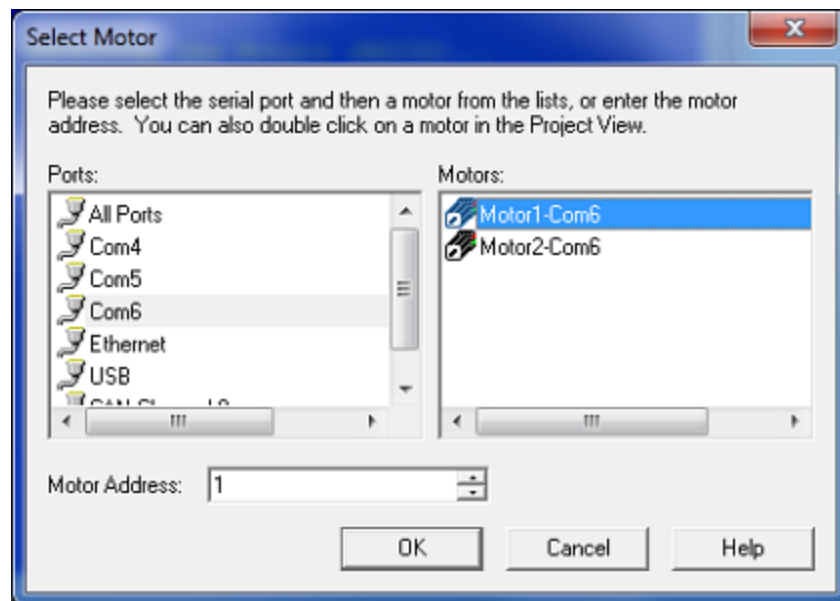
2 - X command.

Monitoring the SmartMotor Status

NOTE: In addition to the software information in this section, there is context-sensitive help available within the SMI software interface, which is accessed by pressing the F1 key or selecting Help from the SMI software main menu.

The Motor View tool is used to monitor the status of the SmartMotor. To see the status of the connected motor, select:

Tools > Motor View > *double-click the desired motor*



Selecting a Motor

After the Motor View window appears, click Poll. The status of the selected motor is updated in the window, as shown in the next figure.

NOTE: The Motor View window provides a view into the status of a SmartMotor.



Motor View (with Polling Enabled)

NOTE: The SmartMotor's hardware limits must be connected (and properly tied low or high, depending on the motor type) or disabled for motion to occur. For Class 5 D-style motors, which have *sinking* inputs, the connected limits must be tied low; for Class 5 M-style and Class 6 motors, which have *sourcing* inputs, the connected limits must be tied high.¹

Therefore, if your SmartMotor doesn't move when adjusting the SmartMotor Playground's slider or issuing a motion command, verify that you've either connected the limits (and properly tied them low or high, depending on your motor type) or selected both Disable Hardware Limits check boxes (located at the lower-right corner of the SmartMotor Playground screen).



WARNING: DO NOT disable the hardware limits if this action creates a safety hazard for personnel or equipment.

Optionally, if you see limit errors and want to move the motor without wiring the limits or disabling them in the SmartMotor Playground, you can issue terminal commands to disable the limits and reset the errors. To do this, issue the next commands in the Terminal window (be sure to use all caps and don't enter the comments to the right). For more details on using the Terminal window, see Terminal Window on page 67.

```
EIGN (2) 'Disable Left Limit
EIGN (3) 'Disable Right Limit
ZS 'Reset errors
```

¹Allen-Bradley / Rockwell uses opposite meanings for the terms *sourcing* and *sinking*.

Normally, when the motor is attached to an application that relies on proper limit operation, you would not disable them. If your motors are connected to an application that is capable of causing damage or injury, it would be essential to properly install the limits before experimenting.

System Status

This chapter provides information on using system status words and bits with the SmartMotor.

Introduction	214
Retrieving and Manipulating Status Words/Bits	214
System and Motor Status Bits	214
Reset Error Flags	217
System Status Examples	217
Timer Status Bits	218
Interrupt Status Bits	218
I/O Status	219
User Status Bits	219
Multiple Trajectory Support Status Bits	220
Cam Status Bits	221
Interpolation Status Bits	222
Motion Mode Status	222
RMODE, RMODE(arg)	222

Introduction

The SmartMotor System Status is divided among 16-bit status words. Many status bits are predefined and offer information about the state of the SmartMotor operating system or the motor itself. However, there are status words that contain user bits and have been set aside for use by the programmers and their specific applications.

NOTE: Status bits may not be cleared or reset if the condition that set it still exists (for example, the Bh bit).

Status bits can be used to cause interrupts within an application program. The state of a status bit can also be tested by IF and WHILE instructions. Therefore status bits can determine the flow or path of execution of an application program.

In addition to the information in this chapter, see Status Words - SmartMotor on page 921. Also, the Moog Animatics website contains a useful tool for working with status bits, the *SmartMotor Developer's Worksheet*, which is available at:

<https://www.animatics.com/support/downloads.knowledgebase.html>.

Retrieving and Manipulating Status Words/Bits

These commands are used in retrieving and manipulating status words and bits.

x=W(word)	Assign to a variable the value of the 16-bit status Word.
RW(word)	Report the value of the 16-bit status Word.
x=B(word,bit)	Assign to a variable the value of the status Bit, word=Word#, bit=Bit#.
RB(word,bit)	Report the value of the status Bit, word=Word#, bit=Bit#.
Z(word,bit)	Clears the status Bit, word=Word#, bit=Bit#.
x=B@	Assign to a variable the value of a status Bit through direct addressing, where @ is replaced with a lower case alpha character.
RB@	Report the value of the status Bit through direct addressing, where @ is replaced with a lower case alpha character.
Z@	Clears a status bit through direct addressing, where @ is replaced with a lower case alpha character.
ZS	Clears a defined set of status bits; its intent is to clear the faults of a motor allowing motion to continue from a G command.

For more details, see Part 2: SmartMotor Command Reference on page 247.

System and Motor Status Bits

There are many system and motor status bits available to govern the application program and motor behavior. The next sections show many of the useful, directly-addressed status bits.

Note that the next sections are not the complete list of status bits. You should also refer to Status Words - SmartMotor on page 921.

General System Directly-Addressed Status Bits

- Bk Program check sum/EEPROM failure
- Bs Syntax error occurred

General Motor Directly-Addressed Status Bits

- Bo Motor off
- Bt Trajectory in progress
- Bw Position wraparound occurred
- Bv Velocity limit reached
- Ba Over current state occurred
- Bh Excessive temperature
- Be Excessive position error

NOTE: In cases where the motor has gone beyond the EL (error limit) but the trajectory generator is still active with the previously calculated trajectory, the ZS command may not clear the Be bit. If you are unable to reset Be with the ZS command, issue an OFF command before issuing the ZS command, which clears the current commanded trajectory and allows the reset to complete.

Motor Hardware Limits Directly-Addressed Status Bits

- Bm Real time negative hardware limit, left limit
- Bp Real time positive hardware limit, right limit
- Bl Historical negative hardware limit, left limit
- Br Historical positive hardware limit, right limit

Motor Software Limits Directly-Addressed Status Bits

- Bms Real time negative hardware limit, left limit
- Bps Real time positive hardware limit, right limit
- Bls Historical negative software limit, CCW limit
- Brs Historical positive software limit, CW limit

Motor Index/Capture Directly-Addressed Status Bits

- Bi(0) Rising index/capture available on the internal motor encoder
- Bi(1) Rising index/capture report available on the external encoder
- Bj(0) Falling index/capture value available on the internal motor encoder
- Bj(1) Falling index/capture value available on the external encoder
- Bx(0) Hardware index/capture input level on the internal motor encoder
- Bx(1) Hardware index/capture input level on the external encoder

For more details, see Part 2: SmartMotor Command Reference on page 247.

Reset Error Flags

If action is taken based on some of the error flags, the flag will need to be reset in order to look out for the next occurrence, or in some cases, depending on how the code is written, prevent repeated action on the same occurrence.

Za	Reset over current state occurred
Zh	Reset excessive temperature
Ze	Reset excessive position error
Zl	Reset historical left limit occurred
Zr	Reset historical right limit occurred
Zls	Reset historical left limit occurred
Zrs	Reset historical right limit occurred
Zs	Reset syntax error occurred
Zv	Reset velocity limit occurred
Zw	Reset encoder wrap occurred
ZS	Resets all above Z status flags, and the status of Bi(#) and Bj(#)

NOTE: In cases where the motor has gone beyond the EL (error limit) but the trajectory generator is still active with the previously calculated trajectory, the ZS command may not clear the Be bit. If you are unable to reset Be with the ZS command, issue an OFF command before issuing the ZS command, which clears the current commanded trajectory and allows the reset to complete.

For more details, see Part 2: SmartMotor Command Reference on page 247.

System Status Examples

An example of where you could use a System status bit would be to replace the TWAIT command. The TWAIT command pauses program execution until motion is complete, but interrupt subroutines will still take place. To avoid a routine simply resting on the TWAIT command, a routine can be written that does much more.

This code example performs the same function as the TWAIT command:

```
WHILE Bt      'While trajectory
LOOP        'Loop back
```

As shown in the next example, the previous routine can be augmented with code that takes specific action in the event of an index signal:

```

EIGN(W,0)           'Set all I/O to be general inputs a=0
ZS                  'Clear all faults
Ai(0)               'Arm motor's capture register
MV VT=1000 ADT=10   'Set up slow velocity mode, slow accel/decel
G                   'Start motion
WHILE Bt            'While trajectory
    IF Bi(0)==0     'Check index captured of encoder
        GOSUB(1)   'Call subroutine
    ELSE
        X
    ENDIF           'End checking
LOOP                'Loop back
OFF
END                 'SUB 1: Increment a every 1 second
C1
    IF B(4,0)==0    'Check Timer 0 status
        a=a+1      'Updating a every second
        TMR(0,1000) 'Set Timer 0 counting
    ENDIF
RETURN
END

```

Timer Status Bits

Timer Status Bits are true while a timer is actively counting. Timers have resolution of 1 millisecond.

TMR(0,1000)	Sets Timer Status bit 0 true for 1 second.
x=TMR(3)	Assign to a variable the value of Timer 3.
RTMR(3)	Report the value of Timer 3.

For more details, see Part 2: SmartMotor Command Reference on page 247.

Interrupt Status Bits

Interrupt Status Bits are true if an interrupt is enabled. It is important to note the interrupts need to be configured before being enabled for proper operation. For details on configuring interrupts, refer to Interrupt Programming on page 195.

These commands directly affect the state of the interrupt status bits:

ITRE	Enables the interrupt handler, sets Interrupt Status Bit 15
ITRD	Disables the interrupt handler, clears Interrupt Status Bit 15
EITR(0)	Enables the highest priority interrupt, sets Interrupt Status Bit 0
DITR(0)	Disables the highest priority interrupt, clears Interrupt Status Bit 0

For more details, see Part 2: SmartMotor Command Reference on page 247.

I/O Status

Typically, to get an I/O port logical status, you would use the IN() instructions for zero-based addressing of the I/O Ports. As with any status of the SmartMotor, you can also retrieve the I/O port status, but not change its state, using the W() and B() status word/bit commands.

x=IN(IO)	Assign to a variable the value of the specified Input. NOTE: Other forms are available, see the command description for details.
RIN(IO)	Report the value of the specified Input (see NOTE above).
x=W(word)	Assign to a variable the value of the specified status word.
RW(word)	Report the value of the specified status word.
x=B(word,bit)	Assign to a variable the value of the specified status word and bit number.
RB(word,bit)	Report the value of the specified status word and bit number.

For more details, see Part 2: SmartMotor Command Reference on page 247.

User Status Bits

Status words 12 and 13 contain user status bits (status bits that can be set by the user). User bits allow you to keep track of events or status within an application program. Their functions are defined by the application program of the SmartMotor. User bits are addressed individually starting at 0 (zero based). Likewise, the user bits words are addressed starting at 0 (zero based).

A powerful feature of user bits is their ability to be addressed over networks such as Combitronic or CANopen. This feature allows a hosting application to cause a SmartMotor to run an interrupt routine. For details, see Interrupt Programming on page 195.

The user bits can also be addressed as words, with or without a mask, to define which bits are affected. These are examples of commands that directly affect the user bits:

US(0)	Set user bit 0
US(W,0,a)	Set first three user bits when a=7
UO(0)=a&b	Set user bit to 1 if the bit-wise operation result is odd, else set it to 0
UO(W,0)=x	Set status word 12 to the value of x
UO(W,1)=123	Set status word 13 to the value 123
UO(W,1,7)=a	Set user bits 16, 17 and 18 to the value of the lower three bits in a
UR(19)	Reset user bit 3 in second user bit status word
UR(W,0)	Reset all user bits in first user bit status word
UR(W,1,7)	Reset user bits 16, 17 and 18

NOTE: The G command also resets several system state flags.

For more details, see Part 2: SmartMotor Command Reference on page 247.

Multiple Trajectory Support Status Bits

The SmartMotor system provides the ability to have multiple trajectory generators operating at the same time. The outputs of the trajectory generators can be manipulated to affect the SmartMotor in a combination of ways, which are discussed in other section of this manual. The trajectory generator status bits help you properly control the use of the trajectory generators from an application program or over a network.

The next example exercises the trajectory status bits of Status Word 7 in a standard Class 5 SmartMotor:

```
EIGN (w, 0)
ZS
MFA (1000)      'Set up Gearing Profile
MFR (2)         'Gearing profile for Trajectory Generator (TG)2
O=0            'Establish actual position to zero
PT=0           'Set target position to zero
VT=1000        'Set target velocity
ADT=100        'Set accel/decel
MP (1)         'Position mode for Trajectory Generator (TG)1
G (2)          'Start TG 2, TG2 in Progress is ON
G (1)          'Start TG 1, PC=PT so TG 1 in progress is OFF
PT=10000
G (1)          'Until PC=PT TG1 in progress is ON
TWAIT (1)
OFF
END
```

Cam Status Bits

The Class 5 SmartMotor supports cams running in Spline and/or Linear Interpolated Position modes.

Each individual cam segment can be interpolated in one of these two modes. While the cam is being executed, the Cam Segment Mode bits can be interrogated to determine which mode is presently being used for that segment. The Cam User Bits can also be turned on and off, which is defined when each segment is written into cam memory through the CTW() command.

Cam User Bits offer a periodic signal based on the phase of a cam, and they can be programmed to come on and off within any given section of the cam. They function much like a standard Programmable Limit Switch (PLS). In a standard Class 5 SmartMotor, these bits reside in Status Word 8.

The next example exercises each Cam User Bit during the programmed cam profile.

```
EIGN (W, 0)
ZS
CTA (7, 0, 0)           'Add table into RAM al[0]-al[8]
CTW (0, 0, 1)          'Add 1st point, Cam User Bit 0 ON
CTW (1000, 4000, 1)    'Add 2nd point, Cam User Bit 0 ON
CTW (3000, 8000, 2)    'Add 3rd point, Cam User Bit 1 ON
CTW (4000, 12000, 132) 'Add 4th, Spline Mode, Cam Bit 2 ON
CTW (1000, 16000, 136) 'Add 5th, Spline Mode, Cam Bit 3 ON
CTW (-2000, 20000, 16) 'Add 6th point. Cam Bit 4 ON
CTW (0, 24000, 32)     'Add 7th point. Cam Bit 5 ON
MC                     'Select Cam Mode
SRC (2)                'Use the virtual controller encoder.
MCE (0)                'Force Linear interpolation.
MCW (0, 0)             'Use table 0 in RAM from point 0.
MFMUL=1                'Simple 1:1 ratio from virtual enc.
MFDIV=1                'Simple 1:1 ratio from virtual enc.
MFA (0) MFD (0)        'Disable virtual enc. ramp-up/ramp-
                        'down sections.
MFSLEW (24000, 1)     'Table is 6 segments *4000 encoder
                        'counts each. Specify the second
                        'argument as a 1 to force this
                        'number as the output total of the
                        'virtual controller encoder into the cam.
MFSDC (-1, 0)         'Disable virtual controller (Gearing)
                        'repeat.
G                       'Begin move.
END                    'Obligatory END
```

Interpolation Status Bits

The Class 5 SmartMotor supports Interpolated Position modes (IP modes) from data sent over a CANopen network. The same bits supported for cams also exist as a separate set of status bits when operating in IP mode. In a standard Class 5 SmartMotor, these bits reside in Status Word 8. When Moog Animatics SMNC multi-axis contouring software is used, there is built-in support for these status bits.

For details on the Moog Animatics SMNC software, see this address:

<https://www.animatics.com/support/downloads/software/smnc.html>

Motion Mode Status

The SmartMotor supports many different motion modes. Keeping them straight can present a challenge. The RMODE command provides a tool for reporting the motion mode.

RMODE, RMODE(arg)

Report Motion Mode

The RMODE command will report the current active motion mode. Insert an argument to specify move generator 1 or 2. The value returned has these meanings:

- 7 CANopen Interpolation
- 6 CANopen Homing
- 4 Torque
- 3 Velocity
- 1 Position
- 0 Null (move generator inactive)
- 2 Quadrature Follow
- 3 Step/Direction Follow
- 4 Cam
- 5 Mixed

For more details, see Part 2: SmartMotor Command Reference on page 247.

I/O Control Details

This chapter provides information on the extensive I/O control functionality that has been designed into the SmartMotor.

I/O Port Hardware	224
I/O Connections Example (Class 5 D-Style Motors)	225
I/O Voltage Protection	225
Discrete Input and Output Commands	225
Discrete Input Commands	226
Discrete Output Commands	226
Output Condition and Fault Status Commands	227
Output Condition Commands	227
Output Fault Status Reports	227
General-Use Input Configuration	228
Multiple I/O Functions Example	228
Analog Functions of I/O Ports	230
5 Volt Push-Pull I/O Analog Functions (Class 5 D-Style Motors)	230
24 Volt I/O Analog Functions (Class 5 D-Style AD1 Option Motors, Class 5 M-Style Motors)	230
24 Volt I/O Analog Functions (Class 6 M-Style Motors)	230
24 Volt I/O Analog Functions (Class 6 D-Style Motors)	231
Special Functions of I/O Ports	232
Class 5 D-Style Motors: Special Functions of I/O Ports	233
Class 5 M-Style Motors: Special Functions of I/O Ports	235
Class 6 Motors: Special Functions of I/O Ports	237
I/O Brake Output Commands	238
I²C Expansion (D-Style Motors)	239

I/O Port Hardware

The extensive and flexible I/O gives the SmartMotor the capability to control an entire machine.

Each point of SmartMotor I/O can be used or configured as a digital input or digital output. For example, on the D-style SmartMotor, there are seven points of 5V I/O located in the 15-pin D-sub connector, and an optional ten points of isolated 24V I/O located in a circular, M-12 connector. The 5V I/O is push-pull; the 24V I/O is sourcing, for machine safety reasons. Regardless of the I/O setting, the analog value can also be read. On the M-style SmartMotors, there are eleven I/O points over two connectors, as well as a drive-enable input and a not faulted signal output.

All I/O are organized into 16-bit status words starting at Status Word 16 of the controller, but within I/O commands it is word 0 (zero). The I/O ports are initially inputs at power-up; once the state is set using a discrete output command, it then controls the state of the I/O pin.

On-board I/O in any standard Class 5 motor is in the first I/O Status Word 0. The I/O can be addressed through commands, and it is zero based — the first I/O number is 0 (zero). There can be as many as 17 on-board I/O ports.

NOTE: Individual motor specifications must be reviewed to determine the number and physical nature of the I/O. The physical nature of the I/O will address the voltage levels and isolation characteristics of each I/O point.

Expanded I/O in a Class 5 motor starts at I/O Status Word 1, and the first expanded I/O number is then 16. Again, individual motor specifications determine the number and physical nature of the expanded I/O.

For all commands listed in the next sections:

- IO is the I/O bit number. This can be passed in from a variable.
- word is the status word number. This can be passed in from a variable.
- mask is a bit-wise mask used to screen out certain bits. This can be passed in from a variable.
- W (capital W letter by itself) refers to "Word", or 16 bits of information. Selects the word format of the command.

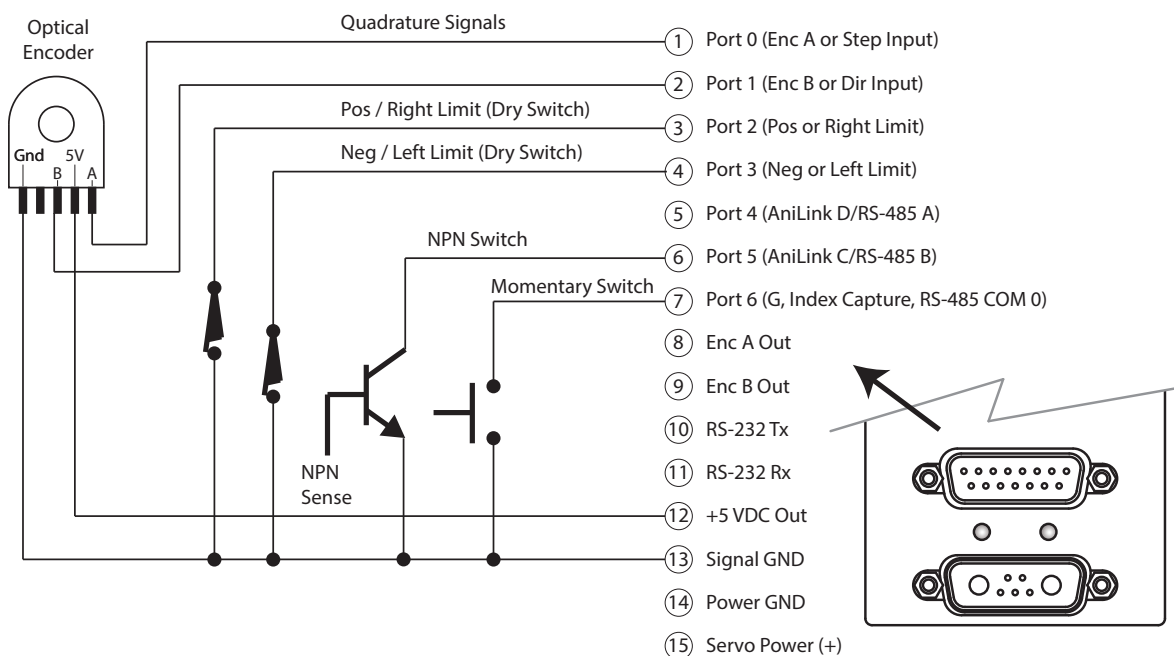
For your SmartMotor's connector specifications, refer to Connector Pinouts in the *SmartMotor Installation & Startup Guide* for your motor.

I/O Connections Example (Class 5 D-Style Motors)

Encoders, ports, switches and buttons can be connected directly to the SmartMotor's I/O pins.

The next figure provides an example of some common I/O connections to the Class 5 D-style SmartMotor's 15-pin D-sub connector.

NOTE: The next figure is an example for Class 5 D-style motors only. For additional I/O connection examples, and connector, pinout and cabling information, refer to the *SmartMotor Installation & Startup Guide* for your motor.



Example of Common I/O Signals and Connections for Class 5 D-style

NOTE: Class 5 D-style SmartMotor I/O is Sinking type (NPN); Class 5 D-style AD1, Class 5 M-style and all Class 6 SmartMotor I/O are Sourcing type (PNP).

I/O Voltage Protection

This section provides information on I/O voltage protection provided by the SmartMotor.

All SmartMotor I/O is confined to operate within specified VDC limits, and some circuitry exists to accommodate occurrences outside the operational range. For both its analog and discrete inputs, the SmartMotor protects against persistent overvoltage to 2x the rated voltage of that input (not to exceed 50 VDC), and to negative persistent voltage of 1/2 the rated voltage. This is permitted as long as those occurrences are moderate and have a short duration.

The specified VDC limits and impedance for the I/O ports differ based on the particular SmartMotor class (e.g., Class 5) and style (e.g., D-style). For details, refer to the corresponding Connector Pinout table in the *SmartMotor Installation & Startup Guide* for your motor.

Discrete Input and Output Commands

This section describes the discrete input and output commands available for the SmartMotor.

NOTE: For the 5V I/O in the Class 5 motor's D-Sub connector, the value can be 0 - 6 for I/Os 0 - 6. For the 24V I/O, the value can be 16 - 25 for the ten I/Os 16 - 25.

Discrete Input Commands

<code>x=IN(IO)</code>	Assign to a variable a value representing the state of the specified I/O bit.
<code>RIN(IO)</code>	Report the value representing the state of the specified I/O bit.
<code>x=IN(W,word)</code>	Assign to a variable a value representing the state of the specified I/O word.
<code>RIN(W,word)</code>	Report the value representing the state of the specified I/O word.
<code>x=IN(W,word[,mask])</code>	Assign to a variable a value representing the state of the specified I/O word after applying a mask.
<code>RIN(W,word[,mask])</code>	Report the value representing the state of the specified I/O word after applying a mask.

Discrete Output Commands

<code>OS(IO)</code>	Set a single output to logic 1 or ON.
<code>OS(W,word[,mask])</code>	Set multiple outputs at once, applying a bit mask first.
<code>OR(IO)</code>	Reset a single output to logic 0 or OFF.
<code>OR(W,word[,mask])</code>	Reset multiple outputs at once, applying a bit mask first.
<code>OUT(IO)=formula</code>	If the bit in formula to the right of the "=" is odd, then set I/O ON; when even or zero, turn it OFF.
<code>OUT(W,word)=formula</code>	Set the I/O group to a value to the right of the "=".
<code>OUT(W,word[,mask]) =formula</code>	Set the I/O group with mask.

For more details, see Part 2: SmartMotor Command Reference on page 247.

Output Condition and Fault Status Commands

This section describes the Output Condition (OC) and Output Fault (OF) status report commands available for the SmartMotor.

Output Condition Commands

NOTE: The output condition commands are only available for Class 5 24V I/O motors (Class 5 M-style motors or D-style motors with the AD1 option). For other motors, refer to the IN/RIN commands (with bitmask), see IN(...) on page 509.

x=OC(IO)	Assign to a variable the value of the individual output status of the specified I/O number — the value is 1 if output is on, 0 if it is off.
ROC(IO)	Report the value of the individual output status of the specified I/O number — the value is 1 if output is on, 0 if it is off.
x=OC(W,word)	Assign to a variable the value of the output status within a word.
ROC(W,word)	Report the value of the output status within a word.

NOTE: Inputs return OFF even if external condition is logic 1 in the OC() commands.

Output Fault Status Reports

NOTE: The output fault status reports are only available for Class 5 24V I/O motors (Class 5 M-style motors or D-style motors with the AD1 option).

x =OF(IO)	Assign to a variable the value of the present fault state for the specified I/O, where: 0 = no Fault , 1 = over current, 2 = possible shorted.
ROF(IO)	Report the value of the present fault state for the specified I/O.
x=OF(S,word)	Assign to a variable the value of the bit mask of present faulted I/O points. Where word is the 16-bit word number, 0 is the controller I/O Status Word 16. If the value is ever greater than zero, then the I/O fault status flag (Controller Status Word 3) is set.
ROF(S,word)	Report the value of the bit mask of present faulted I/O points.
x=OF(L,word)	Assign to a variable the value of the bit mask of latched faulted I/O points. Where word is the 16-bit word number, a read of a 16-bit word will attempt to clear the I/O words latch.
ROF(L,word)	Report the value of the bit mask of latched faulted I/O points.
x=OF(D,word)	Assign to a variable the value of an error code from the controller associated with this I/O word.
ROF(D,word)	Report the value of an error code from the controller associated with this I/O word.

For more details, see Part 2: SmartMotor Command Reference on page 247.

General-Use Input Configuration

This section describes the general-use input configuration commands available for the SmartMotor.

These general-use input configuration commands are available:

EIGN(I/O)	Sets a given I/O port to or back to an input with no function attached. In other words, to remove the function of travel limit from I/O port 2, execute the instruction EIGN(2).
EIGN(W,word)	Sets all I/O in a given I/O word back to input.
EIGN(W,word[,mask])	Sets all I/O in a given I/O word back to input if mask bit is set.

For more details, see Part 2: SmartMotor Command Reference on page 247.

Multiple I/O Functions Example

The next example shows multiple I/O functions:

```

EIGN(W,0)           'deactivate default on-board I/O functions
ab[10]=W(16)       'read the status of on-board I/O.
ab[11]=IN(W,0)    'Same as above, so ab[10]=ab[11] assuming
                  'I/O states didn't change.

a=0
WHILE a<4
    ab[a]=IN(a)    'get first 4 I/O states into ab[0]-ab[3]
    a=a+1

LOOP
a=0
WHILE a<4
    OS(a+4)       'turn ON I/O Ports 4 through 7.
    a=a+1

LOOP
a=1
OUT(W,1)=aw[0]    'set expansion I/O to value in aw[0]
OR(W,1,a)         'reset only I/O 16
END
EIGN(W,0)         'remove default on-board I/O functions
ab[10]=W(16)     'read the status of on-board I/O via
                  'controllers status word.
ab[11]=IN(W,0)   'same as above, so ab[10]=ab[11] assuming
                  'I/O states didn't change.

a=0
WHILE a<4
    ab[a]=IN(a)   'get first 4 I/O states into ab[0] through ab[3]
    a=a+1

LOOP
a=0
WHILE a<4
    OS(a+4)       'turn ON I/O ports 4 through 7.
    a=a+1

LOOP
a=1
OUT(W,1)=aw[0]   'set expansion I/O to value in aw[0]
OR(W,1,a)        'reset only I/O 16
END

```

Analog Functions of I/O Ports

An I/O port's analog value can be monitored with the commands described in this section. The 24V I/O of a SmartMotor offers more flexibility than the 5V I/O, as shown below. All scaled readings are in millivolts.

The analog reads can help diagnose wiring issues external to the SmartMotor. For example, while Ports 4 and 5 are being used as RS-485, the signal bias could be monitored; if a 5V I/O pin is being driven as an output, the analog reading can help find a short.

These commands are used to access the analog functions of the 5V and 24V I/O:

5 Volt Push-Pull I/O Analog Functions (Class 5 D-Style Motors)

INA(A,IO)	Raw analog read: 10-bit res. 0-32736=0-5 VDC
INA(V1,IO)	Scaled voltage reading in millivolts, where 3456 would be 3.456 VDC. 0-5000=0-5 VDC

NOTE: For the 5V I/O in the Class 5 motor's D-Sub connector, the value can be 0 - 6 for I/Os 0 - 6. For the 24V I/O, the value can be 16 - 25 for the ten I/Os 16 - 25.

24 Volt I/O Analog Functions (Class 5 D-Style AD1 Option Motors, Class 5 M-Style Motors)

NOTE: I/O for the -AD1 option starts at 16; I/O for the M-style starts at 0.

INA(A,IO)	Raw analog read: 10-Bit res. 0-19000=0-24 VDC (Produces a raw value of -19000 at 24 VDC, -14000 at 18 VDC)
INA(V,IO)	Scaled read 0-24000, where 24000 is 24.0 Volts, 18000 is 18.0 Volts, etc.
INA(V1,IO)	Scaled read 0-5100, where 550 is 0.55 Volts
INA(V2,IO)	Scaled read 0-610, where 60 is 0.06 Volts
INA(S,x)	Sourcing voltage for the I/O port (when output pin), where x is 16-25 for the Class 5 D-style, use 0 for the M-style.
INA(T,x)	I/O chip temperature, where x is 16-25 for the Class 5 D-style, use 0 for the M-style.

NOTE: With the 24V I/O, the V1 and V2 settings focus the 10 bits of resolution on the finer voltage spans of 5V and 0.6V, respectively.



CAUTION: For Class 5 D-style AD1 option motors, at no time should the voltage to any input exceed the level on the I/O power input (Pin 11). Doing so could cause immediate damage to the expanded I/O hardware.

24 Volt I/O Analog Functions (Class 6 M-Style Motors)

NOTE: I/O for the Class 6 M-style starts at 0.

INA(A,IO)	Raw analog read: 10-Bit res. 0-32736 (Produces the max. value of 32736 nominally at 18 VDC; note that 24 VDC still reads 32736)
-----------	--

INA(V,IO)	Scaled read 0-18000, where 15500 is 15.5 Volts (Produces a max. millivolt value of 18000 at 18 VDC; note that 24 VDC still reads 18000, where 24 VDC is still a safe value, it has simply saturated the scale range)
INA(V1,IO)	Not supported on Class 6 M-Style motors
INA(V2,IO)	Not supported on Class 6 M-Style motors
INA(S,x)	Not supported on Class 6 M-Style motors
INA(T,x)	Not supported on Class 6 M-Style motors

24 Volt I/O Analog Functions (Class 6 D-Style Motors)

NOTE: I/O for the Class 6 D-Style starts at 0.

INA(A,0)	Raw analog read: 10-Bit res. 0-32736 (Produces the max. value of 32736 nominally at 10.67 VDC; note that 24 VDC still reads 32736)
INA(A,1)	
INA(A,11)	Nominal input voltage: 4-20 mA; nominal reported value: 4000-20000, 0-21483 min/max
INA(V,IO)	Scaled read 0-10000, where 10000 is 10 Volts (Produces a max. millivolt value of 10670 at 10.67 VDC; note that 24 VDC still reads 10670, where 24 VDC is still a safe value, it has simply saturated the scale range)
INA(V1,IO)	Not supported on Class 6 D-Style motors
INA(V2,IO)	Not supported on Class 6 D-Style motors
INA(S,x)	Not supported on Class 6 D-Style motors
INA(T,x)	Not supported on Class 6 D-Style motors

For more details, see Part 2: SmartMotor Command Reference on page 247.

Special Functions of I/O Ports

This chapter provides information on the special functions of I/O ports for Class 5 and Class 6 motors.

Class 5 D-Style Motors: Special Functions of I/O Ports	233
Class 5 M-Style Motors: Special Functions of I/O Ports	235
Class 6 Motors: Special Functions of I/O Ports	237

Class 5 D-Style Motors: Special Functions of I/O Ports

The on-board I/O ports provide these special functions:

Ports 0 and 1	External encoder (quadrature or step/direction) inputs, brake output
Ports 2 and 3	Travel limit inputs, brake output
Ports 4 and 5	Communications, brake output
Port 6	Go function, capture input, brake output

NOTE: The brake output function can be pointed to any one of the on-board I/O or expanded I/O ports.

I/O Ports 0 and 1 – External Encoder Function Commands

Ports 0 and 1 can be wired to an external encoder. Below are the supporting configuration commands:

NOTE: For proper counting, the commands OS(), OR() and OUT() should be avoided for ports 0 and 1.

MFO	Set encoder counter to zero; put it in Quadrature Count mode
MSO	Set encoder counter to zero; put it in Step and Direction Count mode (default count mode)

I/O Ports 2 and 3 – Travel Limit Inputs

Ports 2 and 3 are defaulted to travel limit inputs. They can be changed to a general-purpose I/O points by using the EIGN() commands and then returned to the travel limit function with these commands:

EILN	Set I/O 3 as negative overtravel limit
EILP	Set I/O 2 as positive overtravel limit

I/O Ports 4 and 5 – Communications

Ports 4 and 5 can be configured as a second communications channel. Channel 0 is the main communications channel. Ports 4 and 5 are associated with commands for communications across channel 1. For more details on communications, see Communication Details on page 96.

Modbus and DMX protocols are additional communications options that can be used instead of RS4. For more details on these optional protocols, see the documentation for the specified protocol. The communication protocol is specified with the OCHN command type parameter. For details, see OCHN (type,channel,parity,bit rate,stop bits,data bits,mode,timeout) on page 104.



CAUTION: The secondary RS-485 port is non-isolated and not properly biased by the two internal 5k ohm pull-up resistors. Therefore, it is suitable for communication with a bar code reader or light curtain, but it cannot be used to cascade motors because of the heavy biasing and ground bounce resulting from variable shaft loading.

These are examples of the supporting configuration commands:

NOTE: These functions are not supported on the Class 5 IP-65 rated motors with on-board 24V I/O.

OCHN(IIC,1,N,200000,1,8,D)	Set I/O 4 and 5 for I ² C mode
CCHN(IIC,1)	Close the I ² C channel
OCHN(RS4,1,N,38400,1,8,D)	Set I/O 4 and 5 to RS-485 mode
CCHN(RS4,1)	Close the RS-485 channel

I/O Port 6 - Go Command, Encoder Index Capture Input

These commands are used to issue a G (Go) command, and capture input from internal or external encoders:

EISM(6)	Issues G when specified I/O goes low
EIRE	Index/registration input capture of the external encoder count (default setting)
EIRI	Index/registration input capture of the internal motor encoder count

The next table provides a matrix of the index capture functions for the D-style motors.

Capture	Internal index input ^a	External index input ^b	Disable index input ^c
Internal encoder	EIRE command (default at startup)	EIRI command	Not supported
External encoder	Not supported	EIRE command (default at startup)	EIRI command
a) From the internal encoder b) I/O port 6, single-ended, 5 volts c) Disables capture on indicated encoder			

For additional information on index capture, see Index Capture Commands on page 130. For more details, see Part 2: SmartMotor Command Reference on page 247.

Class 5 M-Style Motors: Special Functions of I/O Ports

The on-board I/O ports provide these special functions:

COM Port pins 4, 5, 6, and 8	A-quad-B or Step-and-Direction counting modes
Ports 2 and 3	Travel limit inputs, brake output
Port 5	Capture input, brake output
Port 6	Go function, brake output
Drive enable input	Dedicated input to enable drive
No fault output	Dedicated output to indicate when no faults are preventing motion

NOTE: The brake output function can be pointed to any one of the on-board I/O or expanded I/O ports.

COM Port Pins 4, 5, 6, and 8 - A-quad-B or Step-and-Direction Modes

These pins are for encoder A+/-, B+/- inputs or outputs. They can also be used for step and direction mode. Below are the supporting configuration commands:

MF0	Set encoder counter to zero; put it in Quadrature Count mode
MS0	Set encoder counter to zero; put it in Step-and-Direction Count mode (default count mode)

I/O Ports 2 and 3 - Travel Limit Inputs

Ports 2 and 3 are defaulted to travel limit inputs. They can be changed to general-purpose I/O points by using the EIGN() commands and then returned to the travel limit function with these commands:

EILN	Set I/O 3 as negative overtravel limit
EILP	Set I/O 2 as positive overtravel limit

I/O Port 5 - Encoder Index Capture Input

These commands are used to capture input from internal or external encoders:

EIRE	Index/registration input capture of the external encoder count (default setting)
EIRI	Index/registration input capture of the internal motor encoder count

The next table provides a matrix of the index capture functions for the M-style motors.

Capture	Internal index input ^a	External index input ^b	Disable index input ^c
Internal encoder	EIRE command (default at startup)	EIRI command	Not supported
External encoder	Not supported	EIRE command (default at startup)	EIRI command
a) From the internal encoder b) I/O port 5, single-ended, 24 volts c) Disables capture on indicated encoder			

For additional information on index capture, see Index Capture Commands on page 130. For more details, see Part 2: SmartMotor Command Reference on page 247.

I/O Port 6 - Go Command

This command is used to issue a G (Go) command:

EISM(6) Issues G when specified I/O goes low

Class 6 Motors: Special Functions of I/O Ports

The on-board I/O ports provide these special functions:

For Class 6 M-Style Motors:

For this list, COM Port = 8-pin connector; I/O Port = 12-pin connector.

COM Port pins 4, 5, 6, and 8	A-quad-B or Step-and-Direction counting modes
I/O Ports 2 and 3	Travel limit inputs
I/O Port 4 and 5	Capture input for external and internal encoder
I/O Port 6	Go function
I/O Port 7	Drive enable input (dedicated input to enable drive)
I/O Port 8	Brake output
I/O Port 9	Not Fault output - dedicated output to indicate when no faults are preventing motion

NOTE: The brake output function can be pointed to any one of the on-board I/O or expanded I/O ports.

For Class 6 D-Style Motors:

These are on the HD26 (26-pin) connector:

I/O Pins 13, 14, 15 and 16	A-quad-B or Step-and-Direction counting modes
I/O Ports 2 and 3	Travel limit inputs or GP
I/O Port 4 and 5	GP or capture input for external and internal encoder
I/O Port 6	GP, Go command or homing input
I/O Port 7	Drive enable input (dedicated input to enable drive)
I/O Port 8	GP or brake line output
I/O Port 9	GP or Not Fault output (indicates when no faults are preventing motion)

NOTE: The brake output function can be pointed to any one of the on-board I/O or expanded I/O ports.

A-quad-B or Step-and-Direction Modes

NOTE: See the above listings for the Class 6 M-style and Class 6 D-style pins supporting this functionality.

These pins are for encoder A+/-, B+/- inputs or outputs (output is only available for Class 6 M-style, see ENCD(in_out) on page 437). They can also be used for step and direction mode. Below are the supporting configuration commands:

MF0	Set encoder counter to zero; put it in Quadrature Count mode
MS0	Set encoder counter to zero; put it in Step-and-Direction Count mode (default count mode)

I/O Ports 2 and 3 - Travel Limit Inputs

Ports 2 and 3 are defaulted to travel limit inputs. They can be changed to general-purpose I/O points by using the EIGN() commands and then returned to the travel limit function with these commands:

EILN	Set I/O 3 as negative overtravel limit
EILP	Set I/O 2 as positive overtravel limit

I/O Port 4 and 5 - Encoder Index Capture Input

These commands are used to capture input from internal or external encoders:

EIRE	Index/registration input capture of the external encoder count (default setting)
EIRI	Index/registration input capture of the internal motor encoder count

The next table provides a matrix of the index capture functions for the M-style motors.

Capture	Internal index input ^a	External index input ^b	Disable index input ^c
Internal encoder	EIRE command (default at startup)	EIRI command	Not supported
External encoder	Not supported	EIRE command (default at startup)	EIRI command
a) From the internal encoder b) I/O port 5, single-ended, 24 volts c) Disables capture on indicated encoder			

For additional information on index capture, see Index Capture Commands on page 130. For more details, see Part 2: SmartMotor Command Reference on page 247.

I/O Port 6 - Go Command

This command is used to issue a G (Go) command:

EISM(6)	Issues G when specified I/O goes low
---------	--------------------------------------

I/O Brake Output Commands

The brake output function can be configured to any I/O port including the expanded I/O ports where IO is the bit number.

These commands are used to configure the brake output:

EOBK(IO)	Configure a given output to control an external brake
EOBK(-1)	Remove the brake function from the I/O port

For more details, see Part 2: SmartMotor Command Reference on page 247.

I²C Expansion (D-Style Motors)

I/O ports 4 and 5 can perform as an I²C port. I²C is an "Inter-IC-Communication" scheme that is simple and powerful. There are dozens of low-cost I²C devices on the market that message over I²C and deliver many resources. I²C chips include: I/O expanders, analog input and output, nonvolatile memory, temperature sensors, etc. I²C provides a low cost means of expanding the functionality of a SmartMotor. For more details on I²C, see I²C Communications (Class 5 D-Style Motors) on page 118.

Tuning and PID Control

This chapter provides information on the PID control functionality that has been designed into the SmartMotor.

Introduction	241
Tuning and PID Control on the DS2020 Combitronic System	241
Understanding the PID Control	241
Tuning the PID Control	242
Using F	243
Setting KP	243
Setting KD	243
Setting KI and KL	244
Setting EL=formula	244
Other PID Tuning Parameters	244
KG=formula	245
KV=formula	245
KA=formula	245
Current Limit Control	246
AMPS=formula	246

Introduction

The SmartMotor includes a brushless servomotor with powerful rare-earth magnets and a stator (the outside, stationary part), which is a densely-wound, multi-slotted electromagnet. Controlling the position of a brushless servo's rotor with only electromagnetism is like pulling a sled with a rubber band—accurate control would seem impossible.

The parameters that make it all work are found in the PID (Proportional, Integral, Derivative) control section. These are the three fundamental coefficients to a mathematical algorithm that intelligently recalculates and delivers the power needed by the motor. The input to the PID control is the instantaneous desired position minus the actual position, whether at rest or part of an ongoing trajectory. This difference is called the position error.

NOTE: The PID control is off when operating in Torque mode.

In the Class 5 SmartMotor, the PID update rate defaults to 125 microseconds (8,000 times per second). Optionally it may be decreased or increased to a maximum of 62.5 microseconds. The faster 62.5 microsecond update rate allows for smoother high-speed operation and faster acceleration/deceleration correction under varying load conditions.

Tuning and PID Control on the DS2020 Combitronic System

The DS2020 Combitronic system, with its separate drive and larger motors, uses a somewhat different tuning approach than the fully integrated SmartMotor products. For example, it does not offer the KL, KG or KA parameters, but it does include the KP, KD, KI and KV parameters, which can be set directly through those commands. Other tuning-related commands, like F and AMPS, are also supported. For information on those commands, see the next sections and the individual command description pages in Part 2 of this guide.

The DS2020 Combitronic system internally uses three regulators:

- PI for position loop
- PI for velocity loop
- PI for current loop

The first two are tuned as a single regulator by KP, KI, KD and KV; the last one is automatically set on the basis of motor resistance and inductance.

Additionally, the DS2020 Combitronic system provides a set of seven filters (0-6) for refining the performance of the system. These are accessed through the SMI software. Filter 0 has a default setting of Low Pass Filter (LPF) at 400 Hz; whereas, the other filters (1-6) are disabled by default. For more details on setting the control loop filters, see the Commissioning chapter in the *Moog Animatics DS2020 Combitronic™ Installation and Startup Guide*.

NOTE: The tuning settings must be adjusted by qualified personnel; incorrect settings can cause the system to become unstable. In that case, the X and S commands will not work (because they use control loops to be executed), and only an OFF command will effectively disable the drive (and engage the brake if available and configured).

Understanding the PID Control

The Proportional parameter (KP) of the PID control creates a simple spring constant. The further the shaft is rotated away from its target position, the more power is delivered to return it. With this as the only parameter, the motor shaft would respond just as the end of a spring would if it was grabbed and twisted. If the spring is twisted and let go, it will vibrate wildly. This sort of vibration is hazardous to

most mechanisms. In this scenario, a shock absorber is added to dampen the vibrations, which is the equivalent of what the Derivative parameter (KD) does.

For example, when you sit on the fender of a car, it dips down because of the additional weight based on the constant of the car's spring. It gives you no indication if the shock absorbers are good or bad. However, if you jump up and down on the bumper, you would quickly see if the shock absorbers are working or not. That's because they are not activated by position but rather by speed.

The Derivative parameter steals power away as a function of the rate of change of the overall PID control output. The parameter gets its name from the fact that the derivative of position is speed. Electronically stealing power based on the magnitude of the motor shaft's vibration has the same effect as putting a shock absorber in the system.

NOTE: While the Derivative parameter usually acts to dampen instability, this is not the true definition of the term. Therefore, it is also possible to cause instability by setting the Derivative parameter too high.

Even with the Proportional and Derivative parameters working properly, a situation created by "dead weight" can cause the servo to leave its target. If a constant torque is applied to the end of the shaft, the shaft complies until the deflection causes the Proportional parameter to rise to the equivalent torque. Because there is no speed, the Derivative parameter has no effect. As long as the torque is there, the motor's shaft position will be off target.

That's where the Integral parameter (KI) comes in. The Integral parameter mounts an opposing force that is a function of time. As time passes and there is a deflection present, the Integral parameter adds a little force to bring it back on target with each PID cycle. There is also a separate Integral Limit parameter (KL) (not available for the DS2020 Combitronic system), which limits the Integral parameter's scope of what it can do and help prevent overreaction.

Each of these parameters has its own scaling factor to tailor the overall performance of the PID control to the specific load conditions of any one particular application. The scaling factors are:

KP	Proportional coefficient
KI	Integral coefficient
KD	Derivative coefficient
KL	Integral limit (not available for the DS2020 Combitronic system)

Tuning the PID Control

The task of tuning the PID control is made difficult by the fact that the parameters are so interdependent. A change in one can shift the optimal settings of the others. The SMI software has a Tuner tool, which is an automated utility for optimizing the settings. For details, refer to the Tuner on page 85. Even if you use the Tuner tool, you should still read through this section to understand how to tune a servo and the interaction between the tuning parameters.

When tuning the motor, it is useful to have the Motor View tool running, which will monitor various bits of information that describe the motor's performance. For details on the Motor View tool, see Motor View on page 72.

NOTE: In most cases, it is unnecessary to tune a SmartMotor. They are factory tuned, and stable in virtually any application.

These are the tuner parameters and their descriptions:

KP=formula	Set KP, proportional coefficient
KI=formula	Set KI, integral coefficient

KD=formula	Set KD, derivative coefficient
KS=formula	Set KS, velocity filter option for KD (value is 0, 1, 2, or 3; larger number is longer filter time) (not available for the DS2020 Combitronic system)
KL=formula	Set KL, integral coefficient limit (not available for the DS2020 Combitronic system)
F	Update PID control

Using F

The F (update PID filter) command is used to update changes to any of the tuning parameter settings. Keep in mind that the new parameter settings do not take effect until the F command is issued. For example:

```

KP=100      'Initialize KP to some value
F           'Load into present PID filter

```

Setting KP

The main objective in tuning a servo is to get the KP (proportional coefficient) value as high as possible, while maintaining stability. The higher the KP value, the stiffer the system and the more "under control" it is. Also, when initially setting KP, it is a good idea to start with KI equal to zero (for details, see Setting KI and KL on page 244).

To begin, use the RKP (report KP) command to view the current setting and then increase it by 10% to 20%. For example:

```

RKP        3000    'Report current KP value
KP=3500     'Increase KP value
F           'Load new value into PID filter

```

Each time KP is raised, try to physically destabilize the system by bumping it, twisting it or using a looping program that invokes abrupt motions. As long as the motor always settles to a quiet rest, keep raising KP.

NOTE: If the SMI Tuning Utility is being used, it will employ a step function and graphically show the reaction.

Setting KD

As soon as the SmartMotor starts to find it difficult to maintain stability, find the appropriate value for KD (derivative compensation).

To do this, use the RKD (report KD command) to view the current value. Then move KD up and down until a value is found that gives the quickest stability. Remember to use the F command to update the new value.

Note that if KD is too high, there will be a grinding sound — it is not really grinding, but it is a sign to go the other way. A properly-tuned motor is not only stable but reasonably quiet. The level of noise immunity in the KD term is controlled by KS (velocity filter option for KD).

NOTE: Although the DS2020 Combitronic system doesn't offer a KS term, it does provide a set of seven filters for refining the performance of the system. These are accessed through the SMI software. For details on setting the control loop filters, see the Commissioning chapter in the *Moog Animatics DS2020 Combitronic™ Installation and Startup Guide*.

The derivative term KD requires estimating the derivative of the position error. The simplest method is a backward difference, $KS=0$, which is no velocity filtering and can result in excessive noise. The choices of $KS=1, 2$ and 3 provide increasing levels of noise immunity (velocity filtering) at the expense of increasing latency. Because higher latency typically results in lower achievable PID loop gains, choose the best compromise between smoothness and tracking performance. The default setting is $KS=1$.

After optimizing KD, it may be possible to raise KP a bit more. Keep going back and forth between KP and KD until you've maximized the stiffness of the system. After that, it's time to take a look at KI.

Setting KI and KL

Typically, KI (integral coefficient) is used to compensate for friction; without it, the SmartMotor will never exactly reach the target. Begin with KI equal to zero and KL equal to 1000. Move the motor off target and start increasing KI and KL. Keep KL at least ten times greater than KI during this phase. Use the RKI (report KI) and RKL (report KL) commands to view the values; use the F command to update changes to the values.

NOTE: Although the DS2020 Combitronic system doesn't offer a KL term, it does provide a set of seven filters for refining the performance of the system. These are accessed through the SMI software. For details on setting the control loop filters, see the Commissioning chapter in the *Moog Animatics DS2020 Combitronic™ Installation and Startup Guide*.

Continue to increase KI until the motor always reaches its target. When that happens, add about 30% to KI and start bringing down KL until it prevents the KI term from closing the position precisely on target. After that point is reached, increase KL by about 30%. The integral term needs to be strong enough to overcome friction. However, the limit needs to be set so that a surge of power will not be delivered if the mechanism were to jam or reach one of the physical (hard stop) limits.

Setting EL=formula

Set Maximum Position Error

The difference between where the motor shaft is supposed to be and where it is actually positioned is called the "position error". The magnitude and sign of the error are delivered to the motor in the form of torque after it is put through the PID control. As the error increases, the motor becomes more uncontrolled. Therefore, it is useful to put a limit on the allowable error, which will turn the motor off.

The EL command serves that purpose. It defaults to 1,000, but it can be set from 0 to 262,143. You can view the current value with the REL (report EL) command.

For more details, see Part 2: SmartMotor Command Reference on page 247.

Other PID Tuning Parameters

There are additional parameters that can be used to reduce the position error of a dynamic application. Most of the forces that aggravate a PID loop through the execution of a motion trajectory are unpredictable. However, there are some that can be predicted and eliminated.

NOTE: Although the DS2020 Combitronic system doesn't offer a KG or KA term, it does provide a set of seven filters for refining the performance of the system. These are accessed through the SMI software. For details on setting the control loop filters, see the Commissioning chapter in the *Moog Animatics DS2020 Combitronic™ Installation and Startup Guide*.

KG=formula***Set KG, Gravitational Offset (not available for the DS2020 Combitronic system)***

The simplest force to eliminate is gravity. When power is off, if motion would occur due to gravity, a constant offset can be incorporated into the PID control to balance the system. The KG (gravitational offset) provides this offset. The value for KG can range from -16777216 to 16777215.

To set KG, use the RKG (report KG) command to view the current value, and then make changes to KG. Remember to use the F command to update the new value. Continue adjusting KG until the load equally favors upward and downward motion.

KV=formula***Set KV, Velocity Feed Forward***

Another predictable cause of position error is the natural latency of the PID loop itself. At higher speeds, because the calculation takes a finite amount of time, the result is somewhat delayed — the higher the speed, the more the actual motor position will slightly lag the trajectory-calculated position. This can be programmed out with the KV (velocity feed forward) parameter. KV can range from zero to 65535; typical values range in the low hundreds.

To tune KV, use the RKV command to view the current value, and then make changes to KV while running the motor at a constant speed if the application will allow. Remember to use the F command to update the new value. Continue increasing the value of KV until the error is reduced to near zero and stays there. The error can be seen in real time by activating the Motor View window in the SMI software. For details, see Motor View on page 72.

Position:	297634
Pos. Error:	-23
Velocity:	-256566
Mode:	Position
None	
None	

Position Error Value (in Motor View Window)

In the DS2020 Combitronic system, the velocity feed-forward path directly feeds the velocity control system with the speed value computed by the trajectory generator. A very effective approach for improved position tracking is to use the default value KV=1000 (that corresponds to unity feed-forward gain). In this way, the position control system acts only to adjust position in case of external disturbances.

KA=formula***Set KA, Acceleration Feed Forward (not available for the DS2020 Combitronic system)***

If the SmartMotor is accelerating a mass, it will be exerting a force during that acceleration (force = mass X acceleration), which disappears immediately on reaching the cruising speed. This momentary torque during acceleration is also predictable, and its effects can be programmed out with the KA (acceleration feed forward) parameter. KA can range from zero to 65535.

It is a little more difficult to tune KA, especially with hardware attached. The objective is to arrive at a value that will close the position error during the acceleration and deceleration phases. It is better to tune KA with KI set to zero because KI will address this constant force in another way. It is best to have KA address 100% of the forces due to acceleration, and use the KI term to adjust for friction.

The PID update rate of the SmartMotor can be slowed down with the values:

PID1	Set highest PID update rate, 16 kHz
PID2	(Default) Divide highest PID update rate by 2, 8 kHz
PID4	Divide highest PID update rate by 4, 4 kHz
PID8	Divide highest PID update rate by 8, 2 kHz

NOTE: A reduction in the PID rate can result in an increase in the SmartMotor application program execution rate.

The trajectory and PID control calculations occur within the SmartMotor at the "sample rate" selected by the PIDn command. Although 16 kHz (PID1) is available, 8 kHz (PID2, the default) provides a reasonable compromise between optimum control and the SmartMotor application program execution rate. The program execution rate can be increased by reducing the PID rate using PID4 and PID8 in applications where the lower PID sample rate still results in satisfactory control.

If the PID sample rate is lowered, remember that it is the basis for velocity values, acceleration values and the PID coefficients. If the rate is cut in half, expect to do this to keep all else the same:

- Double velocity
- Increase acceleration by a factor of four

NOTE: If proper care is taken to keep the PID filter stable, the PID# command can be issued on the fly.

Current Limit Control

In some applications, if the motor is misapplied at full power, the attached mechanism could be damaged. Therefore, it can be useful to reduce the maximum amount of current available, which limits the torque the motor can produce.

AMPS=formula

Set Current Limit, 0 to 1023

The AMPS (PWM limit) command is used to set the current limit. Use the AMPS command with a number, variable or formula within the range of 0 to 1023, where the value 1023 corresponds to the maximum commanded torque to the motor. Current is controlled by limiting the maximum PWM duty cycle, which will reduce the maximum speed of the motor as well.

NOTE: The AMPS command has no effect in Torque mode.

For more details, see Part 2: SmartMotor Command Reference on page 247.

Part 2: SmartMotor Command Reference

Part 2 of this guide provides the reference pages for the SmartMotor command set. The commands are listed in alphabetical order. In addition:

- A quick-reference command list sorted alphabetically is available at the end of this manual. For details, see *Commands Listed Alphabetically* on page 946.
- A quick-reference command list sorted by function is available at the end of this manual. For details, see *Commands Listed by Function* on page 954.

NOTE: In the command syntax, when optional bracketed arguments are shown, the comma within the brackets is only used with the optional argument. For example, the comma is used with the optional "m/s" argument in the command `MFSLEW(distance[,m/s])`.

Each command description includes these items:

- **Summary Table**

A table at the beginning of each command page provides a summary list of information about the command. It includes these categories: Application, Description, Execution, Conditional To, Limitations, Read/Report, Write, Language Access, Units, Range Of Values, Typical values, Default Value, Firmware Version, and Combitronic Support.

NOTE: If an item does not apply to the particular command, it is marked with N/A.

- **Detailed Description**

This section provides details about the command. Notes, Cautions and Warnings are used to highlight any critical information that must be adhered to for proper use of the command. For more details on Notes, Cautions and Warnings, see *Safety Information* on page 31.

- **Example Code**

This section provides some example code to show the use of the command. In some cases, the example may be a "snippet" (one or a few lines); in other cases, the example may be a complete program.

NOTE: The programs and code samples in this manual are provided for example purposes only. It is the user's responsibility to decide if a particular code sample or program applies to the application being developed and to adjust the values to fit that application.

In addition, note that:

- Code examples can be copied and pasted into the SMI program editor.
 - When copying from a PDF file, the pasted code will have line indents removed. However, the code will still work properly.
 - All programs must include an END statement. For details, see *END* on page 439.
- **Related Commands**

This section lists commands that are functionally related to the current command.

NOTE: A superscript "R" character preceding the command indicates there is a corresponding "report" version of that command.

(Single Space Character) Single Space Delimiter and String Terminator

APPLICATION:	Program execution and flow control
DESCRIPTION:	Single spaces placed between a series of user variables or commands
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	Serial communications channel data
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

A single space character may be placed between a series of user commands in a single ASCII string as delimiter. If it is sent from a PLC or PC, the same space character can be used as a string terminating character.

NOTE: When sending commands through the serial port from a PC, PLC or other controller, a space character can be used as both a delimiter and a string terminator. It can be used equally and interchangeably with a carriage return as a string terminator.

EXAMPLE: (as delimiter and null terminator in PRINT command)

```
PRINT("a=1 b=2 ")
```

```
'Note space after b=2 as null terminator.'
```

equivalent:

```
PRINT("a=1 b=2", #13)
```

```
'Note carriage return as null terminator.'
```

RELATED COMMANDS:

N/A



a...z

aa...zz

aaa...zzz

32-Bit Variables

APPLICATION:	Variables
DESCRIPTION:	Signed 32-bit user variables
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	Ra...Rz Raa...Rzz Raaa...Rzzz
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	Signed 32-bit integer
RANGE OF VALUES:	-2147483648 to 2147483647
TYPICAL VALUES:	-2147483648 to 2147483647
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	a:3=1 or Ra:3 or a=a:3 aa:3=1 or Raa:3 or a=aa:3 aaa:3=1 or Raaa:3 or a=aaa:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The SmartMotor™ has three groups of predefined user variables:

- The first group consists of the variables a through z
- The second group consists of the variables aa through zz
- The third group consists of the variables aaa through zzz

They are general-purpose, read/write, 32-bit, signed integer variables that can be reported and used on either side of an equal sign in an equation.



CAUTION: These variables are stored in dynamic RAM, which means their values are lost when power is lost.

The value of any variable a through z is reported with the R, PRINT() or PRINT1() functions.

NOTE: These examples and descriptions use the single-character variables. However, you can substitute the double- or triple-character variables, if desired.

EXAMPLE:

```
g=123      'Assign the value of 123 to "g".
Rg        'Report the value of g to the primary serial port.
PRINT ("g=",g,#13)    'Print to the primary serial port.
PRINT1 ("g=",g,#13)  'Print to the secondary serial port.
END
```

Program output is:

```
123
g=123
```

These variables are 32-bit signed integers, so they are limited to whole numbers from -2147483648 to 2147483647. Math operations that result in digits after the decimal point are truncated toward zero. Therefore, the value 2.9 becomes 2, and the value -2.9 becomes -2.

If you assign or perform an operation that normally results in a value outside this range, the Bs bit indicates an overflow and the operation aborts before assigning the value to the left of the equal sign.

For modulo behavior, the operation must be promoted to a float and use the modulo operator "%".

EXAMPLE:

```
c=123      'Sets initial value of c to 123.
Rc        'Reports 123; initial value of c.
RBs      'Reports 0; no error stored.
b=70000
c=b*b     'This will overflow.
Rc        'Reports 123; the value of c is unchanged.
RBs      'Reports 1; error is indicated.
Zs       'Clears error bit.
RBs      'Reports 0; no error stored.
a=-2147483648 'Used as modulo range.
c=((b*1.0)*b)%a 'Equation that promotes to a float internally
           'and modulo divides.
Rc        'Reports 605032704; this is what is expected if
           'the original value 'wrapped' on 32-bit boundaries.
RBs      'Reports 0; this does not cause an error.
```

Program output is:

```
123
0
123
1
0
605032704
0
```

It is also possible to use these variables in certain array index operations:

EXAMPLE:

```
a=10
Raw[a]
Raw[a+1]
```

These are other restrictions:

- If $a+b$ exceeds 32 signed bits, the operation $c=a+b$ will abort, and an error flag is set.
- If $a-b$ exceeds 32 signed bits, the operation $c=a-b$ will abort, and an error flag is set.
- If $a*b$ exceeds 32 signed bits, the operation $c=a*b$ will abort, and an error flag is set.

The system flag, Bs, is set. Note that many different types of command errors will also set the Bs bit. The RERRC command can be used to retrieve the last command error. For a math overflow, that is error code 23. For details on the RERRC command, see ERRRC on page 451.

If one of these variables is used with a variable of another type, it will be appropriately converted (the variable will be "type cast").

For example, assigning the variable $aw[27]=yy$ directly stores the 16 least-significant bits of yy to $aw[27]$. The sign bit of yy is not considered, the sign is determined based on bit 15 of yy . The higher bits of variable yy are ignored.

Similarly, if the left-hand variable is an 8-bit one, such as $ab[167]$, only the lowest 8 bits are preserved. The sign is determined by bit 7 of the value on the right-side of the equals sign.

Conversely, if the left-hand value is a 32-bit variable and the right-hand side contains 16-bit variables, the 16-bit variables will be "upgraded" to 32 bits. The sign is preserved when casting to a longer format. For example, in the equation $cc=ab[4]-aw[7]$, both $ab[4]$ and $aw[7]$ are converted into 32-bit numbers before the subtraction occurs.

In the SmartMotor language, all user variables are written as lowercase letters, while functions and commands have at least one uppercase character. The term "a" is a general-purpose variable, while "A" is the acceleration function. As previously described, any user variable can be assigned a value through an equation.

EXAMPLE:

```
c=123           'Assign the value of 123 to "c".
d=345           'Assign the value of 345 to "d".
e=-599          'Assign the value of -599 to "e".
f=346           'Assign the value of 346 to "f".
g=678678        'Assign the value of 678678 to "g".
```

All user variables are initialized to the value 0 at power up or on execution of the Z system-reset command. Other than by direct assignment, this is the only way the SmartMotor sets all of the user variables to 0. Issuing a RUN command does *not* perform this automatic initialization. For this reason, it is better to test a program, whether it is auto-execution or not, by power cycling the SmartMotor or issuing the Z system-reset command.

NOTE: To understand the relationship between user assigned letter variables a-z, aa-zz and aaa-zzz, and variable arrays $ab[]$, $al[]$ and $aw[]$, see Array Variable Memory Map on page 897. The arrays and the letter variables do not overlap in the Class 5 motor.

RELATED COMMANDS:

R $ab[index]=formula$ *Array Byte [index] (see page 252)*

R $al[index]=formula$ *Array Long [index] (see page 278)*

R $aw[index]=formula$ *Array Word [index] (see page 294)*

**ab[index]=formula**
Array Byte [index]

APPLICATION:	Variables
DESCRIPTION:	User signed 8-bit variables
EXECUTION:	Immediate
CONDITIONAL TO:	Index values range 0 to 203
LIMITATIONS:	An expression used as an index within the [] brackets is limited to no more than one operator (two values). No Combitronic requests or functions with parenthesis are supported within the [] brackets. This data space is shared with Cam motion (MC) if a RAM table location is selected. However, the aw command must not be used to access this space during that time.
READ/REPORT:	Rab[index]
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	Signed 8-bit number
RANGE OF VALUES:	-128 to 127
TYPICAL VALUES:	-128 to 127
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	ab[0]:3=1234, a=ab[0]:3, Rab[0]:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The SmartMotor™ has 8, 16 and 32-bit arrays. The 8-bit array takes the form of the variables ab[index]. These are general-purpose, 8-bit, signed-integer variables that can be reported, used on either side of an equation, and mixed in an expression with variables other than 8-bit. Like all user variables, they are always lowercase, and are automatically initialized to zero at power up or reset.

The syntax of the 8-bit array is ab[index], which stands for "array byte", and accepts an index value between 0 and 203. This index can be specified explicitly or through another variable. For example, ab[4] refers to the fifth element in the 8-bit array, while ab[n] refers to an element of the array where the variable "n" must be between 0 and 203.

The value of any array variable is reported with the R, PRINT() or PRINT1() functions.

EXAMPLE:

```

ab[47]=20  'Assign the value of 20 to ab[47]
Rab[47]    'Report the value of ab[47] to the primary serial port
PRINT ("ab[47]=",ab[47],#13)      'Print to the primary serial port
PRINT1 ("ab[47]=",ab[47],#13)    'Print to the secondary serial port
END

```

Program output is:

```

20
ab[47]=20

```

The $ab[]$ array is classified as read/write, meaning that it can be assigned a value or can be assigned to some other variable or function. In other words, these variables can be left-hand or right-hand values.

EXAMPLE:

```
ab[24]=ab[43]+ab[7]
```

The above is a valid equation that combines the contents of $ab[43]$ with $ab[7]$ and sends the total into $ab[24]$. As signed 8-bit variables, they are limited to whole numbers ranging from -128 and 127. Math operations that result in digits after the decimal point are truncated toward zero. Therefore, a value of 2.9 becomes 2, and a value of -2.9 becomes -2.

If you assign or perform an operation that would normally result in a value outside of this range, the variable will "wrap" or take on the corresponding modulo. For example, $127+1 = -128$; the result wrapped around to the negative extreme.

These are other restrictions:

- If $ab[1]+a$ exceeds 32 signed bits, the operation $c=ab[1]+a$ will abort and an error flag is set.
- If $a-ab[1]$ exceeds 32 signed bits, the operation $c=a-ab[1]$ will abort and an error flag is set.
- If $a*ab[1]$ exceeds 32 signed bits, the operation $c=a*ab[1]$ will abort and an error flag is set.

The system flag, Bs, is set. Note that many different types of command errors will also set the Bs bit. The RERRC command can be used to retrieve the last command error. For a math overflow, that is error code 23. For details on the RERRC command, see ERRC on page 451.

If one of these variables is used with a variable of another type, it will be appropriately converted (the variable will be "type cast").

If the left-hand variable is an 8-bit one like $ab[167]$, only the lowest 8 bits are preserved. The sign is determined by bit 7 of the value on the right-side of the equals sign.

Conversely, if the left-hand value is a 32-bit variable and the right-hand side contains 8-bit variables, the 8-bit variables will be "upgraded" to 32-bits. The sign is preserved when casting to a longer format. In the equation $cc=ab[4]-aw[7]$, both $ab[4]$ and $aw[7]$ are converted into 32-bit numbers before the subtraction occurs.

In the SmartMotor language, all user variables are written as lowercase letters, while functions and commands have at least one uppercase character. The term "a" is a general-purpose variable, while "A" is the acceleration function. As previously described, any user variable can be assigned a value through an equation.

All user variables are initialized to the value 0 at power up or on execution of the Z system-reset command. Other than by direct assignment, this is the only way the SmartMotor sets all of the user variables to 0. Issuing a RUN command does *not* perform this automatic initialization. For this reason, it

is better to test a program, whether it is auto-execution or not, by power cycling the SmartMotor or issuing the Z system-reset command.

NOTE: To understand the relationship between user assigned letter variables a-z, aa-zz and aaa-zzz, and variable arrays *ab[]*, *al[]* and *aw[]*, see Array Variable Memory Map on page 897. The arrays and the letter variables do not overlap in the Class 5 motor.

RELATED COMMANDS:

R *a...z* 32-Bit Variables (see page 249)

R *aa...zz* 32-Bit Variables (see page 249)

R *aaa...zzz* 32-Bit Variables (see page 249)

R *af[index]=formula* Array Float [index] (see page 267)

R *al[index]=formula* Array Long [index] (see page 278)

R *aw[index]=formula* Array Word [index] (see page 294)

VLD(variable,number) Variable Load (see page 820)

VST(variable,number) Variable Save (see page 824)

ABS(value) Absolute Value of ()

APPLICATION:	Math function
DESCRIPTION:	Gets the <i>absolute integer</i> value of the specified variable or number
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RABS(value)
WRITE:	N/A
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	Input: -2147483648 to 2147483647 Output: 0 to 2147483647
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The ABS command gets (reads) the *absolute integer* value of the specified variable or number. For example:

```
x=ABS(value)
```

sets the variable x to the absolute integer value of the variable or number specified in (value).

The ABS command cannot have math arguments and cannot be a variable or value from another motor. For example, x=ABS(PA) is allowed, but x=ABS(PA:3) is not allowed.

There is a special case when using this function—the input value of -2147483648 will output 2147483647. The positive value 2147483648 cannot be represented in a 32-bit value. If the user finds this special case unacceptable, then the user must first test the input value for this special case and provide an alternative action.

EXAMPLE:

```
a=ABS (-5)           'Set variable = ABS(-5)
PRINT (a, #13)      'Print value of variable a
RABS (-5)           'Report ABS(-5)
END
```

Program output is:

```
5
5
```

RELATED COMMANDS:

R FABS(value) *Floating-Point Absolute Value of ()* (see page 463)



APPLICATION:	Motion control
DESCRIPTION:	Get (reads) the commanded acceleration
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RAC
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	(encoder counts / (sample ²)) * 65536
RANGE OF VALUES:	-2147483648 to 2147483647
TYPICAL VALUES:	-1000 to 1000
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	RAC:3, x=AC:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

NOTE: This command is affected by the SCALEA command. For details, see SCALEA(m,d) on page 724. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

The AC command gets (reads) the commanded acceleration:

- =AC
Reads the real-time commanded acceleration from trajectory generator 1 (MV or MP modes only)

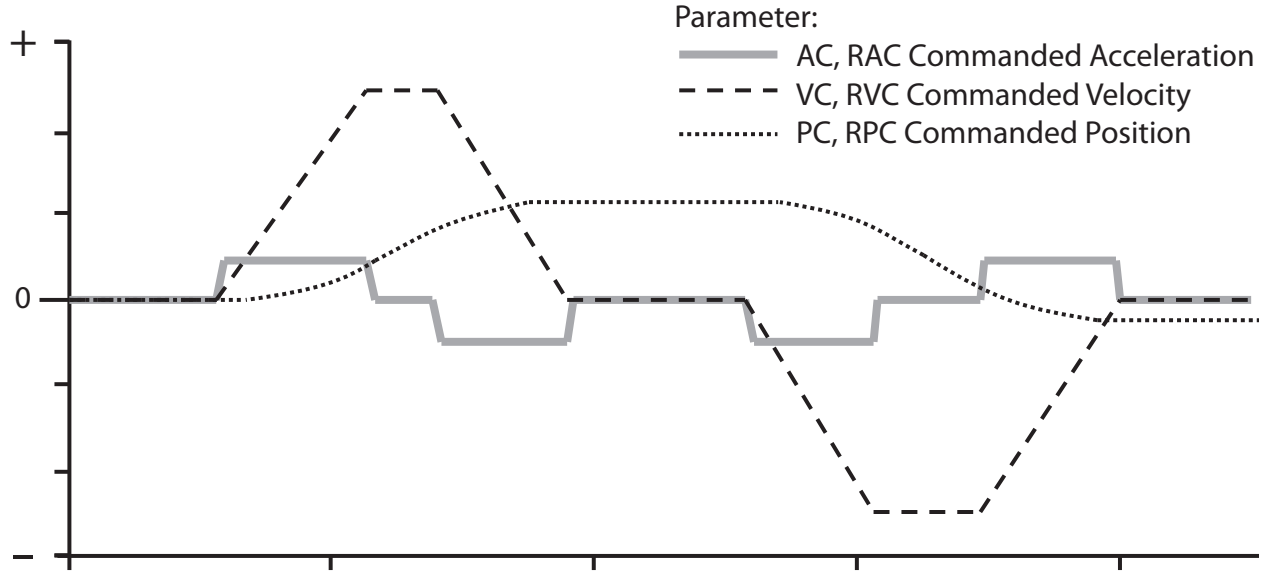
When a velocity or position profile move is commanded, the velocity is ramped up and down according to the settings of ADT=, AT=, or DT=. At any instant, the calculated acceleration or deceleration of the motion profile can be reported. The sign (positive or negative) of this reported acceleration depends on the direction of travel and the command type (begin motion or end motion).

The next table provides example values that illustrate how the sign of AC is reported. It assumes MV (velocity mode).

VT	Command	Direction	Reported Sign of AC
100000	G (start motion)	Positive (increasing position)	Positive
100000	X (end motion)	Positive (increasing position)	Negative
-100000	G (start motion)	Negative (decreasing position)	Negative

VT	Command	Direction	Reported Sign of AC
-100000	X (end motion)	Negative (decreasing position)	Positive

Also, refer to the next figure, which shows the sign (positive or negative) of the reported commanded-acceleration value compared to command velocity and command position.



Reported Sign of AC (Commanded Acceleration) Compared to VC and PC

Equations for Real-World Units:

Encoder resolution and sample rate can vary. Therefore, the general equations shown in the next table can be used to convert the value of AC to various units of acceleration. These equations force floating-point calculations to avoid overflow and maintain resolution. They can be placed in a user program, or they can be precalculated if the values of SAMP and RES are known (SAMP and RES can be reported from the terminal using the RSAMP and RRES commands, respectively). SAMP can change if the PID command is used. The value of RES can differ between motor models.

Output	Equation
Radians/(Sec ²)	=AC*PI*2*(((SAMP*1.0)*SAMP)/65536.0)/RES)
Encoder Counts/(Sec ²)	=AC*(((SAMP*1.0)*SAMP)/65536.0)
Rev/(Sec ²)	=AC*(((SAMP*1.0)*SAMP)/65536.0)/RES)
RPM/Sec	=AC*60.0*(((SAMP*1.0)*SAMP)/65536.0)/RES)
RPM/Min	=AC*3600.0*(((SAMP*1.0)*SAMP)/65536.0)/RES)

EXAMPLE:

```
ADT=10
VT=100000
MV
G
WAIT=10 'Wait to make sure move has started
WHILE AC>0
LOOP
PRINT("Acceleration Complete",#13)
```

RELATED COMMANDS:

R AT=formula *Acceleration Target (see page 286)*
ADT=formula *Acceleration/Deceleration Target (see page 263)*
R DT=formula *Deceleration Target (see page 396)*

ACOS(value)

Arccosine

APPLICATION:	Math function
DESCRIPTION:	Gets the arccosine of the input value
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RACOS(value)
WRITE:	N/A
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Degrees output
RANGE OF VALUES:	Input (floating-point): -1.0 to 1.0 Output in degrees: 180.0 to 0.0 (floating-point)
TYPICAL VALUES:	Input (floating-point): -1.0 to 1.0 Output in degrees: 180.0 to 0.0 (floating-point)
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

ACOS takes an input and returns a floating-point arccosine in degrees:

```
af[1]=ACOS(arg)
```

where *arg* may be an integer (e.g., *a* or *aw*[0]) or floating-point variable (e.g., *af*[0]). Integer or floating-point constants may also be used (e.g., 23 or 23.7, respectively).

This command cannot have within the parenthesis: math operators, other parenthetical functions, or a Combitronic request from another motor. For example, *x*=FABS(PA) is allowed, but *x*=FABS(PA:3) is not allowed.

The result of this function is a floating-point type. If used in an equation, the operations in the equation that are processed after this function are automatically promoted to a float. This is dependent on the mathematical order of operations in the equation. As with other equations (e.g., *x*=*a*+*b*), the variable to the left of "=" may be an integer variable to accept the result. However, the value will be truncated to fit to that integer type. For example, the assignment "*aw*[0]=" will drop any fractional amount and truncate the result to the range -32768 to 32767 (*aw*[0]=100.5 will report as 100, and *aw*[0]=40000.0 will report as -25536).

Although the floating-point variables and their standard binary operations conform to IEEE-754 double precision, the floating-point square root and trigonometric functions only produce IEEE-754 single-precision results. For more details, see Variables and Math on page 198.

EXAMPLE:

```
af[0]=ACOS(.5)      'Set array variable = ACOS(.5)
Raf[0]              'Report variable af[0]
RACOS(.5)           'Report ACOS(.5)
af[1]=0.4332
af[0]=ACOS(af[1])  'Variables may be put in the parenthesis
Raf[0]              'Output in degrees
END
```

Program output is:

```
60.000000000
60.000000000
64.329193115
```

RELATED COMMANDS:

- R ASIN(value) *Arcsine (see page 284)*
- R ATAN(value) *Arctangent (see page 289)*
- R COS(value) *Cosine (see page 372)*
- R SIN(value) *Sine (see page 738)*
- R TAN(value) *Tangent (see page 775)*

ADDR=formula Address (for RS-232 and RS-485)

APPLICATION:	Communications control; supports the DS2020 Combitronic system
DESCRIPTION:	Motor address
EXECUTION:	N/A
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	PRINT(ADDR), <variable>=ADDR RADDR
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	Address
RANGE OF VALUES:	0 to 120 DS2020 Combitronic system: 1 to 120
TYPICAL VALUES:	1 to 120
DEFAULT VALUE:	0 on power-up and until an address is assigned to the motor
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The SmartMotor™ is designed to be used as much in multiple-axis systems as in single-axis systems. For that reason, each SmartMotor can be uniquely addressed through the ADDR command. Used within a program, ADDR permits an identical program stored in different motors to differentiate between motors and provide individual runtime controls. For example, ADDR=5 sets the motor's address to 5.

ADDR is a read/write function, so it can also be used to access the address of the current SmartMotor. For example, to read the motor address, use this ADDR command:

```
var=ADDR
```

where var is any variable. Then you can use the PRINT(var) command to print the motor's serial address to the Terminal window.

To set the motor's serial address to the CAN address, use this ADDR command:

```
<SerialMotorNumber>ADDR=CADDR
```

For example, 3ADDR=CADDR sets the motor 3 serial address to its CAN address; 0ADDR=CADDR would globally set every serial motor's address to its CAN address.

NOTE: SmartMotor commands like 0CADDR=... or 0ADDR=... with a leading number really send a corresponding address byte (i.e., "0", which is hex 80 or decimal 128). This can be seen by viewing the serial data with the Serial Data Analyzer ("sniffer") tool, which is available on the SMI software View menu.

The ADDR command also allows you to retrieve a value over the Combitronic network. See the second example section for details.

EXAMPLE:

```

SWITCH ADDR
  CASE 1          'Motors 1,2 and 3 "GO"
  CASE 2
  CASE 3 G
    BREAK
  CASE 4 S          'Motor 4 "STOP"
ENDS              'Start motion (or stop)

```

EXAMPLE:

The ADDR command allows you to retrieve a value over the Combitronic network, as shown in these examples.

For example: ADDR=x:3

where (assuming the motor processing the command is motor 1):

1. Motor 1 parses the line of text: ADDR=x:3
2. Motor 1 asks motor 3 for motor 3's x variable.
3. Motor 1 completes any other operations on the right side of the equation.
4. Motor 1 assigns motor 1's ADDR with the value from the right side of the equation.

For example: b:2=a+a:3+a:4

where (assuming the motor processing the command is motor 1):

1. Motor 1 parses the text: b:2=a+a:3+a:4
2. Motor 1 retrieves variable a from motors 3 and 4.
3. Motor 1 completes the right side of the equation, including in this case, its own variable a.
4. Motor 1 sends result of the right side of the equation into motor 2's variable b.

EXAMPLE: (Code sets CAN address to motor address and resets all motors. Motors should be addressed on serial RS-232 chain first.)

NOTE: Issue these commands at serial port (SMI Terminal window) only. The "0" in front of these commands will not be recognized by a user program.

```

0CADDR=ADDR      'Set if not same as motor address
0Z               'Reset all motors to enable CAN address

```

RELATED COMMANDS:

R CADDR=formula *CAN Address (see page 355)*

SADDR# *Set Address (see page 720)*



APPLICATION:	Motion control; supports the DS2020 Combitronic system
DESCRIPTION:	Sets the buffered acceleration target (AT) /deceleration target (DT) at the same time
EXECUTION:	Buffered until a G command is issued or an X command is issued
CONDITIONAL TO:	MP, MV, G, X, PID n (sample rate), encoder resolution
LIMITATIONS:	Must not be negative; effective value is rounded down to next even number
READ/REPORT:	There is no direct report for ADT; use RAT and RDT
WRITE:	Write only
LANGUAGE ACCESS:	Assignment
UNITS:	(encoder counts / (sample ²)) * 65536 DS2020 Combitronic system: user increments / sec ² , see FD=e-expression on page 461
RANGE OF VALUES:	0 to 2147483647 DS2020 Combitronic system: 0 to 4294967295
TYPICAL VALUES:	2 to 5000 DS2020 Combitronic system: depends on FD
DEFAULT VALUE:	See: AT, DT DS2020 Combitronic system: 0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	ADT:3=1234 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

NOTE: This command is affected by the SCALEA command. For details, see SCALEA(m,d) on page 724. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

The ADT command sets the AT and DT parameters of the motion profile. Those values are individually accessible with the AT=, =AT, DT=, and =AT commands. ADT is provided as a convenience when those values do not need to be different from each other. See the respective commands for specific details about how they apply to a motion profile.

The ADT= command cannot be reported back directly, because it simply passes the value to AT= and DT=.

A useful Scale Factor Multiplier code example, which also illustrates the use of af[], SAMP and RES, is shown in RES on page 702 and SAMP on page 722.

EXAMPLE: (Shows use of ADT, PT and VT)

```
MP           'Set mode position
ADT=5000    'Set target accel/decel
PT=20000   'Set absolute position
VT=10000   'Set velocity
G           'Start motion
END        'End program
```

EXAMPLE: (Routine homes motor against a hard stop)

```
MDS           'Using Sine mode commutation
KP=3200      'Increase stiffness from default
KD=10200     'Increase damping from default
F            'Activate new tuning parameters
AMPS=100     'Lower current limit to 10%
VT=-10000    'Set maximum velocity
ADT=100      'Set maximum accel/decel
MV           'Set Velocity mode
G            'Start motion
WHILE EA>-100 'Loop while position error is small
LOOP         'Loop back to WHILE
O=-100       'While pressed, declare home offset
S            'Abruptly stop trajectory
MP           'Switch to Position mode
VT=20000     'Set higher maximum velocity
PT=0         'Set target position to be home
G            'Start motion
TWAIT       'Wait for motion to complete
AMPS=1023    'Restore current limit to maximum
END          'End program
```

RELATED COMMANDS:

R AT=formula *Acceleration Target (see page 286)*

R DT=formula *Deceleration Target (see page 396)*

R EL=formula *Error Limit (see page 426)*

R FD=expression *Resolution to Set Units of Position/Velocity/Acceleration (see page 461)*

G *Start Motion (G0) (see page 473)*

R HM_ADT=formula *Homing Accel/Decel Target (see page 491)*

MP *Mode Position (see page 613)*

MV *Mode Velocity (see page 624)*

PID# *Proportional-Integral-Differential Filter Rate (see page 654)*

R PRT=formula *Position, Relative Target (see page 683)*

R PT=formula *Position, (Absolute) Target (see page 690)*

R VT=formula *Velocity Target (see page 828)*

X *Decelerate to Stop (see page 844)*

ADTS=formula Acceleration/Deceleration Target, Synchronized

APPLICATION:	Motion control
DESCRIPTION:	Sets the synchronized (path) acceleration/deceleration target
EXECUTION:	Must be set before issuing PTS or PRTS; will not be effective after that point
CONDITIONAL TO:	PID n
LIMITATIONS:	Must not be negative; 0 is not valid; effective value is rounded up to next even number
READ/REPORT:	None
WRITE:	Write only
LANGUAGE ACCESS:	Assignment
UNITS:	(encoder counts / (sample ²)) * 65536 in 2D or 3D space
RANGE OF VALUES:	0 to 2147483647
TYPICAL VALUES:	10 to 500
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M), not ver. 5.32.x.x; 6.x (D/M), M requires EIP option
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

NOTE: This command requires a Combitronic-supported motor. Although this command does not support Combitronic syntax, it does use Combitronic communication to pass information between other motors.

NOTE: This command is affected by the SCALEA command. For details, see SCALEA(m,d) on page 724. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

The ADTS value determines the synchronized (path) acceleration/deceleration that will be used by subsequent position or velocity moves to calculate the required trajectory. Changing ADTS during a move will not alter the current trajectory unless a new G command is issued.

Acceleration is pre-scaled by 65536 and may range from 2 to 2147483647. A value of 0 is not valid. Due to internal calculations, odd values for this command are rounded up to an even value.

A useful Scale Factor Multiplier code example, which also illustrates the use of af[], SAMP and RES, is shown in RES on page 702 and SAMP on page 722.

EXAMPLE: (2-axis synchronized absolute move to position x:y for motors 1 and 2)

This sample code may be executed by any motor sharing the same CAN bus network with the motors being commanded to move.

NOTE: Ensure no motor drive faults exist prior to calling this subroutine.

```

. . .
C20
OFF:0 MP:0 PRT:0=0 G TWAIT      'Initialize to stationary in position mode
PT:1=PC:1  PT:2=PC:2          'Set target and commanded positions equal
WAIT=50
VTS=v                      'Set target path velocity
ADTS=a                      'Set target path accel/decel
PTS (x;1, y;2)              'Use Position Target Synchronized moves
PTSS (a;3)                  'Supplemental synchronized target
IF PTSD!=0                  'Prevent 0-length (divide by zero) move
    GS                      'Go Synchronized
    TSWAIT                  'Wait until path move time is complete
ENDIF
RETURN

```

For additional examples, see A Note About PTS and PRTS on page 181.

EXAMPLE: (3-axis synchronized relative move to position x:y:z for motors 1, 2 and 3)

This sample code may be executed by any motor sharing the same CAN bus network with the motors being commanded to move.

NOTE: Ensure no motor drive faults exist prior to calling this subroutine.

```

. . .
C20
OFF:0 MP:0 PRT:0=0 G TWAIT      'Initialize to stationary in position mode
PT:1=PC:1  PT:2=PC:2  PT:3=PC:3 'Set target and commanded positions equal
WAIT=50
VTS=v                      'Set target path velocity
ADTS=a                      'Set target path accel/decel
PRTS (x;1, y;2, z;3)        'Use Position Target Synchronized moves
PRTSS (a;4)                 'Supplemental synchronized relative target
IF PTSD!=0                  'Prevent 0-length (divide by zero) move
    GS                      'Go Synchronized
    TSWAIT                  'Wait until path move time is complete
ENDIF
RETURN

```

For additional examples, see A Note About PTS and PRTS on page 181.

RELATED COMMANDS:

PID# *Proportional-Integral-Differential Filter Rate (see page 654)*

PRTS(...) *Position, Relative Target, Synchronized (see page 685)*

PRTSS(...) *Position, Relative Target, Synchronized, Supplemental (see page 688)*

PTS(...) *Position Target, Synchronized (see page 692)*

PTSS(...) *Position Target, Synchronized Supplemental (see page 696)*

VTS=formula *Velocity Target, Synchronized Move (see page 831)*



af[index]=formula

Array Float [index]

APPLICATION:	Variables
DESCRIPTION:	Floating-point array variables
EXECUTION:	Immediate
CONDITIONAL TO:	Index values 0 to 7
LIMITATIONS:	Index limited to two values with single operator; index values may be a constant or variables a-zzz
READ/REPORT:	Raf[index]
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	Signed floating-point number
RANGE OF VALUES:	Very large: more than $\pm 10^{300a,b}$
TYPICAL VALUES:	Depends on required precision (approximate): To maintain integer precision: -4503599627370496 to +4503599627370496 ^b To maintain 3 decimal digits: -4503599627370.496 to +4503599627370.496 ^b
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	af[0]:3=1234.0, a=af[0]:3, Raf[0]:3 where ":3" is the motor address — use the actual address or a variable

^a Use of full range is not recommended because precision of the number is lost. At this extreme range, precision is worse than a whole number.

^b Entering and reporting values through user commands is limited to the range: -2147483648.0 to 2147483647.0

DETAILED DESCRIPTION:

The floating-point array variables meet IEEE-754 specifications. Therefore, they are true floating-point variables — the location of the decimal point can vary with the exponent from very small (approximately 1×10^{-300}) to very large (approximately 1×10^{300}). The user is encouraged to make use of floating-point variables in equations where a typical 32-bit integer might overflow (see examples).

While floating-point numbers seem to have nearly limitless range from large to small, the user must exercise caution. The precision of the number is limited to approximately 15 decimal digits. The number is stored in a base-2 format, including the fractional part. This can result in subtle issues with precision and representation of base-10 values. This is generally avoided by only displaying nine rounded digits

after the decimal place. This allows for small numbers (less than 100000) to show all nine decimal digits.

When using floating-point values or floating-point variables in equations, there are some rules to be aware of. The equation parser will not perform floating-point operations until at least one of the input values is a floating-point value. After a floating-point value is seen, subsequent operations (in the order of operations) in that equation will proceed as floating-point operations. Note that it is a common mistake to divide two integers and expect a floating-point result — at least one of those input values must be entered as a floating-point value or first multiplied by 1.0, to invoke floating-point operations.

Floating-point variables will remember the full range possible but can only display a limited range. The display is limited to nine digits after the decimal point, and -2147483648 to 2147483647 before the decimal point.

When assigning a floating-point variable to an integer, the integer cannot accept a value outside of the range: -2147483648 to 2147483647. This will result in a command error (Code 23: Math Overflow) and the integer will remain at its previous value.

A floating-point number can be assigned to an integer, but it will round toward 0. For example, the value 1.9 becomes 1; the value -1.9 becomes -1.

Basic math operations (+, -, *, /) are performed at 64-bit precision. However, the trigonometric functions are only calculated with 32-bit precision.

For more details, see Variables and Math on page 198.

EXAMPLE:

```
af[0]=123.5      'Assign the value of 123.5 to af[0].
Raf[0]
af[0]=1/10      'Perform the integer divide and store result to af[0].
Raf[0]
af[0]=1.0/10    'Perform the floating-point divide and store
                'result to af[0].
Raf[0]
af[0]=1300000.0*2700000 'The product would overflow a 32-bit integer,
                        'but af[0] can handle it.
a=af[0]/1000000 'Reduce the size of the value to something an
                'integer can handle.
Ra
a=(1300000.0*2700000)/1000000 'This has the same result; the equation
                              'still performs a floating-point divide
                              'and stores the integer result.
Ra
END
```

Program output is:

```
123.500000000
0.000000000
0.100000000
3510000
3510000
```

RELATED COMMANDS:

- R a...z *32-Bit Variables (see page 249)*
- R aa...zz *32-Bit Variables (see page 249)*
- R aaa...zzz *32-Bit Variables (see page 249)*
- R ab[index]=formula *Array Byte [index] (see page 252)*
- R al[index]=formula *Array Long [index] (see page 278)*
- R aw[index]=formula *Array Word [index] (see page 294)*
- R DFS(value) *Dump Float, Single (see page 393)*
- R LFS(value) *Load Float Single (see page 547)*
- VLD(variable,number) *Variable Load (see page 820)*
- VST(variable,number) *Variable Save (see page 824)*



APPLICATION:	I/O control; supports the DS2020 Combitronic system over RS-232 only
DESCRIPTION:	Arms the index register for capturing the rising edge of the encoder
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	Input: 0 or 1
TYPICAL VALUES:	Input: 0 or 1
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	Ai(0):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The Ai(enc) command arms the index register for capturing the rising edge of the encoder. To capture the falling edge, see Aj(enc) on page 274.

NOTE: The rising and falling edges are stored to different index registers.

For the DS2020 Combitronic system, the command is used for a procedure to find the position that corresponds to the physical zero position of the feedback sensor.

The enc parameter specifies the encoder to be captured; it does not specify the source of the index signal.

- Ai(0) specifies internal encoder; for the DS2020 Combitronic system, starts the procedure to find the zero position of motor shaft
- Ai(1) specifies external encoder

EXAMPLE:

```

EIGN(W,0)      'Set all I/O as general inputs
a=0
ZS            'Clear all faults
Ai(0)        'Arm motor's capture register
MV           'Set up slow velocity mode
VT=1000
ADT=10       'Set up accel/decel
G            'Start motion
WHILE Bt     'While trajectory
  IF Bi(0)==0 'Check index capture of encoder
    GOSUB(1)  'Call subroutine
  ELSE
    X
  ENDIF      'End checking
LOOP        'Loop back
RI(0)      'Report rising edge
OFF
END
'SUB 1: Increment a every 1 second
C1
  IF B(4,0)==0 'Check Timer 0 status
    a=a+1      'Updating a every second
    TMR(0,1000) 'Set Timer 0 counting
  ENDIF
RETURN

```

RELATED COMMANDS:

Aij(enc) *Arm Index Rising Edge Then Falling Edge (see page 272)*

Aj(enc) *Arm Index Falling Edge (see page 274)*

Aji(enc) *Arm Index Falling Edge Then Rising Edge (see page 276)*

R Bi(enc) *Bit, Index Capture, Rising (see page 309)*

EIRE *Enable Index Register, Encoder Capture (see page 419)*

EIRI *Enable Index Register, Input Capture (see page 421)*

R I(enc) *Index, Rising-Edge Position (see page 502)*



Arm Index Rising Edge Then Falling Edge

APPLICATION:	I/O control
DESCRIPTION:	Arms the index registers for capturing the rising edge and then the falling edge of the encoder
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	Input: 0 or 1
DEFAULT VALUE:	Input: 0 or 1
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	Aij(0):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The Aij(enc) command arms the index registers for capturing the rising edge and then the falling edge of the encoder. To first capture the falling edge and then the rising edge, see Aji(enc) on page 276.

NOTE: The rising and falling edges are stored to different index registers.

The enc parameter specifies the encoder to be captured; it does not specify the source of the index signal.

- Aij(0) specifies internal encoder
- Aij(1) specifies external encoder

EXAMPLE:

```

EIGN(W,0)      'Set all I/O as general inputs
a=0
ZS             'Clear all faults
Aij(0)        'Arm motor's capture register
MV            'Set up slow velocity mode
VT=1000
ADT=10        'Set up slow accel/decel
G             'Start motion
WHILE Bi(0)==0 'While waiting for rising edge
LOOP         'Loop back
RI(0)        'Report rising edge
WHILE Bj(0)==0 'While waiting for falling edge
LOOP         'Loop back
RJ(0)        'Report falling edge
OFF
END

```

RELATED COMMANDS:

Ai(enc) *Arm Index Rising Edge (see page 270)*
 Aj(enc) *Arm Index Falling Edge (see page 274)*
 Aji(enc) *Arm Index Falling Edge Then Rising Edge (see page 276)*
 R Bi(enc) *Bit, Index Capture, Rising (see page 309)*
 R Bj(enc) *Bit, Index Capture, Falling (see page 312)*
 EIRE *Enable Index Register, Encoder Capture (see page 419)*
 EIRI *Enable Index Register, Input Capture (see page 421)*
 R I(enc) *Index, Rising-Edge Position (see page 502)*
 R J(enc) *Index, Falling-Edge Position (see page 524)*



APPLICATION:	I/O control
DESCRIPTION:	Arms the index register for capturing the falling edge of the encoder
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	Input: 0 or 1
TYPICAL VALUES:	Input: 0 or 1
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	Aj(0):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The Aj(enc) command arms the index register for capturing the falling edge of the encoder. To capture the falling edge, see Ai(enc) on page 270.

NOTE: The rising and falling edges are stored to different index registers.

The enc parameter specifies the encoder to be captured; it does not specify the source of the index signal.

- Aj(0) specifies internal encoder
- Aj(1) specifies external encoder

EXAMPLE:

```

EIGN(W,0)      'Set all I/O as general inputs
a=0
ZS            'Clear all faults
Aj(0)         'Arm motor's capture register
MV           'Set up slow velocity mode
VT=1000
ADT=10       'Set up slow accel/decel
G            'Start motion
WHILE Bt     'While trajectory
  IF Bj(0)==0 'Check index capture of encoder
    GOSUB(1)  'Call subroutine
  ELSE
    X
  ENDIF      'End checking
LOOP        'Loop back
RJ(0)      'Report falling edge
OFF
END
'SUB 1: Increment a every 1 second
C1
  IF B(4,0)==0 'Check Timer 0 status
    a=a+1      'Updating a every second
    TMR(0,1000) 'Set Timer 0 counting
  ENDIF
RETURN

```

RELATED COMMANDS:

Ai(enc) *Arm Index Rising Edge (see page 270)*

Aij(enc) *Arm Index Rising Edge Then Falling Edge (see page 272)*

Aji(enc) *Arm Index Falling Edge Then Rising Edge (see page 276)*

R Bj(enc) *Bit, Index Capture, Falling (see page 312)*

EIRE *Enable Index Register, Encoder Capture (see page 419)*

EIRI *Enable Index Register, Input Capture (see page 421)*

R J(enc) *Index, Falling-Edge Position (see page 524)*



Arm Index Falling Edge Then Rising Edge

APPLICATION:	I/O control
DESCRIPTION:	Arms the index registers for capturing the falling edge and then the rising edge of the encoder
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	Input: 0 or 1
DEFAULT VALUE:	Input: 0 or 1
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	Aji(0):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The Aji(enc) command arms the index registers for capturing the falling edge and then the rising edge of the encoder. To first capture the rising edge and then falling edge, see Aij(enc) on page 272.

NOTE: The rising and falling edges are stored to different index registers.

The enc parameter specifies the encoder to be captured; it does not specify the source of the index signal.

- Aji(0) specifies internal encoder
- Aji(1) specifies external encoder

EXAMPLE:

```

EIGN (W, 0)      'Set all I/O as general inputs
a=0
ZS              'Clear all faults
Aji (0)         'Arm motor's capture register
MV             'Set up slow velocity mode
VT=1000
ADT=10         'Set up slow accel/decel
G              'Start motion
WHILE Bj (0) ==0 'While waiting for falling edge
LOOP          'Loop back
RJ (0)         'Report falling edge
WHILE Bi (0) ==0 'While waiting for rising edge
LOOP          'Loop back
RI (0)         'Report rising edge
OFF
END

```

RELATED COMMANDS:

Ai(enc) *Arm Index Rising Edge (see page 270)*
 Aij(enc) *Arm Index Rising Edge Then Falling Edge (see page 272)*
 Aj(enc) *Arm Index Falling Edge (see page 274)*
 R Bi(enc) *Bit, Index Capture, Rising (see page 309)*
 R Bj(enc) *Bit, Index Capture, Falling (see page 312)*
 EIRE *Enable Index Register, Encoder Capture (see page 419)*
 EIRI *Enable Index Register, Input Capture (see page 421)*
 R I(enc) *Index, Rising-Edge Position (see page 502)*
 R J(enc) *Index, Falling-Edge Position (see page 524)*



`al[index]=formula` Array Long `[index]`

APPLICATION:	Variables
DESCRIPTION:	User signed 32-bit variables
EXECUTION:	Immediate
CONDITIONAL TO:	Index values range 0 to 50
LIMITATIONS:	<p>An expression used as an index within the [] brackets is limited to no more than one operator (two values).</p> <p>No Combitronic requests or functions with parenthesis are supported within the [] brackets.</p> <p>This data space is shared with Cam motion (MC) if a RAM table location is selected. However, the <code>aw</code> command must not be used to access this space during that time.</p>
READ/REPORT:	<code>Ral[index]</code>
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	Signed 32-bit number
RANGE OF VALUES:	-2147483648 to 2147483647
TYPICAL VALUES:	-2147483648 to 2147483647
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	<code>al[0]:3=1234, a=al[0]:3, Ral[0]:3</code> where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The SmartMotor™ has 8, 16 and 32-bit arrays. The 32-bit array takes the form of the variables `al[index]`. These are general-purpose, 32-bit signed integer variables that can be reported, used on either side of an equation, and can be mixed in an expression with variables other than 32-bit. Like all user variables, they are always lowercase, and they are automatically initialized to zero at power up or reset.

The syntax of the 32-bit array is `al[index]` which stands for array long and accepts an index value between 0 and 50. This index can be specified explicitly or through another variable. For example, `al[4]` refers to the fifth element in the 32-bit array, while `aw[n]` refers to an element of the array, where the variable "n" must be between 0 and 50.

The value of any array variable is reported with the `R`, `PRINT()` or `PRINT1()` functions.

EXAMPLE:

```

a1[47]=20  'Assign the value of 20 to a1[47]
Ra1[47]    'Report the value of a1[47] to the primary serial port.
PRINT ("a1[47]=", a1[47], #13)      'Print to the primary serial port.
PRINT1 ("a1[47]=", a1[47], #13)     'Print to the secondary serial port.
END

```

Program output is:

```

20
a1[47]=20

```

The $a1[]$ array is classified as read/write, meaning that it can be assigned a value, or it can be assigned to some other variable or function. In other words, these variables can be left- or right-hand values.

EXAMPLE:

```
a1[24]=a1[43]+a1[7]
```

The above is a valid equation, combining the contents of $a1[43]$ and $a1[7]$ and sending the total into $a1[24]$.

These variables are 32-bit signed integers, so they are limited to whole numbers from -2147483648 to 2147483647. Math operations that result in digits after the decimal point are truncated toward zero. Therefore, the value 2.9 becomes 2, and the value -2.9 becomes -2.

If you assign or perform an operation that normally results in a value outside this range, the Bs bit indicates an overflow and the operation aborts before assigning the value to the left of the equal sign.

These are other restrictions:

- If $a1[1]+a$ exceeds 32 signed bits, the operation $c=a1[1]+a$ aborts and an error flag is set.
- If $a-a1[1]$ exceeds 32 signed bits, the operation $c=a-a1[1]$ aborts and an error flag is set.
- If $a*a1[1]$ exceeds 32 signed bits, the operation $c=a*a1[1]$ aborts and an error flag is set.

The system flag, Bs, is set. Note that many different types of command errors will also set the Bs bit. The RERRC command can be used to retrieve the last command error. For a math overflow, that is error code 23. For details on the RERRC command, see ERRC on page 451.

If one of these variables is used with a variable of another type, it will be appropriately converted (the variable will be "type cast").

For example, assigning the variable $aw[27]=a1[0]$ directly stores the 16 least-significant bits of $a1[0]$ to $aw[27]$. The sign bit of $a1[0]$ is not considered, the sign is determined based on bit 15 of $a1[0]$. The higher bits of the variable $a1[0]$ are ignored.

Conversely, if the left-hand value is a 32-bit variable and the right-hand side contains 16-bit variables, the 16-bit variables will be "upgraded" to 32-bits. The sign is preserved when casting to a longer format. In the equation $a1[0]=ab[4]-aw[7]$, both $ab[4]$ and $aw[7]$ are converted into 32-bit numbers before the subtraction occurs.

In the SmartMotor language, all user variables are written as lowercase letters, while functions and commands have at least one uppercase character. The term "a" is a general-purpose variable, while "A" is the acceleration function. As previously described, any user variable can be assigned a value through an equation.

All user variables are initialized to the value 0 at power up or on execution of the Z system-reset command. Other than by direct assignment, this is the only way the SmartMotor sets all of the user variables to 0. Issuing a RUN command does *not* perform this automatic initialization. For this reason, it

is better to test a program, whether it is auto-execution or not, by power cycling the SmartMotor or issuing the Z system-reset command.

NOTE: To understand the relationship between user assigned letter variables a-z, aa-zz and aaa-zzz, and variable arrays ab[], al[] and aw[], see Array Variable Memory Map on page 897. The arrays and the letter variables do not overlap in the Class 5 motor.

RELATED COMMANDS:

R a...z *32-Bit Variables (see page 249)*

R aa...zz *32-Bit Variables (see page 249)*

R aaa...zzz *32-Bit Variables (see page 249)*

R ab[index]=formula *Array Byte [index] (see page 252)*

R af[index]=formula *Array Float [index] (see page 267)*

R aw[index]=formula *Array Word [index] (see page 294)*

VLD(variable,number) *Variable Load (see page 820)*

VST(variable,number) *Variable Save (see page 824)*



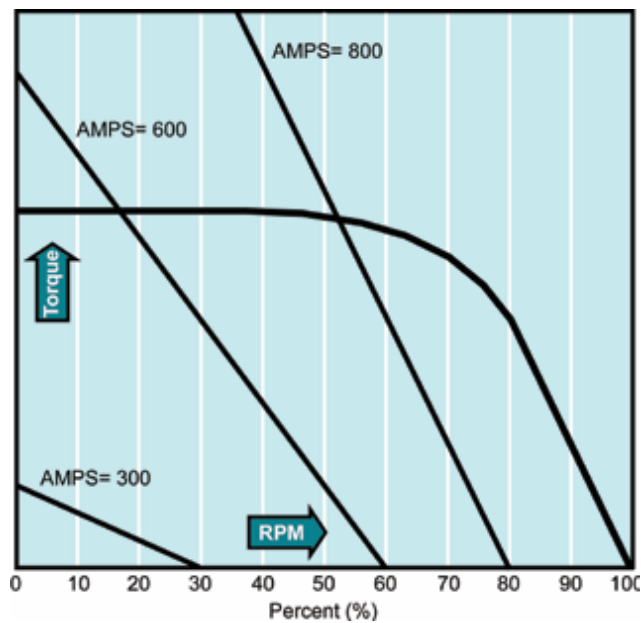
APPLICATION:	Motion control; supports the DS2020 Combitronic system
DESCRIPTION:	Sets maximum allowed pulse width modulation (PWM) to motor windings
EXECUTION:	Next PID sample
CONDITIONAL TO:	N/A
LIMITATIONS:	Must not be negative
READ/REPORT:	RAMPS
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	1/1023 of maximum PWM permitted
RANGE OF VALUES:	0 to 1023
TYPICAL VALUES:	0 to 1023
DEFAULT VALUE:	1023
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	AMPS:3=100, a=AMPS:3, RAMPS:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The AMPS command limits both the continuous torque and speed of the SmartMotor™.

In the DS2020 Combitronic system, this command does not act directly on PWM limitation. Instead, it limits the current reference that can be required to the current loop. Units are 1/1023 of maximum motor current set by the SMI software DS2020 Configuration tool. For details, see the *Moog Animatics DS2020 Combitronic™ Installation and Startup Guide*.

To set the SmartMotor to use maximum available PWM, issue the command AMPS=1023. Setting AMPS=0 limits PWM to 0 and prevents any output torque. To conceptually understand what happens when you use values between 0 and 1023, consider the next torque-speed diagram:



AMPS torque-speed diagram

The AMPS function essentially cuts the torque-speed characteristic of the motor by slicing off the part of the curve to the right of the AMPS line. Note that there are some values of AMPS that will limit top speed but not peak torque. The slope of the line is highly dependent on the voltage of the power source.

AMPS is often used to limit torque and speed. AMPS has no effect in torque mode (MT or T). In this mode, the value of T controls the commanded torque of the motor without limitation by AMPS.

EXAMPLE: (Routine homes motor against a hard stop)

```

MDS           'Using Sine mode commutation
KP=3200      'Increase stiffness from default
KD=10200    'Increase damping from default
F           'Activate new tuning parameters
AMPS=100     'Lower current limit to 10%
VT=-10000   'Set maximum velocity
ADT=100     'Set maximum accel/decel
MV          'Set Velocity mode
G           'Start motion
WHILE EA>-100 'Loop while position error is small
LOOP        'Loop back to WHILE
O=-100     'While pressed, declare home offset
S          'Abruptly stop trajectory
MP        'Switch to Position mode
VT=20000  'Set higher maximum velocity
PT=0      'Set target position to be home
G         'Start motion
TWAIT    'Wait for motion to complete
AMPS=1023 'Restore current limit to maximum
END      'End program

```

RELATED COMMANDS:

MT *Mode Torque* (see page 620)

R T=formula *Torque, Open-Loop Commanded* (see page 769)

APPLICATION:	Math function
DESCRIPTION:	Gets the arcsine of the input value
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RASIN(value)
WRITE:	N/A
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Degrees output
RANGE OF VALUES:	Input (floating-point): -1.0 to 1.0 Output in degrees: -90.0 to 90.0 (floating-point)
TYPICAL VALUES:	Input (floating-point): -1.0 to 1.0 Output in degrees: -90.0 to 90.0 (floating-point)
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

ASIN takes an input and returns a floating-point arcsine in degrees:

```
af[1]=ASIN(arg)
```

where *arg* may be an integer (e.g., *a* or *aw[0]*) or floating-point variable (e.g., *af[0]*). Integer or floating-point constants may also be used (e.g., 23 or 23.7, respectively).

This command cannot have within the parenthesis: math operators, other parenthetical functions, or a Combitronic request from another motor. For example, *x=FABS(PA)* is allowed, but *x=FABS(PA:3)* is not allowed.

The result of this function is a floating-point type. If used in an equation, the operations in the equation that are processed after this function are automatically promoted to a float. This is dependent on the mathematical order of operations in the equation. As with other equations (e.g., *x=a+b*), the variable to the left of "=" may be an integer variable to accept the result. However, the value will be truncated to fit to that integer type. For example, the assignment "*aw[0]=*" will drop any fractional amount and truncate the result to the range -32768 to 32767 (*aw[0]=100.5* will report as 100, and *aw[0]=40000.0* will report as -25536).

Although the floating-point variables and their standard binary operations conform to IEEE-754 double precision, the floating-point square root and trigonometric functions only produce IEEE-754 single-precision results. For more details, see Variables and Math on page 198.

EXAMPLE:

```
af[0]=ASIN(.5)      'Set array variable = ASIN(.5)
Raf[0]              'Report variable af[0]
RASIN(.5)           'Report ASIN(.5)
af[1]=0.4332
af[0]=ASIN(af[1])  'Variables may be put in the parenthesis
Raf[0]              'Output in degrees
END
```

Program output is:

```
30.000000000
30.000000000
25.670810699
```

RELATED COMMANDS:

- R ACOS(value) *Arccosine (see page 259)*
- R ATAN(value) *Arctangent (see page 289)*
- R COS(value) *Cosine (see page 372)*
- R SIN(value) *Sine (see page 738)*
- R TAN(value) *Tangent (see page 775)*



AT=formula

Acceleration Target

APPLICATION:	Motion control; supports the DS2020 Combitronic system
DESCRIPTION:	Sets the target acceleration only (does not change deceleration unless no deceleration has been set)
EXECUTION:	Buffered until a G command is issued
CONDITIONAL TO:	MP, MV, ADT=, G, PID n (sample rate), encoder resolution
LIMITATIONS:	Must not be negative; effective value is rounded down to next even number
READ/REPORT:	RAT
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	(encoder counts / (sample ²)) * 65536 DS2020 Combitronic system: user increments / sec ² , see FD=e-expression on page 461
RANGE OF VALUES:	0 to 2147483647 DS2020 Combitronic system: 0 to 4294967295
TYPICAL VALUES:	2 to 5000 DS2020 Combitronic system: depends on FD
DEFAULT VALUE:	0 (for firmware 5.x.4.x and later, the default is set to 4) DS2020 Combitronic system: 0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	AT:3=1234, a=AT:3, RAT:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

NOTE: This command is affected by the SCALEA command. For details, see SCALEA(m,d) on page 724. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

The buffered AT value determines the acceleration used by subsequent position or velocity moves to calculate the required trajectory. Changing AT during a move will not alter the current trajectory unless a new G command is issued.

Acceleration is pre-scaled by 65536 and may range from 2 to 2147483647. A value of 0 is not valid. Due to internal calculations, odd values for this command are rounded up to an even value.

If the value for DT has not been set since powering up the motor, the value of AT= will be automatically applied to DT=. However, this should be avoided. Instead, always use the ADT= command to specify the value for AT and DT when they are the same. If the value needed for DT is different than AT, specify it with the DT= command.

Equations for Real-World Units:

Encoder resolution and sample rate can vary. Therefore, the general equations in the next table can be used to convert the real-world units of acceleration to a value for AT, where af[0] is already set with the real-world unit value. These equations force floating-point calculations to avoid overflow and maintain resolution. They can be placed in a user program, or they can be precalculated if the values of SAMP and RES are known (SAMP and RES can be reported from the terminal using the RSAMP and RRES commands, respectively). SAMP can change if the PID command is used. The value of RES can differ between motor models.

Input as value in af[0]	Equation
Radians/(Sec ²)	AT=((af[0]*RES)/(PI*2*((SAMP*1.0)*SAMP)))*65536 DS2020 Combitronic system: AT=((af[0]*FD)/(PI*2))
Encoder Counts/(Sec ²) DS2020 Combitronic system: User Increments/(Sec ²)	AT=(af[0]/((SAMP*1.0)*SAMP))*65536 DS2020 Combitronic system: AT=(af[0])
Rev/(Sec ²)	AT=((af[0]*RES)/((SAMP*1.0)*SAMP))*65536 DS2020 Combitronic system: AT=(af[0]*FD)
RPM/Sec	AT=((af[0]*RES)/(60.0*((SAMP*1.0)*SAMP)))*65536 DS2020 Combitronic system: AT=((af[0]*FD)/60)
RPM/Min	AT=((af[0]*RES)/(3600.0*((SAMP*1.0)*SAMP)))*65536 DS2020 Combitronic system: AT=((af[0]*FD)/3600)

A useful Scale Factor Multiplier code example, which also illustrates the use of af[], SAMP and RES, is shown in RES on page 702 and SAMP on page 722.

EXAMPLE:

```
MP      'Set mode position
AT=5000 'Set target acceleration
PT=20000 'Set absolute position
VT=10000 'Set velocity
G       'Start motion
```

EXAMPLE:

```
AT=100   'Set buffered acceleration
VT=750   'Set buffered velocity
MV       'Set buffered velocity mode
G        'Start motion
```

RELATED COMMANDS:

ADT=formula *Acceleration/Deceleration Target (see page 263)*
ADTS=formula *Acceleration/Deceleration Target, Synchronized (see page 265)*
ATS=formula *Acceleration Target, Synchronized (see page 292)*
R DT=formula *Deceleration Target (see page 396)*
DTS=formula *Deceleration Target, Synchronized (see page 399)*
R EL=formula *Error Limit (see page 426)*
R FD=expression *Resolution to Set Units of Position/Velocity/Acceleration (see page 461)*
G *Start Motion (GO) (see page 473)*

MP *Mode Position (see page 613)*

MV *Mode Velocity (see page 624)*

PID# *Proportional-Integral-Differential Filter Rate (see page 654)*

R PT=formula *Position, (Absolute) Target (see page 690)*

R RES *Resolution (see page 702)*

R SAMP *Sampling Rate (see page 722)*

R VT=formula *Velocity Target (see page 828)*

X *Decelerate to Stop (see page 844)*

ATAN(value)

Arctangent

APPLICATION:	Math function
DESCRIPTION:	Gets the arctangent of the input value
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RATAN(value)
WRITE:	N/A
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Degrees output
RANGE OF VALUES:	Input (floating-point): any value Output in degrees (floating-point): -90.0 to 90.0
TYPICAL VALUES:	Input (floating-point): any value Output in degrees (floating-point): -90.0 to 90.0
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

ATAN takes an input and returns a floating-point arctangent in degrees:

```
af[1]=ATAN(arg)
```

where *arg* may be an integer (e.g., *a* or *aw*[0]) or floating-point variable (e.g., *af*[0]). Integer or floating-point constants may also be used (e.g., 23 or 23.7, respectively).

This command cannot have within the parenthesis: math operators, other parenthetical functions, or a Combitronic request from another motor. For example, *x=FABS(PA)* is allowed, but *x=FABS(PA:3)* is not allowed.

The result of this function is a floating-point type. If used in an equation, the operations in the equation that are processed after this function are automatically promoted to a float. This is dependent on the mathematical order of operations in the equation. As with other equations (e.g., *x=a+b*), the variable to the left of "=" may be an integer variable to accept the result. However, the value will be truncated to fit to that integer type. For example, the assignment "*aw*[0]=" will drop any fractional amount and truncate the result to the range -32768 to 32767 (*aw*[0]=100.5 will report as 100, and *aw*[0]=40000.0 will report as -25536).

Although the floating-point variables and their standard binary operations conform to IEEE-754 double precision, the floating-point square root and trigonometric functions only produce IEEE-754 single-precision results. For more details, see Variables and Math on page 198.

EXAMPLE:

```
af[0]=ATAN(50.5)  'Set array variable = ATAN(50.5)
Raf[0]           'Report value of af[0] variable
RATAN(50.5)      'Report ATAN(50.5)
af[1]=0.4332
af[0]=ATAN(af[1]) 'Variables may be put in the parenthesis
Raf[0]           'Output in degrees
END
```

Program output is:

```
88.865577697
88.865577697
23.422260284
```

RELATED COMMANDS:

```
R ACOS(value) Arccosine (see page 259)
R ASIN(value) Arcsine (see page 284)
R COS(value)  Cosine (see page 372)
R SIN(value)  Sine (see page 738)
R TAN(value)  Tangent (see page 775)
```

ATOF(index) ASCII to Float

APPLICATION:	Data conversion
DESCRIPTION:	Gets (reads) the ASCII to float conversion
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	Integer portion of input string must be in 32-bit range
READ/REPORT:	RATOF(index)
WRITE:	N/A
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	-2147483648.000000000 to +2147483647.000000000
TYPICAL VALUES:	-2147483648.000000000 to +2147483647.000000000
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The ATOF command is used to read an ASCII string containing a number in the format:

```
-123456789.123456789
```

The string is stored in the ab[index] array. The argument in the ATOF(index) function is the index of ab[] where the string begins. The parsing ends when a character other than a digit, decimal point or minus sign is met.

The value returned is a float, which can be assigned to a floating-point variable or an integer.

EXAMPLE:

```
ab [10]=50
ab [11]=51
ab [12]=48
ab [13]=46
ab [14]=49
ab [15]=0
af [0]=ATOF (10)
Raf [0]
END
```

Program output is:

```
230.099999999
```

RELATED COMMANDS:

R HEX(index) *Decimal Value of a Hex String (see page 489)*

ATS=formula Acceleration Target, Synchronized

APPLICATION:	Motion control
DESCRIPTION:	Sets the synchronized (path) target acceleration (does not change deceleration)
EXECUTION:	Immediate
CONDITIONAL TO:	PID n
LIMITATIONS:	Must not be negative; 0 is not valid; effective value is rounded up to next even number
READ/REPORT:	None
WRITE:	Write only
LANGUAGE ACCESS:	Assignment
UNITS:	(encoder counts / (sample ²)) * 65536
RANGE OF VALUES:	0 to 2147483647
TYPICAL VALUES:	0 to 5000
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M), not ver. 5.32.x.x; 6.x (D/M), M requires EIP option
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

NOTE: This command requires a Combitronic-supported motor. Although this command does not support Combitronic syntax, it does use Combitronic communication to pass information between other motors.

NOTE: This command is affected by the SCALEA command. For details, see SCALEA(m,d) on page 724. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

Setting the synchronized (path) ATS value determines the acceleration that will be used by subsequent position or velocity moves to calculate the required trajectory. Changing ATS during a move will not alter the current trajectory unless a new G command is issued.

Acceleration is pre-scaled by 65536 and may range from 2 to 2147483647. A value of 0 is not valid. Due to internal calculations, odd values for this command are rounded up to an even value.

Equations for Real-World Units:

Encoder resolution and sample rate can vary. Therefore, the general equations in the next table can be used to convert the real-world units of acceleration to a value for ATS, where af[0] is already set with the real-world unit value. These equations force floating-point calculations to avoid overflow and maintain resolution. They can be placed in a user program, or they can be precalculated if the values of SAMP and RES are known (SAMP and RES can be reported from the terminal using the RSAMP and RRES commands, respectively). SAMP can change if the PID command is used. The value of RES can differ between motor models.

Input as value in af[0]	Equation
Radians/(Sec ²)	$ATS=((af[0]*RES)/(PI*2*((SAMP*1.0)*SAMP)))*65536$
Encoder Counts/(Sec ²)	$ATS=(af[0]/((SAMP*1.0)*SAMP))*65536$
Rev/(Sec ²)	$ATS=((af[0]*RES)/((SAMP*1.0)*SAMP))*65536$
RPM/Sec	$ATS=((af[0]*RES)/(60.0*((SAMP*1.0)*SAMP)))*65536$
RPM/Min	$ATS=((af[0]*RES)/(3600.0*((SAMP*1.0)*SAMP)))*65536$

A useful Scale Factor Multiplier code example, which also illustrates the use of af[], SAMP and RES, is shown in RES on page 702 and SAMP on page 722.

EXAMPLE: (Shows use of ATS, DTS and VTS)

```

EIGN (W, 0)      'Set all I/O as general inputs.
ZS              'Clear errors.
ATS=100         'Set synchronized acceleration target.
DTS=500         'Set synchronized deceleration target.
VTS=100000000   'Set synchronized target velocity.
PTS (500;1,1000;2,10000;3) 'Set synchronized target position
                  'on motor 1, 2 and 3.
GS              'Initiate synchronized move.
TSWAIT         'Wait until synchronized move ends.
END            'Required END.

```

RELATED COMMANDS:

ADTS=formula *Acceleration/Deceleration Target, Synchronized (see page 265)*
DTS=formula *Deceleration Target, Synchronized (see page 399)*
PID# *Proportional-Integral-Differential Filter Rate (see page 654)*
PRTS(...) *Position, Relative Target, Synchronized (see page 685)*
PRTSS(...) *Position, Relative Target, Synchronized, Supplemental (see page 688)*
PTS(...) *Position Target, Synchronized (see page 692)*
R PTSD *Position Target, Synchronized Distance (see page 695)*
PTSS(...) *Position Target, Synchronized Supplemental (see page 696)*
R PTST *Position Target, Synchronized Time (see page 698)*
VTS=formula *Velocity Target, Synchronized Move (see page 831)*



aw[index]=formula

Array Word [index]

APPLICATION:	Variables
DESCRIPTION:	User signed 16-bit data variables
EXECUTION:	Immediate
CONDITIONAL TO:	Index values range 0 to 101
LIMITATIONS:	<p>An expression used as an index within the [] brackets is limited to no more than one operator (two values).</p> <p>No Combitronic requests or functions with parenthesis are supported within the [] brackets.</p> <p>This data space is shared with Cam motion (MC) if a RAM table location is selected. However, the aw command must not be used to access this space during that time.</p>
READ/REPORT:	Raw[index]
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	Signed 16-bit number
RANGE OF VALUES:	-32768 to 32767
TYPICAL VALUES:	-32768 to 32767
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	aw[0]:3=1234, a=aw[0]:3, Raw[0]:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The SmartMotor™ has 8, 16 and 32-bit arrays. The 16-bit array takes the form of the variables aw [index]. These are general-purpose, 16-bit signed integer variables that can be reported, used on either side of an equation, and can be mixed in an expression with variables other than 16-bit. Like all user variables, they are always lowercase, and are automatically initialized to zero at power up or reset.

The syntax of the 16-bit array is aw[index], which stands for array word, and accepts an index value between 0 and 101. This index can be specified explicitly or through another variable. For example, aw [4] refers to the fifth element in the 16-bit array, while aw[n] refers to an element of the array, where the variable "n" must be between 0 and 101.

The value of any array variable is reported with the R, PRINT() or PRINT1() functions.

EXAMPLE:

```
aw[47]=20  'Assign the value of 20 to aw[47]
Raw[47]    'Report the value of aw[47] to the primary serial port
PRINT ("aw[47]=",aw[47],#13)      'Print to the primary serial port
PRINT1 ("aw[47]=",aw[47],#13)    'Print to the secondary serial port
END
```

Program output is:

```
20
aw[47]=20
```

The $aw[]$ array is classified as read/write, meaning that it can be assigned a value, or it can be assigned to some other variable or function. In other words, these variables can be left- or right-hand values.

EXAMPLE:

```
aw[24]=aw[43]+aw[7]
```

The above is a valid equation, combining the contents of $aw[43]$ and $aw[7]$ and sending the total into $aw[24]$.

As signed 16-bit variables, they are limited to whole numbers ranging from -32768 and 32767. Math operations that result in digits after the decimal point are truncated toward zero. So a value of 2.9 becomes 2, and a value of -2.9 becomes -2.

If you assign or perform an operation that would normally result in a value outside of this range, the variable will "wrap," or take on the corresponding modulo. As an example, because of this, $32767+1=-32768$. The result "wrapped around" to the negative extreme.

These are other restrictions:

- If $aw[1]+a$ exceeds 32 signed bits, the operation $c=aw[1]+a$ will abort and an error flag is set.
- If $a-aw[1]$ exceeds 32 signed bits, the operation $c=a-aw[1]$ will abort and an error flag is set.
- If $a*aw[1]$ exceeds 32 signed bits, the operation $c=a*aw[1]$ will abort and an error flag is set.

The system flag, Bs, is set. Note that many different types of command errors will also set the Bs bit. The RERRC command can be used to retrieve the last command error. For a math overflow, that is error code 23. For details on the RERRC command, see ERRRC on page 451.

If one of these variables is used with a variable of another type, it will be appropriately converted (the variable will be "type cast").

If the left-hand variable is a 16-bit one like $aw[100]$, only the lowest 16 bits are preserved. The sign is determined by bit 15 of the value on the right-side of the equals sign.

Conversely, if the left-hand value is a 32-bit variable and the right-hand side contains 16-bit variables, the 16-bit variables will be "upgraded" to 32-bits. The sign is preserved when casting to a longer format. In the equation $cc=ab[4]-aw[7]$, both $ab[4]$ and $aw[7]$ are converted into 32-bit numbers before the subtraction occurs.

In the SmartMotor language, all user variables are written as lowercase letters, while functions and commands have at least one uppercase character. The term "a" is a general-purpose variable, while "A" is the acceleration function. As previously described, any user variable can be assigned a value through an equation.

All user variables are initialized to the value 0 at power up or on execution of the Z system-reset command. Other than by direct assignment, this is the only way the SmartMotor sets all of the user variables to 0. Issuing a RUN command does *not* perform this automatic initialization. For this reason, it

is better to test a program, whether it is auto-execution or not, by power cycling the SmartMotor or issuing the Z system-reset command.

NOTE: To understand the relationship between user assigned letter variables a-z, aa-zz and aaa-zzz, and variable arrays ab[], al[] and aw[], see Array Variable Memory Map on page 897. The arrays and the letter variables do not overlap in the Class 5 motor.

RELATED COMMANDS:

R a...z *32-Bit Variables (see page 249)*

R aa...zz *32-Bit Variables (see page 249)*

R aaa...zzz *32-Bit Variables (see page 249)*

R ab[index]=formula *Array Byte [index] (see page 252)*

R af[index]=formula *Array Float [index] (see page 267)*

R al[index]=formula *Array Long [index] (see page 278)*

VLD(variable,number) *Variable Load (see page 820)*

VST(variable,number) *Variable Save (see page 824)*



B(word,bit)

Status Byte

APPLICATION:	System; supports the DS2020 Combitronic system
DESCRIPTION:	Reads the status from the specified status word and bit number
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RB(word,bit)
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Binary bit
RANGE OF VALUES:	Input: word argument range: 0-17 bit argument range: 0-15 Output: 0 or 1
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	RB(2,4):3, x=B(2,4):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The B command reads the status from the specified status word and bit number:

=B(word,bit)

where "word" is the word number and "bit" is the bit number.

EXAMPLE: Check timer status

```
IF B(4,0)==0      'Check Timer 0 status
  a=a+1           'Updating a every second
  TMR(0,1000)    'Set Timer 0 counting
ENDIF
```

EXAMPLE: (For CANopen only, using firmware 5.x.4.21 or later) Read status word 10, bits 1-5, and then clear the event flag.

```

WHILE 1
  IF B(10,1)==1
    Z(10,1) ' Clear event flag
    PRINT("Rx PDO 1",#13)
  ENDIF
  IF B(10,2)==1
    Z(10,2) ' Clear event flag
    PRINT("Rx PDO 2",#13)
    . . .
  ENDIF
  IF B(10,5)==1
    Z(10,5) ' Clear event flag
    PRINT("Rx PDO 5",#13)
  ENDIF
LOOP
END

```

EXAMPLE: Read Status Words for faults, and print fault messages to the screen. You can use the tables in Status Words - SmartMotor on page 921 to get the bits and meanings, or you can use the "bit" commands (like Ba, Be, etc.) where available.

```

IF(B(0,0)!=1) 'Read Status Word 0, bit 0 for Drive Enable fault

' This section looks at low voltage and drive enable input state,
' which can prevent drive ready state regardless of any faults.

Z(6,13) WAIT=10 ' Clear drive voltage low flag to check true state
IF B(6,13) == 1 ' Not a fault but will prevent drive ready
  PRINT("Bus voltage is too low.",#13)
ENDIF

IF (FW/(2^24)) == 6
' For Class 6 motors:
  IF IN(7) == 0
    PRINT("Drive enable input low",#13)
  ENDIF
  IF Ba==1 ' Look for overcurrent fault
    PRINT("Overcurrent occurred",#13)
  ENDIF
ELSEIF (FW/(2^24)) == 5
' For Class 5 Class 5 M-series 5.98.x.x or 5.97.x.x
' NOTE: Not a fault in D-series
  IF ((FW/65536)&255) == 98 | ((FW/65536)&255) == 97)
    IF IN(12) == 0
      PRINT("Drive enable input low",#13)
    ENDIF
    IF Ba==1 ' Look for overcurrent fault
      PRINT("Overcurrent occurred",#13)
    ENDIF

```

```

    ENDIF
ENDIF

' This section look at faults.

IF Bh==1      ' Look for overtemperature fault
    PRINT("Over temperature",#13)
ENDIF

IF Be==1      ' Look for position error fault
    PRINT("Excessive Position Error",#13)
ENDIF

IF(B(0,3)==1) ' Look for bus voltage fault
    PRINT("Bus voltage Fault",#13)
ENDIF

IF Bv==1      ' Look for velocity limit fault
    PRINT("Velocity limit fault",#13)
ENDIF

IF(B(0,9)==1) ' Look for dE/dt fault
    PRINT("DE/DT Fault",#13)
ENDIF

IF Br==1      ' Look for historical positive hardware limit fault
    PRINT("Historical Positive H/W limit",#13)
ENDIF

IF Bl==1      ' Look for historical negative hardware limit fault
    PRINT("Historical Negative H/W limit",#13)
ENDIF

IF (B(1,10)==1)&(B(1,11)==1) ' Software limits enabled with
                               ' mode to cause faults
    IF Brs==1 ' Look for historical positive software limit fault
        PRINT("Historical Positive S/W limit",#13)
    ENDIF
    IF Bls==1 ' Look for historical negative software limit fault
        PRINT("Historical Negative S/W limit",#13)
    ENDIF
ENDIF

IF B(2,8)==1  ' Look for Watchdog fault on supported motors
               ' NOTE: Not all firmware supports this notification
    PRINT("Watchdog",#13)
ENDIF

IF B(2,9)==1  ' Look for ADB (Animatics Data Block) fault
               ' If ADB is corrupt, motion is prevented
    PRINT("ADB checksum Fault",#13)

```

```
ENDIF

IF B(6,5)==1    ' Look for encoder feedback fault
    PRINT("Feedback Fault (encoder)",#13)
ENDIF

IF B(6,7)==1    ' Look for drive enable fault
    PRINT("Drive Enable Fault",#13)
ENDIF

IF B(6,12)==1   ' Look for absolute encoder battery fault
    ' NOTE: This will print even if configured to not fault motor
    PRINT("ABS Battery Fault",#13)
ENDIF

ELSE            ' The drive is ready for motion
    PRINT("Drive is ready!",#13)
ENDIF
```

RELATED COMMANDS:

R FAUSTS(x) *Returns Fault Status Word (see page 459)*

R W(word) *Report Specified Status Word (see page 833)*

APPLICATION:	System
DESCRIPTION:	Overcurrent detected state
EXECUTION:	Historical, latched by PID sample
CONDITIONAL TO:	Hardware Detection
LIMITATIONS:	N/A
READ/REPORT:	RBa RB(0,4)
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Binary bit
RANGE OF VALUES:	0 or 1
TYPICAL VALUES:	0
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The SmartMotor™ firmware checks each PID Sample to see whether or not a peak overcurrent condition exists. The setpoint is in hardware and depends on the model motor and drive stage. If the setpoint is reached, the system flag Ba is to 1.

For a Class 5 D-style motor:

When an overcurrent condition is detected, the SmartMotor will turn off the amplifier for several servo samples to reduce the peak load and then turn on the amplifier to complete the commanded motion. During the off state, if the position error exceeds the allowable following error (EL), the servo will indicate a position error (shown by the Be status bit).

For an M-style motor:

The M-style motor has more sophisticated current sensing and protection. Other mechanisms in the firmware are used to try to prevent the peak current from occurring. This bit is an indication that a more severe situation has occurred. If this bit is indicated on an M-style motor, then the motor will shut down, similar to other faults. This fault must be cleared (typically with a ZS command) to resume motion.

The Ba bit is reset by any of these methods:

- Power reset
- Z command (total reset of software)
- Z(0,4) command
- ZS command
- Za command

When the Ba flag is repeatedly set, it indicates an underlying problem:

- If the Ba flag is frequently set, it typically indicates that the motor is undersized in the peak range.
- If the Ba bit is set during every machine cycle, the acceleration value may be too high. Therefore, try lowering the acceleration value. If the flag is still set for every cycle, then the motor may be under sized for the application.

For details on motor sizing, see the *Moog Animatics Product Catalog*.

EXAMPLE: (Subcomponent of system check routine)

```
IF Ba                'If Peak overcurrent is detected
  PRINT("OVERCURRENT") 'Inform host
  Za                'Clear overcurrent state latch
ENDIF
```

EXAMPLE: (Subroutine finds and prints errors)

```
C10                'Subroutine label
  IF Be            'Check for position error
    PRINT("Position Error", #13)
  ENDIF
  IF Bh            'Check for overtemp error
    PRINT("Overtemp Error", #13)
  ENDIF
  IF Ba            'Check for overcurrent error
    PRINT("Overcurrent Error", #13)
  ENDIF
RETURN            'Return to subroutine call
```

RELATED COMMANDS:

R B(word,bit) *Status Byte (see page 297)*

R W(word) *Report Specified Status Word (see page 833)*

Z(word,bit) *Reset Specified Status Bit (see page 848)*

Z *Total CPU Reset (see page 846)*

Za *Reset Overcurrent Flag (see page 850)*

ZS *Global Reset System State Flag (see page 858)*

BAUD(channel)=formula Set BAUD Rate (RS-232 and RS-485)

APPLICATION:	Communications control; supports the DS2020 Combitronic system
DESCRIPTION:	Set serial communication BAUD rate for transmitting data
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	There may be limitations based on the motor type. For details, see Product-Specific Table on page 303.
READ/REPORT:	RBAUD(channel)
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	bits per second (Baud)
RANGE OF VALUES:	2400, 4800, 9600, 19200, 38400, 57600 or 115200
TYPICAL VALUES:	2400, 4800, 9600, 19200, 38400, 57600 or 115200
DEFAULT VALUE:	9600
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020) RBAUD(1) requires: 5.x (D/M); 6.4.2.x (D)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The BAUD command sets the speed or baud rate of the specified serial channel. To do this, use:

```
BAUD(channel)=formula
```

where (channel) is 0 or 1 for channel 0 or channel 1, respectively, and formula is the desired baud rate: 2400, 4800, 9600, 19200, 38400, 57600 or 115200.

NOTE: BAUD(1)= and RBAUD(1) are not available on M-style motors.

Additionally, the baud rate of the primary communications channel can be set by the command:

```
BAUDrate
```

where rate is the desired baud rate: 2400, 4800, 9600, 19200, 38400, 57600 or 115200. For example, BAUD9600 would be equivalent to BAUD(0)=9600.

You can also set the baud rate for all motors, for example:

```
0BAUD9600
```

would globally set all motors on the serial network to a baud rate of 9600 bps.

NOTE: SmartMotor commands like 0CADDR=... or 0ADDR=... with a leading number really send a corresponding address byte (i.e., "0", which is hex 80 or decimal 128). This can be seen by viewing the serial data with the Serial Data Analyzer ("sniffer") tool, which is available on the SMI software View menu.

Product-Specific Table

This table provides product-specific information on the BAUD command.

Motor	Chnl ¹	COM Chnls	COM Pins	BAUD Cmd Forms ²	Report Cmd Forms ³	Combi Cmd Forms
Class 5 D-Style	COM0	RS-232 (RS-485 with adapter)	7W2; 15-pin (not CDS)	BAUDrate BAUD(channel)=baud	RBAUD(channel)	N/A
	COM1	RS-485	15-pin	BAUD(channel)=baud	RBAUD(channel)	N/A
Class 5 M-Style	COM0	RS-485	8-pin circular	BAUDrate BAUD(channel)=baud	RBAUD(channel)	N/A
	COM1	N/A	N/A	N/A	RBAUD(channel)	N/A
Class 6 D-Style	COM0	RS-232	7W2	BAUDrate BAUD(channel)=baud	RBAUD(channel)	N/A
	COM1	RS-485	26-pin	BAUD(channel)=baud	RBAUD(channel)	N/A
Class 6 M-Style	COM0	RS-485	8-pin circular	BAUDrate BAUD(channel)=baud	RBAUD(channel)	N/A
	COM1	N/A	N/A	N/A	N/A	N/A
Class 6 SL17	COM0	RS-232	15-pin	BAUDrate BAUD(channel)=baud	RBAUD(channel)	N/A
	COM1	N/A	N/A	N/A	N/A	N/A

1. See the corresponding motor installation guide for I/O and channel details.
2. For details, see BAUD(channel)=formula on page 303.
3. Not the same as Class 5 M-style or Class 6 IE motors.

EXAMPLE: (Shows use of BAUD(channel)=formula, CBAUD=formula)

```
ECHO      'Turn echo on
EL=-1     'Disable error limits
BAUD(0)=115200 'Set serial chan. 0 communications to 115,200
CBAUD=1000000 'Set CAN bus to 1000000
. . .
```

EXAMPLE: (Shows use of BAUDrate)

```
EIGN(W,0) 'Set all local I/O as general-use inputs
ZS        'Clear errors
BAUD115200 'Set baud rate of channel 0 to 115,200
END
```

RELATED COMMANDS:

R CAN, CAN(arg) *CAN Bus Status (see page 357)*

R CBAUD=formula *CAN Baud Rate (see page 363)*

Be Bit, Position Error Limit

APPLICATION:	System; supports the DS2020 Combitronic system
DESCRIPTION:	Position error declared
EXECUTION:	Historical, latched by PID sample
CONDITIONAL TO:	Position error exceeded EL value during trajectory move
LIMITATIONS:	Torque modes have no position error
READ/REPORT:	RBe RB(0,6)
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Binary bit
RANGE OF VALUES:	0 or 1
TYPICAL VALUES:	0
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The Be status bit indicates the detection of a position error. At each PID sample, the magnitude of the measured position error is compared to the user-specified position error (EL) value. If this value is exceeded, the servo will be immediately turned off.

The Be bit is reset by any of these methods:

- Power reset
- Z command (total reset of software)
- Z(0,6) command
- ZS command
- Ze command

NOTE: In cases where the motor has gone beyond the EL (error limit) but the trajectory generator is still active with the previously calculated trajectory, the ZS command may not clear the Be bit. If you are unable to reset Be with the ZS command, issue an OFF command before issuing the ZS command, which clears the current commanded trajectory and allows the reset to complete.

EXAMPLE: (test for position error)

```
TWAIT          'wait for trajectory in progress
               'to complete
IF Be          'unsuccessful, position error?
    PRINT("POSITION ERROR")    'inform host
ENDIF
```

EXAMPLE: (loop while Be is 0)

```
WHILE Be==0 LOOP 'Loop while Be is 0
    'Then proceed when Be is 1
```

NOTE: An extended period of overcurrent condition may result in a position error because this condition will cause a reduction in power to the motor and cause it to fall behind, possibly enough to exceed EL (maximum allowable position error).

If position errors are continuously received, check for loss of drive power, increased load or locked load.

EXAMPLE: (Subroutine finds and prints errors)

```
C10          'Subroutine label
IF Be        'Check for position error
    PRINT("Position Error", #13)
ENDIF
IF Bh        'Check for overtemp error
    PRINT("Overtemp Error", #13)
ENDIF
IF Ba        'Check for overcurrent error
    PRINT("Overcurrent Error", #13)
ENDIF
RETURN      'Return to subroutine call
```

RELATED COMMANDS:

R B(word,bit) *Status Byte (see page 297)*

R W(word) *Report Specified Status Word (see page 833)*

Z(word,bit) *Reset Specified Status Bit (see page 848)*

Z *Total CPU Reset (see page 846)*

Ze *Reset Position Error Flag (see page 851)*

ZS *Global Reset System State Flag (see page 858)*

APPLICATION:	System
DESCRIPTION:	Hardware motor overheat state
EXECUTION:	Historical, latched by PID sample
CONDITIONAL TO:	Motor temperature, temperature setpoint (TH)
LIMITATIONS:	N/A
READ/REPORT:	RBh RB(0,5)
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Binary bit
RANGE OF VALUES:	0 or 1
TYPICAL VALUES:	0
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The SmartMotor has an internal temperature sensor on the control circuit board. This provides a simple safety mechanism to shut down the motor when the average temperature exceeds 85 degrees Celsius.

Under continuous heavy loads, all motors will generate heat. If the heat sink or ventilation method is inadequate, eventually the motor will overheat. If this situation repeatedly occurs, it may mean that the motor does not have enough power for the assigned task (inadequate motor size) or excessive resistance (friction) to motion is occurring. Therefore, check the design of your motion system. For details on motor sizing, see the *Moog Animatics Product Catalog*.

The overheat temperature limit is specified using the TH command, but it cannot exceed 85 degrees Celsius. If the temperature exceeds the TH value, the motor will turn off, and Bh will be set to 1. The SmartMotor will reject any motion command until the temperature has dropped below the trip point by 5 degrees Celsius (below 80 degrees Celsius).

The Ba bit is reset by any of these methods:

- Power reset
- Z command (total reset of software)
- Z(0,5) command
- ZS command
- Zh command

EXAMPLE: (Subcomponent of system check routine)

```
IF Bh
  PRINT("MOTOR TOO HOT") 'Inform host
  GOSUB123               'Deal with condition
ENDIF
C123 'Some code to deal with condition
```

EXAMPLE:

```
PRINT(#13,"Default value of TH = ",TH) 'default=85
PRINT(#13,"Motor Temperature = ",TEMP)
PRINT(#13,"START MOTION")
ADT=222 'Set accel/decel
VT=44444
MV 'Set velocity mode
G 'Start motion
TH=TEMP-5 'Force an overheat condition
'units are degrees Celsius
'TH maximum setting is 85

a=CLK
WHILE Bh==0 LOOP 'Loop while Bh is 0
WHILE Bt LOOP
b=CLK
PRINT(#13,"Servo OFF after ",b-a," milliseconds")
```

EXAMPLE: (Subroutine finds and prints errors)

```
C10 'Subroutine label
IF Be 'Check for position error
  PRINT("Position Error", #13)
ENDIF
IF Bh 'Check for overtemp error
  PRINT("Overtemp Error",#13)
ENDIF
IF Ba 'Check for overcurrent error
  PRINT("Overcurrent Error",#13)
ENDIF
RETURN 'Return to subroutine call
```

RELATED COMMANDS:

R B(word,bit) *Status Byte (see page 297)*
 R TEMP, TEMP(arg) *Temperature, Motor (see page 777)*
 R TH=formula *Temperature, High Limit (see page 779)*
 R W(word) *Report Specified Status Word (see page 833)*
 Z *Total CPU Reset (see page 846)*
 Z(word,bit) *Reset Specified Status Bit (see page 848)*
 Zh *Reset Temperature Fault (see page 852)*
 ZS *Global Reset System State Flag (see page 858)*

Bi(enc) Bit, Index Capture, Rising

APPLICATION:	System
DESCRIPTION:	Rising edge capture on encoder
EXECUTION:	Historical, latched by PID sample
CONDITIONAL TO:	Index capture, arming index: Ai(enc), Aij(enc), Aji(enc), EIRI, EIRE
LIMITATIONS:	N/A
READ/REPORT:	RBi(enc); supports the DS2020 Combitronic system over RS-232 only RBi(0), RBi(1), RB(1,2), RB(1,6)
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Binary bit
RANGE OF VALUES:	0 or 1
TYPICAL VALUES:	0
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The Bi(enc) flag is set to 1 when the associated encoder Z pulse (index mark) is detected. The value of the associated encoder capture is valid and can be read using the I(enc) command.

For the DS2020 Combitronic system, the report version of this command is used for a procedure to find the position that corresponds to the physical zero position of the feedback sensor.

Before a capture can occur, arming is required using the Ai(enc), Aij(enc) or Aji(enc) command. After a capture has occurred, the value stored in I(enc) will remain the same until another arming command is issued and another index is detected. Reading the captured value does not change the capture state or captured value.

The enc parameter specifies the encoder to be captured; it does not specify the source of the index signal.

- Bi(0) specifies the internal encoder
- Bi(1) specifies the external encoder

The command RI(enc) reports the captured index reading.

The Bi(0) bit is reset by any of these methods:

- Power reset
- Z command (total reset of software)
- Z(1,2) command
- ZS command

The Bi(1) bit is reset by any of these methods:

- Power reset
- Z command (total reset of software)
- Z(1,6) command
- ZS command

For the DS2020 Combitronic system, RBi(0) is initially 0 and returns 1 if the procedure to find the zero position of motor shaft is finished (zero found).

EXAMPLE: (simple homing)

```

MV                'Set buffered velocity mode
ADT=10            'Set buffered accel/decel
VT=-4000         'Set low buffered maximum velocity
i=0              'A flag to know if any index was found
ZS G             'Start slow motion profile
Ai (0)           'Clear and arm index capture
WHILE Bm==0     'Travel until negative limit reached
  IF Bi (0)==1
    PT=I (0)    'Save this target
    Ai (0)     'Clear and arm index capture
    i=1        'Set flag to indicate index was found
  ENDIF
LOOP
X                'Decelerate to a stop
IF i==0
  'Index not seen, must have started close to limit
  VT=4000       'Set low buffered maximum velocity
  Ai (0)       'Clear and arm index capture
  ZS G
  WHILE Bi (0)==0 'Travel positive until index reached
  LOOP
  PT=I (0)     'Go back to index
ENDIF
MP ZS G        'Start motion
TWAIT         'Wait till end of trajectory
O=0           'Set origin at index

```

EXAMPLE: (fast index find)

```

MP           'Set buffered velocity mode
ADT=1000    'Set fast accel/decel
VT=4000000 'Set fast velocity
PRT=RES+100 'Set relative distance just beyond
            'One shaft turn
Ai (0)      'Clear and arm index capture
O=0         'Force change to position register
G           'Start fast move
TWAIT      'Wait till end of trajectory
PT=I (0)+100 'Go back to index
G           'Start motion
TWAIT      'Wait until end of trajectory
O=0         'Set origin at index

```

Index used as High Speed Position Capture:

When enabled through EIRI, the Bi(0) flag is set to 1 when a rising edge is seen at I/O pin 6. As a result, I/O pin 6 can be used to capture position for high-speed registration applications.

EXAMPLE: (fast position capture)

```

EIGN (6)    'Set port 6 as input port
EIRI        'Set port 6 to register internal encoder
            'Set F command flags
VT=100000  'Set Velocity
ADT=100     'Set accel/decel
MV          'Set to Velocity Mode
Ai (0)      'Arm registration
G           'Start moving
  WHILE Bi (0) ==0 'Travel until index reached
  LOOP
    X           'Decelerate to a stop
    RI (0)      'Report registered position
END

```

RELATED COMMANDS:

Ai(enc) *Arm Index Rising Edge (see page 270)*
 Aij(enc) *Arm Index Rising Edge Then Falling Edge (see page 272)*
 Aji(enc) *Arm Index Falling Edge Then Rising Edge (see page 276)*
 R B(word,bit) *Status Byte (see page 297)*
 R Bx(enc) *Bit, Index Input, Real-Time (see page 351)*
 EIRE *Enable Index Register, Encoder Capture (see page 419)*
 EIRI *Enable Index Register, Input Capture (see page 421)*
 R I(enc) *Index, Rising-Edge Position (see page 502)*
 R W(word) *Report Specified Status Word (see page 833)*
 Z *Total CPU Reset (see page 846)*
 Z(word,bit) *Reset Specified Status Bit (see page 848)*
 ZS *Global Reset System State Flag (see page 858)*

APPLICATION:	System
DESCRIPTION:	Falling edge capture on encoder
EXECUTION:	Historical, latched by PID sample
CONDITIONAL TO:	Index capture, arming index: Aj(enc), Aij(enc), Aji(enc), EIRI, EIRE
LIMITATIONS:	N/A
READ/REPORT:	RBj(enc) RBj(0), RB(1,3) RBj(1), RB(1,7)
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Binary bit
RANGE OF VALUES:	0 or 1
TYPICAL VALUES:	0
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The Bj(enc) flag is set to 1 when the associated encoder Z pulse (index mark) is detected. The value of the associated encoder capture is valid and can be read using the J(enc) command.

Before a capture can occur, arming is required using the Aj(enc), Aij(enc) or Aji(enc) command. After a capture has occurred, the value stored in J(enc) will remain the same until another arming command is issued and another index is detected. Reading the captured value does not change the capture state or captured value.

The enc parameter specifies the encoder to be captured; it does not specify the source of the index signal.

- Bj(0) specifies the internal encoder
- Bj(1) specifies the external encoder

The command RJ(enc) reports the captured index reading.

The Bj(0) bit is reset by any of these methods:

- Power reset
- Z command (total reset of software)
- Z(1,3) command
- ZS command

The Bj(1) bit is reset by any of these methods:

- Power reset
- Z command (total reset of software)
- Z(1,7) command
- ZS command

EXAMPLE: (simple homing)

```

MV                'Set buffered velocity mode
ADT=10            'Set buffered accel/decel
VT=-4000          'Set low buffered maximum velocity
i=0               'A flag to know if any index was found
ZS G              'Start slow motion profile
Aj (0)            'Clear and arm index capture
WHILE Bm==0       'Travel until negative limit reached
  IF Bj (0) ==1
    PT=J (0)      'Save this target
    Aj (0)        'Clear and arm index capture
    i=1           'Set flag to indicate index was found
  ENDIF
LOOP
X                  'Decelerate to a stop
IF i==0
  'Index not seen, must have started close to limit
  VT=4000         'Set low buffered maximum velocity
  Aj (0)          'Clear and arm index capture
  ZS G
  WHILE Bj (0) ==0 'Travel positive until index reached
  LOOP
  PT=J (0)        'Go back to index
ENDIF
MP ZS G           'Start motion
TWAIT            'Wait till end of trajectory
O=0              'Set origin at index

```

Index used as High Speed Position Capture:

When enabled through EIRI, the Bj(0) flag is set to 1 when a falling edge is seen at I/O pin 6. As a result, I/O pin 6 can be used to capture position for high-speed registration applications.

EXAMPLE: (fast position capture)

```

EIGN(6)      'Set port 6 as input port
EIRI         'Set port 6 to register internal encoder
             'Set F command flags
VT=100000   'Set velocity
ADT=100     'Set accel/decel
MV          'Set to Velocity Mode
Aj(0)       'Arm registration
G           'Start moving
  WHILE Bj(0)==0  'Travel until index reached
  LOOP
    X           'Decelerate to a stop
    RJ(0)      'Report registered position
END

```

RELATED COMMANDS:

Aij(enc) *Arm Index Rising Edge Then Falling Edge (see page 272)*

Aj(enc) *Arm Index Falling Edge (see page 274)*

Aji(enc) *Arm Index Falling Edge Then Rising Edge (see page 276)*

R B(word,bit) *Status Byte (see page 297)*

R Bx(enc) *Bit, Index Input, Real-Time (see page 351)*

EIRE *Enable Index Register, Encoder Capture (see page 419)*

EIRI *Enable Index Register, Input Capture (see page 421)*

R J(enc) *Index, Falling-Edge Position (see page 524)*

R W(word) *Report Specified Status Word (see page 833)*

Z *Total CPU Reset (see page 846)*

Z(word,bit) *Reset Specified Status Bit (see page 848)*

ZS *Global Reset System State Flag (see page 858)*

Bk Bit, Program EEPROM Data Status

APPLICATION:	System
DESCRIPTION:	Program EEPROM checksum failure state
EXECUTION:	Historical, set on EEPROM data check (during startup and program download)
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RBk RB(2,15)
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Binary bit
RANGE OF VALUES:	0 or 1
TYPICAL VALUES:	0= valid EEPROM user program checksum 1= Invalid EEPROM user program checksum
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

Bk indicates whether a user-program checksum write error has been detected. If Bk is 1, the user program and/or program header has been corrupted. You should not run the program in the SmartMotor™. This can occur if communications was lost or corrupted during a download of a program. Bk is reset to zero by a power reset or Z command, and a valid (pass) checksum is detected through RCKS.

RCKS scans the entire program, including the header, and returns two 6-bit checksums and then a "P" (pass) or "F" (fail) at the end. If RCKS reports a failure, Bk is set to 1. RCKS sends its value through the primary serial port.

EXAMPLE: (terminal window commands with responses)

```
RCKS      000049 0025E0 P
RBk       0
```

RELATED COMMANDS:

R B(word,bit) *Status Byte (see page 297)*

LOAD *Download Compiled User Program to Motor (see page 548)*

RCKS *Report Checksum (see page 701)*

R W(word) *Report Specified Status Word (see page 833)*

Z *Total CPU Reset (see page 846)*

Bl Bit, Left Hardware Limit, Historical

APPLICATION:	System
DESCRIPTION:	Hardware left travel limit
EXECUTION:	Historical, sampled each PID update until latched
CONDITIONAL TO:	EIGN(3), OUT(3)=, EILN
LIMITATIONS:	N/A
READ/REPORT:	RBl RB(0,13)
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Binary bit
RANGE OF VALUES:	0 or 1
TYPICAL VALUES:	0= Left/negative limit has not been active 1= Left/negative limit has been active
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

Bl is the historical left limit flag. If the left limit is found to be active during any servo sample, Bl is set to 1; it remains at 1 until you reset it. In addition, the motion will stop, and the motor will either servo in place or turn off the amplifier, depending on the value of the FSA function. The historical left/negative limit flag Bl provides a latched limit value, in case the limit was already reached or exceeded but is not currently active.

The real-time left/negative limit flag is Bm. It remains at 1 only while there is an active signal level on user pin 3. When Bm is set to 1, Bl is set to 1.

If the pin's function is assigned as general-purpose I/O through the EIGN(3) command, neither Bm nor Bl will be affected by the pin state. Changing pin states will not elicit limit behavior from the motor. For the pin to again elicit limit behavior, including the setting of Bl, the EILN command must be used to assign the pin's function as a limit switch.

The Bl bit is reset by any of these methods:

- Power reset
- Z command (total reset of software)
- Z(0,13) command
- ZS command
- Zl command

EXAMPLE:

```
IF Bm
    PRINT ("LEFT LIMIT PRESENTLY ACTIVE")
ELSEIF B1
    PRINT ("LEFT LIMIT PREVIOUSLY CONTACTED")
ELSE
    PRINT ("LEFT LIMIT NEVER REACHED")
ENDIF
```

RELATED COMMANDS:

R B(word,bit) *Status Byte (see page 297)*

R Bm *Bit, Left Hardware Limit, Real-Time (see page 320)*

R Bp *Bit, Right Hardware Limit, Real-Time (see page 325)*

R Br *Bit, Right Hardware Limit, Historical (see page 329)*

EIGN(...) *Enable as Input for General-Use (see page 412)*

EILN *Enable Input as Limit Negative (see page 415)*

R W(word) *Report Specified Status Word (see page 833)*

Z *Total CPU Reset (see page 846)*

Z(word,bit) *Reset Specified Status Bit (see page 848)*

Zl *Reset Historical Left Limit Flag (see page 853)*

ZS *Global Reset System State Flag (see page 858)*

Bls Bit, Left Software Limit, Historical

APPLICATION:	System
DESCRIPTION:	Software left travel limit
EXECUTION:	Historical, sampled each PID update until latched
CONDITIONAL TO:	SLD, SLM, SLE, SLN, motor actual position (RPA)
LIMITATIONS:	N/A
READ/REPORT:	RBlS RB(1,13)
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Binary bit
RANGE OF VALUES:	0 or 1
TYPICAL VALUES:	0= Left/negative software limit has not been active 1= Left/negative software limit has been active
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

Bls is the historical software left-limit flag. Bls provides a latched value in case the software limit was already reached or exceeded but is not currently active.

The software limits are an indication that the motor's actual position has exceeded the set range. If the software left limit is found to be active during any servo sample, Bls is set to 1 and remains 1 until you reset it. In addition, the motion will stop, and the motor will either servo in place or turn off the amplifier, depending on the value of the SLM and/or FSA function.

The left/negative software limit position can be set through the SLN command. The left and right software-limit functionality can be enabled and disabled with the SLE and SLD commands, respectively.

The real-time software left/negative limit flag is Bms, which only remains set to 1 while the motor is past the software limit. When Bms is set to 1, Bls is set to 1.

The Bls bit is reset by any of these methods:

- Power reset
- Z command (total reset of software)
- Z(1,13) command
- ZS command
- Zls command

EXAMPLE:

```
IF Bms
    PRINT ("SOFTWARE LEFT LIMIT PRESENTLY ACTIVE")
ELSEIF Bls
    PRINT ("SOFTWARE LEFT LIMIT PREVIOUSLY CONTACTED")
ELSE
    PRINT ("SOFTWARE LEFT LIMIT NEVER REACHED")
ENDIF
```

RELATED COMMANDS:

R B(word,bit) *Status Byte (see page 297)*

R Bms *Bit, Left Software Limit, Real-Time (see page 322)*

R Bps *Bit, Right Software Limit, Real-Time (see page 327)*

R Brs *Bit, Right Software Limit, Historical (see page 341)*

FSA(cause,action) *Fault Stop Action (see page 465)*

SLD *Software Limits, Disable (see page 740)*

SLE *Software Limits, Enable (see page 742)*

R SLM(mode) *Software Limit Mode (see page 748)*

R SLN=formula *Software Limit, Negative (see page 750)*

R W(word) *Report Specified Status Word (see page 833)*

Z *Total CPU Reset (see page 846)*

Z(word,bit) *Reset Specified Status Bit (see page 848)*

Zls *Reset Left Software Limit Flag, Historical (see page 854)*

ZS *Global Reset System State Flag (see page 858)*

Bm Bit, Left Hardware Limit, Real-Time

APPLICATION:	System
DESCRIPTION:	Left/negative hardware limit state
EXECUTION:	Real time, sampled each PID update
CONDITIONAL TO:	EIGN(3), OUT(3)=, EILN
LIMITATIONS:	N/A
READ/REPORT:	RBm RB(0,15)
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Binary bit
RANGE OF VALUES:	0 or 1
TYPICAL VALUES:	0 = Left/negative limit switch not active, or pin not assigned as a limit switch 1 = Left/negative limit switch active
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

Bm indicates if the left/negative pin is currently active. When Bm is set to 1, the historical left limit flag (Bl) is set to 1.

The left/negative software travel limit may be disabled by being assigned as a general-purpose input using the EIGN(3) command or as an output using the OUT(3)= command. To re-enable the left/negative hardware travel limit, issue the EILN command.

EXAMPLE:

```
IF Bm
    PRINT("LEFT LIMIT PRESENTLY ACTIVE")
ELSEIF Bl
    PRINT("LEFT LIMIT PREVIOUSLY CONTACTED")
ELSE
    PRINT("LEFT LIMIT NEVER REACHED")
ENDIF
```


RELATED COMMANDS:

- R B(word,bit) Status Byte (see page 297)*
- R Bl Bit, Left Hardware Limit, Historical (see page 316)*
- R Bp Bit, Right Hardware Limit, Real-Time (see page 325)*
- R Br Bit, Right Hardware Limit, Historical (see page 329)*
- EIGN(...) Enable as Input for General-Use (see page 412)*
- EILN Enable Input as Limit Negative (see page 415)*
- R W(word) Report Specified Status Word (see page 833)*
- Z Total CPU Reset (see page 846)*
- Z(word,bit) Reset Specified Status Bit (see page 848)*
- ZS Global Reset System State Flag (see page 858)*

Bms Bit, Left Software Limit, Real-Time

APPLICATION:	System
DESCRIPTION:	Left (negative) software limit state flag
EXECUTION:	Real time, sampled each PID update
CONDITIONAL TO:	SLD, SLM, SLE, SLN, motor actual position (RPA)
LIMITATIONS:	N/A
READ/REPORT:	RBms RB(1,15)
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Binary bit
RANGE OF VALUES:	0 or 1
TYPICAL VALUES:	0 = Software left/negative limit switch not active, or pin not assigned as a limit switch 1 = Software left/negative limit switch active
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

Bms indicates if the left/negative software limit is currently active. The software limits are an indication that the motor's actual position has exceeded the set range. When Bms is set to 1, the historical left limit flag (BlS) is set to 1.

The left/negative software limit position can be set through the SLN command. The left and right software-limit functionality can be enabled and disabled with the SLE and SLD commands, respectively.

EXAMPLE:

```
IF Bms
    PRINT("SOFTWARE LEFT LIMIT PRESENTLY ACTIVE")
ELSEIF Bls
    PRINT("SOFTWARE LEFT LIMIT PREVIOUSLY CONTACTED")
ELSE
    PRINT("SOFTWARE LEFT LIMIT NEVER REACHED")
ENDIF
```

RELATED COMMANDS:

- R B(word,bit) *Status Byte (see page 297)*
- R Bls *Bit, Left Software Limit, Historical (see page 318)*
- R Bps *Bit, Right Software Limit, Real-Time (see page 327)*
- R Brs *Bit, Right Software Limit, Historical (see page 341)*
- FSA(cause,action) *Fault Stop Action (see page 465)*
- SLD *Software Limits, Disable (see page 740)*
- SLE *Software Limits, Enable (see page 742)*
- R SLM(mode) *Software Limit Mode (see page 748)*
- R SLN=formula *Software Limit, Negative (see page 750)*
- R W(word) *Report Specified Status Word (see page 833)*
- Z *Total CPU Reset (see page 846)*
- Z(word,bit) *Reset Specified Status Bit (see page 848)*
- ZS *Global Reset System State Flag (see page 858)*

APPLICATION:	System
DESCRIPTION:	Motor OFF state
EXECUTION:	Immediate
CONDITIONAL TO:	Motor is off
LIMITATIONS:	N/A
READ/REPORT:	RBo RB(0,1)
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Binary flag
RANGE OF VALUES:	0 or 1
TYPICAL VALUES:	1 = Motor is off 0 = Motor is on
DEFAULT VALUE:	1
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The Bo bit represents the SmartMotor™ drive stage state. When Bo is 0, the drive stage is on; when Bo is 1, the drive stage is off.

Bo is set to 1 for any of these conditions:

- The motor has been powered on but no command has activated the drive stage
- The Z command is used to reset the motor
- The OFF command is issued or triggered by a motor fault

EXAMPLE:

```
C1          'Subroutine C1
IF Bo==1   'If Bo (motor off) is true
            PRINT("MOTOR DRIVE IS OFF",#13)
ENDIF
RETURN
```

RELATED COMMANDS:

G *Start Motion (GO)* (see page 473)

OFF *Off (Drive Stage Power)* (see page 636)

Z *Total CPU Reset* (see page 846)

Bp Bit, Right Hardware Limit, Real-Time

APPLICATION:	System
DESCRIPTION:	Right/positive hardware limit state
EXECUTION:	Sampled each PID update
CONDITIONAL TO:	EIGN(2), OUT(2)=, EILP
LIMITATIONS:	N/A
READ/REPORT:	RBp RB(0,14)
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Binary bit
RANGE OF VALUES:	0 or 1
TYPICAL VALUES:	0= Right/positive limit switch not active, or pin not assigned as a limit switch 1= Right/positive limit switch is active
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

Bp indicates if the right/positive pin is active. When Bp is set to 1, the historical right limit flag (Br) is set to 1.

The right/positive software travel limit may be disabled by being assigned as a general-purpose input using the EIGN(2) command or as an output using the OUT(2)= command. To re-enable the right/positive software travel limit, issue the EILP command.

EXAMPLE:

```
IF Br
    PRINT("RIGHT LIMIT PRESENTLY ACTIVE")
ELSEIF Bp
    PRINT("RIGHT LIMIT PREVIOUSLY CONTACTED")
ELSE
    PRINT("RIGHT LIMIT NEVER REACHED")
ENDIF
```

RELATED COMMANDS:

- R B(word,bit) Status Byte (see page 297)*
- R Bl Bit, Left Hardware Limit, Historical (see page 316)*
- R Bm Bit, Left Hardware Limit, Real-Time (see page 320)*
- R Br Bit, Right Hardware Limit, Historical (see page 329)*
- EIGN(...) Enable as Input for General-Use (see page 412)*
- EILP Enable Input as Limit Positive (see page 417)*
- R W(word) Report Specified Status Word (see page 833)*
- Z Total CPU Reset (see page 846)*
- Z(word,bit) Reset Specified Status Bit (see page 848)*
- ZS Global Reset System State Flag (see page 858)*

Bps Bit, Right Software Limit, Real-Time

APPLICATION:	System
DESCRIPTION:	Right (positive) software limit state flag
EXECUTION:	Real time, sampled each PID update
CONDITIONAL TO:	SLD, SLM, SLE, SLP, motor actual position (RPA)
LIMITATIONS:	N/A
READ/REPORT:	RBps RB(1,14)
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Binary flag
RANGE OF VALUES:	0 or 1
TYPICAL VALUES:	0= Right/positive limit switch not active, or pin not assigned as a limit switch 1= Right/positive limit switch is active
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

Bps indicates if the right/positive software limit is currently active. The software limits are an indication that the motor's actual position has exceeded the set range. When Bps is set to 1, the historical right limit flag (Brs) is set to one.

The right/positive software limit position can be set through the SLP command. The left and right software-limit functionality can be enabled and disabled with the SLE and SLD commands, respectively.

EXAMPLE:

```
IF Brs
    PRINT("SOFTWARE RIGHT LIMIT PRESENTLY ACTIVE")
ELSEIF Bps
    PRINT("SOFTWARE RIGHT LIMIT PREVIOUSLY CONTACTED")
ELSE
    PRINT("SOFTWARE RIGHT LIMIT NEVER REACHED")
ENDIF
```

RELATED COMMANDS:

- R B(word,bit) *Status Byte (see page 297)*
- R Bls *Bit, Left Software Limit, Historical (see page 318)*
- R Bms *Bit, Left Software Limit, Real-Time (see page 322)*
- R Brs *Bit, Right Software Limit, Historical (see page 341)*
- FSA(cause,action) *Fault Stop Action (see page 465)*
- SLD *Software Limits, Disable (see page 740)*
- SLE *Software Limits, Enable (see page 742)*
- R SLM(mode) *Software Limit Mode (see page 748)*
- R SLP=formula *Software Limit, Positive (see page 752)*
- R W(word) *Report Specified Status Word (see page 833)*
- Z *Total CPU Reset (see page 846)*
- Z(word,bit) *Reset Specified Status Bit (see page 848)*
- ZS *Global Reset System State Flag (see page 858)*

Br Bit, Right Hardware Limit, Historical

APPLICATION:	System
DESCRIPTION:	Hardware right travel limit
EXECUTION:	Historical, sampled each PID update until latched
CONDITIONAL TO:	EIGN(2), OUT(2)=, EILP
LIMITATIONS:	N/A
READ/REPORT:	RBr RB(0,12)
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Binary flag
RANGE OF VALUES:	0 or 1
TYPICAL VALUES:	0= Right/positive limit has not been active 1= Right /positive limit has been active
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

Br is the historical right limit flag. If the right limit is found to be active during any servo sample, Br is set to 1, and it remains at 1 until you reset it. In addition, depending on the value of the F function, the motion will stop and the motor will either servo or turn the amplifier OFF. The historical right/positive limit flag Br provides a latched limit value, in case the limit was already reached or exceeded but is not currently active.

The real-time right/positive limit flag is Bp; it remains at 1 only while there is an active signal level on user pin 2. When Bp is set to 1, Br is set to 1.

If the pin's function is assigned as general-purpose I/O through the EIGN(2) command, neither Bp nor Br will be affected by the pin state. Changing pin states will not elicit limit behavior from the motor. It will be necessary to issue the EILP command to assign the pin's function to being a limit switch for the pin to again elicit limit behavior, including the setting of Br.

The Br bit is reset by any of these methods:

- Power reset
- Z command (total reset of software)
- Z(0,12) command
- ZS command
- Zr command

EXAMPLE:

```
IF Br
    PRINT("RIGHT LIMIT PRESENTLY ACTIVE")
ELSEIF Bp
    PRINT("RIGHT LIMIT PREVIOUSLY CONTACTED")
ELSE
    PRINT("RIGHT LIMIT NEVER REACHED")
ENDIF
```

RELATED COMMANDS:

R B(word,bit) *Status Byte (see page 297)*

R Bl *Bit, Left Hardware Limit, Historical (see page 316)*

R Bm *Bit, Left Hardware Limit, Real-Time (see page 320)*

R Bp *Bit, Right Hardware Limit, Real-Time (see page 325)*

EIGN(...) *Enable as Input for General-Use (see page 412)*

EILP *Enable Input as Limit Positive (see page 417)*

R W(word) *Report Specified Status Word (see page 833)*

Z *Total CPU Reset (see page 846)*

Z(word,bit) *Reset Specified Status Bit (see page 848)*

Zr *Reset Right Limit Flag, Historical (see page 855)*

ZS *Global Reset System State Flag (see page 858)*

BREAK

Break from CASE or WHILE Loop

APPLICATION:	Program execution and flow control
DESCRIPTION:	Causes immediate exit from a WHILE or SWITCH control block (loop)
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	For downloaded code only; cannot be used through a serial port!
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

BREAK is used by the WHILE...LOOP and SWITCH...ENDS flow-control blocks (loops). In both structures, if BREAK is encountered, the program jumps out of that particular WHILE loop or SWITCH structure. If the control blocks are nested, BREAK only exits the WHILE loop or SWITCH structure that it is currently in.

The most common use of BREAK is to end each CASE of a SWITCH control structure. Without the BREAK statement, the program would continue to execute into the next CASE, even if it is not true.

EXAMPLE:

```

SWITCH a
  CASE 1
    PRINT("Hiya!", #13)
  CASE 2
    PRINT("Lo there!", #13)
    BREAK
  CASE 3
    PRINT("Me here!", #13)
    BREAK
  DEFAULT
    PRINT("Urp!", #13)
    BREAK

```

ENDS

If a=2, the SmartMotor™ will print "Lo there!" However, if a=1, the SmartMotor will print both "Hiya!" and "Lo there!" There is no BREAK statement to stop the program from running into case 2.

The BREAK statement can always be replaced by a GOTO statement, and this is how it is actually executed using the precompiled program location. BREAK has the advantage of not requiring a statement label to define the program branch location and, therefore, not conforming to structured programming methodology.

BREAK is not a valid terminal command; it is only valid from within a user program. If you want to be able to "break out of" a control block by remote (terminal) commands, you will need to use GOTO# or GOSUB# with appropriate statement labels. The next example illustrates this concept.

EXAMPLE:

```
a=1
WHILE a
  PRINT("I am still here ...",#13)
  WAIT=12000
  IF a==100
    BREAK      'a=100 could be sent through serial command
  ENDIF
LOOP
GOTO20

C10
  PRINT("EXITED with a==100",#13)
END

C20
  PRINT("EXITED with a<0",#13)
END
```

RELATED COMMANDS:

CASE formula *Case Label for SWITCH Block (see page 360)*
 DEFAULT *Default Case for SWITCH Structure (see page 388)*
 ENDS *End SWITCH Structure (see page 443)*
 LOOP *Loop Back to WHILE Formula (see page 553)*
 SWITCH formula *Switch, Program Flow Control (see page 766)*
 WHILE formula *While Condition Program Flow Control (see page 841)*



BRKENG

Brake Engage

APPLICATION:	Motion control
DESCRIPTION:	Immediately engages the hardware brake
EXECUTION:	Immediate; remains in effect until otherwise commanded
CONDITIONAL TO:	Hardware brake option is required
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	See BRKSRV on page 337
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	BRKENG:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:



CAUTION: It is important to turn the servo off when the brake is engaged. Otherwise, the motor could drive against the brake and overheat. When the SmartMotor™ powers up or comes out of a soft reset, the brake control is set to BRKSRV, by default, to automatically enforce this safety rule.

The SmartMotor™ may be purchased with optional, internal, zero-backlash brakes, which are used to hold a load for safety purposes. They are fail-safe, magnetic-clutch, disk brakes. The default power-on state is to disengage the brake when the drive stage is turned on. When power is lost, the brake engages. The brake takes from 3 to 5 milliseconds to actuate or release.

When BRKENG is issued, the brake is de-energized, which allows the magnetic brake to lock the shaft in place.

BRKENG terminates the brake control modes BRKSRV, BRKTRJ and BRKRLS.

NOTE: BRKENG is a manual override for the BRKSRV and BRKTRJ commands. When BRKENG is used, you must subsequently issue a BRKSRV, BRKTRJ or BRKRLS command to resume shaft movement.

EXAMPLE:

```
OFF           'Turn motor off
WHILE VA     'Wait for zero velocity
LOOP         ' before
BRKENG       ' applying the brake (shaft locked)
```

RELATED COMMANDS:

BRKRLS *Brake Release (see page 335)*

BRKSRV *Brake Servo, Engage When Not Servoing (see page 337)*

BRKTRJ *Brake Trajectory, Engage When No Active Trajectory (see page 339)*

EOBK(IO) *Enable Output, Brake Control (see page 445)*



BRKRLS

Brake Release

APPLICATION:	Motion control
DESCRIPTION:	Immediately release hardware brake
EXECUTION:	Immediate; remains in effect until otherwise commanded
CONDITIONAL TO:	Hardware brake option is required
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	See BRKSRV on page 337
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	BRKRLS:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:



CAUTION: It is important to turn the servo off when the brake is engaged. Otherwise, the motor could drive against the brake and overheat. When the SmartMotor™ powers up or comes out of a soft reset, the brake control is set to BRKSRV, by default, to automatically enforce this safety rule.

The SmartMotor™ may be purchased with optional, internal, zero-backlash brakes, which are used to hold a load for safety purposes. They are fail-safe, magnetic-clutch, disk brakes. The default power-on state is to disengage the brake when the drive stage is turned on. When power is lost, the brake engages. The brake takes from 3 to 5 milliseconds to actuate or release.

When BRKRLS is issued, the brake is energized (disengaged), which allows full shaft movement.

BRKRLS terminates the brake control modes BRKSRV, BRKTRJ and BRKENG.

To release MTB mode even if no brake is installed, manually "freewheel" the motor by issuing a BRKRLS command and then an OFF command (in that order). Those two commands do not need to be in immediate sequence—i.e., other commands, except MTB, can be between them. For details on MTB mode, see MTB on page 622.

NOTE: To ensure the motor remains in "freewheel" state, issue the FSA command (with action 1, servo off / freewheel) before issuing the BRKRLS OFF command sequence. For details, see FSA (cause,action) on page 465.

EXAMPLE:

```

BRKENG 'Assuming motion has stopped
OFF    ' or almost stopped
WAIT=1000
VT=0   'Set buffered velocity
ADT=0  'Set buffered accel/decel
MP     'Set buffered mode
PT=PA  'Set Target position to current position
G      'Begin servo at current position
BRKRLS 'Release (disengage brake)

```

EXAMPLE: (Shows use of "freewheel")

```

PT=PA  'Set Target position to current position
G      'Holds position
OFF    'Drive stage off, but MTB (dynamic braking) active
G      'Holds again
BRKRLS 'No change seen yet
OFF    'Now motor freewheels
G      'Motor holds in place again
OFF    'Motor freewheels
MTB    'Motor has dynamic braking
G      'Motor holds position
OFF    'Motor off but WITH dynamic braking
BRKRLS 'No change seen yet
G      'Motor holds position
OFF    'Motor freewheels
G      'Motor holds position
OFF    'Motor freewheels
MTB    'Returns back to default of dynamic braking when OFF is issued

```

RELATED COMMANDS:

BRKENG *Brake Engage (see page 333)*
BRKSRV *Brake Servo, Engage When Not Servoing (see page 337)*
BRKTRJ *Brake Trajectory, Engage When No Active Trajectory (see page 339)*
EOBK(IO) *Enable Output, Brake Control (see page 445)*
FSA(cause,action) *Fault Stop Action (see page 465)*
MTB *Mode Torque Brake (see page 622)*
OFF *Off (Drive Stage Power) (see page 636)*



Brake Servo, Engage When Not Servoing

APPLICATION:	Motion control
DESCRIPTION:	Release hardware brake while motor is on; engage hardware brake while motor is off
EXECUTION:	Immediate; remains in effect until otherwise commanded
CONDITIONAL TO:	Hardware brake option is required
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	This is the default brake command at power up
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	BRKSRV:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:



CAUTION: It is important to turn the servo off when the brake is engaged. Otherwise, the motor could drive against the brake and overheat. When the SmartMotor™ powers up or comes out of a soft reset, the brake control is set to BRKSRV, by default, to automatically enforce this safety rule.

The SmartMotor™ may be purchased with optional, internal, zero-backlash brakes, which are used to hold a load for safety purposes. They are fail-safe, magnetic-clutch, disk brakes. The default power-on state is to disengage the brake when the drive stage is turned on. When power is lost, the brake engages. The brake takes from 3 to 5 milliseconds to actuate or release.

BRKSRV terminates the brake control modes BRKRLS, BRKTRJ and BRKENG.

NOTE: A position error will terminate both the trajectory in progress state and servo on state. In this instance, the brake would then be asserted automatically.

EXAMPLE:

```
BRKSRV      'Set brake mode assuming it is safe
MP          'Set buffered mode
ADT=100     'Set buffered accel/decel
VT=100000  'Set buffered maximum velocity
PT=1000    'Set target
G           'Servo on, brake release, go to target
```

RELATED COMMANDS:

BRKENG *Brake Engage (see page 333)*

BRKRLS *Brake Release (see page 335)*

BRKTRJ *Brake Trajectory, Engage When No Active Trajectory (see page 339)*

EOBK(IO) *Enable Output, Brake Control (see page 445)*



Brake Trajectory, Engage When No Active Trajectory

APPLICATION:	Motion control
DESCRIPTION:	Release hardware brake while a trajectory is in progress; engage hardware brake while no trajectory is in progress
EXECUTION:	Immediate; remains in effect until otherwise commanded
CONDITIONAL TO:	Hardware brake option is required
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	See BRKSRV on page 337
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	BRKTRJ:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The SmartMotor™ may be purchased with optional, internal, zero-backlash brakes, which are used to hold a load for safety purposes. They are fail-safe, magnetic-clutch, disk brakes. The default power-on state is to disengage the brake when the drive stage is turned on. When power is lost, the brake engages. The brake takes from 3 to 5 milliseconds to actuate or release.

BRKTRJ automatically coordinates movement and brake application. When a trajectory is started by a G command, the brake is released. When the trajectory completes, the brake is engaged and the servo is simultaneously turned off. In this mode, and whenever the motor is not performing a trajectory, the brake is automatically engaged and the servo is turned off for any reason that clears the Bt (Busy Trajectory) bit.

As a result, any non-trajectory mode, like Torque mode, will not result in motion because the brake will be engaged and the servo will be off. Because the motor-off flag Bo is 0 (false), this behavior could be confusing to a user who is unaware of the function of BRKTRJ. However, from an operation/control perspective, the motor has not changed modes to OFF, which would be coincidental with Bo set to 1. When running in Torque mode or some other non-trajectory mode, it is more appropriate to use BRKSRV. For details, see BRKSRV on page 337.

BRKTRJ terminates the brake control modes BRKRLS, BRKENG and BRKSRV.

BRKTRJ immediately resets the trajectory flag to zero when the trajectory generator declares the trajectory has completed. At that instant, the BRKTRJ will engage (de-energize) the brake.

EXAMPLE:

```

BRKTRJ      'Set brake mode to respond to Bt bit
MP          'Set buffered mode
ADT=100     'Set buffered accel/decel
VT=100000  'Set buffered maximum velocity
C1         'Program statement label
PT=1000    'Set buffered target position
G          'Servo on, start trajectory
'The brake will automatically be energized and released
TWAIT      'Wait for trajectory to end,
           ' now brake will be on and servo off
WAIT=1000  'Brake on for ~one second
PT=0       'Set new buffered target position
G         'Servo on, brake off, trajectory
WAIT=1000
GOTO1     'Effective loop forever

```

NOTE: A position error will terminate the trajectory-in-progress state. In this case, the brake would then be asserted.

When in BRKTRJ mode, the brake will click on at the beginning of each move and click off at the end of each move. The clicking sound is normal and indicates proper operation.

RELATED COMMANDS:

BRKENG *Brake Engage (see page 333)*
BRKRLS *Brake Release (see page 335)*
BRKSRV *Brake Servo, Engage When Not Servoing (see page 337)*
EOBK(10) *Enable Output, Brake Control (see page 445)*

Brs Bit, Right Software Limit, Historical

APPLICATION:	System
DESCRIPTION:	Software right travel limit
EXECUTION:	Historical, sampled each PID update until latched
CONDITIONAL TO:	SLD, SLM, SLE, SLP, motor actual position (RPA)
LIMITATIONS:	N/A
READ/REPORT:	RBrS RB(1,12)
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Binary flag
RANGE OF VALUES:	0 or 1
TYPICAL VALUES:	0= Right/positive limit has not been active 1= Right /positive limit has been active
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

Brs is the historical software right-limit flag. Brs provides a latched value in case the software limit was already reached or exceeded but is not currently active.

The software limits are an indication that the motor's actual position has exceeded the set range. If the software right limit is found to be active during any servo sample, Brs is set to 1 and remains 1 until you reset it. In addition, the motion will stop, and the motor will either servo in place or turn off the amplifier, depending on the value of the SLM and/or FSA function.

The right/positive software limit position can be set through the SLP command. The left and right software-limit functionality can be enabled and disabled with the SLE and SLD commands, respectively.

The real-time software right/positive limit flag is Bps, which only remains set to 1 while the motor is past the software limit. When Bps is set to 1, Brs is set to 1.

The Brs bit is reset by any of these methods:

- Power reset
- Z command (total reset of software)
- Z(1,12) command
- ZS command
- Zrs command

EXAMPLE:

```
IF Brs
    PRINT ("SOFTWARE RIGHT LIMIT PRESENTLY ACTIVE")
ELSEIF Bps
    PRINT ("SOFTWARE RIGHT LIMIT PREVIOUSLY CONTACTED")
ELSE
    PRINT ("SOFTWARE RIGHT LIMIT NEVER REACHED")
ENDIF
```

RELATED COMMANDS:

R B(word,bit) *Status Byte (see page 297)*

R Bls *Bit, Left Software Limit, Historical (see page 318)*

R Bms *Bit, Left Software Limit, Real-Time (see page 322)*

R Bps *Bit, Right Software Limit, Real-Time (see page 327)*

FSA(cause,action) *Fault Stop Action (see page 465)*

SLD *Software Limits, Disable (see page 740)*

SLE *Software Limits, Enable (see page 742)*

R SLM(mode) *Software Limit Mode (see page 748)*

R SLP=formula *Software Limit, Positive (see page 752)*

R W(word) *Report Specified Status Word (see page 833)*

Z *Total CPU Reset (see page 846)*

Z(word,bit) *Reset Specified Status Bit (see page 848)*

Zrs *Reset Right Software Limit Flag, Historical (see page 856)*

ZS *Global Reset System State Flag (see page 858)*

APPLICATION:	System
DESCRIPTION:	Command syntax error occurred state
EXECUTION:	Immediate
CONDITIONAL TO:	Syntax error found while executing commands
LIMITATIONS:	N/A
READ/REPORT:	RBs RB(2,14)
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Binary flag
RANGE OF VALUES:	0 or 1
TYPICAL VALUES:	0 = Syntax error has not occurred 1 = Syntax error has occurred
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

If a syntax error is encountered in either a serial command or user program, the Bs flag is set to 1. This flag only indicates that a syntax error was encountered. The most common syntax errors are misspellings of commands, but the improper use of variables is also flagged. For example, trying to access the array element `aw[20000]` will produce a syntax error. The command that contains the syntax error is ignored.

The Bs bit is reset by any of these methods:

- Power reset
- Z command (total reset of software)
- Z(2,14) command
- ZS command
- Zs command

EXAMPLE:

Suppose the host should have transmitted `ADT=100` but actually transmitted `ADT=L00` due to noise in transmission. Bs would be set.

```
IF Bs
    PRINT("Syntax Error",#13)
ENDIF
```

RELATED COMMANDS:

R ERRC Error Code, Command (see page 451)

R ERRW Communication Channel of Most Recent Command Error (see page 453)

Z(word,bit) Reset Specified Status Bit (see page 848)

ZS Global Reset System State Flag (see page 858)

APPLICATION:	System
DESCRIPTION:	Trajectory in progress state flag
EXECUTION:	Updated each PID sample
CONDITIONAL TO:	Trajectory in progress
LIMITATIONS:	N/A
READ/REPORT:	RBt RB(0,2)
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Binary flag
RANGE OF VALUES:	0 or 1
TYPICAL VALUES:	0 = No trajectory in progress 1 = Trajectory in progress
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The flag Bt is set to 1 when the motor performs a calculated trajectory path from one point to another. After the trajectory generator has requested the final target position, the Bt flag is reset to zero. At this point, the PID positioning control takes over the motion — this means the motor shaft may still be moving due to mechanical settling.

Mode Torque (MT) will set the Bt bit to 1 only when the torque slope (TS) is active. When the torque reaches a constant value, the Bt will be 0.

Mode Velocity (MV) will maintain the Bt bit to 1 regardless of commanded velocity or acceleration, even they are set to zero.

Mode Follow (MFR) and Mode Step (MSR) will maintain the Bt bit to 1 even if there is no change to incoming counts.

If a relative or absolute move is commanded in Mode Position (MP), and there is no (zero) commanded acceleration or velocity, the Bt bit will be set to 1 and the motor shaft will not move. In other words, if ADT=0 and VT=0, then the motor shaft will not move, but the Bt bit will show a trajectory in progress.

EXAMPLE:

```

WHILE Bt           'While trajectory in progress
LOOP
WHILE VA           'While still settling or while velocity not zero
LOOP
OFF                'Motor off
BRKENG             'Brake engage

```

EXAMPLE:

```

MP          'Buffer a position move request
ADT=10
VT=440000
PT=10000
G          'Start the first buffered move
WHILE Bt   'Wait for first trajectory to be done
LOOP      'NOTE: TWAIT could have been used!
ADT=20     'Buffer another move
VT=-222000
PT=20000
G          'Now begin the second move

```

EXAMPLE: (Routine changes speed on digital input)

```

EIGN(W,0)  'Disable hardware limit IO
KD=10010   'Changing KD value in tuning
F          'Accept new KD value
O=0        'Reset origin
ADT=100    'Set maximum accel/decel
VT=10000   'Set maximum velocity
PT=40000   'Set final position
MP         'Set Position mode
G          'Start motion
WHILE Bt   'Loop while motion continues
  IF IN(0)==0 'If input is low
    IF VT==10000 'Check VT so change happens once
      VT=12000  'Set new velocity
      G         'Initiate new velocity
    ENDIF
  ENDIF
LOOP      'Loop back to WHILE
END

```

RELATED COMMANDS:

BRKTRJ *Brake Trajectory, Engage When No Active Trajectory (see page 339)*
G *Start Motion (GO) (see page 473)*
OFF *Off (Drive Stage Power) (see page 636)*
X *Decelerate to Stop (see page 844)*

APPLICATION:	System
DESCRIPTION:	Velocity limit reached
EXECUTION:	Historical, latched by PID sample
CONDITIONAL TO:	VL=
LIMITATIONS:	N/A
READ/REPORT:	RBv RB(0,7)
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Binary flag
RANGE OF VALUES:	0 or 1
TYPICAL VALUES:	0= Velocity limit not reached 1= Velocity limit reached
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

Bv reports the status of the velocity limit fault. It returns a 1 if the velocity limit was reached or a 0 if not. It is reported by the RBv command. The equivalent reporting PRINT() command is PRINT(Bv,#13).

When this bit is indicated, the motor has exceeded the speed set in the VL command. The motor will stop according to the fault action. This bit must be cleared to resume motion.

The Bv bit is reset by any of these methods:

- Power reset
- Z command (total reset of software)
- Z(0,7) command
- ZS command
- Zv command

EXAMPLE:

```

C1          'Subroutine C1
IF Bv==1   'If Bv (Velocity Limit) is true
            PRINT("VELOCITY LIMIT EXCEEDED", #13)
ENDIF
RETURN

```

RELATED COMMANDS:

R B(word,bit) *Status Byte (see page 297)*

R VL=formula *Velocity Limit (see page 818)*

R W(word) *Report Specified Status Word (see page 833)*

Z *Total CPU Reset (see page 846)*

Z(word,bit) *Reset Specified Status Bit (see page 848)*

ZS *Global Reset System State Flag (see page 858)*

Zv *Reset Velocity Limit Fault (see page 860)*

Bw Bit, Wrapped Encoder Position

APPLICATION:	System
DESCRIPTION:	Encoder overflow or underflow occurred
EXECUTION:	Historical, latched by PID sample
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RBw RB(3,3)
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Binary flag
RANGE OF VALUES:	0 or 1
TYPICAL VALUES:	0= No encoder wrap around occurred 1= Encoder wrap around occurred by position mode move
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

If Bw is 1, it indicates that the encoder position has exceeded or "wrapped" beyond the maximum value for the 32-bit position register. Specifically, the position has gone outside of the range -2147483648 to 2147483647.

This *does not* mean that the SmartMotor™ has lost its position information — it is still tracking its position.

The Bw bit can be set during any type of motion, it is not a fault and will not stop motion.

The Bw bit is reset by any of these methods:

- Power reset
- Z command (total reset of software)
- Z(3,3) command
- ZS command
- Zw command

EXAMPLE: (Test for wraparound and then reset flag)

```
IF Bw      'Test flag
    PRINT("Wraparound Occurred")
    Zw     'Reset flag
ENDIF
```

RELATED COMMANDS:

R B(word,bit) Status Byte (see page 297)

R W(word) Report Specified Status Word (see page 833)

Z Total CPU Reset (see page 846)

Z(word,bit) Reset Specified Status Bit (see page 848)

ZS Global Reset System State Flag (see page 858)

Zw Reset Encoder Wrap Status Flag (see page 861)

APPLICATION:	System
DESCRIPTION:	Index input state
EXECUTION:	Updated each PID sample
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RBx(enc) RBx(0), RB(1,8) RBx(1), RB(1,9)
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Binary flag
RANGE OF VALUES:	0 or 1
TYPICAL VALUES:	0 = Index capture input is not in contact (low) 1 = Index capture input is in contact (high)
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

Bx(enc) indicates the real-time state of the index input level. The Bx bit is set to 1 only while the motor is sitting on the index marker. Note that the index marker is only one encoder count wide. Therefore, this function is mainly used to verify the exact position of the index. For other uses, it is more efficient to use the functions like Bi and I.

The value of enc determines which encoder is being referred to:

- Bx(0) specifies the internal encoder
- Bx(1) specifies the external encoder

EXAMPLE: (Fast Index Find, Report Bx)

```
MP           'Set buffered velocity mode
ADT=1000    'Set fast accel/decel
VT=4000000 'Set fast velocity
PRT=RES*1.1 'Set relative distance just beyond one shaft turn
Ai (0)      'Clear and arm index capture
O=0         'Force change to position register
G           'Start fast move
TWAIT      'Wait till end of trajectory
PT=I (0)    'Go back to index
G           'Start motion
TWAIT      'Wait until end of trajectory
O=0         'Set origin at index
IF Bx (0)
    PRINT("On Index Pulse", #13)
ENDIF
```

RELATED COMMANDS:

R Bi(enc) *Bit, Index Capture, Rising* (see page 309)

R I(enc) *Index, Rising-Edge Position* (see page 502)

APPLICATION:	Program execution and flow control
DESCRIPTION:	Program statement label
EXECUTION:	N/A
CONDITIONAL TO:	N/A
LIMITATIONS:	Labels C0...C999 are permitted
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

C{number} is a statement label, where "number" is a value between 0 and 999. Statement labels provide the internal addressing required to support the GOSUB{number} and GOTO{number} language commands. For example, GOTO1 directs the program to label C1, whereas GOSUB37 directs the program to the subroutine that starts at label C37. You can also use labels to simply enhance program clarity. Statement labels may be placed anywhere within a program except in the middle of an expression.

NOTE: Program labels work by using a jump table in the header of the compiled program. The header contains the location of every label from 0 up to the highest label value used.

EXAMPLE: (consider these two programs)

```

C0
END
and
C999
END

```

Both programs behave exactly the same. However, the first compiled program (C0...END) will be much smaller than the second (C999...END) because the second contains all the label locations from 0-999.

The program header is read whenever the SmartMotor™ powers up or is reset. This means that the SmartMotor knows how to jump to any label location, even if the program has never been run, and start executing the program from there. This is a common means of making a single program with several routines that can be invoked on demand from a host.

EXAMPLE:

```
END
C0
    PRINT("Routine 0",#13)
END

C1
    PRINT("Routine 1",#13)
END
C2
    PRINT("Routine 2",#13)
END
```

To run routine 1, the host simply issues GOTO1 to the SmartMotor. If the host issues GOTO2, the motor runs routine 2. You can use a similar technique to allow the host to control where the program starts.

Using GOTO_{nnn} to jump to a location within a SWITCH block may be syntactically valid. However, it can yield unexpected runtime program execution when a CASE number is encountered.

RELATED COMMANDS:

GOSUB(label) *Subroutine Call (see page 480)*

GOTO(label) *Branch Program Flow to a Label (see page 482)*

CADDR=formula CAN Address

APPLICATION:	Communications control; supports the DS2020 Combitronic system
DESCRIPTION:	Set and read the CAN address (also used for DeviceNet and PROFIBUS); it can be different from the serial address. Value is retained in EEPROM between power cycles.
EXECUTION:	Requires reboot of motor for change to take effect
CONDITIONAL TO:	N/A
LIMITATIONS:	Requires reboot of motor for change to take effect
READ/REPORT:	PRINT(CADDR), <variable>=CADDR, RCADDR
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	1 to 127, -1 and 255 available for SL17 motor only
TYPICAL VALUES:	1 to 127
DEFAULT VALUE:	63 (factory EEPROM setting)
FIRMWARE VERSION:	5.x (D/M); 6.4.2.x (D); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The CADDR command is used to set and read the CAN address of the motor. The CAN address can be different from the serial address. The PROFIBUS firmware will also use this to set/read the PROFIBUS address.

The SmartMotor supports different protocols over the optional CAN port. CANopen and DeviceNet are popular industrial networks that use CAN. If a controller is communicating to a group of SmartMotors as follower devices under either of these standard protocols, the Combitronic protocol can still function, and it will not be detected by the CANopen or DeviceNet controller.

NOTE: To operate properly, the CAN network must have all devices set to the same baud rate.

The next command sets up and supports the CAN port of the motor:

```
CADDR=formula
```

where formula may be from 1 to 127. The setting is stored in the EEPROM, but you must reboot the motor to activate the setting.

```
CBAUD=formula
```

where formula may be one of these: 1000000, 800000, 500000, 250000, 125000, 50000 or 20000.

To set the SL17 SmartMotor to "auto address" (automatically assign the CAN address), use this CADDR command:

```
CADDR=-1
```

(CADDR=255 has the same effect.) Note that this setting is for the SL17 SmartMotor only; if used with any other motor, it will have no effect.

To read the CAN address, use this CADDR command:

```
var=CADDR
```

where var is any variable. Then you can use the PRINT(var) command to print the CAN address to the Terminal window.

RCADDR also reports the current value of the CAN ID either set from EEPROM as static address 1-127, or from auto-address 1-127. When the CADDR=formula command is issued, it will cause RCADDR to report the new address even through a reboot is required to actually activate and use the new address. If CADDR=-1 or 255 (auto-address for the SL17 SmartMotor only), then 255 is reported until a reboot. Then RCADDR reports a default address of 120. After an auto-address is assigned by a controller, RCADDR reports the address that the motor was assigned, i.e., 1, 2, 3..., etc.

To set the CAN address to the serial address on all motors, use this CADDR command:

```
<SerialMotorNumber>CADDR=<address>
```

For example, 3CADDR=10 sets serial motor 3's CAN address to 10; 0CADDR=ADDR would globally set every serial motor's CAN address to its serial address.

NOTE: SmartMotor commands like 0CADDR=... or 0ADDR=... with a leading number really send a corresponding address byte (i.e., "0", which is hex 80 or decimal 128). This can be seen by viewing the serial data with the Serial Data Analyzer ("sniffer") tool, which is available on the SMI software View menu.

EXAMPLE: (Code sets CAN address to motor address and resets all motors. Motors should be addressed on serial RS-232 chain first.)

NOTE: Issue these commands at serial port (SMI Terminal window) only. The "0" in front of these commands will not be recognized by a user program.

```
0CADDR=ADDR    'Set if not same as motor address
0Z             'Reset all motors to enable CAN address
```

RELATED COMMANDS:

R ADDR=formula *Address (for RS-232 and RS-485) (see page 261)*

CANCTL(function,value) *CAN Control (see page 359)*

R CBAUD=formula *CAN Baud Rate (see page 363)*

APPLICATION:	Communications control
DESCRIPTION:	Get CAN bus error or status information
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RCAN, RCAN(arg) See the detailed description for values of arg
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	See the detailed description
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M) requires CAN option; 6.4.2.x (D)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The CAN command is used to get (read) an error or other status information about the CAN bus. The user should not assume an error is indicated simply because a value greater than 0 is reported. Several of the bits are for information only and do not indicate an error.

If an error condition exists, then status word 2, bit 4 will indicate so. A user program or a manual report using RCAN provides further detail about the error condition after the CAN error status bit is indicated in status word 2, bit 4. RCAN also indicates information about the homing operation.

The SmartMotor supports different protocols over the optional CAN port. CANopen and DeviceNet are popular industrial networks that use CAN. If a controller is communicating to a group of SmartMotors as follower devices under either of these standard protocols, the Combitronic protocol can still function, and it will not be detected by the CANopen or DeviceNet controller.

NOTE: To operate properly, the CAN network must have all devices set to the same baud rate.

The next command sets up and supports the CAN port of the motor:

CADDR=formula

where formula may be from 1 to 127. The setting is stored in the EEPROM, but you must reboot the motor to activate the setting.

CBAUD=formula

where formula may be one of these: 1000000, 800000, 500000, 250000, 125000, 50000 or 20000.

Specific features are based on the fieldbus network being used. See the corresponding SmartMotor fieldbus guide for more details.

Homing commands (MH, HM_MTHD, etc.) rely on RCAN(3) to get feedback on the current state of the homing process. Bit 13 (value 8192) indicates homing error. While bits 10 (value 1024) and 12 (value 4096) indicate complete.

For more details on SmartMotor homing operations, see the *SmartMotor Homing Procedures and Methods Application Note*.

EXAMPLE:

```
C1          'Subroutine C1
IF CAN&16   'Test CAN word at value 16 (position 5)
    PRINT("USER TRIED COMBITRONIC READ FROM BROADCAST ADDRESS",#13)
ENDIF
RETURN
```

RELATED COMMANDS:

R CADDR=formula *CAN Address (see page 355)*

CANCTL(function,value) *CAN Control (see page 359)*

R CBAUD=formula *CAN Baud Rate (see page 363)*

CANCTL(function,value)

CAN Control

APPLICATION:	Communications control
DESCRIPTION:	Sets the CAN attributes for industrial network protocols (CANopen, DeviceNet, etc.)
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	Input: CANCTL(function,value) function: >= 0 value: -2147483648 to 2147483647
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M) requires CAN option; 6.4.2.x (D)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The CANCTL command is used to control CAN network features. Commands execute based on the function argument to control CAN functions. For example:

- **function = 1:** Reset the CAN MAC and all errors. Resets the CANopen stack, PROFIBUS stack or DeviceNet stack depending on firmware type. Value is ignored.
- **function = 5:** Set timeout for Combitronic. Value is in milliseconds; the default is 30.

Specific features are based on the fieldbus network being used. See the corresponding SmartMotor fieldbus guide for more details.

For more details on SmartMotor homing operations, see the *SmartMotor Homing Procedures and Methods Application Note*.

EXAMPLE:

```
EIGN (W, 0)      'Make all onboard I/O inputs
ZS              'Clear errors
CANCTL (5, 100) 'Set Combitronic timeout to 100ms
END
```

RELATED COMMANDS:

R CADDR=formula *CAN Address (see page 355)*

R CAN, CAN(arg) *CAN Bus Status (see page 357)*

R CBAUD=formula *CAN Baud Rate (see page 363)*

CASE formula

Case Label for SWITCH Block

APPLICATION:	Program execution and flow control
DESCRIPTION:	CASE label for SWITCH program control block
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	Can only be executed from within user program
REPORT VALUE:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The CASE command provides a label for defining a specific block of code that is referred to from a SWITCH formula.

The execution time is similar to the equivalent IF formula control block. Therefore, placing the most likely CASE values at the top of the CASE list will yield faster program execution times.

At execution time, the program interpreter evaluates the SWITCH formula value and then tests the CASE numbers for an equal value in the programmed order.

- If the SWITCH formula value does equal the CASE number, then program execution continues with the command immediately after.
- If the SWITCH formula value does not equal the CASE number, then the next CASE statement is evaluated.
- If the SWITCH formula value does not equal any CASE number, then the DEFAULT entry point is used.
- If the SWITCH formula value does not equal any CASE number and there is no DEFAULT case, then program execution passes through the SWITCH to the ENDS without performing any commands.

If a BREAK is encountered, then program execution branches to the instruction or label after the ENDS of the SWITCH control block. BREAK can be used to isolate CASEs. Without BREAK, the CASE number syntax is transparent and program execution continues at the next instruction. That is, you will run into the next CASE number code sequence.

Each SWITCH control block must have at least one CASE number defined plus one, and only one, ENDS statement. SWITCH is not a valid terminal command — it is only valid within a user program.

EXAMPLE:

Consider this code fragment:

```

SWITCH v
  CASE 1
    PRINT (" v = 1 ",#13)
  BREAK
  CASE 2
    PRINT (" v = 2 ",#13)
  BREAK
  CASE 3
    PRINT (" v = -23 ",#13)
  BREAK
  DEFAULT
    PRINT ("v IS NOT 1, 2 OR -23",#13)
  BREAK
ENDS

```

The first line, SWITCH v, lets the SmartMotor™ know that it is checking the value of the variable v. Each subsequent CASE begins the section of code that tells the SmartMotor what to do if v is equal to that case.

EXAMPLE:

```

a=-3                                     'Assign a value
WHILE a<4
  PRINT (#13,"a=",a," ")
  SWITCH a                               'Test the value
    CASE 3
      PRINT ("MAX VALUE",#13)
    BREAK
    CASE -1                               'Negative test values are valid
    CASE -2                               'Note no BREAK here
    CASE -3
      PRINT ("NEGATIVE")
    BREAK                                 'Note use of BREAK
    CASE 0
      PRINT ("ZERO")                     'Zero test value is valid
      PRINT ("ZERO")                     'Note order is random
      DEFAULT                             'The default case
      PRINT ("NO MATCH VALUE")
    BREAK
  ENDS                                   'Need not be numerical
  a=a+1
LOOP
END

```

Program output is:

```
a=-3 NEGATIVE
a=-2 NEGATIVE
a=-1 NEGATIVE
a=0 ZERO
a=1 NO MATCH VALUE
a=2 NO MATCH VALUE
a=3 MAX VALUE
```

RELATED COMMANDS:

BREAK Break from CASE or WHILE Loop (see page 331)
DEFAULT Default Case for SWITCH Structure (see page 388)
ENDS End SWITCH Structure (see page 443)
SWITCH formula Switch, Program Flow Control (see page 766)

CBAUD=formula CAN Baud Rate

APPLICATION:	Communications control; supports the DS2020 Combitronic system
DESCRIPTION:	Gets/sets the CAN baud rate (not used for PROFIBUS); value is retained in EEPROM between power cycles
EXECUTION:	Requires reboot of motor for change to take effect
CONDITIONAL TO:	N/A
LIMITATIONS:	Requires reboot of motor for change to take effect
READ/REPORT:	RCBAUD
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	Baud
RANGE OF VALUES:	20000, 50000, 125000, 250000, 500000, 800000,1000000
TYPICAL VALUES:	20000, 50000, 125000, 250000, 500000, 800000,1000000
DEFAULT VALUE:	125000 (Factory EEPROM setting)
FIRMWARE VERSION:	5.x (D/M); 6.4.2.x (D); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The CBAUD command is used to get (read) and set the CAN baud rate:

- =CBAUD
Get CAN baud rate.
- CBAUD=
Set CAN baud rate.

The SmartMotor supports different protocols over the optional CAN port. CANopen and DeviceNet are popular industrial networks that use CAN. If a controller is communicating to a group of SmartMotors as follower devices under either of these standard protocols, the Combitronic protocol can still function, and it will not be detected by the CANopen or DeviceNet controller.

NOTE: To operate properly, the CAN network must have all devices set to the same baud rate.

The next command sets up and supports the CAN port of the motor:

CADDR=formula

where formula may be from 1 to 127. The setting is stored in the EEPROM, but you must reboot the motor to activate the setting.

CBAUD=formula

where formula may be one of these: 1000000, 800000, 500000, 250000, 125000, 50000 or 20000.

NOTE: Unlike the CADDR command, the CBAUD value has no effect on PROFIBUS, where the baud rate is auto-detected.

EXAMPLE:

```
EIGN (W,0)      'Make all onboard I/O inputs
ZS              'Clear errors
CBAUD=1000000  'Set CAN network baud rate to 1,000,000 bps
END
```

RELATED COMMANDS:

R CADDR=formula *CAN Address (see page 355)*
R CAN, CAN(arg) *CAN Bus Status (see page 357)*
CANCTL(function,value) *CAN Control (see page 359)*

CCHN(type,channel)

Close Communications Channel (RS-232 or RS-485)

APPLICATION:	Communications control
DESCRIPTION:	Close a communications channel
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	Type= RS2, RS4 Channel = 0 or 1
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M) CCHN channel 1 requires: 5.x (D/M); 6.4.2.x (D)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

CCHN(type,channel) closes the specified communications channel, where "type" is the communications mode, and "channel" is the COM port to close. This command flushes the serial port buffer — any characters in the buffer will be lost.

For a D-style motor, the channel 0 COM port (COM0) is opened by default at power-up; that COM port can only be RS-232 or RS-485, while channel 1 (COM1) can only be RS-485.

These are valid CCHN commands:

```
CCHN(RS2,0) 'Close the channel 0 RS-232 port
CCHN(RS4,1) 'Close the channel 1 RS-485 port
```

After power up or a Z reset command, channel 0 is opened as RS-232 by default.

For an M-style motor, only the channel 0 COM port can be opened. It is only RS-485 and is opened by default at power-up.

These are valid CCHN commands:

```
CCHN(RS4,0) 'Close the channel 0 RS-485 port
```

After power up or a Z reset command, channel 0 is opened as RS-485 by default.

For systems using DMX, these CCHN commands are used to close a DMX channel:

- CCHN(DMX,1) for Class 5 and Class 6 D-style motors
- CCHN(DMX,0) for Class 5 and Class 6 M-style motors

EXAMPLE:

```
EIGN (W, 0)   'Make all onboard I/O inputs  
ZS           'Clear errors  
CCHN (RS2, 0) 'Close main RS-232 communications port 0  
END
```

RELATED COMMANDS:

OCHN(...) *Open Channel (see page 632)*
Z *Total CPU Reset (see page 846)*

CHN(channel) Communications Error Flag

APPLICATION:	Communications control
DESCRIPTION:	Get the communications error flags from the specified channel: CHN(0): RS-232 communications error flag CHN(1): RS-485 communications error flag
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RCHN(channel)
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Set of four binary state flags
RANGE OF VALUES:	Channel = 0 or 1
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The read-only function, CHN, holds binary-code, historical error information for serial channels 0 or 1 on the SmartMotor™, which are specified as:

- CHN(0): Channel 0 communications error flag
- CHN(1): Channel 1 communications error flag

The command gives the 5-bit status of either serial port channels 0 or 1:

- Bit 0 = 1: Receive buffer has overflowed on the specified channel
- Bit 1 = 1: Framing error occurred on the specified channel
- Bit 2 = 1: N/A
- Bit 3 = 1: Parity error occurred on the specified channel
- Bit 4 = 1: Timeout occurred in command mode on the specified channel

CHN is read only. It can be reported through RCHN. For example, if RCHN(0) returns an 8, it means that a parity error was detected on channel 0; if RCHN(0) equals zero, no error has been detected since opening channel 0.

EXAMPLE: (test individual flags)

```
IF CHN(0) &8
    PRINT("HOST CHANNEL - parity error occurred")
ELSEIF CHN(0) &1
    PRINT("HOST CHANNEL - buffer overflow")
ENDIF
```

EXAMPLE: (test all flags)

```
IF CHN(0) !=0
    PRINT("SERIAL ERROR !!")
ENDIF
```

RELATED COMMANDS:

CCHN(type,channel) *Close Communications Channel (RS-232 or RS-485) (see page 365)*

OCHN(...) *Open Channel (see page 632)*

ZS *Global Reset System State Flag (see page 858)*



APPLICATION:	System
DESCRIPTION:	Value (in milliseconds) of the firmware system clock
EXECUTION:	N/A
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RCLK
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	Number
RANGE OF VALUES:	-2147483648 to 2147483647
TYPICAL VALUES:	N/A
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	CLK:3=1234, a=CLK:3, RCLK:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

CLK is an independent, free-running, read/write counter. It is reset to zero on a hardware or software reset, and it counts once per millisecond.

The user may assign a value to this counter at any time. The size of CLK is 32 bits (signed). At the value 2147483647, it will roll over to the value -2147483648 and continue counting up. The time required to start from 0 and reach the roll-over value is 24.8 days.

EXAMPLE: The next example pauses for one second.

```
WAIT=1000           'Pause for one sec
```

EXAMPLE:

This example uses code within a WHILE loop that executes during the pause.

```
CLK=0              'Initialize clock
WHILE CLK<1000    'Loop one sec
LOOP
```

RELATED COMMANDS:

WAIT=formula *Wait for Specified Time (see page 835)*

COMCTL(function,value)

Serial Communications Control

APPLICATION:	Communications control
DESCRIPTION:	Special configuration options for communications channels
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	Input: COMCTL(function,value) function: >= 1 value: See details in next table
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The COMCTL(function, value) command is used for serial communications control with the DMX protocol. The next table provides details on the possible action settings, their corresponding functions and allowed value settings.

'function'	'value'	Comm. protocol	Description
1	1 to 512	DMX	Set base DMX slot/channel; default is 1 at power-up.
2	1 to 102	DMX	Set number of DMX slots/channels to accept; default is 1 at power-up.
3	1 to 512	DMX	Sync on DMX slot/channel; default is 512 at power-up.
4	0 to 101	DMX	Allows for the selection of the aw[] register where the DMX data begins loading; default is 0 at power-up.

For more details on using the SmartMotor with the DMX protocol, see the *Moog Animatics SmartMotor™ DMX Guide*.

EXAMPLE:

```

'Variables for DMX control:
b=1 'Set DMX base address Valid Address: 1 thru 512.
n=3 'Set number of DMX channels to use.
'NOTE: max that may be summed is 3 or 24 bit position unsigned int.
s=0 'First motor array variable index to use starting with aw[s].

'Configure DMX data usage and motor variable storage:
IF n>3 PRINT("n too large.",#13) END ENDIF
    'Limit "n" based on a max of 3 bytes.
IF b>(513-n) PRINT("b too large.",#13) END ENDIF
    'Limit "b" based on max data slot.
IF s>(102-n) PRINT("s too large.",#13) END ENDIF
    'Limit "s" to max array value.
q=b+n-1 'Last data channel used (will be trigger when data received).
COMCTL(1,b) 'Set base DMX channel to value from CADDR.
COMCTL(2,n) 'Accept 1 DMX channel of data.
COMCTL(3,q) 'Status word 12 bit 2 will be set to 1 when
    'channel "q" arrives.
COMCTL(4,s) 'Set start of array index storage (good for
    'bypassing Cam mode dynamic array).
OCHN(DMX,1,N,250000,2,8,D) 'Open DMX channel: COM 1 (Class 5/6 D), no parity,
    '250 kBd, 2 stop, 8 data, datamode.

```

RELATED COMMANDS:

OCHN(...) *Open Channel (see page 632)*

APPLICATION:	Math function
DESCRIPTION:	Gets the cosine of the input value
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RCOS(value)
WRITE:	N/A
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Degrees input
RANGE OF VALUES:	Input in degrees (floating-point): 0.0 to 360.0 (larger values can be used, but it is not recommended; user should keep range within modulo 360) Output: +/- 1.0 (floating-point)
TYPICAL VALUES:	Input in degrees (floating-point): 0.0 to 360.0 (larger values can be used, but it is not recommended; user should keep range within modulo 360) Output: +/- 1.0 (floating-point)
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

COS takes an input angle in degrees and returns a floating-point cosine:

$$af[1]=\text{COS}(arg)$$

where *arg* is in degrees, and may be an integer (i.e., *a*, *aw*[0]) or floating-point variable (i.e., *af*[0]). Integer or floating-point constants may also be used (i.e., 23 or 23.7, respectively).

This command cannot have within the parenthesis: math operators, other parenthetical functions, or a Combitronic request from another motor. For example, *x*=FABS(PA) is allowed, but *x*=FABS(PA:3) is not allowed.

The result of this function is a floating-point type. If used in an equation, the operations in the equation that are processed after this function are automatically promoted to a float. This is dependent on the mathematical order of operations in the equation. As with other equations (e.g., *x*=*a*+*b*), the variable to the left of "=" may be an integer variable to accept the result. However, the value will be truncated to fit to that integer type. For example, the assignment "*aw*[0]=" will drop any fractional amount and truncate the result to the range -32768 to 32767 (*aw*[0]=100.5 will report as 100, and *aw*[0]=40000.0 will report as -25536).

Although the floating-point variables and their standard binary operations conform to IEEE-754 double precision, the floating-point square root and trigonometric functions only produce IEEE-754 single-precision results. For more details, see Variables and Math on page 198.

EXAMPLE:

```
af[0]=COS(57.3)      'Set array variable = COS(57.3)
Raf[0]              'Report value of af[0] variable
RCOS(57.3)          'Report COS(57.3)
af[1]=42.3          '42.3 degrees
af[0]=COS(af[1])    'Variables may be put in the parenthesis.
Raf[0]
END
```

Program output is:

```
0.540240287
0.540240287
0.739631056
```

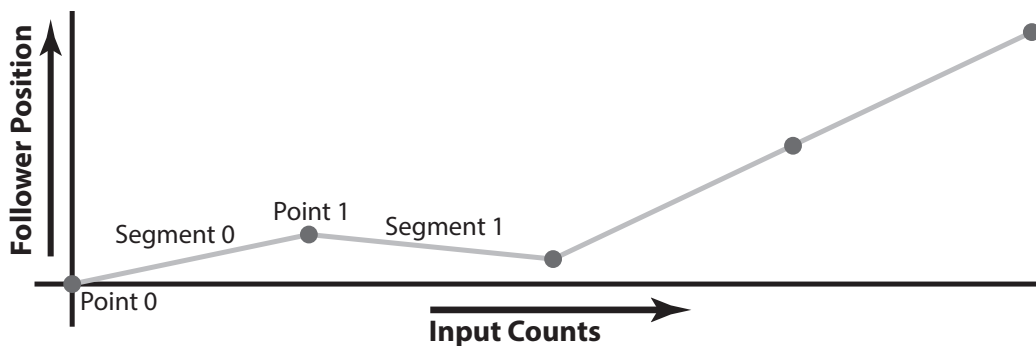
RELATED COMMANDS:

R ACOS(value) *Arccosine (see page 259)*
R ASIN(value) *Arcsine (see page 284)*
R ATAN(value) *Arctangent (see page 289)*
R SIN(value) *Sine (see page 738)*
R TAN(value) *Tangent (see page 775)*

APPLICATION:	Motion control
DESCRIPTION:	Reads the current cam pointer for the Cam table
EXECUTION:	Immediate
CONDITIONAL TO:	Cam mode active
LIMITATIONS:	N/A
READ/REPORT:	RCP
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Cam segments (not controller counts)
RANGE OF VALUES:	0-65536
TYPICAL VALUES:	0-750 (maximum storage capacity of data points in EEPROM)
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The CP command reads the current Cam pointer used by the Cam table. When running in Cam mode, the trajectory interpolates between the points entered in the Cam table. These segments in between are numbered starting from 0. For example, segment 0 has point 0 at the low end of the segment and point 1 at the high end.



Note that CP reports the segment and not the controller input value to the Cam table. Segments are typically many controller counts long. The length of segments can either be fixed or variable depending on the initial configuration used when the table is created. In either case, the length of segments in terms of controller counts is specified when writing the table.

The number of segments is primarily limited by the amount of storage available in the Cam table.

EXAMPLE:

```

EIGN (w, 0)
ZS
MF0
O=0
CTE (1)           'Erase all cam tables
CTA (5,1000)      'Add cam table, 5 points, 1000 counts apart
CTW (0)           '1st cam point
CTW (500)
CTW (4000)
CTW (2000)
CTW (0)
SRC (1)           'Use external encoder input
MCE (1)           'Spline cam table
MFMUL=1           'Numerator
MFDIV=1           'Denominator
MC                'Mode: Cam
G                'Go
WHILE CP<3 LOOP  'While we are below cam point 3, loop
OS (4)           'After cam point 3 is passed, turn on output 4
END

```

RELATED COMMANDS:

CTA(points,seglen[,location]) *Cam Table Attribute (see page 376)*

CTE(table) *Cam Table Erase (see page 378)*

R CTT *Cam Table Total in EEPROM (see page 382)*

CTW(pos[,seglen][,user]) *Cam Table Write Data Points (see page 383)*

MC *Mode Cam (Electronic Camming) (see page 555)*

MCE(arg) *Mode Cam Enable () (see page 558)*

MCW(table,point) *Mode Cam Where (Start Point) (see page 562)*

CTA(points,seglen[,location])

Cam Table Attribute

APPLICATION:	Motion control
DESCRIPTION:	Creates a new Cam table in memory: number of records, segment length, memory area (volatile or nonvolatile)
EXECUTION:	Immediate
CONDITIONAL TO:	Any pre-existing tables in nonvolatile memory
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	Input: points: 2-2147483647 seglen: 0-65535 location: 0, 1
TYPICAL VALUES:	Input: points: 2-750 seglen: 0-65535 location: 0, 1
DEFAULT VALUE:	See details. Parameter 3 (location) is optional; if omitted, then location 1 is used.
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

This command adds a Cam table to memory with the supplied parameters. The table can use either EEPROM memory (default) or the data variable space. After this table has been added, the CTW command will write data points into it (CTW writes to the most recent table).

- Parameter 1 "points": Specifies the number of points in the table.
- Parameter 2 "Segment length": Specifies the controller encoder distance between each point. If exp2 is set to 0, then the distance is specified per data record through the CTW command.
- Parameter 3 (Optional): Specifies if this is a table in user variables or EEPROM. By default, if parameter 3 is omitted or set to 1, then EEPROM is chosen. Up to ten tables (numbered from 1 through 10) can exist in EEPROM location. The tables are added sequentially after any existing tables in EEPROM.

If parameter 3 is 0, then the user array location is chosen (a1[0] through a1[50]). Only one table can exist in the user variables.

EXAMPLE: (Subroutine erases EEPROM tables and sets up Cam table)

C40

```

CTE(1)           'Erase all EEPROM tables
CTA(15,8000)
CTW(0)           'CP=0 {cam pointer or cam index pointer}
CTW(500)         'CP=1
CTW(4000)        'CP=2
CTW(20000)
CTW(45000)
CTW(50000)
CTW(60000)
CTW(65000)
CTW(55000,0,1)  'Turn on Bit 0 Status Word 8
CTW(46000)      'Will turn off at this point
CTW(45000,0,2)  'Turn on Bit 1 Status Word 8
CTW(8000)       'Will turn off at this point
CTW(4000)
CTW(500)
CTW(0)          'CP=14

```

RETURN

RELATED COMMANDS:

R CP *Cam Pointer for Cam Table (see page 374)*

CTE(table) *Cam Table Erase (see page 378)*

R CTT *Cam Table Total in EEPROM (see page 382)*

CTW(pos[,seglen][,user]) *Cam Table Write Data Points (see page 383)*

MC *Mode Cam (Electronic Camming) (see page 555)*

MCE(arg) *Mode Cam Enable () (see page 558)*

MCW(table,point) *Mode Cam Where (Start Point) (see page 562)*

CTE(table) Cam Table Erase

APPLICATION:	Motion control
DESCRIPTION:	Erase the Cam tables
EXECUTION:	Immediate
CONDITIONAL TO:	Existence of tables in nonvolatile memory
LIMITATIONS:	Tables cannot be individually deleted
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	Input: table: 1-10
TYPICAL VALUES:	1
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

CTE is used to erase the Cam tables. To erase all EEPROM tables, choose CTE(1).

By choosing a number higher than 1, lower table numbers can be preserved. If, for example, there were three tables stored, CTE(2) would erase tables 2 and 3, but not table 1.

NOTE: The feature that allows partial erasure may not be available in future generations of motors.

CTE(0) is not defined; therefore, it will not change the table in RAM memory.

EXAMPLE: (Subroutine erases EEPROM tables and sets up Cam table)

C40

```

CTE (1)           'Erase all EEPROM tables
CTA (15,8000)
CTW (0)           'CP=0 {cam pointer or cam index pointer}
CTW (500)         'CP=1
CTW (4000)        'CP=2
CTW (20000)
CTW (45000)
CTW (50000)
CTW (60000)
CTW (65000)
CTW (55000,0,1)  'Turn on Bit 0 Status Word 8
CTW (46000)      'Will turn off at this point
CTW (45000,0,2)  'Turn on Bit 1 Status Word 8
CTW (8000)        'Will turn off at this point
CTW (4000)
CTW (500)
CTW (0)           'CP=14

```

RETURN

RELATED COMMANDS:

R CP *Cam Pointer for Cam Table (see page 374)*

CTA(points,seglen[,location]) *Cam Table Attribute (see page 376)*

R CTT *Cam Table Total in EEPROM (see page 382)*

CTW(pos[,seglen][,user]) *Cam Table Write Data Points (see page 383)*

MC *Mode Cam (Electronic Camming) (see page 555)*

MCE(arg) *Mode Cam Enable () (see page 558)*

MCW(table,point) *Mode Cam Where (Start Point) (see page 562)*



APPLICATION:	Motion control
DESCRIPTION:	Encoder counter reading: CTR(0)=internal encoder counter; CTR(1)=external encoder counter
EXECUTION:	Updated once each PID sample
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RCTR(enc)
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Encoder counts
RANGE OF VALUES:	-2147483648 to 2147483647
TYPICAL VALUES:	0
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	RCTR(0):3, x=CTR(0):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The CTR command can be used as shown:

- The CTR(enc) command allows access to the internal or external encoders
- The CTR(0) command will always access the internal encoder
- The CTR(1) command will always access the external encoder

If the ENCO or ENC1 commands are used, the CTR commands will still access the encoders as described above.

Note that the O= or OSH= commands will affect the encoder selected by the ENCO (internal encoder) or ENC1 (external encoder) command. Therefore, if ENCO is commanded, then O= and OSH= commands will affect CTR(0). However, if ENC1 is commanded, then O= or OSH= will affect CTR(1) because the external encoder is being used to measure the motor's position and to close the PID loop.

MF0 and MS0 will both set CTR(1) to zero in addition to selecting the type of input (quadrature or step/direction, respectively). However, those commands will only have an effect in ENCO mode (the default mode at power-up).

EXAMPLE:

```
EIGN (w, 0)
ZS
WHILE CTR(1)==0 LOOP    'Wait for motion on external encoder
PRINT ("INPUT ENCODER HAS MOVED", #13)
END
```

RELATED COMMANDS:

ENC0 *Encoder Zero (Close Loop on Internal Encoder) (see page 432)*
ENC1 *Encoder Zero (Close Loop on External Encoder) (see page 433)*
MC *Mode Cam (Electronic Camming) (see page 555)*
MF0 *Mode Follow, Zero External Counter (see page 578)*
MFR *Mode Follow Ratio (see page 600)*
MS0 *Mode Step, Zero External Counter (see page 616)*
MSR *Mode Step Ratio (see page 618)*
O=formula, O(trj#)=formula *Origin (see page 628)*
OSH=formula, OSH(trj#)=formula *Origin Shift (see page 642)*

CTT Cam Table Total in EEPROM

APPLICATION:	Motion control
DESCRIPTION:	Gets the total number of Cam tables in EEPROM
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RCTT
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	Output: 0-10
TYPICAL VALUES:	Output: 0-10
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The CTT command reports the number of Cam tables stored in EEPROM. The table in RAM memory is not included in that number.

EXAMPLE:

```

C1          'Cam check subroutine
IF CTT==0   'If no cam tables have been created
    GOSUB2   'Go to cam creation subroutine
ENDIF
RETURN

```

```

C2  'Cam creation subroutine code here
RETURN

```

RELATED COMMANDS:

R CP *Cam Pointer for Cam Table (see page 374)*
CTA(points,seglen[,location]) *Cam Table Attribute (see page 376)*
CTE(table) *Cam Table Erase (see page 378)*
CTW(pos[,seglen][,user]) *Cam Table Write Data Points (see page 383)*
MC *Mode Cam (Electronic Camming) (see page 555)*
MCE(arg) *Mode Cam Enable () (see page 558)*
MCW(table,point) *Mode Cam Where (Start Point) (see page 562)*

CTW(pos[,seglen][,user])

Cam Table Write Data Points

APPLICATION:	Motion control
DESCRIPTION:	Writes data points into Cam table
EXECUTION:	Immediate
CONDITIONAL TO:	CTA command must be called before CTW
LIMITATIONS:	Data points (pos) cannot exceed a difference of -8388608 or +8388607 from one to the next.
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	Input: pos: -2147483648 to 2147483647 seglen (segment length): 0 to 65535 user (user bits): 0 to 255
TYPICAL VALUES:	Input: pos: -2147483648 to 2147483647 seglen (segment length): 0 to 65535 user (user bits): 0 to 255
DEFAULT VALUE:	See details. The seglen (segment length) and user (user bits) parameters are optional; if omitted, then segment length must have been defined in CTA command; user bits assume 0 if omitted.
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The CTW command writes to the table addressed by the most recent CTA command. CTW writes to either the tables stored in the EEPROM or the user array.

NOTE: Typically, the actual Cam table would not be part of the program that executes the mode. The SMI software contains an Import Cam tool to facilitate Cam table generation.

- pos (position): The position coordinate of the motor for that data point. The first point in the table should be set to 0 to avoid confusion. When the table is run, the currently commanded motor position seamlessly becomes the starting point of the table. The first point of the table is kept at 0, which makes it easier to see that all of the data points are relative to that starting point.

- **seglen** (segment length): If the Cam table was specified as variable length in the CTA command, then parameter 2 is required for each data point. It is optional when using a fixed-length Cam table (specified in the CTA command). This parameter represents the absolute distance of the encoder source beginning from the start of the table. For reasons similar to *pos*, *seglen* should also be 0 for the first data point specified.

If you wish to use the optional user parameter, then the *seglen* parameter must be used (set to the default: 0).

- **user** (user bits): (Optional) Defines Cam user bits and Spline mode override. It is an 8-bit binary value where:
 - Bit 0-5: User may apply as desired to Cam status bits 0-5 of status word 8; the segment between the *previous* point and this point will apply these bits.
 - Bit 6: Factory reserved — user should leave as 0.
 - Bit 7: When set to 0, there is no special override of Spline mode; when set to 1, the segment between the *previous* point and this point are forced into linear interpolation. Bit 7 has no effect when MCE has chosen linear mode.

When loading Cam tables, it is important to note the table capacity. As mentioned previously:

- When a Cam table is stored in user array memory (a[0]-a[50]), 52 points can be stored as fixed-length segments; 35 points are possible when variable-length segments are used.
- When Cam tables are written to EEPROM memory, significantly more data can be written:
 - For fixed-length segments, there is space for at least 750 points.
 - For variable length segments, at least 500 points can be written.

EXAMPLE: (Subroutine erases EEPROM tables and sets up Cam table)

C40

```

CTE (1)           'Erase all EEPROM tables
CTA (15,8000)
CTW (0)           'CP=0 {cam pointer or cam index pointer}
CTW (500)         'CP=1
CTW (4000)        'CP=2
CTW (20000)
CTW (45000)
CTW (50000)
CTW (60000)
CTW (65000)
CTW (55000,0,1)  'Turn on Bit 0 Status Word 8
CTW (46000)      'Will turn off at this point
CTW (45000,0,2)  'Turn on Bit 1 Status Word 8
CTW (8000)       'Will turn off at this point
CTW (4000)
CTW (500)
CTW (0)          'CP=14

```

RETURN

RELATED COMMANDS:

R CP *Cam Pointer for Cam Table (see page 374)*

CTA(points,seglen[,location]) *Cam Table Attribute (see page 376)*

CTE(table) *Cam Table Erase (see page 378)*

R CTT *Cam Table Total in EEPROM (see page 382)*

MC *Mode Cam (Electronic Camming) (see page 555)*

MCE(arg) *Mode Cam Enable () (see page 558)*

MCW(table,point) *Mode Cam Where (Start Point) (see page 562)*



APPLICATION:	Motion control
DESCRIPTION:	Read actual derivative error value
EXECUTION:	Updated once each PID sample
CONDITIONAL TO:	Servo active (MP, MV, etc., but not MT mode)
LIMITATIONS:	N/A
READ/REPORT:	RDEA
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Units of position error per PID cycle *65536
RANGE OF VALUES:	Output: -2147483648 to 2147483647
TYPICAL VALUES:	N/A
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	RDEA:3, x=DEA:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The DEA command is used to read back the rate of change of the PID position error. This value is averaged over four consecutive PID cycles and is in units of position error per PID cycle *65536. This measured value is used in dE/dt limiting, which can be set through the DEL command.

DEL (Derivative Error Limit) provides the safest and quickest method to fault a motor on sudden changes in load or detection of human interference.

The purpose of this limit is to act as a look-ahead on position error. Instead of just triggering on a raw position error based on how far behind the motor may be in a move, the processor is looking at how fast that position error changes.

dE/dt refers to the dynamic rate of change of position error. This results in an instant release of energy and less chance of damage to equipment or injury to machine operator. Under normal servo control position error limits, if the load collides against an object, the motor will not fault until the position error limit is reached. As a result, applied current and torque will increase until that condition is met. By adding an additional derivative limit on position error, the servo will fault within milliseconds of contact with an object.

EXAMPLE:

```
EIGN(W,0)   'Make all onboard I/O inputs
ZS          'Clear errors
MV          'Mode Velocity
VT=500000  'Velocity target
ADT=50      'Accel/Decel target
G           'Go
WHILE DEA <32768 & Bt LOOP
PRINT("Possible collision",#13)
END
```

RELATED COMMANDS:

R DEL=formula *Derivative Error Limit (see page 390)*

DELM(arg) *Derivative Error Limit Mode (see page 392)*

DEFAULT

Default Case for SWITCH Structure

APPLICATION:	Program execution and flow control
DESCRIPTION:	Default for SWITCH program control block
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	Must reside within a SWITCH and ENDS structure
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

DEFAULT allows controlled code execution in a SWITCH structure for results that are not evaluated by CASE. There can only be one DEFAULT statement per SWITCH control block.

DEFAULT is not a valid terminal command. It is only valid within a user program.

In the next example, DEFAULT is used when no CASE can be executed for the value of "v".

EXAMPLE:

```

SWITCH x
  CASE 1
    PRINT (" x = 1 ",#13)
  BREAK
  CASE 2
    PRINT (" x = 2 ",#13)
  BREAK
  CASE 3
    PRINT (" x = -23 ",#13)
  BREAK
  DEFAULT
    PRINT ("x IS NOT 1, 2 OR -23",#13)
  BREAK
ENDS

```

The first line, SWITCH x, tells the SmartMotor™ that it is checking the value of the variable x. The second line, CASE 1, begins the section of code that tells the SmartMotor what to do if x is equal to 1.

Similarly, the eighth line, CASE 3, tells what to do if x=3. Finally, DEFAULT tells what to do if none of the CASEs match the value of x.

EXAMPLE:

```

a=20
WHILE a
    SWITCH a-12
        CASE -4 PRINT ("-4 ") BREAK
        CASE -3 PRINT ("-3 ") BREAK
        CASE -2 PRINT ("-2 ") BREAK
        CASE -1 PRINT ("-1 ") BREAK
        CASE 0  BREAK
        CASE 1  PRINT ("+1 ") BREAK
        CASE 2  PRINT ("+2 ") BREAK
        CASE 3  PRINT ("+3 ") BREAK
        CASE 4  PRINT ("+4 ") BREAK
        DEFAULT PRINT ("D ")
    ENDS
a=a-1
LOOP

```

Program output is:

```
D D D D +4 +3 +2 +1 -1 -2 -3 -4 D D D D D D D
```

RELATED COMMANDS:

BREAK *Break from CASE or WHILE Loop (see page 331)*
CASE formula *Case Label for SWITCH Block (see page 360)*
ENDS *End SWITCH Structure (see page 443)*
SWITCH formula *Switch, Program Flow Control (see page 766)*



DEL=formula

Derivative Error Limit

APPLICATION:	Motion control
DESCRIPTION:	Get/set the derivative error limit
EXECUTION:	Immediate
CONDITIONAL TO:	Servo active (MP, MV, etc., not MT mode)
LIMITATIONS:	N/A
READ/REPORT:	RDEL
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	Units of position error per PID cycle *65536
RANGE OF VALUES:	Input: 0 to 2147483647
TYPICAL VALUES:	N/A
DEFAULT VALUE:	2147483647
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	DEL:3=1234, a=DEL:3, RDEL:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The DEL command is used to get (read) or set the dE/dt fault limit:

- =DEL
Gets the setting for dE/dt fault limit
- DEL=frm
Sets the dE/dt fault limit

When the actual dE/dt reaches the value of this setting, then the dE/dt fault is tripped and the motor will perform the fault reaction. The absolute value of the actual value is used so that both positive and negative values of dE/dt will be compared against the DEL setting. Also, the DELM command can modify the functionality to behave differently depending on direction of dE/dt relative to motor direction.

DEL (Derivative Error Limit) provides the safest and quickest method to fault a motor on sudden changes in load or detection of human interference.

The purpose of this limit is to act as a look-ahead on position error. Instead of just triggering on a raw position error based on how far behind the motor may be in a move, the processor is looking at how fast that position error changes.

dE/dt refers to the dynamic rate of change of position error. This results in an instant release of energy and less chance of damage to equipment or injury to machine operator. Under normal servo control position error limits, if the load collides against an object, the motor will not fault until the position error limit is reached. As a result, applied current and torque will increase until that condition is met. By

adding an additional derivative limit on position error, the servo will fault within milliseconds of contact with an object.

EXAMPLE:

`DEL=VT 'Set limit to commanded speed`

If dE/dt equals commanded velocity, then the motor just hit a hard stop. Normally, the motor would have to continue applying torque until the normal position error is exceeded. However, if DEL (dE/dt limit) is set to target velocity (VT), then as soon as contact is made with a hard stop, the controller will immediately fault without any wind up.

RELATED COMMANDS:

`R DEA` *Derivative Error, Actual (see page 386)*

`DELM(arg)` *Derivative Error Limit Mode (see page 392)*

DELM(arg)

Derivative Error Limit Mode

APPLICATION:	Motion control
DESCRIPTION:	Derivative error limit mode
EXECUTION:	Immediate
CONDITIONAL TO:	Servo active (MP, MV, etc., but not MT mode)
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	Assignment
UNITS:	N/A
RANGE OF VALUES:	0, 1
TYPICAL VALUES:	0, 1
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The DELM command allows the dE/dt fault limit to be reconfigured. By default, DELM is set to the value 0. This means that the absolute value of DEA is compared to DEL when determining if the dE/dt limit has been exceeded. This means that a disturbance to the motor in either direction can potentially lead to a fault if the disturbance is large enough.

This behavior can be changed by setting DELM(1). When that value is issued, the DEL limit is only reactive to disturbances that block the commanded direction of motion (i.e., something that attempts to cause the motor to run slower). Disturbances that attempt to cause the motor to run faster are ignored.

EXAMPLE:

```
EIGN(W,0)      'Make all onboard I/O inputs
ZS             'Clear
IF IN(6)==0    'Check Input 6
    DELM(1)    'Set derivative error limit mode to 1
ELSE DELM(0)   'Otherwise, set to 0
ENDIF
END
```

RELATED COMMANDS:

R DEA *Derivative Error, Actual (see page 386)*

R DEL=*formula* *Derivative Error Limit (see page 390)*

DFS(value)

Dump Float, Single

APPLICATION:	Data conversion
DESCRIPTION:	Get the af[index] variable in its raw 32-bit IEEE format
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RDFS(value)
WRITE:	N/A
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	Input: Any float value within 32-bit float range: ±1x10 ³⁸ Output: 32-bit integer -2147483648 to 2147483647
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The DFS(value) command allows the creation of a 32-bit value in IEEE-754 format. This allows export of floating-point values to external systems. This format may be needed for interchange.

The 32-bit value output by this function is *not* the integer rounded off from the input. It is an encoded number that includes the exponent of the floating point value. The output of this function does not have any usefulness within the SmartMotor programming language.

EXAMPLE:

```
af[0]=(a+b)/3.0      'Create a floating-point result.
al[0]=DFS(af[0])    'Dump the IEEE-754 32-bit representation
                    'into al[0].
PRINT(#ab[0],#ab[1],#ab[2],#ab[3]) 'Print the 4 bytes (32-bits)
                    'of this result to the serial port for the
                    'host PLC to interpret.
```

RELATED COMMANDS:

- R af[index]=formula *Array Float [index] (see page 267)*
- R ATOF(index) *ASCII to Float (see page 291)*
- R HEX(index) *Decimal Value of a Hex String (see page 489)*
- R LFS(value) *Load Float Single (see page 547)*



APPLICATION:	Program execution and flow control
DESCRIPTION:	Disable the specified interrupt or combination of interrupts
EXECUTION:	Immediate
CONDITIONAL TO:	Interrupts configured and enabled; ITR, EITR and ITRE
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	Input: DITR(interrupt), where interrupt is 0-7 DITR(W,mask), where mask is 0-255
TYPICAL VALUES:	Input: DITR(interrupt), where interrupt is 0-7 DITR(W,mask), where mask is 0-255
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	DITR(1):3 or DITR(W,7):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The DITR (Disable Interrupts) command is used to disable one or more interrupts.

NOTE: Use DITR() or EITR() before the STACK command to stop any pending interrupt events from reoccurring. Additionally, DITR() will prevent future calls.

DITR is written as:

- DITR(interrupt)
Where interrupt is used to specify an interrupt (0-7).
- DITR(W,mask)
A literal "W" is used as the first argument; the mask argument can select a combination of interrupts.

NOTE: The (W,mask) input requires this firmware:
for Class 5: 5.x.4.46 and later; for Class 6: 6.0.2.37 and later.

For an interrupt to work, it must be enabled at two levels: first, enable *individual* interrupts with the EITR() command using the interrupt number from 0 to 7 in the parentheses; second, enable *all* interrupts with the ITRE command. Similarly, *individual* interrupts can be disabled with the DITR()

command, and *all* interrupts can be disabled with the ITRD command. For more details, see the corresponding command-description pages.

NOTE: The user program must also be running for interrupts to take effect, the END and RUN commands will reset the state of the interrupts to defaults.

For more details on interrupt programming, see Interrupt Programming on page 195.

EXAMPLE: (Subroutine shows use of DTR, ETR, TMR and TWAIT)

```

C10          'Place a label
  IF PA>47000 'Just before 12 moves
    DTR(0)   'Disable interrupt
    TWAIT    'Wait till reaches 48000
    p=0      'Reset variable p
    PT=p     'Set target position
    G        'Start motion
    TWAIT    'Wait for move to complete
    ETR(0)   'Re-enable interrupt
    TMR(0,1000) 'Restart timer
  ENDIF
GOTO10      'Go back to label

```

RELATED COMMANDS:

EISM(x) *E-Configure Input as Sync Controller (see page 423)*

ETR(int) *Enable Interrupts (see page 424)*

ITR(Int#,StatusWord,Bit#,BitState,Label#) *Interrupt Setup (see page 517)*

ITRD *Interrupt Disable, Global (see page 520)*

ITRE *Enable Interrupts, Global (see page 522)*

RETURNI *Return Interrupt (see page 708)*



DT=formula

Deceleration Target

APPLICATION:	Motion control; supports the DS2020 Combitronic system
DESCRIPTION:	Get/set target deceleration only (does not change acceleration)
EXECUTION:	Buffered until a G command is issued
CONDITIONAL TO:	MP, MV, ADT=, AT=, G, X, PID _n (sample rate), encoder resolution
LIMITATIONS:	Must not be negative; effective value is rounded down to next even number
READ/REPORT:	RDT
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	(encoder counts / (sample ²)) * 65536 DS2020 Combitronic system: user increments / sec ² , see FD=e-expression on page 461
RANGE OF VALUES:	0 to 2147483647 DS2020 Combitronic system: 0 to 4294967295
TYPICAL VALUES:	2 to 5000 DS2020 Combitronic system: depends on FD
DEFAULT VALUE:	0 (for firmware 5.x.4.x and newer, default is set to 4) DS2020 Combitronic system: 0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	DT:3=1234, a=DT:3, RDT:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

NOTE: This command is affected by the SCALEA command. For details, see SCALEA(m,d) on page 724. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

Setting the buffered DT value determines the deceleration that will be used by subsequent position or velocity moves to calculate the required trajectory. Changing DT during a move will not alter the current trajectory unless a new G or X command is issued.

Acceleration is pre-scaled by 65536 and may range from 2 to 2147483647. A value of 0 is not valid. Due to internal calculations, odd values for this command are rounded up to an even value.

If the value for DT has not been set since powering up the motor, the value of AT= will be automatically applied to DT=. However, this should be avoided. Instead, always use the ADT= command to specify the value for AT and DT when they are the same. If the value needed for DT is different than AT, specify it with the DT= command.

Equations for Real-World Units:

Encoder resolution and sample rate can vary. Therefore, the general equations in the next table can be used to convert the real-world units of deceleration to a value for DT, where af[0] is already set with the real-world unit value. These equations force floating-point calculations to avoid overflow and maintain resolution. They can be placed in a user program, or they can be precalculated if the values of SAMP and RES are known (SAMP and RES can be reported from the terminal using the RSAMP and RRES commands, respectively). SAMP can change if the PID command is used. The value of RES can differ between motor models.

Input as value in af[0]	Equation
Radians/(Sec ²)	DT=((af[0]*RES)/(PI*2*((SAMP*1.0)*SAMP)))*65536 DS2020 Combitronic system: DT=((af[0]*FD)/(PI*2))
Encoder Counts/(Sec ²) DS2020 Combitronic system: User Increments/(Sec ²)	DT=(af[0]/((SAMP*1.0)*SAMP))*65536 DS2020 Combitronic system: AT=(af[0])
Rev/(Sec ²)	DT=((af[0]*RES)/((SAMP*1.0)*SAMP))*65536 DS2020 Combitronic system: DT=(af[0]*FD)
RPM/Sec	DT=((af[0]*RES)/(60.0*((SAMP*1.0)*SAMP)))*65536 DS2020 Combitronic system: DT=((af[0]*FD)/60)
RPM/Min	DT=((af[0]*RES)/(3600.0*((SAMP*1.0)*SAMP)))*65536 DS2020 Combitronic system: DT=((af[0]*FD)/3600)

A useful Scale Factor Multiplier code example, which also illustrates the use of af[], SAMP and RES, is shown in RES on page 702 and SAMP on page 722.

EXAMPLE:

```
MP           'Set position mode
DT=5000     'Set target deceleration
PT=20000   'Set absolute position
VT=500     'Set velocity
G           'Start motion
```

EXAMPLE:

```
DT=100     'Set buffered deceleration
VT=750     'Set buffered velocity
MV         'Set buffered velocity mode
G         'Start motion
```

RELATED COMMANDS:

ADT=formula *Acceleration/Deceleration Target (see page 263)*
ADTS=formula *Acceleration/Deceleration Target, Synchronized (see page 265)*
R AT=formula *Acceleration Target (see page 286)*
ATS=formula *Acceleration Target, Synchronized (see page 292)*
DTS=formula *Deceleration Target, Synchronized (see page 399)*
R EL=formula *Error Limit (see page 426)*
R FD=expression *Resolution to Set Units of Position/Velocity/Acceleration (see page 461)*
G *Start Motion (GO) (see page 473)*
MP *Mode Position (see page 613)*
MV *Mode Velocity (see page 624)*
PID# *Proportional-Integral-Differential Filter Rate (see page 654)*
R PT=formula *Position, (Absolute) Target (see page 690)*
R RES *Resolution (see page 702)*
R SAMP *Sampling Rate (see page 722)*
R VT=formula *Velocity Target (see page 828)*
X *Decelerate to Stop (see page 844)*

DTS=formula Deceleration Target, Synchronized

APPLICATION:	Motion control
DESCRIPTION:	Sets the synchronized (path) deceleration for a move
EXECUTION:	Immediate
CONDITIONAL TO:	PID _n
LIMITATIONS:	Must not be negative; 0 is not valid; effective value is rounded up to next even number
READ/REPORT:	None
WRITE:	Write only
LANGUAGE ACCESS:	Assignment
UNITS:	(encoder counts / (sample ²)) * 65536
RANGE OF VALUES:	0 to 2147483647
TYPICAL VALUES:	0 to 5000
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M), not ver. 5.32.x.x; 6.x (D/M), M requires EIP option
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

NOTE: This command requires a Combitronic-supported motor. Although this command does not support Combitronic syntax, it does use Combitronic communication to pass information between other motors.

NOTE: This command is affected by the SCALEA command. For details, see SCALEA(m,d) on page 724. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

Setting the synchronized (path) DTS value determines the deceleration that will be used by subsequent position or velocity moves to calculate the required trajectory. Changing DTS during a move will not alter the current trajectory unless a new G command is issued.

Acceleration is pre-scaled by 65536 and may range from 2 to 2147483647. A value of 0 is not valid. Due to internal calculations, odd values for this command are rounded up to an even value.

Equations for Real-World Units:

Encoder resolution and sample rate can vary. Therefore, the general equations in the next table can be used to convert the real-world units of acceleration to a value for DTS, where af[0] is already set with the real-world unit value. These equations force floating-point calculations to avoid overflow and maintain resolution. They can be placed in a user program, or they can be precalculated if the values of SAMP and RES are known (SAMP and RES can be reported from the terminal using the RSAMP and RRES commands, respectively). SAMP can change if the PID command is used. The value of RES can differ between motor models.

Input as value in af[0]	Equation
Radians/(Sec ²)	$DTS=((af[0]*RES)/(PI*2*((SAMP*1.0)*SAMP)))*65536$
Encoder Counts/(Sec ²)	$DTS=(af[0]/((SAMP*1.0)*SAMP))*65536$
Rev/(Sec ²)	$DTS=((af[0]*RES)/((SAMP*1.0)*SAMP))*65536$
RPM/Sec	$DTS=((af[0]*RES)/(60.0*((SAMP*1.0)*SAMP)))*65536$
RPM/Min	$DTS=((af[0]*RES)/(3600.0*((SAMP*1.0)*SAMP)))*65536$

A useful Scale Factor Multiplier code example, which also illustrates the use of af[], SAMP and RES, is shown in RES on page 702 and SAMP on page 722.

EXAMPLE: (Shows use of ATS, DTS and VTS)

```

EIGN (W, 0)      'Set all I/O as general inputs.
ZS              'Clear errors.
ATS=100         'Set synchronized acceleration target.
DTS=500         'Set synchronized deceleration target.
VTS=100000000  'Set synchronized target velocity.
PTS (500;1,1000;2,10000;3) 'Set synchronized target position
                  'on motor 1, 2 and 3.
GS              'Initiate synchronized move.
TSWAIT         'Wait until synchronized move ends.
END            'Required END.

```

RELATED COMMANDS:

ATS=formula *Acceleration Target, Synchronized (see page 292)*
ADTS=formula *Acceleration/Deceleration Target, Synchronized (see page 265)*
PID# *Proportional-Integral-Differential Filter Rate (see page 654)*
PRTS(...) *Position, Relative Target, Synchronized (see page 685)*
PRTSS(...) *Position, Relative Target, Synchronized, Supplemental (see page 688)*
PTS(...) *Position Target, Synchronized (see page 692)*
R PTSD *Position Target, Synchronized Distance (see page 695)*
PTSS(...) *Position Target, Synchronized Supplemental (see page 696)*
R PTST *Position Target, Synchronized Time (see page 698)*
VTS=formula *Velocity Target, Synchronized Move (see page 831)*



APPLICATION:	Motion control
DESCRIPTION:	Get current position error in real time
EXECUTION:	Updated once each PID sample
CONDITIONAL TO:	Servo active (MP, MV, etc., not MT mode)
LIMITATIONS:	N/A
READ/REPORT:	REA ; supports the DS2020 Combitronic system
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Encoder counts DS2020 Combitronic system: user increments / sec ² , see FD=e-expression on page 461
RANGE OF VALUES:	Output: -2147483648 to 2147483647
TYPICAL VALUES:	Magnitude limited to user set value of EL
DEFAULT VALUE:	1000
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	REA:3, x=EA:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

NOTE: This command is affected by the SCALEP command. For details, see SCALEP(m,d) on page 726. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

The EA command provides the current position error in real time. Position error is the difference in encoder counts between the desired trajectory position and the measured position. If the absolute value of EA exceeds the user value EL, the fault reaction is performed and Be (Position Error) status bits will be set to 1, within that PID servo sample. When the servo is off, EA reverts to zero because there is no longer a desired position.

NOTE: As acceleration (AC) is increased, a larger value of EL may be required. EL is unsigned, but EA may be positive or negative.

EXAMPLE:

```
IF EA<1234    'test value of EA
              'then do something
ENDIF

WHILE EA<1234 LOOP    'Loop while EA is less than 1234
                    'Then continue with program
```

EXAMPLE: (Routine homes motor against a hard stop)

```
MDS           'Using Sine mode commutation
KP=3200       'Increase stiffness from default
KD=10200      'Increase damping from default
F             'Activate new tuning parameters
AMPS=100      'Lower current limit to 10%
VT=-10000     'Set maximum velocity
ADT=100       'Set maximum accel/decel
MV            'Set Velocity mode
G             'Start motion
WHILE EA>-100 'Loop while position error is small
LOOP          'Loop back to WHILE
O=-100        'While pressed, declare home offset
S             'Abruptly stop trajectory
MP            'Switch to Position mode
VT=20000     'Set higher maximum velocity
PT=0          'Set target position to be home
G             'Start motion
TWAIT        'Wait for motion to complete
AMPS=1023     'Restore current limit to maximum
END           'End program
```

RELATED COMMANDS:

R EL=formula *Error Limit (see page 426)*

R FD=expression *Resolution to Set Units of Position/Velocity/Acceleration (see page 461)*

Echo Incoming Data on Communications Port 0

APPLICATION:	Communications control; supports the DS2020 Combitronic system
DESCRIPTION:	Enable motor echo of received channel 0 serial data
EXECUTION:	Immediate; remains in effect until otherwise commanded
CONDITIONAL TO:	N/A
LIMITATIONS:	Not for use with an RS-485 port such as an M-style motor, or a D-style motor with the RS485-ISO adapter
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	Motor defaults to ECHO_OFF (non-echo)
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The ECHO command causes the SmartMotor™ to retransmit (or echo out) all serial bytes on the transmit line that were received on the receive line of COM port 0. This retransmission occurs when the SmartMotor reads these bytes from the buffer, regardless of whether these bytes are command or individual data bytes. ECHO_OFF terminates the echo capability.

ECHO is required to pass serial bytes through a motor to the next motor in an RS-232 serial daisy-chain configuration, such as when the Add-A-Motor cables are used. Also, it is often used in single-motor applications for transmit verification.

NOTE: It is not recommended to use ECHO when the communications channel is an RS-485 port such as an M-style motor or a D-style motor with the RS485-ISO adapter. This mode of communication is half-duplex and is not compatible with the ECHO command.

EXAMPLE:

```
SADDR1  'Address the first motor
ECHO    'Echo for host data
PRINT(#128,"SADDR2",#13) '0SADDR2
WAIT=10                               'Allow time
PRINT(#130,"ECHO",#13)  '2ECHO
WAIT=10
PRINT(#130,"SLEEP",#13) '2SLEEP
WAIT=10
PRINT(#128,"SADDR3",#13) '0SADDR3
WAIT=10
PRINT(#131,"ECHO",#13)  '3ECHO
WAIT=10
PRINT(#128,"WAKE",#13)  '0WAKE
WAIT=10
```

RELATED COMMANDS:

ECHO1 *Echo Incoming Data on Communications Port 1 (see page 406)*
ECHO_OFF *Turn Off Echo on Communications Port 0 (see page 407)*
ECHO_OFF1 *Turn Off Echo on Communications Port 1 (see page 409)*

ECH00

Echo Incoming Data on Communications Port 0

APPLICATION:	Communications control
DESCRIPTION:	Enable motor echo of received channel 0 serial data
EXECUTION:	Immediate; remains in effect until otherwise commanded
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	Motor defaults to ECHO_OFF0 (non-echo)
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The ECH00 command causes the SmartMotor™ to retransmit (or echo out) all serial bytes on the transmit line that were received on the receive line of COM port 0. This retransmission occurs when the SmartMotor reads these bytes from the buffer, regardless of whether these bytes are command or individual data bytes. ECHO_OFF0 terminates the echo capability.

NOTE: It is not recommended to use ECHO when the communications channel is an RS-485 port such as an M-style motor or a D-style motor with the RS485-ISO adapter. This mode of communication is half-duplex and is not compatible with the ECHO command.

EXAMPLE:

```
EIGN(W,0)    'Make all onboard I/O inputs
ZS          'Clear errors
OCHN(RS4,0,N,9600,1,8,D) 'Open com port 0
ECH00      'Enable echo on com port 0
```

RELATED COMMANDS:

ECHO *Echo Incoming Data on Communications Port 0 (see page 403)*
 ECH01 *Echo Incoming Data on Communications Port 1 (see page 406)*
 ECHO_OFF *Turn Off Echo on Communications Port 0 (see page 407)*
 ECHO_OFF1 *Turn Off Echo on Communications Port 1 (see page 409)*
 ECHO_OFF1 *Turn Off Echo on Communications Port 1 (see page 409)*

Echo Incoming Data on Communications Port 1

APPLICATION:	Communications control
DESCRIPTION:	Enable motor echo of received channel 1 serial data
EXECUTION:	Immediate; remains in effect until otherwise commanded
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	Motor defaults to ECHO_OFF1 (non-echo)
FIRMWARE VERSION:	5.x (D/M); 6.4.2.x (D)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The ECHO1 command causes the SmartMotor™ to retransmit (or echo out) all serial bytes on the transmit line that were received on the receive line of COM port 1. This retransmission occurs when the SmartMotor reads these bytes from the buffer, regardless of whether these bytes are command or individual data bytes. ECHO_OFF1 terminates the echo capability.

NOTE: It is not recommended to use ECHO when the communications channel is an RS-485 port such as an M-style motor or a D-style motor with the RS485-ISO adapter. This mode of communication is half-duplex and is not compatible with the ECHO command.

EXAMPLE:

```
EIGN(W,0)    'Make all onboard I/O inputs
ZS          'Clear errors
OCHN(RS4,1,N,9600,1,8,D) 'Open auxiliary com port
ECHO1      'enable echo on aux com port
```

RELATED COMMANDS:

ECHO *Echo Incoming Data on Communications Port 0 (see page 403)*
ECHO_OFF *Turn Off Echo on Communications Port 0 (see page 407)*
ECHO_OFF1 *Turn Off Echo on Communications Port 1 (see page 409)*

ECHO_OFF

Turn Off Echo on Communications Port 0

APPLICATION:	Communications control; supports the DS2020 Combitronic system
DESCRIPTION:	Disable motor echo of received channel 0 serial data
EXECUTION:	Immediate; remains in effect until otherwise commanded
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	Motor defaults to ECHO_OFF (non-echo)
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The ECHO_OFF command causes the SmartMotor™ channel 0 COM port to stop echoing. This is the default power-up state of any SmartMotor. No incoming channel 0 characters are retransmitted.

In order to automatically detect and differentiate between multiple motors on a serial daisy-chain cable, the ECHO state can be alternately turned on and off to ensure the motors are properly addressed.

NOTE: It is not possible to maintain communications on a serial chain without issuing ECHO.

EXAMPLE:

```
ECHO_OFF      'Remove echo from channel 0
EIGN(W,0)     'Make all onboard I/O inputs
ZS           'Clear errors
```

RELATED COMMANDS:

ECHO *Echo Incoming Data on Communications Port 0 (see page 403)*

ECHO1 *Echo Incoming Data on Communications Port 1 (see page 406)*

ECHO_OFF1 *Turn Off Echo on Communications Port 1 (see page 409)*

ECHO_OFF0

Turn Off Echo on Communications Port 0

APPLICATION:	Communications control
DESCRIPTION:	Disable motor echo of received channel 0 serial data
EXECUTION:	Immediate; remains in effect until otherwise commanded
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	Motor defaults to ECHO_OFF0 (non-echo)
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The ECHO_OFF0 command causes the SmartMotor™ channel 0 serial port to stop echoing. No incoming channel 0 characters are retransmitted.

EXAMPLE: (Shows use of ECHO_OFF0 and OCHN)

```
EIGN (W,0)    'Make all onboard I/O inputs
ZS           'Clear errors
OCHN (RS4,0,N,9600,1,8,C) 'Open communications channel 0
ECHO_OFF0    'Turn echo off for communications channel 0
END
```

RELATED COMMANDS:

ECHO *Echo Incoming Data on Communications Port 0 (see page 403)*
 ECHO0 *Echo Incoming Data on Communications Port 0 (see page 405)*
 ECHO1 *Echo Incoming Data on Communications Port 1 (see page 406)*
 ECHO_OFF *Turn Off Echo on Communications Port 0 (see page 407)*
 ECHO_OFF1 *Turn Off Echo on Communications Port 1 (see page 409)*

ECHO_OFF1

Turn Off Echo on Communications Port 1

APPLICATION:	Communications control
DESCRIPTION:	Disable motor echo of received channel 1 serial data
EXECUTION:	Immediate; remains in effect until otherwise commanded
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	Motor defaults to ECHO_OFF1 (non-echo)
FIRMWARE VERSION:	5.x (D/M); 6.4.2.x (D)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The ECHO_OFF1 command causes the SmartMotor™ channel 1 serial port to stop echoing. No incoming channel 1 characters are retransmitted.

EXAMPLE: (Shows use of ECHO_OFF1 and OCHN)

```

EIGN (W,0)      'Make all onboard I/O inputs
ZS             'Clear errors
OCHN (RS4,1,N,9600,1,8,C) 'Open aux communications channel
ECHO_OFF1     'Turn echo off for aux communications channel
END

```

RELATED COMMANDS:

ECHO *Echo Incoming Data on Communications Port 0 (see page 403)*

ECHO1 *Echo Incoming Data on Communications Port 1 (see page 406)*

ECHO_OFF *Turn Off Echo on Communications Port 0 (see page 407)*



ECS(counts)

Encoder Count Shift

APPLICATION:	Motion control
DESCRIPTION:	External encoder counter shift
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	None
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	Encoder counts
RANGE OF VALUES:	Input counts: -2147483648 to 2147483647
TYPICAL VALUES:	-100 to +100
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	ECS(100):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The ECS(value) command immediately shifts the external encoder counter by the specified value. In Follow mode or Cam mode this is interpreted as incoming controller counts, so motion may result. In other words, ECS adds the specified value to the incoming controller counts as if they actually had an instantaneous change in value.

NOTE: When issued, ECS is dynamic and immediate! It is not buffered. No G command is required.

For example, if the external encoder count is 4000 and ECS(1234) is issued, the count would immediately shift to 5234. It is instantaneously shifted by 1234 counts, as seen by the trajectory generator.

ECS accounts for changes in material width on traverse and take up winding applications to allow for full placement of material onto spools. These applications require the means to dynamically detect material width as close as possible to where it is being wound onto the controller spool.

This command works on top of *any* gear or Cam mode. It should be used with care because it can cause abrupt changes to position.

EXAMPLE:

```
C1           'Instant step routine
WHILE 1     'Forever loop
  IF IN(1)==0 'Check input 1 for low state
    ECS(100) 'Instantly add 100 counts to CTR(1)
    WHILE IN(1)==0 LOOP 'Hold while input is triggered
  ENDIF
LOOP
RETURN
END
```

RELATED COMMANDS:

R CTR(enc) *Counter, Encoder, Step and Direction (see page 380)*
MFCTP(arg1,arg2) *Mode Follow Control Traverse Point (see page 583)*
MFH(distance[,m/s]) *Mode Follow, High Ascend/Descend Rate (see page 590)*
R MFHTP=formula *Mode Follow, High Traverse Point (see page 592)*
MFL(distance[,m/s]) *Mode Follow, Low Ascend/Descend Rate (see page 594)*
R MFLTP=formula *Mode Follow, Low Traverse Point (see page 596)*



APPLICATION:	I/O control; supports the DS2020 Combitronic system
DESCRIPTION:	Configure I/O as general-use input
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	Input: EIGN(IO) EIGN(W,word) EIGN(W,word[,mask]) See details for range of IO, word and mask — depends on motor series
TYPICAL VALUES:	N/A
DEFAULT VALUE:	See details — depends on motor series
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	EIGN(0):3, EIGN(W,0):3, or EIGN(W,0,m):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The EIGN command is used to configure I/O pins for general-use input as shown:

- EIGN(IO)
Sets the specified I/O port to or back to an input with no function attached. In other words, to remove the travel-limit function from I/O port 2, execute the instruction EIGN(2). See the next table for allowed range of IO.
- EIGN(W,word)
Sets all I/O in the specified I/O word back to inputs. A literal "W" is used as the first argument. See the next table for allowed values for "word".
- EIGN(W,word[,mask])
Set all I/O in the specified I/O word back to input if mask bit is set. A literal "W" is used as the first argument. See the next table for allowed values for "word" and for the mask range.

NOTE: The range of IO and word depends on the motor series:

Motor Type	word Allowed Values	IO Allowed Range	Word 0 Bitmask Range	Word 1 Bitmask Range
D-style	0	0-6	0 to 127	N/A
D-style with AD1 option	0,1	0-6, 16-25	0 to 127	0 to 1023
M-style	0	0-10	0 to 2047	N/A
DS2020 Combitronic system	N/A	0, 2, 3	N/A	N/A

Ports 2 and 3 are travel limit inputs by default. However, the EIGN() commands can change them to general-purpose I/O points. They can be returned to travel limits with the EILN and EILP commands.

EXAMPLE:

```
EIGN (6)      'Assigns a single I/O point (I/O 6) as
              'general-use input.
EIGN (W, 0)   'Assigns all local I/O in word 1 as general-use
              'inputs and disables the travel limits.
EIGN (W, 0, 12) 'Assigns inputs 2 and 3 as general-use inputs at
              'once (disabling overtravel limits).
EIGN (W, 0, m) 'Assign a masked word-sized set of local I/O as
              'general-use inputs at once.
```

EXAMPLE: (assign inputs 2 and 3 as general-use inputs at once; disable overtravel limits)

```
x=12
```

```
y=0
```

```
EIGN (W, y, x) 'EIGN(W,y)&x will also do the same thing
```

EXAMPLE: (configuring individual ports as inputs)

```
          EIGN (0)    'Set User port 0 as Input
EIGN (1)   'Set User port 1 as Input
EIGN (2)   'Set User port 2 as Input
EIGN (3)   'Set User port 3 as Input
EIGN (4)   'Set User port 4 as Input
EIGN (5)   'Set User port 5 as Input
EIGN (6)   'Set User port 6 as Input
```

EXAMPLE: (disabling left and right limits)

```

EIGN(2)    'Disable left limit
EIGN(3)    'Disable right limit
ZS         'Clear faults
VT=700000 'Set Target Velocity
ADT=100    'Set accel/decel
MV         'Set Mode Velocity
ITR(0,3,15,1,20) 'Set interrupt
EITR(0)    'Enable interrupt zero
ITRE       'Enable all interrupts
G          'Start motion
C10        'Place a label
GOTO10     'Loop..., required for interrupt operation
END        'End (never reached)

C20        'Interrupt subroutine code here
RETURNI    'Return from interrupt subroutine

```

RELATED COMMANDS:

EILN *Enable Input as Limit Negative (see page 415)*
 EILP *Enable Input as Limit Positive (see page 417)*
 EISM(x) *E-Configure Input as Sync Controller (see page 423)*
 EOBK(IO) *Enable Output, Brake Control (see page 445)*
 R IN(...) *Specified Input (see page 509)*
 R INA(...) *Specified Input, Analog (see page 512)*
 SLD *Software Limits, Disable (see page 740)*
 SLE *Software Limits, Enable (see page 742)*



APPLICATION:	I/O control; supports the DS2020 Combitronic system
DESCRIPTION:	Activate left/negative hardware limit
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	Limit switch
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	EILN:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The EILN command activates the left/negative hardware limit.

The EILN command sets I/O port 3 as the negative overtravel limit. User I/O port 3 can be a general-purpose analog or digital input, output or act as the negative-limit input (which is the default state). EILN explicitly defines I/O port 3 as the negative limit, while the EIGN command configures it as a general-purpose input, which disables the limit behavior.

NOTE: I/O port 3 cannot be set as an output until the EIGN command is issued first (i.e., the OR, OS or OUT command will not change I/O 3 until EIGN is issued to set I/O 3 as an input).

Ports 2 and 3 are travel limit inputs by default. However, the EIGN() commands can change them to general-purpose I/O points. They can be returned to travel limits with the EILN and EILP commands.

EXAMPLE: (Subroutine enables negative hardware limit)

```

C1
c=0
PRINT("Enter c=1 to enable negative HW limit...",#13)
WHILE c==0 LOOP      'Wait for user to change variable c
EILN                  'Enable negative hardware limit
PRINT("Negative HW limit enabled!",#13)
RETURN

```

Program output is:

Enter c=1 to enable negative HW limit...

(The user enters c=1)

Negative HW limit enabled!

RELATED COMMANDS:

EIGN(...) *Enable as Input for General-Use (see page 412)*

EILP *Enable Input as Limit Positive (see page 417)*

EIRE *Enable Index Register, Encoder Capture (see page 419)*

EIRI *Enable Index Register, Input Capture (see page 421)*

EISM(x) *E-Configure Input as Sync Controller (see page 423)*

EOBK(IO) *Enable Output, Brake Control (see page 445)*

OR(value) *Output, Reset (see page 638)*

OS(...) *Output, Set (see page 640)*

OUT(...)=formula *Output, Activate/Deactivate (see page 644)*



APPLICATION:	I/O control; supports the DS2020 Combitronic system
DESCRIPTION:	Activate right/positive hardware limit
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	Limit switch
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	EILP:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The EILP command activates the right/positive hardware limit.

The EILP command sets I/O port 2 as the positive overtravel limit. User I/O port 2 can be a general-purpose analog or digital input, output or act as the positive-limit input (which is the default state). EILP explicitly defines I/O port 2 as the positive limit, while the EIGN command configures it as a general-purpose input, which disables the limit behavior.

NOTE: I/O port 2 cannot be set as an output until the EIGN command is issued first (i.e., the OR, OS or OUT command will not change I/O 2 until EIGN is issued to set I/O 2 as an input).

Ports 2 and 3 are travel limit inputs by default. However, the EIGN() commands can change them to general-purpose I/O points. They can be returned to travel limits with the EILN and EILP commands.

EXAMPLE: (Subroutine enables positive hardware limit)

```

C1
c=0
PRINT("Enter c=1 to enable positive HW limit...",&13)
WHILE c==0 LOOP      'Wait for user to change variable c
EILP                  'Enable positive hardware limit
PRINT("Positive HW limit enabled!",&13)
RETURN

```

Program output is:

Enter c=1 to enable positive HW limit...

(The user enters c=1)

Positive HW limit enabled!

RELATED COMMANDS:

EIGN(...) *Enable as Input for General-Use (see page 412)*

EILN *Enable Input as Limit Negative (see page 415)*

EIRE *Enable Index Register, Encoder Capture (see page 419)*

EIRI *Enable Index Register, Input Capture (see page 421)*

EISM(x) *E-Configure Input as Sync Controller (see page 423)*

EOBK(IO) *Enable Output, Brake Control (see page 445)*

OR(value) *Output, Reset (see page 638)*

OS(...) *Output, Set (see page 640)*

OUT(...)=formula *Output, Activate/Deactivate (see page 644)*

EIRE

Enable Index Register, Encoder Capture

APPLICATION:	I/O control
DESCRIPTION:	Capture internal encoder using its index; capture external encoder from input.
EXECUTION:	Immediate; remains in effect until otherwise commanded
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	Motor defaults to EIRE (external input captures the external encoder)
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

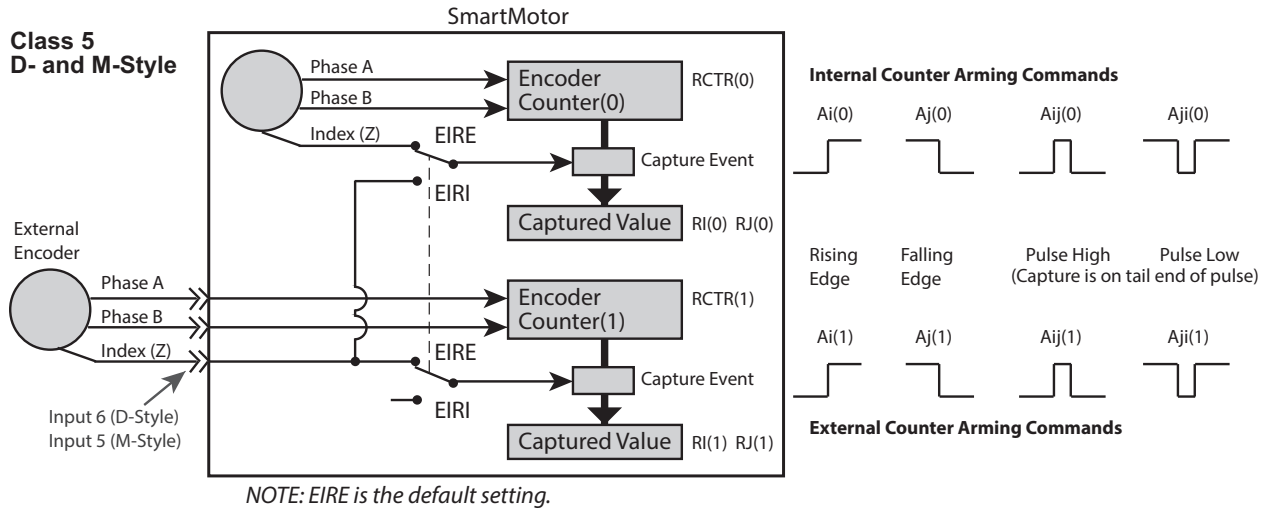
DETAILED DESCRIPTION:

Captures the external or internal *encoder's* position as described in the next table. This is also known as index capture, registration or touch probes.

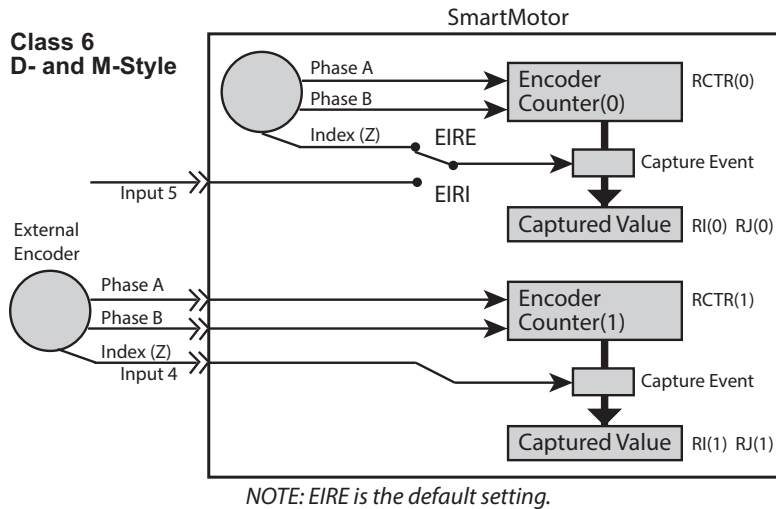
Class 5 D-Style	Class 5 M-Style	Class 6 D- and M-style
Internal encoder capture using internal encoder index	Internal encoder capture using internal encoder index	Internal encoder capture using internal encoder index
External encoder captured from input 6	External encoder captured from input 5	External encoder capture using input 4

Refer to the next figure for more details.

**Class 5
D- and M-Style**



**Class 6
D- and M-Style**



Encoder Capture Diagrams for EIRE

EXAMPLE:

```
EIGN (W, 0)    'Make all onboard I/O inputs
ZS            'Clear errors
EIRE          'Use external IO to capture external encoder position
END
```

RELATED COMMANDS:

- $A_i(\text{enc})$ Arm Index Rising Edge (see page 270)
- $A_j(\text{enc})$ Arm Index Falling Edge (see page 274)
- EIGN(...) Enable as Input for General-Use (see page 412)
- EIRI Enable Index Register, Input Capture (see page 421)
- $R I(\text{enc})$ Index, Rising-Edge Position (see page 502)
- $R J(\text{enc})$ Index, Falling-Edge Position (see page 524)

EIRI

Enable Index Register, Input Capture

APPLICATION:	I/O control
DESCRIPTION:	Capture internal encoder using input; external encoder behavior may change depending on model
EXECUTION:	Immediate; remains in effect until otherwise commanded
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	Motor defaults to EIRE (external input captures the external encoder)
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

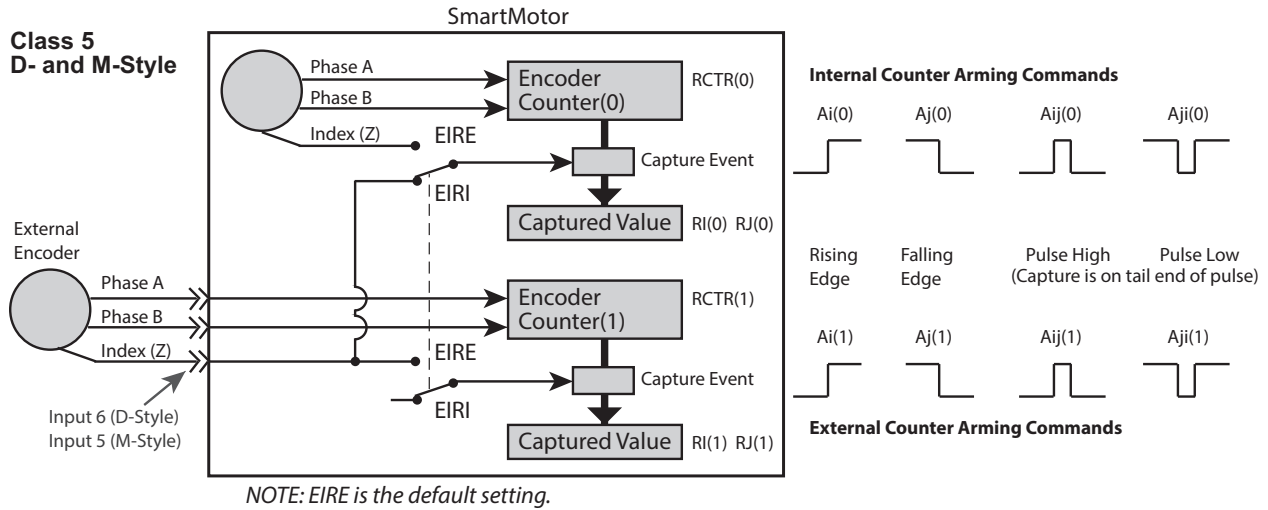
DETAILED DESCRIPTION:

Captures the internal encoder using an input; external encoder behavior may change depending on model as described in the next table. This is also known as index capture, registration or touch probes.

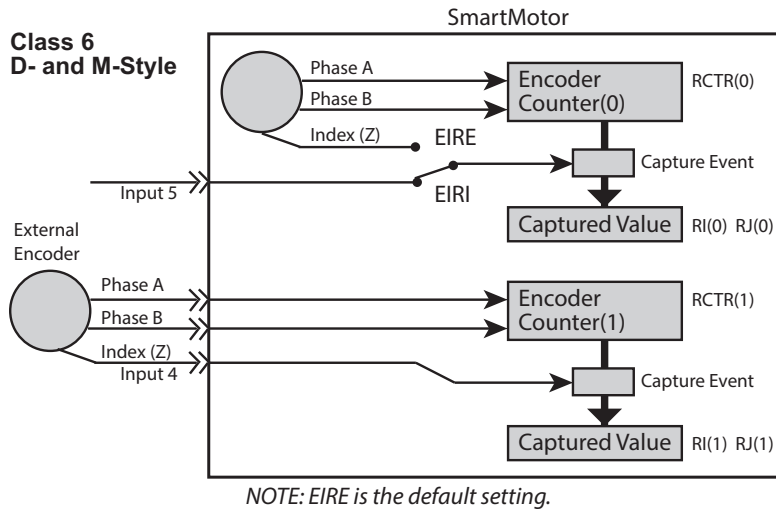
Class 5 D-Style	Class 5 M-Style	Class 6 D- and M-style
Internal encoder capture using input 6	Internal encoder capture using input 5	Internal encoder capture using input 5
External encoder capture is disabled	External encoder capture is disabled	External encoder capture using input 4

Refer to the next figure for more details.

**Class 5
D- and M-Style**



**Class 6
D- and M-Style**



Encoder Capture Diagrams for EIRI

EXAMPLE:

```
EIGN (W, 0)    'Make all onboard I/O inputs
ZS            'Clear errors
EIRI         'Use external IO to capture internal encoder position
END
```

RELATED COMMANDS:

- Ai(enc) Arm Index Rising Edge (see page 270)
- Aj(enc) Arm Index Falling Edge (see page 274)
- EIRE Enable Index Register, Encoder Capture (see page 419)
- EIGN(...) Enable as Input for General-Use (see page 412)
- R I(enc) Index, Rising-Edge Position (see page 502)
- R J(enc) Index, Falling-Edge Position (see page 524)

EISM(x)

E-Configure Input as Sync Controller

APPLICATION:	I/O control; supports the DS2020 Combitronic system - see Details
DESCRIPTION:	Configure specified I/O as the start-motion input
EXECUTION:	Immediate; remains in effect until otherwise commanded
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	By default, this feature is not enabled
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The EISM(x) command configures the specified input as the start-motion input. For example, EISM(6) issues a G command when input 6 is asserted.

NOTE: For systems using the DS2020 Combitronic system, always use EISM(0).

- For D-style motors, the input must be driven low, EISM(6)
- For M-style motors, the input must be driven high, EISM(6)

Refer to Connecting the System in your motor's user guide for the location of pin I/O-6 on your motor.

EXAMPLE:

```
EIGN (w,0)      'Make all onboard I/O inputs
ZS              'Clear errors
EISM (6)        'Configure motor to receive G when I/O 6 is triggered
MV              'Mode Velocity
ADT=100         'Set accel/decel
VT=100000      'Set target velocity
WHILE 1 LOOP   'Hold program here
```

RELATED COMMANDS:

EIGN(...) *Enable as Input for General-Use (see page 412)*
 G *Start Motion (GO) (see page 473)*



EITR(int)

Enable Interrupts

APPLICATION:	Program execution and flow control
DESCRIPTION:	Enable the specified interrupt or a combination of interrupts
EXECUTION:	Immediate
CONDITIONAL TO:	Interrupts configured and globally enabled (with ITR and ITRE)
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	Input: EITR(interrupt), where interrupt is 0-7 EITR(W,mask) where mask is 0-255
TYPICAL VALUES:	Input: EITR(interrupt), where interrupt is 0-7 EITR(W,mask), where mask is 0-255
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	EITR(1):3 or EITR(W,7):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The EITR (Enable Interrupts) command is used to enable one or more interrupts.

NOTE: Use DITR() or EITR() before the STACK command to stop any pending interrupt events from reoccurring. Additionally, DITR() will prevent future calls.

EITR is written as:

- EITR(interrupt)
Where interrupt is used to specify an interrupt (0-7).
- EITR(W,mask)
A literal "W" is used as the first argument; the mask argument can select a combination of interrupts.

NOTE: The (W,mask) input requires this firmware:
for Class 5: 5.x.4.46 and later; for Class 6: 6.0.2.37 and later.

For an interrupt to work, it must be enabled at two levels: first, enable *individual* interrupts with the EITR() command using the interrupt number from 0 to 7 in the parentheses; second, enable *all* interrupts with the ITRE command. Similarly, *individual* interrupts can be disabled with the DITR()

command, and *all* interrupts can be disabled with the ITRD command. For more details, see the corresponding command-description pages.

NOTE: The user program must also be running for interrupts to take effect, the END and RUN commands will reset the state of the interrupts to defaults.

For more details on interrupt programming, see Interrupt Programming on page 195.

EXAMPLE:

```
EIGN (2)      'Disable left limit
EIGN (3)      'Disable right limit
ZS           'Clear faults
VT=700000    'Set target velocity
ADT=100      'Set target accel/decel
MV           'Set Mode Velocity
ITR(0,3,15,1,10) 'Set interrupt
EITR(0)      'Enable interrupt zero
ITRE        'Enable all interrupts
G           'Start motion
PAUSE       'Pause here so program doesn't end
END         'End would disable interrupts
```

EXAMPLE: (Subroutine shows use of DITR, EITR, TMR and TWAIT)

```
C10          'Place a label
  IF PA>47000 'Just before 12 moves
    DITR(0)   'Disable interrupt
    TWAIT     'Wait till reaches 48000
    p=0       'Reset variable p
    PT=p      'Set target position
    G         'Start motion
    TWAIT     'Wait for move to complete
    EITR(0)   'Re-enable interrupt
    TMR(0,1000) 'Restart timer
  ENDIF
GOTO10      'Go back to label
```

RELATED COMMANDS:

DITR(int) *Disable Interrupts (see page 394)*
 EISM(x) *E-Configure Input as Sync Controller (see page 423)*
 ITR(Int#,StatusWord,Bit#,BitState,Label#) *Interrupt Setup (see page 517)*
 ITRD *Interrupt Disable, Global (see page 520)*
 ITRE *Enable Interrupts, Global (see page 522)*
 RETURNI *Return Interrupt (see page 708)*



APPLICATION:	Motion control; supports the DS2020 Combitronic system
DESCRIPTION:	Set maximum allowable position error (the error limit)
EXECUTION:	Immediate; enforced each PID sample
CONDITIONAL TO:	Servo active (MP, MV, etc., not MT mode)
LIMITATIONS:	Torque mode has no position error
READ/REPORT:	REL
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	Encoder counts DS2020 Combitronic system: user increments, see FD=expression on page 461
RANGE OF VALUES:	0 to 262143 DS2020 Combitronic system: 0 to 4294967295
TYPICAL VALUES:	1000 DS2020 Combitronic system: fraction of FD value
DEFAULT VALUE:	1000 DS2020 Combitronic system: 8192
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	EL:3=1234, a=EL:3, REL:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

NOTE: This command is affected by the SCALEP command. For details, see SCALEP(m,d) on page 726. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

The EL command is used to set the maximum allowable position error in encoder counts. Position error is the difference between the calculated trajectory position (PC), at any instant in time, and the actual position (PA). The SmartMotor™ uses the position error to generate a torque by means of the PID filter. The greater the error or deflection, the more torque the motor applies in attempt to correct it.



CAUTION: An EL setting that is greater than the expected move distance may result in drive saturation and severely limit the life of the SmartMotor.

EL is primarily used as a safety measure. It is a programmable, allowable error beyond which the motor recognizes it is outside of the domain of control being enforced. If EL=100 is commanded and a position error of greater than 100 encoder counts occurs, the motor will perform its fault reaction and the Be (Position Error bit) will be set to 1. All closed-loop modes are bound by this EL value. Non-closed-loop modes, such as Torque mode, ignore the value of EL.

For the DS2020 Combitronic system, position error is strongly related to the FD value. When the absolute value of the position error is greater than EL/FD motor shaft rotations, the error limit is reached. For example, if EL=8192 and FD=65536, as default, the fault is signaled if the position error is greater than 1/8 shaft rotation. Also, note that the position error limit, set by EL, is a RAM (volatile) parameter on the DS2020. This means the user setting will be lost whenever the DS2020 Combitronic system is reset, and it will revert to its default value (8192).

For the DS2020 Combitronic system, the position error timeout can be set through a CANopen command (0x6066). See the *SmartMotor™ CANopen Guide* for details on sending commands over the CANopen network.

This limit triggers fault 45 (shown in the DS2020 Combitronic system status word 0, bit 6); its reaction can be set by FSAD(45) and read by RFSAD(45). By default, this reaction is 2: when position error EA is greater than the limit EL, the drive is disabled.

EXAMPLE:

```
EL=1234      'Set maximum allowable error to 1234
```

If the motor dynamically exceeds 1234, it immediately faults on position error.

RELATED COMMANDS:

R EA *Error Actual (see page 401)*

R FD=expression *Resolution to Set Units of Position/Velocity/Acceleration (see page 461)*

R FSAD(n,m) *Set Reaction to Fault (see page 467)*

G *Start Motion (GO) (see page 473)*

MP *Mode Position (see page 613)*

MV *Mode Velocity (see page 624)*

R PA *Position, Actual (see page 646)*

R PC, PC(axis) *Position, Commanded (see page 650)*

ELSE

IF-Structure Command Flow Element

APPLICATION:	Program execution and flow control
DESCRIPTION:	Alternate action of IF <i>formula</i> or ELSEIF <i>formula</i> within IF...ELSE...ENDIF control block
EXECUTION:	Immediate
CONDITIONAL TO:	Value of previous IF <i>formula</i> or ELSEIF <i>formula</i> .
LIMITATIONS:	Must reside with IF <i>formula</i> ...ENDIF program control block; can be executed only from within a user program
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

An IF formula...ENDIF control block may optionally include an ELSE statement to control execution when none of the test conditions are true. As illustrated in the next example, an IF formula can be used when you want the SmartMotor™ to do one thing if the variable g=43 and another if it does not equal that value.

EXAMPLE:

```
IF g==43
    PRINT("Gee...43!", #13)
ELSE
    PRINT("No 43 for me.", #13)
ENDIF
```

The first line checks to see if g is equal to 43. If so, the string "Gee...43!" is sent out the primary serial port. The ELSE in line 3 tells the SmartMotor what to do otherwise.

An IF control block can have only one ELSE statement. When the language interpreter evaluates the IF formula as false (zero), an ELSE exists and there are no ELSEIF statements, the program branches immediately to the statement after the ELSE. If there are ELSEIF formula clauses within the control block, all the ELSEIF clauses must precede the ELSE clause. In these cases, the ELSE clause is only executed when both the IF formula is false (zero) and all ELSEIF formulas are false (zero).

ELSE is analogous to the DEFAULT case for a SWITCH control block.

ELSE is not a valid terminal command; it is only valid within a user program.

EXAMPLE:

```
a=1      'PRINT("FALSE") is always executed
IF a==2
    PRINT("TRUE")
ELSE
    PRINT("FALSE")
ENDIF
```

EXAMPLE:

```
IF a==1  'Only if a is NOT 1, 2, or 3
         'will GOSUB5 be executed.
    GOSUB2
ELSEIF a==2
    GOSUB3
ELSEIF a==3
    GOSUB4
ELSE
    GOSUB5
ENDIF

C2      'Some subroutine code here
C3      'Some subroutine code here
C4      'Some subroutine code here
C5      'Some subroutine code here
```

RELATED COMMANDS:

ELSEIF formula *IF-Structure Command Flow Element (see page 430)*

ENDIF *End IF Statement (see page 441)*

IF formula *Conditional Program Code Execution (see page 506)*

ELSEIF formula

IF-Structure Command Flow Element

APPLICATION:	Program execution and flow control
DESCRIPTION:	Alternate evaluation of IF...ENDIF control block
EXECUTION:	Immediate
CONDITIONAL TO:	Value of <i>formula</i> and previous IF <i>formula</i> or ELSEIF <i>formula</i>
LIMITATIONS:	Must reside with IF <i>formula</i> ...ENDIF program control block; can be executed only from within user program
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

An IF formula's control block may optionally include any number of ELSEIF formulas to perform multiple evaluations in a specified order. For example, ELSEIF would be used when you want the SmartMotor™ to do one thing if the variable $g=43$, another if $g=43000$, and another if $g=-2$.

For more detail about valid *formulas* that can be used, see IF formula on page 506.

EXAMPLE:

```
IF g==43
    PRINT ("Gee...43!", #13)
ELSEIF g==43000
    PRINT ("43 grand for me.", #13)
ELSEIF g== -2
    PRINT ("2?", #13)
ENDIF
```

The first line checks to see if g is equal to 43. If so, the string "Gee...43!" is sent out the primary serial port and the IF control block terminates. If g is not 43, the program goes on to test if g is 43000. If it is, then "43 grand for me." is sent out the primary serial port and the IF control block terminates. Similarly, if g is not 43000, then the program goes on to test if g is -2. If it is, then "-2?" is sent out the primary serial port and the IF control block terminates.

An IF control block can have multiple ELSEIF statements. If such an ELSEIF clause exists and the language interpreter evaluates the IF formula to be false (zero), then the program will branch immediately to first ELSEIF formula.

If the associated formula is true, then the subsequent clause is executed until an ELSEIF, ELSE or ENDIF is encountered, and then execution branches to the ENDIF of the current IF control block. If the first ELSEIF clause is not executed, then program execution continues at the next ELSEIF formula, and so on, until all the ELSEIF formulas have been tested. In the case all ELSEIFs have false formulas and an ELSE clause exists, that clause will be executed.

The ELSEIF statement is similar to the CASE number in a SWITCH control block. Note the difference—ELSEIF handles formulas, but CASE only handles a fixed number.

ELSEIF is not a valid terminal command. It is only valid within a user program.

EXAMPLE:

```
a=3                                'Will be found false
IF a==2
    PRINT ("222")
ELSEIF a==3                        'Will be found true
    PRINT ("333") 'so "333" will be printed.
ENDIF
```

EXAMPLE:

```
IF a==1                            'Only if a is NOT 1, 2, or 3
                                    'will GOSUB5 be executed.
    GOSUB2
ELSEIF a==2
    GOSUB3
ELSEIF a==3
    GOSUB4
ELSE
    GOSUB5
ENDIF
```

RELATED COMMANDS:

ELSE *IF-Structure Command Flow Element (see page 428)*

ENDIF *End IF Statement (see page 441)*

IF formula *Conditional Program Code Execution (see page 506)*



Encoder Zero (Close Loop on Internal Encoder)

APPLICATION:	Motion control
DESCRIPTION:	Use internal encoder for the PID, actual position, actual velocity
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	Motor defaults to ENCO (use internal encoder for PID, actual position)
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	ENC0:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The SmartMotor™ can accept inputs from either the internal integrated encoder or an external source. ENCO causes the SmartMotor to read its position from the internal encoder; ENC1 uses the secondary (external) encoder. When ENCO is active, PA (position actual) will track the internal encoder. For more details, see PA on page 646.

EXAMPLE:

```
ENC1 'Servo from external encoder
ENC0 'Restore default encoder behavior
```

RELATED COMMANDS:

R CTR(enc) Counter, Encoder, Step and Direction (see page 380)
ENCD(in_out) Set Encoder Bus Port as Input or Output (see page 437)
ENC1 Encoder Zero (Close Loop on External Encoder) (see page 433)
MS0 Mode Step, Zero External Counter (see page 616)
MF0 Mode Follow, Zero External Counter (see page 578)



Encoder Zero (Close Loop on External Encoder)

APPLICATION:	Motion control
DESCRIPTION:	Use external encoder for the PID, actual position, actual velocity
EXECUTION:	Immediate
CONDITIONAL TO:	External encoder attached to motor
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	Motor defaults to ENCO (use internal encoder for PID, actual position)
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	ENC1:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:



WARNING: If the ENC1 command is issued without an external encoder properly electrically connected to the A and B encoder inputs *and* physically connected to the shaft, the motor will run away with full speed and torque.

The SmartMotor™ can accept position information from either the internal integrated encoder or an external source. The ENC1 command will cause the SmartMotor to servo from the secondary (external) encoder channel instead of the internal encoder. When ENC1 is active, PA (Position Actual) will track the external encoder.

The default mode of operation (accept position information from the internal encoder) is restored with the ENCO command.

NOTE: Always issue either MS0 or MF0 to select the input mode of the external encoder. Do not assume one mode or the other.

- MS0 will set step/direction (and clear the position to 0)
- MF0 will set quadrature input (and clear the position to 0)

If the external encoder is not connected or is incorrectly connected, the motor may run away. In this case, use the RPA command to check the position. If you can change the position by rotating the shaft, then the encoder is connected but the A and B signals need to be swapped, which reverses the direction described by the quadrature phasing of the A and B signals.

EXAMPLE:

```
MSO  'Set external encoder to step/direction (zero external encoder)
ENC1  'Servo from external encoder
ENC0  'Restore default encoder behavior
```

RELATED COMMANDS:

^R CTR(enc) *Counter, Encoder, Step and Direction (see page 380)*
ENC0 *Encoder Zero (Close Loop on Internal Encoder) (see page 432)*
ENCD(in_out) *Set Encoder Bus Port as Input or Output (see page 437)*
MS0 *Mode Step, Zero External Counter (see page 616)*
MF0 *Mode Follow, Zero External Counter (see page 578)*

ENCCTL(function,value)

Encoder Control

APPLICATION:	Motion control
DESCRIPTION:	Special configuration options for encoder
EXECUTION:	Immediate
CONDITIONAL TO:	Depends on sub-command; certain specific encoders apply
LIMITATIONS:	Command is not available for Class 6 M-series SmartMotors
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	Input: ENCCTL(function,value) function: >= -1 value: See details in next table
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.97.x / 5.98.x (D/M); 6.4.2.x (D)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The ENCCTL(function,value) command is used to set special configuration options for the encoder. It has two input parameters:

- function: specifies the configuration setting for the encoder
- value: (where applicable) specifies the value to apply to the selected action

This command requires the absolute encoder option for the M-style motor. It does not apply to the typical optical incremental encoder found in most SmartMotors.

NOTE: The D-style motor does not offer an ABS option.

Refer to the next table for details.

'function'	'value' range	Encoder type	Description
-1	N/A	ABS*, C6D**	Removes the effect of the ENCCTL(0,value) command. PA offset is then set as shipped from the factory.
0	-2147483648 to 2147483647	ABS*, C6D**	Sets the current absolute position to 'value'. This calculates the offset required and stores it in nonvolatile memory. Motor firmware will adjust absolute position with offset on every power up.
5	N/A	ABS*	Reset error flags.

'function'	'value' range	Encoder type	Description
6	0,1,2	ABS*	Report encoder firmware version to the terminal. value 0: Report lowest byte (minor rev) value 1: Report mid byte (major rev) value 2: Report high byte (firmware type)
11	-2 to 2	C6D**	Selection of encoder resolution relative to 12-bit (4096 counts per revolution). Positive value increases resolution as a bit shift by that value; negative value decreases resolution as a bit shift by that value. This setting is non-volatile. NOTE: The motor must be rebooted for this resolution change to take effect.
12	0	C6D**	Reload position (RPA) with absolute data from encoder.

*Requires the absolute encoder option for the Class 5 M-series motor.

**Requires the Class 6 D-series motor with batteryless absolute multiturn encoder (no battery backup needed).

EXAMPLE:

```
EIGN(W,0)   'Make all onboard I/O inputs
ZS          'Clear errors
ENCCTL(0,0) 'Set absolute encoder to zero
```

RELATED COMMANDS:

ENCD(in_out) *Set Encoder Bus Port as Input or Output (see page 437)*

ENCD(in_out)

Set Encoder Bus Port as Input or Output

APPLICATION:	I/O control
DESCRIPTION:	Set M-style motor Encoder Bus port as an input or output
EXECUTION:	Immediate
CONDITIONAL TO:	M-style SmartMotor
LIMITATIONS:	Not available for Class 6 D-style
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	Motor defaults to ENCD(0) where encoder is input
FIRMWARE VERSION:	5.97.x / 5.98.x (D/M); 6.0.x (M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The M-style SmartMotor™ is equipped with a bidirectional Encoder Bus port, which can be configured as an input or an output for use as a follower or controller, respectively, in an "electronic line shaft" configuration.

NOTE: The D-style motor does not support this command and will return an error.

Using the Encoder Bus port along with Moog Animatics encoder bus cables and ENCD commands, you can create a daisy-chained series of M-style motors. In this configuration:

- One motor will have ENCD(1) issued — this will be the controller.
- All other motors will have ENCD(0) (default) issued — these will be the follower devices.

ENCODER OUTPUT

For Class 6 M-style motors, the internal encoder is conditioned with error correction and resolution adjustments when used normally for positioning (i.e., as seen in the RPA command "actual position"). However, when directed as an output and received into another motor for the purposes of Follow mode, there are several things to be aware of at the receiving motor:

- The resolution seen at the receiving motor will be 4096 instead of 4000. MFMUL and MFDIV will need to compensate accordingly.
- The direction will be negative (assuming a straight-through connection of the 8-pin "Y" cable).
- There are non-accumulating, single-turn errors that are not compensated.

EXAMPLE:

```
ENCD(1) 'Encoder bus port set as output (controller)
ENCD(0) '(Default) Encoder bus port set as input (follower)
```

RELATED COMMANDS:

^RCTR(enc) *Counter, Encoder, Step and Direction (see page 380)*

MC *Mode Cam (Electronic Camming) (see page 555)*

MF0 *Mode Follow, Zero External Counter (see page 578)*

MFR *Mode Follow Ratio (see page 600)*

MS0 *Mode Step, Zero External Counter (see page 616)*

MSR *Mode Step Ratio (see page 618)*



End Program Code Execution

APPLICATION:	Program execution and flow control
DESCRIPTION:	Terminates execution of the user program
EXECUTION:	Immediate
CONDITIONAL TO:	Valid whether issued by host or user program
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	END:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The END command terminates execution of a user program if there is one running. END may be issued through serial communications channels or from within the user program. Each program must have a minimum of at least one END statement.

NOTE: The SMI program will not compile a source file without at least one END present.

END only terminates the user program and internally resets the program pointer to the beginning of the program; no other state, variable, mode or trajectory is affected.

User program interrupts will stop functioning when the END is encountered in a program or issued from the serial communication channel.

The SMI program provides several toolbar buttons that send an END command. This is especially useful when something prevents you from entering the END command at the terminal screen. For more details, see the SMI software's online help file, which can be accessed from the Help menu or by pressing the F1 key.

EXAMPLE:

```
IF Be END ENDIF    'Terminate user program
                   'on position error
```

EXAMPLE: (Routine homes motor against a hard stop)

```
MDS           'Using Sine mode commutation
KP=3200       'Increase stiffness from default
KD=10200     'Increase damping from default
F            'Activate new tuning parameters
AMPS=100     'Lower current limit to 10%
VT=-10000    'Set maximum velocity
ADT=100      'Set maximum accel/decel
MV           'Set Velocity mode
G            'Start motion
WHILE EA>-100 'Loop while position error is small
LOOP         'Loop back to WHILE
O=-100       'While pressed, declare home offset
S            'Abruptly stop trajectory
MP           'Switch to Position mode
VT=20000    'Set higher maximum velocity
PT=0         'Set target position to be home
G            'Start motion
TWAIT        'Wait for motion to complete
AMPS=1023    'Restore current limit to maximum
END          'End program
```

RELATED COMMANDS:

RCKS *Report Checksum (see page 701)*

RUN *Run Program (see page 714)*

RUN? *Halt Program Execution Until RUN Received (see page 716)*

Z *Total CPU Reset (see page 846)*

ENDIF

End IF Statement

APPLICATION:	Program execution and flow control
DESCRIPTION:	IF <i>formula</i> ... ENDIF control block terminator
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	Requires corresponding IF <i>formula</i> ; can be executed only from within a user program
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

Each control block beginning with an IF formula must have a corresponding ENDIF block exit statement. Regardless of the execution path through the control block at run time, the program statement after ENDIF is the common exit point branched to after processing the IF...ENDIF control block.

NOTE: There can only be one ENDIF statement for each IF statement; every IF structure must be terminated with an ENDIF statement.

The common exit point after ENDIF is branched to when:

- Processing a true IF formula clause and encountering ELSEIF, ELSE or ENDIF
- Processing a true ELSEIF formula and encountering another ELSEIF, ELSE or ENDIF
- Processing an ELSE clause and encountering ENDIF
- All IF and ELSEIF formulas are false, and there is no ELSE clause

ENDIF is not a valid terminal command; it is only valid within a user program.

EXAMPLE:

```
a=1      'PRINT("FALSE") is always executed
IF a==2
    PRINT ("TRUE")
ELSE
    PRINT ("FALSE")
ENDIF
```

RELATED COMMANDS:

ELSE *IF-Structure Command Flow Element (see page 428)*

ELSEIF formula *IF-Structure Command Flow Element (see page 430)*

IF formula *Conditional Program Code Execution (see page 506)*

APPLICATION:	Program execution and flow control
DESCRIPTION:	SWITCH formula...ENDS control block terminator
EXECUTION:	N/A
CONDITIONAL TO:	N/A
LIMITATIONS:	Requires corresponding SWITCH formula; can be executed only from within a user program
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

Each SWITCH formula must have a corresponding ENDS block exit statement. Regardless of the execution path through the control block at run time, any program statement immediately after ENDS is the common exit point branched to when processing the SWITCH...ENDS control block.

NOTE: There can only be one ENDS statement for each SWITCH statement.

The common exit point after ENDS is branched to when:

- Encountering a BREAK
- Encountering ENDS
- The SWITCH formula value is not equal to any CASE number value, and there is no DEFAULT statement label for the control block

ENDS is not a valid terminal command; it is only valid within a user program.

EXAMPLE:

```
SWITCH x
  CASE 1 PRINT ("x=1", #13) BREAK
  CASE 2 PRINT ("x=2", #13) BREAK
  CASE 3 PRINT ("x=3", #13) BREAK
ENDS
'This is the exit point for SWITCH...ENDS code block
```

RELATED COMMANDS:

BREAK *Break from CASE or WHILE Loop (see page 331)*

CASE formula *Case Label for SWITCH Block (see page 360)*

DEFAULT *Default Case for SWITCH Structure (see page 388)*

SWITCH formula *Switch, Program Flow Control (see page 766)*



APPLICATION:	I/O control
DESCRIPTION:	Configure a specified output to control an external brake
EXECUTION:	Immediate; remains in effect until otherwise commanded
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	See below for range of IO — depends on motor series
TYPICAL VALUES:	See below for range of IO — depends on motor series
DEFAULT VALUE:	Class 5 motors default to disabled: EOBK(-1) For Class 6 motors, the default is EOBK(8)
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	EOBK(-1):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:



WARNING: For Class 5 D-series motors, certain special features may override the brake function. In particular, the MFR, MSR, MF0, MS0 commands, or any similar feature from a network interface (including CANopen modes of operation: -1, -3, -11), may interfere with a brake assignment to I/O 0 or 1). Therefore, use of I/O 0 or 1 *is not* recommended for the brake in the Class 5 D-series if follow or step modes are used, regardless of SRC setting. For a programming example, refer to Programming Note on page 446.

If an external brake is used instead of the optional internal brake, the EOBK(IO) command allows automatic control of the external brake through a selectable I/O port pin.

EOBK(-1) disables the brake output from any I/O pin.

The logic state corresponds with the current brake-control method (i.e., BRKRLS, BRKENG, BRKSRV or BRKTRJ).

Motor Type	IO Range	Brake Engaged Voltage ^a	Brake Released Voltage ^b
Class 5 D-style	0-6	5	0
Class 5 D-style with AD1 option	0-6	5	0
	16-25	0	24
Class 5 M-style	0-10	0 ^c	24 ^c
Class 6 D-style, M-style	8	0	24
^a Commanded to lock the motor shaft. ^b Commanded to allow the motor to rotate. ^c Requires firmware 5.x.3.60 or newer for Class 5 M-style motor.			

Programming Note

NOTE: When using the EOBK command in programs with follow or step modes commands, be aware of the information in this section.

In situations where EOBK(0) is used before follow or step modes commands, for example, MFR, note that these commands interfere with I/O 0 and 1. This defeats, for example, EOBK(0), from working properly when it is placed before MFR.

To program this correctly:

- Choose an output value for EOBK that is something other than 0 or 1, e.g., EOBK(2):

```
EOBK (2)
...
MFR
G
```

OR

- If EOBK(0) (or EOBK(1)) must be used, be sure to reissue EOBK *after* MFR but *before* the G command:

```
MFR
EOBK (0)
G
```

EXAMPLE:

```
EIGN(W,0)    'Make all onboard I/O inputs
ZS           'Clear errors
EOBK(22)     'Set output 22 to brake control
```

RELATED COMMANDS:

BRKENG *Brake Engage (see page 333)*

BRKRLS *Brake Release (see page 335)*

BRKSRV *Brake Servo, Engage When Not Servoing (see page 337)*

BRKTRJ *Brake Trajectory, Engage When No Active Trajectory (see page 339)*



APPLICATION:	I/O control
DESCRIPTION:	Configure a specified output for the fault indication.
EXECUTION:	Immediate; remains in effect until otherwise commanded
CONDITIONAL TO:	N/A
LIMITATIONS:	Class 5 motors are not supported.
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	See details for range of I/O — depends on motor series.
TYPICAL VALUES:	See details for range of I/O — depends on motor series.
DEFAULT VALUE:	For Class 6 motors, the factory EEPROM default is EOFT(9). Power-on behavior depends on most recent setting of this command.
FIRMWARE VERSION:	6.0.2.35 (M); 6.4.2.x (D); no Class 5
COMBITRONIC:	EOFT(-1):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

EOFT(-1) disables the fault indication from the assigned I/O pin and returns that pin to a user-controlled state. This allows the "not fault" pin to be used as ordinary user I/O.

To return the "not fault" output function to the intended pin, issue EOFT(9) to set logical I/O 9 as the assigned output for that feature.

NOTE: This command is non-volatile due to the potential for a glitch on start-up when used as user-controlled output because this feature is enabled by default. In other words, if EOFT(-1) is issued to disable this feature, it will stay disabled through a power-cycle so that the user-controlled application of that I/O pin won't initially output the "not fault" signal.

Class	Motor Type	I/O Range
Class 5	D-style	N/A
	D-style with AD1 option	N/A
	M-style	N/A
Class 6	M-style	9
	D-style	9

EXAMPLE:

```
EIGN(W,0)   'Make all onboard I/O inputs  
ZS          'Clear errors  
EOFT(9)     'Set output 9 as to "not fault" indication
```

RELATED COMMANDS:

OR(value) *Output, Reset (see page 638)*

OS(...) *Output, Set (see page 640)*

OUT(...)=formula *Output, Activate/Deactivate (see page 644)*

EOIDX(number) Encoder, Output Index

APPLICATION:	I/O control
DESCRIPTION:	Output encoder index to specified logical I/O
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	Command is not available for any Class 6 motors
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	Depends on the motor model (see details)
TYPICAL VALUES:	Depends on the motor model (see details)
DEFAULT VALUE:	-1 (disabled)
FIRMWARE VERSION:	5.x.4.42 or later; no Class 6
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The EOIDX() command is used to output the encoder's index to the output specified by the number parameter, where number=

- -1 to disable (this is the default state on power-up), or
- a valid logical I/O number is specified based on the motor style:
 - For Class 5 D-Style:
 - number=6 to use logical I/O 6 (pin 7 on DA-15 15-pin connector)
 - For Class 5 M-Style:
 - number=7 to use logical I/O 7 (pin 6 on circular M12 12-pin connector)

EXAMPLE: (enabling and disabling)

```
EOIDX(6)    'Enable for an SM23165D motor
```

```
EOIDX(-1)  'Disable
```

RELATED COMMANDS:

EIRE *Enable Index Register, Encoder Capture (see page 419)*

EIRI *Enable Index Register, Input Capture (see page 421)*

APPLICATION:	EEPROM (Nonvolatile Memory)
DESCRIPTION:	Set user data EEPROM pointer
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	REPTR
WRITE:	Read/write EPTR auto incremented as used
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	EEPROM address pointer (bytes)
RANGE OF VALUES:	0 to 32767
TYPICAL VALUES:	0 to 32767
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

EPTR sets the address location (pointer) within the nonvolatile, user-data EEPROM for the data-retrieval read VLD(variable, number) function, and data-storage write VST(variable, number) function. EPTR auto-increments by 1, 2 or 4 with each read or write access to the physical EEPROM device according to the current data type.

EXAMPLE:

```
EPTR=4000      'Set EPTR = 4000
VST(hh,1)     'Store a 32-bit value; EPTR is now 4004
VST(ab[7],1)  'Store an 8-bit value; EPTR is now 4005
VST(aw[7],1)  'Store a 16-bit value; EPTR is now 4007
VST(x,3)      'Store three consecutive variables: x, y, z
               'EPTR is now 4007+(3*4) or 4019
VST(x,4)      'INVALID !!! EPTR remains 4019 !!!
```

NOTE: You cannot store consecutive variables past their group range (i.e., consecutive variables a-z, aa-zz or aaa-zzz must be stored within their groups).

```
VST(aa,26)    'Perfectly valid !!!
VST(aa,27)    'INVALID !!!
```

RELATED COMMANDS:

VLD(variable,number) *Variable Load (see page 820)*
VST(variable,number) *Variable Save (see page 824)*

ERRC

Error Code, Command

APPLICATION:	System
DESCRIPTION:	Get code for most recent command error
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RERRC
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	Output: 0-65535
TYPICAL VALUES:	0-33
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The ERRC command reports the most recent command error. Command errors originate from SmartMotor commands through a user program, serial port or command encapsulation such as CANopen object 2500h. The command error bit status word 2, bit 14 will be indicated when a new error has occurred. The ERRC command can be used to determine which error has occurred. Refer to the next table.

Code	Description	Notes
0	NO_ERROR	
1	BAD_LENGTH	A command is too long, or contains too many parameters if a variable list of parameters is allowed.
2	BAD_ARGUMENT	One or more of the values given to a command was out of range, so the command was aborted.
3	BAD_PACKET	Command was not recognized. Check the command name and syntax.
4	BAD_OPERATION	General error, command not allowed in the current state of the motor.
5	MISSING_ARGUMENT	Incomplete formula
6	Reserved	Reserved
7	ERROR_PARENTHESIS	Reserved
8	Reserved	Reserved
9	LENGTH_VIOLATION	Embedded address in a user program was not found within the 64-character buffer for IF, SWITCH, GOTO, etc.
10	BAD_ARRAY_ADDR	Array index outside the defined range.
11	DIVIDE_BY_ZERO	Attempt to divide by 0.
12	STACK_OVERFLOW	No room on stack for another 10 GOSUB and INTERRUPTS use the same stack.
13	STACK_UNDERFLOW	RETURN or RETURNI with no place to return.

Code	Description	Notes
14	BAD_LABEL	Label does not exist for GOSUB or GOTO.
15	NESTED_SWITCH	Reserved
16	BAD_FORMULA	Formula used in an assignment, IF, or SWITCH is improperly formatted. Check syntax, parenthesis, operators, etc.
17	BAD_WRITE_LENGTH	VST command amount written too long
18	NOT_USED	Reserved
19	BAD_BIT	Z{letter} command issued for a bit that cannot be reset; ITR command may also show this error if incorrect bit number is specified.
20	INVALID_INTERRUPT	EITR command for interrupt not defined.
21	NO_PERMISSION	Operation or memory range is not user-accessible.
22	OPERATION_FAILED	General error
23	MATH_OVERFLOW	A math operation (add, multiply, etc.) overflowed, or an assignment to a variable where the value is outside of the allowable range.
24	CMD_TIMEOUT	Combitronic timeout
25	IO_NOT_PRESENT	Attempt to access an I/O point that does not exist.
26	NO_CROSS_AXIS_SUPPORT	Attempt to use Combitronic on a command that does not support it.
27	BAD_MOTOR_STATE	Reserved
28	BAD_CROSS_AXIS_ID	Combitronic command issued to an invalid remote device address (greater than 127); requires firmware version 5.x.4.55 or later.
29	BAD_COMBITRONIC_FCODE	The remote Combitronic does not support the command called, possibly due to older firmware or different motor series.
30	BAD_COMBITRONIC_SFCD	The remote Combitronic does not support the command called, possibly due to older firmware or different motor series.
31	EE_WRITE_QUEUE_FULL	EEPROM in the motor was not able to save data as requested.
32	CAM_FULL	Cam table cannot fit a table of the specified size (CTA command).
33	GOSUB_GOTO_BLOCKED	GOSUB attempted but motor cannot process it. This could happen when another command source (such as CANopen) attempts a GOSUB during a user program upload/download.

EXAMPLE:

```
x=ERRC      'Assign error value to the variable x
```

ERRC may be used in SWITCH CASE code:

```
SWITCH ERRC
    CASE 2      PRINT ("BAD_ARGUMENT", #13)      BREAK
    CASE 11     PRINT ("DIVIDE_BY_ZERO", #13)     BREAK
ENDS
```

RELATED COMMANDS:

^R ERRW *Communication Channel of Most Recent Command Error (see page 453)*

Communication Channel of Most Recent Command Error

APPLICATION:	System
DESCRIPTION:	Get communication channel of most recent command error
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RERRW
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	Output: 0-65535
TYPICAL VALUES:	0-3
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The ERRW command reports the command source of the most recent command error. Command errors originate from SmartMotor commands through a user program, serial port or command encapsulation such as CANopen object 2500h. The command error bit status word 2, bit 14 will be indicated when a new error has occurred. The ERRW command can be used to determine the source of the error. Refer to the next table.

Code	Description	Notes
0	Com 0	Communications port 0 (RS-232 for D-style, RS-485 for M-style)
1	Com 1	Communications port 1 (RS-485 for D-style only)
2	Program	From user program running in the motor
3	Reserved	NOTE: Class 6 firmware 6.0.2.36 and previous reported code 3 for program. This has been corrected in newer Class 6 firmware versions.
4	CANopen encapsulation	Object 0x2500 encapsulation (CANopen or EtherCAT/CoE)
5	Modbus encapsulation 0	Serial command encapsulation over Modbus RTU attached to Com 0.
6	Modbus encapsulation 1	Serial command encapsulation over Modbus RTU attached to Com 1.
7	TCP encapsulation	Class 6 only, EtherNet/IP firmware only. Introduced firmware version 6.0.2.37.
8	USB	Class 6 only
9	Special system use	

EXAMPLE:

```
x=ERRW      'Assign error value to the variable x
```

ERRW may be used in SWITCH CASE code:

```
SWITCH ERRW
  CASE 0 PRINT("Command Error on Com Channel 0",#13)  BREAK
  CASE 1 PRINT("Command Error on Com Channel 1",#13)  BREAK
  CASE 2 PRINT("Command Error in User Program",#13)   BREAK
ENDS
```

RELATED COMMANDS:

R ERRC *Error Code, Command (see page 451)*

ETH(arg)

Get Ethernet Status and Errors

APPLICATION:	Communications control
DESCRIPTION:	Assign the result to a variable, or report errors and certain status information for the industrial Ethernet (IE) bus
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RETH(arg)
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	6.x (D/M); no Class 5
COMBITRONIC:	EtherNet/IP supports Combitronic addressing for RETH(5) and RETH(45) through RETH(49). E.g., RETH(5):3, a=ETH(5):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The ETH command is used to gather specific error or status information relating to the industrial Ethernet (IE) bus interface.

- Assign the result to a program variable: x=ETH(arg)
- As a report: RETH(arg)

Specific features are based on the fieldbus network being used. See the corresponding SmartMotor fieldbus guide for more details.

EXAMPLE:

```
x=ETH(0)           'Get the Ethernet status and assign it to x.
```

RELATED COMMANDS:

ETHCTL(function,value) *Control Industrial Ethernet Network Features (see page 456)*

IPCTL(function,"string") *Set IP Address, Subnet Mask or Gateway (see page 515)*

SNAME("string") *Set PROFINET Station Name (see page 754)*

ETHCTL(function,value)

Control Industrial Ethernet Network Features

APPLICATION:	Communications control
DESCRIPTION:	Control features of the industrial Ethernet (IE) network
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	See below
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	6.x (D/M); no Class 5
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The ETHCTL command is used to control the industrial Ethernet (IE) network features. Commands execute based on the action argument, which controls Ethernet functions. After issuing an ETHCTL command, the Ethernet error codes will be checked to determine the state of Object 2304h, sub-index 3, bit 6 (Ethernet error).

Specific features are based on the fieldbus network being used. See the corresponding SmartMotor fieldbus guide for more details.

EXAMPLE:

```
ETHCTL(13,0) 'Disables 402 profile (motion) commands in EtherCAT
ETHCTL(13,1) 'Enables 402 profile (motion) commands in EtherCAT
```

RELATED COMMANDS:

IPCTL(function,"string") *Set IP Address, Subnet Mask or Gateway (see page 515)*

SNAME("string") *Set PROFINET Station Name (see page 754)*



APPLICATION:	Motion control; supports the DS2020 Combitronic system
DESCRIPTION:	Load buffered PID values into PID filter
EXECUTION:	Next PID sample
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	F:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The servo tuning parameters KA, KD, KG, KI, KL, KP, KS, and KV are all buffered parameters. Once requested, these parameters take effect only when the F command is issued. This allows several parameters to be changed simultaneously without intermediate tuning states causing disruptions. Tuning parameters can be changed during a move profile, although caution is urged.



CAUTION: Use caution when changing the servo tuning parameters during a move profile.

Different motor sizes have different optimal PID default gain values. A default set of tuning parameters is in effect at power up or reset of the motor. However, the default tuning parameters are optimized for an unloaded shaft.

EXAMPLE:

```

KP=100           'Initialize KP to a some value
F               'Load into present PID filter
G               'Start motion
WAIT=40000
KP=KP+10        'Increment the present KP gain value
F               'Change into filter
END

```

EXAMPLE: (Routine homes motor against a hard stop)

```

MDS           'Using Sine mode commutation
KP=3200       'Increase stiffness from default
KD=10200      'Increase damping from default
F            'Activate new tuning parameters
AMPS=100      'Lower current limit to 10%
VT=-10000     'Set maximum velocity
ADT=100       'Set maximum accel/decel
MV           'Set Velocity mode
G            'Start motion
WHILE EA>-100 'Loop while position error is small
LOOP         'Loop back to WHILE
O=-100       'While pressed, declare home offset
S            'Abruptly stop trajectory
MP           'Switch to Position mode
VT=20000     'Set higher maximum velocity
PT=0         'Set target position to be home
G            'Start motion
TWAIT        'Wait for motion to complete
AMPS=1023    'Restore current limit to maximum
END          'End program

```

RELATED COMMANDS:

R KA=formula *Constant, Acceleration Feed Forward (see page 526)*

R KD=formula *Constant, Derivative Coefficient (see page 528)*

R KG=formula *Constant, Gravitational Offset (see page 530)*

R KI=formula *Constant, Integral Coefficient (see page 532)*

R KL=formula *Constant, Integral Limit (see page 535)*

R KP=formula *Constant, Proportional Coefficient (see page 537)*

R KS=formula *Constant, Velocity Filter Option (for KD) (see page 540)*

R KV=formula *Constant, Velocity Feed Forward (see page 542)*



FAUSTS(x)

Returns Fault Status Word

APPLICATION:	Motion control, specific to the DS2020 Combitronic system
DESCRIPTION:	Reports the 32-bit fault status word x
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	Command is not available for Class 6 M-series SmartMotors
READ/REPORT:	RFAUSTS(x)
WRITE:	Read only
LANGUAGE ACCESS:	N/A
UNITS:	Number
RANGE OF VALUES:	0 to 2
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.0.4.55/5.98.4.55 (D/M); 6.4.2.x (D); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	RFAUSTS(x):3, a=FAUSTS(x):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The =FAUSTS(x) command is used to report the 32-bit fault word "x" of the DS2020 Combitronic system. The syntax of the command is:

```
a=FAUSTS(x)
```

where "a" is a variable and "x" is the 32-bit fault word.

The RFAUSTS(x) command is used to report the 32-bit fault word "x" of the DS2020 Combitronic system. The syntax of the command is:

```
RFAUSTS(x)
```

where x is the 32-bit fault word.

The range of values is from 0 to 2:

0 = Fault Word 0

1 = Fault Word 1

2 = Fault Word 2

Note that there is also a set of DS2020 Combitronic system 16-bit status words, which can be reported and assigned like the SmartMotor status words. For details, see Fault and Status Words - DS2020 Combitronic System on page 932.

EXAMPLE:

RFAUSTS (2)

RELATED COMMANDS:

R B(word,bit) *Status Byte (see page 297)*

R FSAD(n,m) *Set Reaction to Fault (see page 467)*

R W(word) *Report Specified Status Word (see page 833)*



Resolution to Set Units of Position/Velocity/Acceleration

APPLICATION:	Motion control, specific to the DS2020 Combitronic system
DESCRIPTION:	Select the resolution that sets the units of position/velocity/acceleration
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	Command is not available for Class 6 M-series SmartMotors
READ/REPORT:	RFD
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment only
UNITS:	Number
RANGE OF VALUES:	0 to 4294967295
TYPICAL VALUES:	8192 - 2097152
DEFAULT VALUE:	65536
FIRMWARE VERSION:	5.0.4.55/5.98.4.55 (D/M); 6.4.2.x (D); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	FD:3=1234, a=FD:3, RFD:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The FD command provides the ability to select the resolution that sets the units of position/velocity/acceleration (i.e., how many counts there are in one motor revolution for position; how many counts there are in one motor revolution per second for velocity, and how many counts there are in one motor revolution per squared second for acceleration).

A report command, RFD, is also available that reads the currently set value.

The value range on the DS2020 Combitronic system is from 0 to 4294967295.

EXAMPLE:

FD=1234

RELATED COMMANDS:

ADT=formula *Acceleration/Deceleration Target (see page 263)*

R AT=formula *Acceleration Target (see page 286)*

R DT=formula *Deceleration Target (see page 396)*

R FAUSTS(x) *Returns Fault Status Word (see page 459)*

R FSAD(n,m) *Set Reaction to Fault (see page 467)*

R RES *Resolution (see page 702)*

O=formula, O(trj#)=formula *Origin (see page 628)*

R PA *Position, Actual (see page 646)*

R PT=formula *Position, (Absolute) Target (see page 690)*

R VA *Velocity Actual (see page 807)*

R VT=formula *Velocity Target (see page 828)*

FABS(value) Floating-Point Absolute Value of ()

APPLICATION:	Math function
DESCRIPTION:	Gets the floating-point absolute value of the input value
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RFABS(value)
WRITE:	N/A
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	Input: floating-point variable range; integer inputs or literal values limited are limited to -2147483648 to 2147483647 Output: floating-point value range
TYPICAL VALUES:	Input: floating-point variable range; integer inputs or literal values limited are limited to -2147483648 to 2147483647 Output: floating-point value range
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

FABS takes an input and returns the floating-point absolute value:

$$af[1]=FABS(arg)$$

where arg may be an integer (e.g., a or aw[0]) or floating-point variable (e.g., af[0]). Integer or floating-point constants may also be used (e.g., 23 or 23.7, respectively).

This command cannot have within the parenthesis: math operators, other parenthetical functions, or a Combitronic request from another motor. For example, x=FABS(PA) is allowed, but x=FABS(PA:3) is not allowed.

The result of this function is a floating-point type. If used in an equation, the operations in the equation that are processed after this function are automatically promoted to a float. This is dependent on the mathematical order of operations in the equation. As with other equations (e.g., x=a+b), the variable to the left of "=" may be an integer variable to accept the result. However, the value will be truncated to fit to that integer type. For example, the assignment "aw[0]=" will drop any fractional amount and truncate the result to the range -32768 to 32767 (aw[0]=100.5 will report as 100, and aw[0]=40000.0 will report as -25536).

EXAMPLE:

```
af[0]=FABS(-5.545)      'Set array variable = FABS(-5.545)
PRINT(af[0],#13)      'Print value of array variable af[0]
RFABS(-5.545)         'Report FABS(-5.545)
END
```

Program output is:

```
5.545000076
5.545000076
```

RELATED COMMANDS:

^R ABS(value) *Absolute Value of () (see page 255)*



FSA(cause,action)

Fault Stop Action

APPLICATION:	Motion control
DESCRIPTION:	Fault stop action
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	FSA(0,0)-FSA(1,2)
TYPICAL VALUES:	FSA(0,0)-FSA(1,2)
DEFAULT VALUE:	FSA(0,0)
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	FSA(0,0):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

FSA(cause,action) is used to specify the fault type and fault mode. FSA(0,0) is the default configuration, which sets the fault action of all types of faults to result in Mode Torque Brake (MTB).

FSA(cause,action) sets the fault stop action, where:

cause: specifies the type of fault that will trigger the *action*:

- 0 - All types of faults
- 1 - Hardware travel limit faults
- 2 - Soft limit faults (6.0.x.x firmware only)

action: specifies the desired action:

- 0 - Default action (MTB)
- 1 - Servo off (freewheel)
- 2 - X command
- 3 - Hard stop (6.0.x.x firmware only)

EXAMPLE: (Shows various values for the FSA command)

```
FSA(0,0)  'All Faults: servo will dynamically brake to stop.
FSA(0,1)  'All Faults: servo will turn off (freewheel).
FSA(0,2)  'All Faults: servo will decelerate to stop with "X" command.

FSA(1,0)  'Hardware Travel Limit Faults: servo will dynamically brake to stop.
FSA(1,1)  'Hardware Travel Limit Faults: servo will turn off (freewheel).
FSA(1,2)  'Hardware Travel Limit Faults: servo will decelerate to
           'stop with "X" command.
```

RELATED COMMANDS:

BRKRLS *Brake Release (see page 335)*
MTB *Mode Torque Brake (see page 622)*
OFF *Off (Drive Stage Power) (see page 636)*
X *Decelerate to Stop (see page 844)*



APPLICATION:	Motion control, specific to the DS2020 Combitronic system
DESCRIPTION:	Set reaction m to fault number n
EXECUTION:	Immediate
CONDITIONAL TO:	Can be used only when drive is not enabled
LIMITATIONS:	Command is not available for Class 6 M-series SmartMotors
READ/REPORT:	RFSAD(n)
WRITE:	Read/write
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	n: 1 to 73 m: 0 to 4, (127 as report only)
TYPICAL VALUES:	Depends on the fault
DEFAULT VALUE:	Depends on the fault
FIRMWARE VERSION:	5.0.4.55/5.98.4.55 (D/M); 6.4.2.x (D); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	FSAD(n,m):3, a=FSAD(n):3, RFSAD(n):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The FSAD(n,m) command is used to set reaction m to fault number n for the DS2020 Combitronic system. The commands RFSAD(n) and =FSAD(n) are also available that read the currently set value for fault number n. For fault numbers, refer to the Fault column in Fault Tables on page 932.

During its operation, the DS2020 Combitronic system can signal more than 70 faults, which are organized into three 32-bit status words (0, 1, 2). For details, see Fault and Status Words - DS2020 Combitronic System on page 932.

To set reaction m to fault number n for the DS2020 Combitronic system, the syntax of the command is:

FSAD(n,m)

where n is the fault number and m is the desired reaction as shown:

m	Reaction
0	None
1	Send CANopen emergency message
2	Disable power stage
3	Slow down ramp
4	Quick stop ramp
127	Disable power stage for hard faults (n = 1-8, 12, 13 and 19-21), not selectable or editable

- For hard faults, the reaction is always disable power stage. These faults respond to RFSA(n) with 127; they behave exactly like faults with code 2.
- Reactions 2, 3, 4 and 127 also send an emergency message.
- Reactions 3, 4, once the ramp is terminated, disable the drive. These two commands act like X, S with that difference.
- If the motor has a brake and it is configured, once the drive is disabled (reaction 2, 3, 4, 127), the brake is engaged.

Understanding Fault Reactions

NOTE: Setting fault reactions must be done carefully and with a precise knowledge of the physical implications of the possible reactions to a particular fault.

When setting fault reactions, remember:

- Reactions 3, 4 (slow and quick stops) are executed by actively driving the motor, generating PWM signals to control it
- Only reaction 2 and 127 immediately disables the PWM generation
- Reactions 0 and 1 (none and emergency message) leads to no physical action on the system

The next list describes some typical scenarios:

- Hard faults are locked to reaction 127, because it is impossible, from a hardware point of view, to supply the PWM signal to control the motor.
- Overvoltage and overtemperature should not lead to an active motor stop, because the current flow can worsen the situation; therefore, disabling the drive (reaction 2) is the best choice.
- Faults related to the position/velocity transducer should disable the drive, because if the feedback reports an incorrect position, then an improper magnetic field is created, making the motor no longer controllable.
- Negative and positive limit switches, if asserted, indicate that the maximum position has been reached; therefore, the motor should be immediately stopped to avoid a mechanism crash.

EXAMPLE: (Shows various values for the FSAD command)

```
FSAD(72,2)  'Positive limit switch fault: servo will turn off
FSAD(73,4)  'Negative limit switch fault: servo quickly decelerates to stop
```

RELATED COMMANDS:

FSA(cause,action) *Fault Stop Action (see page 465)*

FSQRT(value)

Floating-Point Square Root

APPLICATION:	Math function
DESCRIPTION:	Gets the floating-point square root of the specified variable or number
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RFSQRT(value)
WRITE:	N/A
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Floating-point number
RANGE OF VALUES:	Input: Non-negative floating-point variable range, integer inputs or literal values are limited to 0 to 2147483647 Output: Non-negative floating-point value range
TYPICAL VALUES:	Input: Non-negative floating-point variable range, integer inputs or literal values are limited to 0 to 2147483647 Output: Non-negative floating-point value range
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

FSQRT takes a value and reports the floating-point square root:

$$af[1]=FSQRT(arg)$$

where arg may be an integer (e.g., a or aw[0]) or floating-point variable (e.g., af[0]). Integer or floating-point constants may also be used (e.g., 23 or 23.7, respectively).

This command cannot have within the parenthesis: math operators, other parenthetical functions, or a Combitronic request from another motor. For example, x=FABS(PA) is allowed, but x=FABS(PA:3) is not allowed.

The result of this function is a floating-point type. If used in an equation, the operations in the equation that are processed after this function are automatically promoted to a float. This is dependent on the mathematical order of operations in the equation. As with other equations (e.g., x=a+b), the variable to the left of "=" may be an integer variable to accept the result. However, the value will be truncated to fit to that integer type. For example, the assignment "aw[0]=" will drop any fractional amount and truncate the result to the range -32768 to 32767 (aw[0]=100.5 will report as 100, and aw[0]=40000.0 will report as -25536).

Although the floating-point variables and their standard binary operations conform to IEEE-754 double precision, the floating-point square root and trigonometric functions only produce IEEE-754 single-precision results. For more details, see Variables and Math on page 198.

EXAMPLE:

```

a=9           'Set variable a = 9
af[0]=FSQRT(4) 'Set array variable af[0] = FSQRT(4)
RFSQRT(4)
af[1]=FSQRT(6.5) 'Set array variable af[1] = FSQRT(6.5)
RFSQRT(6.5)
af[2]=FSQRT(8.5) 'Set array variable af[2] = FSQRT(8.5)
RFSQRT(8.5)
af[3]=FSQRT(a)   'Set array variable af[3] = FSQRT(a)
RFSQRT(a)
PRINT(af[0]," ",af[1]," ",af[2]," ",af[3],#13) 'Print variable values
END

```

Program output is:

```

2.000000000
2.549509763
2.915475845
3.000000000
2.000000000, 2.549509763, 2.915475845, 3.000000000

```

RELATED COMMANDS:

`R SQRT(value)` *Integer Square Root (see page 757)*



APPLICATION:	System
DESCRIPTION:	Gets (reads) the firmware version
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RFW
WRITE:	Read only
LANGUAGE ACCESS:	N/A
UNITS:	Long
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	a=FW:3, RFW:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The FW command gets (reads) the motor firmware version as a long (32-bit) value. This allows user programs to inspect the current firmware version. A comparison can be made to check for an exact version number. This can be a very important part of ensuring the integrity of a system. For instance, a program can check the firmware version and refuse to operate a machine if an improper firmware version has been loaded.



CAUTION: If a machine has been tested and certified using a particular firmware version, then older or newer firmware may produce unexpected results.

The encoding of this 32-bit number uses the format of firmware numbering, but compresses this value into a single (large) number. For example, firmware version 5.0.3.44 is converted to four individual bytes: 5, 0, 3 and 44. These four bytes are then assembled into a 32-bit number — the 5 (class) becomes the most significant byte, and the 44 (minor version) becomes the least significant byte. The resulting number in decimal format is 83886892.

EXAMPLE:

```
PRINT (FW, #13)   'Print the motor firmware version as a long
RFW              'Report the motor firmware version as a long
END
```

Program output is:

83886892
83886892

RELATED COMMANDS:

^R SP2 *Bootloader Version (see page 755)*



APPLICATION:	Motion control; supports the DS2020 Combitronic system
DESCRIPTION:	Initiate motion, or change trajectory parameters of existing motion.
EXECUTION:	Immediate
CONDITIONAL TO:	Drive is ready (status word 0, bit 0)
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	G:3 or G(0):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The G (Go) command is used to start motion or update buffered values such as speed or acceleration.

The G command is required in each of these cases:

- Initiate an Absolute Move in Mode Position (MP):
`VT=10000 ADT=100 PT=1234 MP G`
- Initiate a Relative Move in Mode Position (MP):
`VT=10000 ADT=100 PRT=4000 MP G`
- Initiate a Velocity in Mode Velocity (MV):
`VT=10000 ADT=100 MV G`
- Change to a new Velocity in Mode Position (MP) or Mode Velocity (MV):
`VT=10000 ADT=100 MV G WAIT=1000 VT=VT*2 G`
- Change to a new Acceleration in Mode Position (MP) or Mode Velocity (MV):
`VT=10000 ADT=100 MV G WAIT=1000 ADT=200 G`
- Initiate an Electronic Gear Ratio in Mode Follow with Ratio (MFR):
`MFO MFMUL=1 MFDIV=10 MFR G`

- Initiate an Electronic Gear Ratio in Mode Step with Ratio (MSR):

```
MF0 MFMUL=1 MFDIV=10 MSR G
```

- Initiate Cam Mode (MC):

```
MF0 MC G
```

- Initiate a phase offset move while in Electronic Gear Ratio in either Mode Follow or Mode Step:

```
MF0 MFMUL=1 MFDIV=10 MFR G MP(1) WAIT=2000 PRT=2000 VT=100 G(1)
```

NOTE: The MFR (or MSR) command does NOT need to come after the MFMUL and MFDIV commands. If MFMUL or MFDIV is changed, the G command will enable the corresponding change in ratio.

- Initiate a homing operation when the MH homing mode is selected, and associated commands like HM_ADT, etc. have been set. Due to the nature of the homing mode being associated with the CANopen homing state machine, the firmware remembers when the homing operation was started from G, so that the S and X commands will know to abort that operation. There is finer control that is possible by manipulation of the CANopen control word and other homing options that are not covered here. For more details on SmartMotor homing operations, see the *SmartMotor Homing Procedures and Methods Application Note*.

On power-up, the motor defaults to the off state with MP (Mode Position) buffered with no velocity or acceleration values. As a result, if G is issued, the motor will immediately servo in place.

If a G command is transmitted and no motion results, any of these items may be the cause:

- EL=0 or too small
- ADT=0 or 1
- VT=0 or so small that motion is not visible
- Target position equals current position
- PRT=0
- Bh=1 the motor is hotter than max permitted temperature TH
- AMPS=0 or too small
- T=0 or too small
- Motor is in Torque mode
- Limit input switch(s) asserted
- External encoder signal not present or not changing (in Follow modes)
- Motor is part of a daisy chain that has not been properly configured
- Serial communications are good but target motor is not addressed
- Serial communications at incorrect baud rate
- Serial communications cable not attached or poorly connected
- Motor has no drive power
- Motor has a previous fault that needs to be cleared
- Motor has no connections to limit switch inputs on boot-up and, therefore, has a travel-limit fault
- Drive enable is not asserted (M-style only)
- Bus voltage is too high or too low

EXAMPLE:

```

ADT=100      'Set buffered accel/decel
VT=10000    'Set buffered velocity
PT=1000     'Set buffered position
MP          'Set buffered Position Mode
G           'Load buffered move, start motion

```

To servo in place:

```

PT=PA      'Set buffered position = actual position
G         'Set buffered Velocity

```

EXAMPLE: (Routine homes motor against a hard stop)

```

MDS          'Using Sine mode commutation
KP=3200     'Increase stiffness from default
KD=10200    'Increase damping from default
F           'Activate new tuning parameters
AMPS=100    'Lower current limit to 10%
VT=-10000   'Set maximum velocity
ADT=100     'Set maximum accel/decel
MV          'Set Velocity mode
G           'Start motion
WHILE EA>-100 'Loop while position error is small
LOOP        'Loop back to WHILE
O=-100     'While pressed, declare home offset
S          'Abruptly stop trajectory
MP         'Switch to Position mode
VT=20000   'Set higher maximum velocity
PT=0       'Set target position to be home
G         'Start motion
TWAIT     'Wait for motion to complete
AMPS=1023 'Restore current limit to maximum
END       'End program

```

RELATED COMMANDS:

ADT=formula *Acceleration/Deceleration Target (see page 263)*
R AT=formula *Acceleration Target (see page 286)*
R DT=formula *Deceleration Target (see page 396)*
EISM(x) *E-Configure Input as Sync Controller (see page 423)*
R EL=formula *Error Limit (see page 426)*
GS *Start Synchronized Motion (GO Synchronized) (see page 487)*
MC *Mode Cam (Electronic Camming) (see page 555)*
MFR *Mode Follow Ratio (see page 600)*
MP *Mode Position (see page 613)*
MV *Mode Velocity (see page 624)*
R PRT=formula *Position, Relative Target (see page 683)*
R PT=formula *Position, (Absolute) Target (see page 690)*
R VT=formula *Velocity Target (see page 828)*

GETCHR**Next Character from Communications Port 0**

APPLICATION:	Communications control
DESCRIPTION:	Get the next character in channel 0 serial input buffer
EXECUTION:	Immediate
CONDITIONAL TO:	Requires that a character is in the buffer; communications channel 0 must be open in data mode
LIMITATIONS:	Maximum buffer length is 31 characters
READ/REPORT:	RGETCHR
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	-1 to 255
TYPICAL VALUES:	-1 to 255
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

GETCHR (or GETCHR0) reads and removes the next available character in the channel 0 serial receive buffer. It is recommended to check that $LEN > 0$ before issuing the GETCHR command. Otherwise, it is possible to read an empty receive buffer.

Alternatively, the value returned from GETCHR can be checked to see if it is -1. However, do not use an `ab[]` register to store the value in this case, because the value -1 will be aliased with the character 255. Instead use an `aw[]` or `al[]` register.

Normally, the SmartMotor™ interprets incoming characters on communications channel 0 as commands. However, it is sometimes useful to prevent that from happening and write a custom command interpreter. This is accomplished by reopening the input channel in data mode with the OCHN command. For details see OCHN(...) on page 632.



CAUTION: The OCHN command will cause the SmartMotor to ignore incoming commands and can lock you out. Therefore, during development, prevent this by using the RUN? command at the start of each program.

NOTE: If you get locked out and are unable to communicate with the SmartMotor, you may be able to recover communications using the SMI software's Communication Lockup Wizard. For more details, see Communication Lockup Wizard on page 31.

EXAMPLE:

```
C20           'Place a label
IF LEN>0     'Check to see that LEN>0
  c=GETCHR   'Get character from buffer
  IF c==69   'Check to see if it is an E
    END      'End the program
  ENDIF
ENDIF
GOTO20       'Loop back to C20
```

RELATED COMMANDS:

- R GETCHR1 *Next Character from Communications Port 1 (see page 478)*
- R LEN *Length of Character Count in Communications Port 0 (see page 544)*
- R LEN1 *Length of Character Count in Communications Port 1 (see page 545)*
- OCHN(...) *Open Channel (see page 632)*

GETCHR1

Next Character from Communications Port 1

APPLICATION:	Communications control
DESCRIPTION:	Get the next character in channel 1 serial input buffer
EXECUTION:	Immediate
CONDITIONAL TO:	Requires that a character is in the buffer; communications channel 1 must be open in data mode
LIMITATIONS:	Maximum buffer length is 31 characters
READ/REPORT:	RGETCHR1
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	-1 to 255
TYPICAL VALUES:	-1 to 255
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.0.x, 5.16.x or 5.32.x series (D); 6.4.2.x (D) RGETCHR1 requires: 5.x (D/M); 6.4.2.x (D)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

GETCHR1 reads and removes the next available character in the channel 1 serial receive buffer. It is recommended to check that LEN1>0 before issuing the GETCHR1 command. Otherwise, it is possible to read an empty receive buffer.

Alternatively, the value returned from GETCHR1 can be checked to see if it is -1. However, do not use an ab[] register to store the value in this case, because the value -1 will be aliased with the character 255. Instead use an aw[] or al[] register.

Communications channel 1 can be used to accept special commands and/or data such as those from a light curtain or a barcode reader. This is done by opening the input channel in data mode with the OCHN command. For details see OCHN(...) on page 632.



CAUTION: The OCHN command will cause the SmartMotor to ignore incoming commands and can lock you out. Therefore, during development, prevent this by using the RUN? command at the start of each program.

NOTE: If you get locked out and are unable to communicate with the SmartMotor, you may be able to recover communications using the SMI software's Communication Lockup Wizard. For more details, see Communication Lockup Wizard on page 31.

NOTE: M-style motors do not have the second communications port (COM 1) needed to support the LEN1 and GETCHR1 commands.

EXAMPLE:

```
C20                'Place a label
IF LEN1>0          'Check to see that LEN1>0
    c=GETCHR1      'Get character from buffer
    IF c==69       'Check to see if it is an E
        END        'End the program
    ENDIF
ENDIF
GOTO20            'Loop back to C20
```

RELATED COMMANDS:

- R GETCHR *Next Character from Communications Port 0 (see page 476)*
- R LEN *Length of Character Count in Communications Port 0 (see page 544)*
- R LEN1 *Length of Character Count in Communications Port 1 (see page 545)*
- OCHN(...) *Open Channel (see page 632)*



GOSUB(label)

Subroutine Call

APPLICATION:	Program execution and flow control
DESCRIPTION:	Perform subroutine beginning at C{number}
EXECUTION:	Immediate
CONDITIONAL TO:	C{number} previously defined
LIMITATIONS:	Nesting must be <10 levels deep
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	Input: 0-999
TYPICAL VALUES:	Input: 0-999
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	GOSUB(0):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The GOSUB(label) command redirects program execution to a subroutine of the program marked with a label C{number}. The end of every subroutine is marked by the RETURN statement, which causes execution to return to the line after the corresponding GOSUB command.

NOTE: If a GOSUB is attempted to a label that does not exist, the GOSUB is ignored and the next program line is executed.

Subroutines may call further subroutines — that is called nesting. There may be as many as a thousand GOSUBs, but the nesting cannot exceed nine levels deep. A counter, conditional test or some other method can be used to stay within the nesting limit. A subroutine may call itself, which is called recursion. However, this practice is highly discouraged because it can lead to a stack overflow or nesting limit.

NOTE: Subroutines present a great opportunity to partition and organize your code.

The STACK control flow command explicitly and deliberately destroys the RETURN address history. Therefore, if you issue a STACK command, ensure that the program execution does not encounter a RETURN before the next GOSUB.

The GOSUB command is valid from both serial channels and within a user program. There are three forms of the command that are valid:

- GOSUB1 — Traditional format (no parenthesis)
- GOSUB(1) — With parenthesis

- GOSUB(a) — Any variable may be used

The third format allows for highly flexible programs that can call an array of different subroutines.

EXAMPLE:

```
GOSUB20      'Run subroutine 20
GOSUB21      'Run subroutine 21
a=3
GOSUB25      'Run subroutine 25
END          'End code execution

C20          'Nested subroutine
  GOSUB30
  PRINT("20",#13)
RETURN

C21          'Nested subroutine
  GOSUB30
  PRINT("21",#13)
RETURN

C25          'Recursive subroutine
  PRINT(" 25:",a)
  a=a-1
  IF a==0
    RETURN
  ENDIF
  GOSUB25
RETURN

C30          'Normal subroutine
  PRINT(#13,"Subroutine Call ")
RETURN
```

Program output is:

```
Subroutine Call 20
```

```
Subroutine Call 21
25:3 25:2 25:1
```

Referring to the previous example, the GOSUB commands: GOSUB20, GOSUB21, GOSUB25 or GOSUB30 can also be issued from the terminal.

RELATED COMMANDS:

C{number} *Command Label (see page 353)*

GOTO(label) *Branch Program Flow to a Label (see page 482)*

STACK *Stack Pointer Register, Clear (see page 761)*



GOTO(label)

Branch Program Flow to a Label

APPLICATION:	Program execution and flow control
DESCRIPTION:	Branch program execution to statement C{number}
EXECUTION:	Immediate
CONDITIONAL TO:	C{number} previously defined
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	Input: 0-999
TYPICAL VALUES:	Input: 0-999
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	GOTO(0):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:



CAUTION: Extensive use of IF statements and GOTO branches can quickly make your programs impossible to read or debug. Learn to organize your code with one main loop using a GOTO and write the rest of the program with subroutines (GOSUB). For details, see GOSUB(label) on page 480.

The GOTO{number} command unconditionally redirects program execution control to another part of the program marked by the label C{number}.

NOTE: If a GOTO jump is attempted to a label that does not exist, the GOTO is ignored and the next program line is executed. This can create problems because it would likely not be the correct command order. Therefore, be certain that every GOTO has a corresponding label.

The GOTO{number} command is valid for both serial channels and within a user program. However, take care not to issue a GOTO{number} command unless the corresponding C{number} label exists within the stored program.

The GOTO command is valid from both the serial channels and within a user program. There are three forms of the command that are valid:

- GOTO1 — Traditional format
- GOTO(1) — With parenthesis
- GOTO(a) — Any variable may be used

The third format allows for highly flexible programs that can jump to an array of different labels.

EXAMPLE:

```
C0           'Place main label
  IF IN(0)==0
    PRINT("Input 0 Low",#13)
  ENDIF
  GOTO0     'GOTO allows program to run forever
END
```

RELATED COMMANDS:

BREAK *Break from CASE or WHILE Loop (see page 331)*
C{number} *Command Label (see page 353)*
DEFAULT *Default Case for SWITCH Structure (see page 388)*
ELSE *IF-Structure Command Flow Element (see page 428)*
GOSUB(label) *Subroutine Call (see page 480)*



GROUP(function,value)

Group Address Settings

APPLICATION:	Communications control
DESCRIPTION:	Configure group addressing settings of Combitronic over Ethernet (UDP) communications.
EXECUTION:	Immediate
CONDITIONAL TO:	Availability of Combitronic over Ethernet
LIMITATIONS:	Not available with Combitronic over CAN bus
READ/REPORT:	RGROUP(function)
WRITE:	Read/write
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	See below
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	6.0.2.35 (M); 6.4.2.x (D) requires IE option; no Class 5
COMBITRONIC:	GROUP(y,z):3, a=GROUP(y):3, RGROUP(y):3 where y is the function and z is a data value, see below; where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The GROUP command configures group addressing settings of Combitronic over Ethernet (UDP) communications. The RGROUP command reports the state of those group addressing settings.

Addressing Modes

There are several modes of addressing in Combitronic over Ethernet (UDP):

- Point-to-point: a command with the specific address of a motor after the colon, for example, a:3=5. This type of message is received by the intended target and is not affected by broadcast or group-cast settings. It also has the advantage of being routable, i.e., it can find its destination based on an IP address across IP network segments if necessary.
- Broadcast: a command is directed to ":0" as a target address. Broadcast can be invoked if the group target is set to 0.
- Group-cast: a command is directed to ":255" as a target address.

Group addressing is analogous to tuning a radio to a particular channel—it allows the SmartMotor to "listen" to a particular message group. Note that ordinary point-to-point messages, for example a:3=5, will still be received by the intended destination (address 3) regardless of the group settings (target or filter) of the sender or receiver, respectively. Likewise, broadcast messages, for example a:0=5, will not be affected by group target or filter.

Group-cast messages, e.g., a:255=5, will be sent to and received by motors according to their respective group settings. For example:

- motor 1: group target is set to 4
- motor 2: group filter is set to 4
- motor 3: group filter is set to 5
- motor 4: group filter is set to 4

In this example, a message sent by motor 1 to address 255 will be received by motors 2 and 4 but not by motor 3.

Also, note that commands PTS, PRTS and GS automatically transmit to Combitronic without the ":" operator, and they specifically perform a group-cast. Therefore, the configured group outgoing target setting is the destination group in these case. By default, this is 0, which will be treated by the receiving motors as a broadcast.

GROUP Command

The GROUP command configures several different settings involving group addressing:

GROUP(function,value)

Two arguments are given:

- function: a number associated with the function column below
- value: a value passed to that function

Possible codes, values and meanings are described in the next table.

Function	Value Meanings	Default Value	Description
1	<p>0: Outgoing group-directed messages (":255" or PTS, PRTS, GS) are sent as a broadcast instead of group-cast. All SmartMotors that are reachable by broadcast on this network will receive the message.</p> <p>1-255: the above cases are group-cast to this specified group number. Only SmartMotors configured with this group number as a filter (and reachable in the same network) will receive.</p>	0	Group number outgoing target (which group will outgoing group messages go to).
2	<p>0: Do not listen to any particular group (does not affect the reception of a broadcast message).</p> <p>1-255: listen to this specified group.</p>	0	Group number incoming filter (which group does this motor listen to).
3	<p>0: Allow reception of broadcast.</p> <p>1: Mask (block this type of traffic).</p>	0	Mask (block) the reception of broadcast messages.
4	<p>0: Allow reception of group message.</p> <p>1: Mask (block this type of traffic).</p>	0	Mask (block) the reception of group-cast messages.

RGROUP Command

As mentioned at the beginning of this section, the RGROUP command reports the state of those group addressing settings. The syntax for the RGROUP command is:

```
RGROUP(function)
```

where:

Function	Description
1	Group number outgoing target (the group that outgoing group messages go to)
2	Group number incoming filter (the group that this motor listens to)
3	Mask (block) the reception of broadcast messages
4	Mask (block) the reception of group-cast messages

EXAMPLE: (These GROUP examples are commanded from the terminal window; however, the command can also be used within a program)

NOTE: Comments are for information only and cannot be typed in the terminal window.

Example 1: group-cast

In motor 1:

```
GROUP(1,7)      ' Set this motor (motor 1) to direct group-cast
                  ' messages to group 7.
GROUP(2,7):2    ' Tell motor 2 to listen to group 7.
a:0=6           ' All motors receive this.
b:255=88       ' Group 7 motors receive this.
```

Example 2: broadcast with mask

In motor 1:

```
GROUP(3,1):2    ' Tell motor 2 to ignore broadcast.
a:0=6           ' Motor 2 ignores this message.
GROUP(3,0):2    ' Tell motor 2 to accept broadcast.
a:0=6           ' Motor 2 accepts this message.
```

EXAMPLE: (This GROUP example is commanded from the terminal window)

This example requests the value of motor 4's group number incoming filter.

```
RGROUP(2):4
```

The command reports:

```
3
```

RELATED COMMANDS:

GS *Start Synchronized Motion (GO Synchronized)* (see page 487)

PRTS(...) *Position, Relative Target, Synchronized* (see page 685)

PTS(...) *Position Target, Synchronized* (see page 692)

Start Synchronized Motion (GO Synchronized)

APPLICATION:	Motion control
DESCRIPTION:	Go synchronized; initiates linear-interpolated moves
EXECUTION:	Immediate
CONDITIONAL TO:	Combitronic network of motors is established
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M), not ver. 5.32.x.x; 6.x (D/M), M requires EIP option
COMBITRONIC:	N/A (see details)

DETAILED DESCRIPTION:

NOTE: This command requires a Combitronic-supported motor. Although this command does not support Combitronic syntax, it does use Combitronic communication to pass information between other motors.

The GS (Go Synchronized) command is used to start synchronized motion or update buffered values such as speed or acceleration. Only motors that have been included in the synchronized move group will start when this is commanded. Motors are included in the synchronized move group using the PTS, PTSS, PRTS or PRTSS command issued from a single controller motor.

If a GS command is transmitted and no motion results, refer to the G command for conditions that may prevent motion (for details, see G on page 473). In addition, the PTS command has several aspects that must be considered (for details, see PTS(...) on page 692).

While this command uses Combitronic communications, it does not have the typical Combitronic syntax — that is because this command is intended to be called on the controller within the group of motors. Combitronic communications will automatically be sent to the network so that participating motors will receive this command.

EXAMPLE: (2-axis synchronized absolute move to position x:y for motors 1 and 2)

This sample code may be executed by any motor sharing the same CAN bus network with the motors being commanded to move.

NOTE: Ensure no motor drive faults exist prior to calling this subroutine.

```

. . .
C20
  OFF:0 MP:0 PRT:0=0 G TWAIT      'Initialize to stationary in position mode
  PT:1=PC:1 PT:2=PC:2           'Set target and commanded positions equal
  WAIT=50
  VTS=v                          'Set target path velocity
  ADTS=a                          'Set target path accel/decel
  PTS (x;1,y;2)                  'Use Position Target Synchronized moves
  PTSS (a;3)                      'Supplemental synchronized target
  IF PTSD!=0                      'Prevent 0-length (divide by zero) move
    GS                            'Go Synchronized
    TSWAIT                        'Wait until path move time is complete
  ENDIF
RETURN

```

For additional examples, see A Note About PTS and PRTS on page 181.

RELATED COMMANDS:

ADTS=formula *Acceleration/Deceleration Target, Synchronized (see page 265)*

ATS=formula *Acceleration Target, Synchronized (see page 292)*

DTS=formula *Deceleration Target, Synchronized (see page 399)*

G *Start Motion (GO) (see page 473)*

PRTS(...) *Position, Relative Target, Synchronized (see page 685)*

PRTSS(...) *Position, Relative Target, Synchronized, Supplemental (see page 688)*

PTS(...) *Position Target, Synchronized (see page 692)*

R PTSD *Position Target, Synchronized Distance (see page 695)*

PTSS(...) *Position Target, Synchronized Supplemental (see page 696)*

R PTST *Position Target, Synchronized Time (see page 698)*

VTS=formula *Velocity Target, Synchronized Move (see page 831)*

HEX(index) Decimal Value of a Hex String

APPLICATION:	Data conversion
DESCRIPTION:	Get (read) the integer value of an ASCII-encoded hex string
EXECUTION:	Immediate
CONDITIONAL TO:	ASCII-encoded hexadecimal number stored in the ab[] array
LIMITATIONS:	Hex string maximum length is 8 digits long; lower-case characters "a" though "f" are not supported
READ/REPORT:	RHEX(index)
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	Input: "0" to "FFFFFFF" Output: -2147483648 to 2147483647
TYPICAL VALUES:	Input: "0" to "FFFFFFF" Output: -2147483648 to 2147483647
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The HEX command reads a hexadecimal (hex) string into a variable. The hex string must be ASCII-encoded (i.e., the ASCII digits 0-9 and upper-case characters A-F).

NOTE: For Class 5 SmartMotors, lower-case characters are not supported in firmware versions preceding 5.x.4.8.

This command uses the input ASCII bytes found in the ab[] registers and converts them to a value.

The argument into the HEX command is the starting ab[] register for the string. For example:

```
x=HEX (10)
```

In this case:

- The HEX command will start at ab[10] and proceed up to ab[17].
- The string must be stored with the most significant hex digit at the beginning. In this example, ab[10] is the most significant digit, and ab[17] is the least significant digit.
- The string can be up to eight bytes long and terminated with a NULL value 0. If fewer than eight digits are found, the termination is required so the command knows the size of the hex number. If more than eight digits are found, only the first eight are processed.
- The result is an integer.

EXAMPLE:

```
'Value of: 5862
'Hex of: 16E6
'Decimal representation of hex: 049 054 069 054
ab[10]=49
ab[11]=54
ab[12]=69
ab[13]=54
ab[14]=0           'Null character to end value
x=HEX(10)         'Set x to hex value
Rx               'Report x command
```

Program output is:

5862

RELATED COMMANDS:

- R ATOF(index) *ASCII to Float (see page 291)*
- R DFS(value) *Dump Float, Single (see page 393)*
- R LFS(value) *Load Float Single (see page 547)*



HM_ADT=formula

Homing Accel/Decel Target

APPLICATION:	Motion control
DESCRIPTION:	Sets the homing buffered acceleration target (AT) /deceleration target (DT) at the same time
EXECUTION:	Buffered until homing mode is started
CONDITIONAL TO:	Homing mode operation, PID n (sample rate), encoder resolution
LIMITATIONS:	Must not be negative; effective value is rounded down to next even number
READ/REPORT:	RHM_ADT
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	(encoder counts / (sample ²)) * 65536
RANGE OF VALUES:	0 to 2147483647
TYPICAL VALUES:	2 to 5000
DEFAULT VALUE:	4
FIRMWARE VERSION:	6.x (D/M); no Class 5
COMBITRONIC:	HM_ADT:3=1234, a=HM_ADT:3, RHM_ADT:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

NOTE: This command is affected by the SCALEA command. For details, see SCALEA(m,d) on page 724. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

The HM_ADT command sets the acceleration and deceleration rate of the homing motion profile. For the homing mode, those rates are a single common value and cannot be set individually.

A useful Scale Factor Multiplier code example, which also illustrates the use of af[], SAMP and RES, is shown in RES on page 702 and SAMP on page 722.

For more details on SmartMotor homing operations, see the *SmartMotor Homing Procedures and Methods Application Note*.

EXAMPLE:

```
ADT=20          'Set homing mode target accel/decel
```

RELATED COMMANDS:

- R HM_MTHD=formula *Homing Method (see page 492)*
- R HM_OSET=formula *Homing Offset (see page 496)*
- R HM_VTS=formula *Homing Velocity Target to Switch (see page 498)*
- R HM_VTZ=formula *Homing Velocity Target to Zero (see page 500)*
- R MH *Mode, Homing (see page 607)*



HM_MTHD=formula

Homing Method

APPLICATION:	Motion control
DESCRIPTION:	Get/set the desired homing method
EXECUTION:	Buffered until homing mode is started
CONDITIONAL TO:	Homing mode
LIMITATIONS:	N/A
READ/REPORT:	RHM_MTHD
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	1-14, 17-30, 33-35
TYPICAL VALUES:	1-14, 17-30, 33-35
DEFAULT VALUE:	0
FIRMWARE VERSION:	6.x (D/M); no Class 5
COMBITRONIC:	HM_MTHD:3=14, a=HM_MTHD:3, RHM_MTHD:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

HM_MTHD sets the method for the homing operation. For example, HM_MTHD=1 specifies that the homing operation will use the negative limit switch. The next table shows the available methods.

HM_MTHD=	Reference to motor's index?	Final direction	Switch input (s)	Description
0	N/A	N/A	N/A	Not a valid mode
1	yes	positive	negative limit	Home to negative limit switch, then reference motor index in the positive direction relative to negative limit switch
2	yes	negative	positive limit	Home to positive limit switch, then reference motor index in the negative direction relative to positive limit switch
3	yes	negative	home switch	Home switch transitions to on state in positive direction; seek home switch, then reference motor index in negative direction relative to home switch
4	yes	positive	home switch	Home switch transitions to on state in positive direction; seek home switch, then reference motor index in positive direction relative to home switch

HM_MTHD=	Reference to motor's index?	Final direction	Switch input (s)	Description
5	yes	positive	home switch	Home switch transitions to on state in negative direction; seek home switch, then reference motor index in positive direction relative to home switch
6	yes	negative	home switch	Home switch transitions to on state in negative direction; seek home switch, then reference motor index in negative direction relative to home switch
7	yes	negative	home switch and positive limit	Home switch is on state in the middle of the actuator range with off state at both ends, positive limit switch also required; Seeks the lower end of the home area, then reference motor index in negative direction relative to home switch
8	yes	positive	home switch and positive limit	Home switch is on state in the middle of the actuator range with off state at both ends, positive limit switch also required; Seeks the lower end of the home area, then reference motor index in positive direction relative to home switch
9	yes	negative	home switch and positive limit	Home switch is on state in the middle of the actuator range with off state at both ends, positive limit switch also required; Seeks the higher end of the home area, then reference motor index in negative direction relative to home switch
10	yes	positive	home switch and positive limit	Home switch is on state in the middle of the actuator range with off state at both ends, positive limit switch also required; Seeks the higher end of the home area, then reference motor index in positive direction relative to home switch
11	yes	positive	home switch and negative limit	Home switch is on state in the middle of the actuator range with off state at both ends, negative limit switch also required; Seeks the higher end of the home area, then reference motor index in positive direction relative to home switch
12	yes	negative	home switch and negative limit	Home switch is on state in the middle of the actuator range with off state at both ends, negative limit switch also required; Seeks the higher end of the home area, then reference motor index in negative direction relative to home switch
13	yes	positive	home switch and negative limit	Home switch is on state in the middle of the actuator range with off state at both ends, negative limit switch also required; Seeks the lower end of the home area, then reference motor index in positive direction relative to home switch
14	yes	negative	home switch	Home switch is on state in the middle of the actuator

HM_MTHD=	Reference to motor's index?	Final direction	Switch input (s)	Description
			and negative limit	range with off state at both ends, negative limit switch also required; Seeks the lower end of the home area, then reference motor index in negative direction relative to home switch
15 -16	N/A	N/A	N/A	(reserved)
17	no	positive	negative limit	Home to negative limit switch
18	no	negative	positive limit	Home to positive limit switch
19	no	negative	home switch	Home switch transitions to on state in positive direction; seek home switch
20	no	positive	home switch	Home switch transitions to on state in positive direction; seek home switch
21	no	positive	home switch	Home switch transitions to on state in negative direction; seek home switch
22	no	negative	home switch	Home switch transitions to on state in negative direction; seek home switch
23	no	negative	home switch and positive limit	Home switch is on state in the middle of the actuator range with off state at both ends, positive limit switch also required; Seeks the lower end of the home area
24	no	positive	home switch and positive limit	Home switch is on state in the middle of the actuator range with off state at both ends, positive limit switch also required; Seeks the lower end of the home area
25	no	negative	home switch and positive limit	Home switch is on state in the middle of the actuator range with off state at both ends, positive limit switch also required; Seeks the higher end of the home area
26	no	positive	home switch and positive limit	Home switch is on state in the middle of the actuator range with off state at both ends, positive limit switch also required; Seeks the higher end of the home area
27	no	positive	home switch and negative limit	Home switch is on state in the middle of the actuator range with off state at both ends, negative limit switch also required; Seeks the higher end of the home area
28	no	negative	home switch and negative limit	Home switch is on state in the middle of the actuator range with off state at both ends, negative limit switch also required; Seeks the higher end of the home area

HM_MTHD=	Reference to motor's index?	Final direction	Switch input (s)	Description
29	no	positive	home switch and negative limit	Home switch is on state in the middle of the actuator range with off state at both ends, negative limit switch also required; Seeks the lower end of the home area
30	no	negative	home switch and negative limit	Home switch is on state in the middle of the actuator range with off state at both ends, negative limit switch also required; Seeks the lower end of the home area
31 - 32	N/A	N/A	N/A	(reserved)
33	yes	negative	(none)	Reference motor index in negative direction relative to starting position
34	yes	positive	(none)	Reference motor index in positive direction relative to starting position
35	no	N/A	(none)	No motion, reference the current position
36	N/A	N/A	N/A	(reserved) Not supported
37	N/A	N/A	N/A	(reserved) Not supported
38+	N/A	N/A	N/A	(reserved)

For more details on SmartMotor homing operations, see the *SmartMotor Homing Procedures and Methods Application Note*.

EXAMPLE:

```
HM_MTHD=1      'Home using the negative limit switch
```

RELATED COMMANDS:

R HM_ADT=formula *Homing Accel/Decel Target (see page 491)*

R HM_VTS=formula *Homing Velocity Target to Switch (see page 498)*

R HM_VTZ=formula *Homing Velocity Target to Zero (see page 500)*

R MH Mode, Homing (see page 607)



HM_OSET=formula

Homing Offset

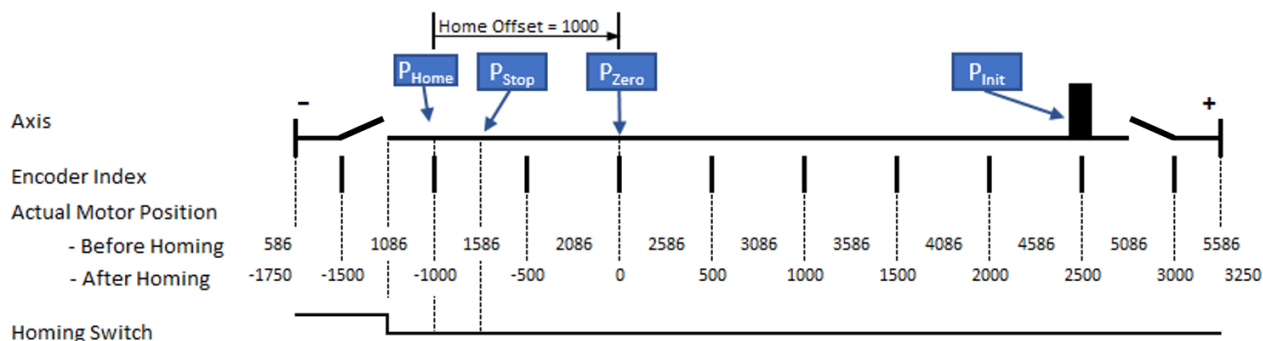
APPLICATION:	Motion control
DESCRIPTION:	Get/set the homing offset
EXECUTION:	Applied when homing process completes
CONDITIONAL TO:	homing mode
LIMITATIONS:	N/A
READ/REPORT:	RHM_OSET
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	Encoder counts
RANGE OF VALUES:	-2147483648 to 2147483647
TYPICAL VALUES:	-1000000000 to 1000000000
DEFAULT VALUE:	0
FIRMWARE VERSION:	6.x (D/M); no Class 5
COMBITRONIC:	HM_OSET:3=1234, a=HM_OSET:3, RHM_OSET:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

NOTE: This command is affected by the SCALEP command. For details, see SCALEP(m,d) on page 726. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

HM_OSET offsets the location of the zero position from the reference home location when the homing operation completes.

NOTE: This does not control where the motor will come to rest when the homing completes. The location where the motor comes to rest is a result of deceleration and speed. The user application is responsible for commanding a deliberate move to its desired location whether that is position: home, position: zero, or some other place.



For more details on SmartMotor homing operations, see the *SmartMotor Homing Procedures and Methods Application Note*.

EXAMPLE:

```
HM_OSET=3000      'Set the homing offset
```

RELATED COMMANDS:

R HM_ADT=formula *Homing Accel/Decel Target (see page 491)*

R HM_MTHD=formula *Homing Method (see page 492)*

R HM_VTS=formula *Homing Velocity Target to Switch (see page 498)*

R HM_VTZ=formula *Homing Velocity Target to Zero (see page 500)*

R MH *Mode, Homing (see page 607)*



APPLICATION:	Motion control
DESCRIPTION:	Gets/sets the homing velocity to the switch position
EXECUTION:	Buffered until homing mode is started
CONDITIONAL TO:	Homing mode operation, PID n (sample rate), encoder resolution
LIMITATIONS:	N/A
READ/REPORT:	RHM_VTS
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	(encoder counts / sample) * 65536
RANGE OF VALUES:	0 to 2147483647
TYPICAL VALUES:	1000 to 3200000
DEFAULT VALUE:	0
FIRMWARE VERSION:	6.x (D/M); no Class 5
COMBITRONIC:	HM_VTS=1234:3, a=HM_VTS:3, RHM_VTS:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

NOTE: This command is affected by the SCALEV command. For details, see SCALEV(m,d) on page 728. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

The HM_VTS command is used to get (read) or set the velocity target to the homing switch position:

- =HM_VTS
Read the current target velocity
- HM_VTS=frm
Set the target velocity

The HM_VTS command specifies a target velocity while in homing mode for the first phase of seeking the limit and/or home switch as prescribed by the selected homing mode. Homing mode will select the direction, so HM_VTS is given in positive values only. The value must be in the range 0 to 2147483647. The value set by the HM_VTS command only governs the calculated trajectory of MH mode. The PID compensator may need to "catch up" if the actual position has fallen behind the trajectory position. In this case, the actual speed will exceed the target speed. The value defaults to zero, so it must be set before any motion can occur. The new value takes effect during the homing process, so it must be set prior to activating homing mode.

For more details on SmartMotor homing operations, see the *SmartMotor Homing Procedures and Methods Application Note*.

Equations for Real-World Units:

Encoder resolution and sample rate can vary. Therefore, the general equations in the next table can be used to convert the real-world units of velocity to a value for HM_VTS, where af[0] is already set with the real-world unit value. These equations force floating-point calculations to avoid overflow and maintain resolution. They can be placed in a user program, or they can be precalculated if the values of SAMP and RES are known (SAMP and RES can be reported from the terminal using the RSAMP and RRES commands, respectively). SAMP can change if the PID command is used. The value of RES can differ between motor models.

Input as Value in af[0]	Equation
Radians/Sec	$HM_VTS = ((af[0] * RES) / (PI * 2.0 * SAMP)) * 65536$
Encoder Counts/Sec	$HM_VTS = (af[0] / (SAMP * 1.0)) * 65536$
Rev/Sec	$HM_VTS = ((af[0] * RES) / (SAMP * 1.0)) * 65536$
RPM	$HM_VTS = ((af[0] * RES) / (60.0 * SAMP)) * 65536$

A useful Scale Factor Multiplier code example, which also illustrates the use of af[], SAMP and RES, is shown in RES on page 702 and SAMP on page 722.

EXAMPLE:

```
HM_VTS=40000      'Set the homing target velocity to the switch position
```

RELATED COMMANDS:

- R HM_ADT=formula *Homing Accel/Decel Target (see page 491)*
- R HM_MTHD=formula *Homing Method (see page 492)*
- R HM_OSET=formula *Homing Offset (see page 496)*
- R HM_VTZ=formula *Homing Velocity Target to Zero (see page 500)*
- R MH Mode, Homing (see page 607)



APPLICATION:	Motion control
DESCRIPTION:	Gets/sets the homing velocity to the zero position
EXECUTION:	Buffered until homing mode is started
CONDITIONAL TO:	Homing mode operation, PID n (sample rate), encoder resolution
LIMITATIONS:	N/A
READ/REPORT:	RHM_VTZ
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	(encoder counts / sample) * 65536
RANGE OF VALUES:	0 to 2147483647
TYPICAL VALUES:	1000 to 3200000
DEFAULT VALUE:	0
FIRMWARE VERSION:	6.x (D/M); no Class 5
COMBITRONIC:	HM_VTZ=1234:3, a=HM_VTZ:3, RHM_VTZ:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

NOTE: This command is affected by the SCALEV command. For details, see SCALEV(m,d) on page 728. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

The HM_VTZ command is used to get (read) or set the velocity target to the homing zero position:

- =HM_VTZ
Read the current target velocity
- HM_VTZ=frm
Set the target velocity

The HM_VTZ command specifies a target velocity while in homing mode for the final phase of seeking the home position as prescribed by the selected homing mode. Homing mode will select the direction, so HM_VTZ is given in positive values only. The value must be in the range 0 to 2147483647. The value set by the HM_VTZ command only governs the calculated trajectory of MH mode. The PID compensator may need to "catch up" if the actual position has fallen behind the trajectory position. In this case, the actual speed will exceed the target speed. The value defaults to zero, so it must be set before any motion can occur. The new value takes effect during the homing process, so it must be set it prior to activating homing mode.

For more details on SmartMotor homing operations, see the *SmartMotor Homing Procedures and Methods Application Note*.

Equations for Real-World Units:

Encoder resolution and sample rate can vary. Therefore, the general equations in the next table can be used to convert the real-world units of velocity to a value for HM_VTZ, where af[0] is already set with the real-world unit value. These equations force floating-point calculations to avoid overflow and maintain resolution. They can be placed in a user program, or they can be precalculated if the values of SAMP and RES are known (SAMP and RES can be reported from the terminal using the RSAMP and RRES commands, respectively). SAMP can change if the PID command is used. The value of RES can differ between motor models.

Input as Value in af[0]	Equation
Radians/Sec	$HM_VTZ = ((af[0] * RES) / (PI * 2.0 * SAMP)) * 65536$
Encoder Counts/Sec	$HM_VTZ = (af[0] / (SAMP * 1.0)) * 65536$
Rev/Sec	$HM_VTZ = ((af[0] * RES) / (SAMP * 1.0)) * 65536$
RPM	$HM_VTZ = ((af[0] * RES) / (60.0 * SAMP)) * 65536$

A useful Scale Factor Multiplier code example, which also illustrates the use of af[], SAMP and RES, is shown in RES on page 702 and SAMP on page 722.

EXAMPLE:

```
HM_VTZ=20000      'Set the homing target velocity to zero position
```

RELATED COMMANDS:

- R HM_ADT=formula *Homing Accel/Decel Target (see page 491)*
- R HM_MTHD=formula *Homing Method (see page 492)*
- R HM_OSET=formula *Homing Offset (see page 496)*
- R HM_VTS=formula *Homing Velocity Target to Switch (see page 498)*
- R MH Mode, Homing (see page 607)



APPLICATION:	I/O control
DESCRIPTION:	Encoder value latched by rising-edge, hardware-index capture
EXECUTION:	Immediate
CONDITIONAL TO:	Index previously captured
LIMITATIONS:	N/A
READ/REPORT:	RI(enc); supports the DS2020 Combitronic system over RS-232 only
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Encoder counts
RANGE OF VALUES:	Input: 0 or 1 Output: -2147483648 to 2147483647
TYPICAL VALUES:	Input: 0 or 1 Output: -2147483648 to 2147483647
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	RI(0):3, x=I(0):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

NOTE: This command is affected by the SCALEP command for ENC1 only. For details, see SCALEP (m,d) on page 726. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

I (capital i) is the function that stores the last hardware-latched, rising-edge encoder index position. It can be read from a host with the RI(enc) command, or it can be read by the program with a line such as a=I(enc). The value of enc determines which encoder is being referred to:

- I(0) reads the internal encoder captured count
- I(1) reads the external encoder captured count

For the DS2020 Combitronic system, the report version of this command is used for a procedure to find the position that corresponds to the physical zero position of the feedback sensor.

The index capture must first be armed with the Ai, Aij or Aji command before a capture occurs. The Bi (enc) command can be used to detect the capture event (due to encoder index or input signal). These capture events can also be detected by status bits in status word 1.

The index is a physical reference mark on the encoder. It is also referred to as a Z pulse, marker pulse, and sometimes a combination of those names. It is typically used in homing sequences requiring a high degree of repeatability.

Class 5.x and later firmware has the ability to redirect port 6 to the Index register input trigger, which allows high-speed position capture through port 6. The internal or external encoder can use this source through the EIRI and EIRE commands, respectively. When using this method, the previously-stated rules for arming and clearing the index still apply.

For the DS2020 Combitronic system, RI(0) returns the position value that corresponds to the last found physical zero position of the feedback sensor.

EXAMPLE: (homing against a hard stop with Index reference)

NOTE: This method of referencing against a hard stop can eliminate an additional switch and cable.

```

AMPS=100      'Current limit 10%
O=0           'Declare this home
MP            'Set Mode Position
ADT=100      'Set accel/decel
VT=100000    'Set Velocity
PT=-1000000  'Move negative
Ai(0)
G             'Start Motion
WHILE Bt     'Wait for motion fault
  IF Bi(0)   'If rising-edge index pulse seen
    a=I(0)   'Record rising-edge index position
  ENDIF
LOOP         'Loop back to wait
O=-a        'Last Index is Home
PT=0        'Move to New Home
G           'Start Motion
AMPS=1023   'Restore power

```

RELATED COMMANDS:

Ai(enc) *Arm Index Rising Edge (see page 270)*

Aij(enc) *Arm Index Rising Edge Then Falling Edge (see page 272)*

Aj(enc) *Arm Index Falling Edge (see page 274)*

Aji(enc) *Arm Index Falling Edge Then Rising Edge (see page 276)*

R Bi(enc) *Bit, Index Capture, Rising (see page 309)*

R Bx(enc) *Bit, Index Input, Real-Time (see page 351)*

EIRE *Enable Index Register, Encoder Capture (see page 419)*

EIRI *Enable Index Register, Input Capture (see page 421)*



IDENT=formula

Set Identification Value

APPLICATION:	EEPROM (Nonvolatile Memory)
DESCRIPTION:	Get/set the SmartMotor identification value
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RIDENT
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	-2147483648 to 2147483647
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	6.x (D/M); no Class 5
COMBITRONIC:	IDENT:3=1234, a=IDENT:3, RIDENT:3 where ":3" is the motor address — use the actual address or a variable
	NOTE: Requires Class 6 EIP motor

DETAILED DESCRIPTION:

The IDENT command gets (reads) and sets the identification value for a SmartMotor. It doesn't have any effect on the motor — it's just a non-volatile ID that is preserved between power cycles. It allows a SmartMotor's user program to self-detect its designated purpose according to the programmer's setting of IDENT on that motor.

- `x=IDENT`
Get the IDENT value and assign it to the variable x.
- `IDENT=formula`
Set IDENT equal to the value of the formula.

The value of IDENT can be reported with the RIDENT command.

EXAMPLE:

In the next example, the programmer has three motors on the machine. He wants to load the same user program in all motors but have each motor do certain operations based on its IDENT value. That part of the program could look like this:

```
IF IDENT=1
' Do the motor 1 operation - motors 2 and 3 ignore this.
ENDIF
```


RELATED COMMANDS:

R IDENT=formula *Set Identification Value (see page 504)*

IF formula *Conditional Program Code Execution (see page 506)*

SWITCH formula *Switch, Program Flow Control (see page 766)*

IF formula

Conditional Program Code Execution

APPLICATION:	Program execution and flow control
DESCRIPTION:	IF <i>formula</i> ...ENDIF control block
EXECUTION:	Immediate
CONDITIONAL TO:	Value of <i>formula</i> after the IF statement
LIMITATIONS:	Requires corresponding ENDIF; can be executed only from within user program
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)

DETAILED DESCRIPTION:

CAUTION: Extensive use of IF statements and GOTO branches can quickly make your programs impossible to read or debug. Learn to organize your code with one main loop using a GOTO and write the rest of the program with subroutines (GOSUB). For details, see GOSUB(label) on page 480.

The IF statement provides a method for an executing program to choose between alternate execution paths at runtime. In its simplest form, the IF control block consists of:

```
IF (formula) 'Evaluates as nonzero
    'Run the code below the "IF" command
ENDIF
```

NOTE: Every IF structure must be terminated with an ENDIF.

Formula is a test condition — both mathematical comparisons and Boolean-logic bitwise comparisons can be used.

- If the result of the *formula* is any value besides 0, then it is considered true and the code immediately after the IF *formula* statement is executed.
- If the result of *formula* is 0, then it is considered false and execution skips the code after IF *formula*. When false, execution skips to the next available ELSE, ELSEIF or ENDIF command.

The next table shows various forms of IF formulas and their descriptions:

IF formula	Description
IF a==b	if a equals b
IF a!=b	if a does not equal b
IF a<b	if a is less than b
IF a<=b	if a is less than or equal to b
IF a>b	if a is greater than b
IF a>=b	if a is greater than or equal to b
IF a&b	if a AND b (bitwise comparison)
IF a b	if a OR b (bitwise comparison)
IF a	if a does not equal zero (common shortcut to IF a==1)

The formula after the IF statement may:

- Include Combitronic values retrieved from other motors
- Consist of multiple variables and math operators

Note that there isn't a logical OR, AND or XOR. The bitwise operators may be used with proper attention paid to the result of those operations. For example:

```
IF (a==b) | (c==d)
```

will be true if a equals b, OR if c equals d. The reason this works is that comparison operators such as "==" return 0 if false and 1 if true. In a bitwise sense, this is setting bit 0 when true. The bitwise OR operator "|" compares all bits. However, only the lowest bit becomes significant.

EXAMPLE: (If true, run some code.)

```
IF PA>12345    'If Position is above 12345
    PRINT("position is greater than 12345",#13)
ENDIF         'This is the next line of code to be executed
              'whether it is true or not.
```

EXAMPLE: (If true, run some code; ELSE if false, run some other code.)

```
IF PA>12345    'If Position is above 12345
    PRINT("position is greater than 12345",#13)
ELSE          'If it is no true
    PRINT("position is not greater than 12345",#13)
ENDIF         'This is the next line of code to be executed
```

EXAMPLE: (If true, run some code; else if something else is true, run that code.)

```
IF PA>12345    'If Position is above 12345
    PRINT("position is greater than 12345",#13)
ELSEIF PA==0  'If Position equals zero
    PRINT("position is at zero",#13)
ENDIF         'This is the next line of code to be executed
              'even if position is not at zero and
              'not greater than 12345.
```

EXAMPLE: (Test for two conditions and default to another line of code.)

```
IF PA>100      'If Position is above 100
    PRINT("position is greater than 100",#13)
ELSEIF PA<=0  'If it less than or equal to zero
    PRINT("position is <= to zero",#13)
ELSE
    PRINT("position is between zero and 100",#13)
ENDIF
```

EXAMPLE: (Binary bit mask comparison.)

```
a=10 'binary 1010
b=5  'binary 0101
c=7  'binary 0111
d=1  'binary 0001
e=0  'binary 0000
IF a&2 'Compare "a" and 2 as binary numbers bit for it.
    PRINT("This is true because 2 is 0010",#13)
ENDIF
IF a&d 'Are any bits in common with a AND d?
    PRINT("This will never PRINT",#13)
ENDIF
IF a|b 'Are there any bits that are 1 in either number?
    PRINT("This will print",#13)
ENDIF
IF d|e 'Even though e is zero, d is nonzero:
    PRINT("This will print",#13)
ENDIF
IF b&c
    PRINT("This is true",#13)
ENDIF
END
```

RELATED COMMANDS:

ELSE *IF-Structure Command Flow Element (see page 428)*
 ELSEIF formula *IF-Structure Command Flow Element (see page 430)*
 ENDIF *End IF Statement (see page 441)*



APPLICATION:	I/O control
DESCRIPTION:	Read the specified input or all inputs
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RIN(...) ; supports the DS2020 Combitronic system
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	Depends on motor series and command options (see details)
TYPICAL VALUES:	Depends on motor series and command options (see details)
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	RIN(0):3, x=IN(0):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The IN command reads one specific input or all inputs. It can be used in these ways:

- $x=IN(IO)$
where $IN(IO)$ specifies the I/O number that is assigned to the variable x . See the next table for allowed range of IO. The result is a value of 0 or 1 assigned to x .
- $x=IN(W,word)$
where $IN(W,word)$ specifies which word of I/O will be assigned to the variable x . A literal "W" is used as the first argument. See the next table for the allowed values for "word" and the output word value range.
- $x=IN(W,word[,mask])$
where $IN(W,word[,mask])$ specifies which word of I/O will be assigned to the variable x . A literal "W" is used as the first argument. The mask argument is ANDed with the resulting response word (equivalent to using the & operator on the result). See the next table for the allowed values for "word", the output word value range, and the bitmask range.

Motor Type	word Allowed Values	IO Allowed Range	Logic 0 Voltage	Logic 1 Voltage	Output Word Value Range	Bitmask Range
Class 5 D-style	0	0-6	0	5	0 to 255	0 to 255
		7 (virtual only, not connected)	N/A	N/A		
Class 5 D-style with AD1 option	0	0-6	0	5	0 to 255	0 to 255
		7 (virtual only, not connected)	N/A	N/A		
	1	16-25	0	24	0 to 1023	0 to 1023
Class 5 M-style	0	0-10	0	24	0 to 2047	0 to 2047
Class 6 M-style	0	0-9	0	24	0 to 2047	0 to 2047
Class 6 D-style	0	0-9	0	24	0 to 2047	0 to 2047
DS2020 Combitronic system (report only)	0	0-5				

NOTE: D-style motor's bit #7 does not connect to any physical I/O but does remember the state it was set to.

Logical I/O User Read Commands Example for Class 5 M-style Motor

The next example describes the RIN() commands used for reading logical I/O status on the Class 5 M-style motor.

Pin	Conn	Desc	User Read Command
1	12 pin	I/O-0	RIN(0)
...			
4	5 pin	I/O-2	RIN(2)
...			
9	12 pin	Not Fault Output	RIN(11)
10	12 pin	Drive Enable Input	RIN(12)

Further, other commands are available for this purpose:

- Bits returned by Status Word 16, RW(16); for example:

```
RW(16)      2048
```

where "RW(16)" is the command typed in the Terminal Window; it returns "2048" indicating bit 7 is on. For more details on Status Word 16, see Status Word 16: On Board Local I/O Status: M-Style Class 5 Motor on page 930. For more details on the RW(16) command, see RW(word) on page 833.

- Bits returned by RIN(W,0); for example:

```
RIN(W, 0)      2048
```

where "RIN(W,0)" is the command typed in the Terminal Window; it returns "2048" indicating bit 7 is on. For more details on Status Word 16, see Status Word 16: On Board Local I/O Status: M-Style Class 5 Motor on page 930. For details on the RIN(W,0) command, see the Detailed Description section of this topic.

EXAMPLE:

This line of code could be written in motor number 1 — it sets variable "a" in motor 2 equal to an I/O of motor 3:

```
a:2=IN(0):3      'Set variable in motor 2 to I/O of motor 3
```

EXAMPLE: (Subroutine checks inputs and calls corresponding subroutines based on state; shows C#, ENDIF, GOSUB, GOTO, IF and IN)

```
C10              'Place label
  IF IN(0)==0    'Check Input 0
    GOSUB20     'If Input 0 low, call Subroutine 20
  ENDIF         'End check Input 0
  IF IN(1)==0    'Check Input 1
    a=30        'as example for below
    GOSUB(a)    'If Input 1 low, call Subroutine 30
  ENDIF         'End check Input 1
GOTO(10)        'Will loop back to C10

C20              'Subroutine 20 code here
RETURN

C30              'Subroutine 30 code here
RETURN
```

RELATED COMMANDS:

EIGN(...) *Enable as Input for General-Use (see page 412)*
 EILN *Enable Input as Limit Negative (see page 415)*
 EILP *Enable Input as Limit Positive (see page 417)*
 EIRE *Enable Index Register, Encoder Capture (see page 419)*
 EIRI *Enable Index Register, Input Capture (see page 421)*
 EISM(x) *E-Configure Input as Sync Controller (see page 423)*
 EOBK(IO) *Enable Output, Brake Control (see page 445)*
 R INA(...) *Specified Input, Analog (see page 512)*
 OR(value) *Output, Reset (see page 638)*
 OS(...) *Output, Set (see page 640)*
 OUT(...)=formula *Output, Activate/Deactivate (see page 644)*
 UO(...)=formula *User Status Bits (see page 795)*
 UR(...) *User Bits, Reset (see page 801)*
 US(...) *User Bits, Set (see page 803)*



INA(...)

Specified Input, Analog

APPLICATION:	I/O control
DESCRIPTION:	Read the desired analog input
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RINA(...)
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	Depends on motor series and command options (see details)
TYPICAL VALUES:	Depends on motor series and command options (see details)
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	RINA(V1,3):3, x=INA(V1,3):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The INA command reads the specified analog input. It can be used in these ways:

NOTE: See the table after these descriptions for their application to each motor type.

- $x=INA(A,IO)$
where $IN(A,IO)$ specifies a raw analog reading with 10-bit resolution and spanned over a signed 16-bit range, which is assigned to variable x .
- $x=INA(V,IO)$
where $IN(V,IO)$ specifies a reading of the input voltage (V) in millivolts of analog input value and for a given I/O (defined by IO), which is assigned to the variable x .
- $x=INA(V1,IO)$
where $INA(V1,IO)$ specifies a scaled 0-5 VDC reading in millivolts (3456 would be 3.456 VDC) for a given I/O (defined by IO) that is assigned to the variable x .
- $x=INA(V2,IO)$
where $INA(V2,IO)$ specifies a scaled 0-.6 VDC reading in millivolts (60 would be 0.06 VDC) for a given I/O (defined by IO) that is assigned to the variable x .
- $x=INA(S,x)$
where $INA(S,x)$ specifies the sourcing voltage for the I/O port (when output pin); x is 16-25 for the Class 5 D-style motor and 0 for the M-style motor.

- $x=INA(T,x)$
 where $INA(T,x)$ specifies the I/O chip temperature; x is 16-25 for the Class 5 D-style motor and 0 for the M-style motor.

Motor type	IO	Command	Nominal input voltage	Nominal value reported
Class 5 D-style	0-6	INA(A,IO)	0-5V	0-32736
		INA(V1,IO)		0-5000
Class 5 D-style with AD1 option	0-6	INA(A,IO)	0-5V	0-32736
		INA(V1,IO)		0-5000
	16-25	INA(A,IO)	0-24V	0-19000
		INA(V,IO)		0-24000
		INA(V1,IO)		0-5100 ^a
		INA(V2,IO)		0-610 ^a
		INA(S,x)		2400
INA(T,x)	30 (°C)			
Class 5 M-style	0-10	INA(A,IO)	0-24V	0-19000
		INA(V,IO)		0-24000
		INA(V1,IO)		0-5100 ^a
		INA(V2,IO)		0-610 ^a
		INA(S,x)		2400
		INA(T,x)		30 (°C)
Class 6 M-style	0-1	INA(A,IO)	0-18V ^b	0-32736
		INA(V,IO)		0-18000
Class 6 D-style	0-1	INA(A,IO)	0-10V ^c	0-32736
		INA(V,IO)		0-10000
	11	INA(A,11)	4-20mA	4000-20000 nom; 0-21483 min/max
<p>a. 99999 indicates out of range. b. Nominal input voltage can go up to 24V, but analog measurement is saturated at 18V (18000 millivolts) c. Nominal input voltage can go up to 24V, but analog measurement is saturated at 10.67V (10670 millivolts)</p>				

EXAMPLE: (Routine maintains velocity during analog drift)

```

EIGN (W, 0)           'Disable hardware limits
KP=3020              'Increase stiffness from default
KD=10010             'Increase damping from default
F                   'Activate new tuning parameters
ADT=100              'Set maximum accel/decel
MV                  'Set to Velocity mode
d=10                 'Analog dead band, 5000 = full scale
o=2500               'Offset to allow negative swings
m=40                 'Multiplier for speed
w=10                 'Time delay between reads
b=0                  'Seed b
C10                  'Label to create infinite loop
  a=INA (V1, 3) -o   'Take analog 5 Volt full-scale reading
  x=a-b              'Set x to determine change in input
  IF x>d             'Check if change beyond dead band
    VT=b*m           'Multiplier for appropriate speed
    G                 'Initiate new velocity
  ELSEIF x<-d        'Check if change beyond dead band
    VT=b*m           'Multiplier for appropriate speed
    G                 'Initiate new velocity
  ENDIF              'End IF statement
  b=a                'Update b for prevention of hunting
  WAIT=w             'Pause before next read
GOTO10               'Loop back to label
END                  'Required END (never reached)

```

RELATED COMMANDS:

R IN(...) *Specified Input (see page 509)*

R OC(...) *Output Condition (see page 630)*

R OF(...) *Output Fault (see page 634)*

IPCTL(function,"string")

Set IP Address, Subnet Mask or Gateway

APPLICATION:	Communications control
DESCRIPTION:	Sets IP address, subnet mask or Gateway
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	ASCII string: decimal values 0-255 separated by "."
RANGE OF VALUES:	"0.0.0.0" to "255.255.255.255"
TYPICAL VALUES:	"192.168.0.10" (IP address) "255.255.255.0" (Subnet mask) "192.168.0.1" (Gateway)
DEFAULT VALUE:	"0.0.0.0" for IP address, subnet mask, gateway (disabled / automatic)
FIRMWARE VERSION:	6.x (D/M); no Class 5
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

NOTE: In PROFINET networks, IP addresses are usually automatically configured and this command is not needed. Therefore, leave the address at the default (0.0.0.0) setting, unless you need to set a specific static IP address.

The IPCTL command sets the IP address, subnet mask, or gateway for the industrial Ethernet network. It uses the form IPCTL(function,"string"), where, for example:

- function is one of these codes:

function	Description
0	Set IP address
1	Set subnet mask
2	Set gateway

- "string" is formatted as an IP address and entered as a string

Specific features are based on the fieldbus network being used. See the corresponding SmartMotor fieldbus guide for more details.

EXAMPLE: (Set a static IP address)

```
IPCTL(0,"192.168.0.10") 'Set the IP address to 192.168.0.10
```

RELATED COMMANDS:

SNAME("string") *Set PROFINET Station Name (see page 754)*

ETHCTL(function,value) *Control Industrial Ethernet Network Features (see page 456)*

ITR(Int#,StatusWord,Bit#,BitState,Label#)

Interrupt Setup

APPLICATION:	Program execution and flow control
DESCRIPTION:	Configure a single user program interrupt
EXECUTION:	Immediate; remains in effect until otherwise commanded
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	See details
TYPICAL VALUES:	N/A
DEFAULT VALUE:	Motor defaults with no ITR interrupts configured
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The ITR command is used to configure an interrupt. It uses the form:

```
ITR(Int#,StatusWord,Bit#,BitState,Label#)
```

where:

- Int# — the interrupt number: 0-7 (lower number is a higher priority)
- StatusWord — the status word number containing the bit to monitor: 0-17
- Bit# — the bit number in a status word to monitor: 0-15
- BitState — the transition to this state will cause the interrupt: 0 or 1
- Label# — the label number to jump to for the interrupt routine: 0-999

After this command is called, the user-program process will be monitoring for the specified condition. The interrupts are always triggered by an edge (transition) of a bit in the status words. Any status word/bit can be chosen.

For the interrupt to function, a program must be running, ITRE must enable the global interrupt scanner, and the individual interrupt must be enabled with the EITR() command. Often, a program is completely interrupt driven and has no need for a main loop. In this case, the PAUSE command can be placed in a program after the point where the interrupts are configured. This will halt the main loop of the program but will leave the interrupts active. To understand where the PAUSE command will continue on with the main program, see RESUME on page 704.

The routine called as label# must have a RETURNI at the end instead of a RETURN. Therefore, interrupt routines should not be called as subroutines, and subroutines should not be called as interrupt routines.

If there is a need for an interrupt or a subroutine to use a common section of code, then the interrupt routine should use a GOSUB to call the common section as a subroutine. This method ensures that the return path will still go through the RETURNI at the end of the interrupt routine.

Lower interrupt ID numbers have a higher priority. This means that they can be called even while a lower-priority interrupt routine is in progress. In some cases, this may cause a conflict, which is typically referred to as a race condition. For example, if both routines need to read and modify the same variable, one or more interrupts can be disabled while this critical operation is taking place. For more details, see the command pair DITR(int) on page 394 and EITR(int) on page 424, or the command pair ITRD on page 520 and ITRE on page 522.

NOTE: Each instance of the ITR command must have a unique interrupt level. It is not possible to configure two different events with the same priority level.

An interrupt will block itself, but an interrupt does not automatically disable itself. In other words, the interrupt-causing event will call the interrupt routine. If the same event occurs again while inside the interrupt routine, then that same interrupt routine will be called again. This occurs immediately after the first instance of the interrupt routine completes.

EXAMPLE: (Fault handler routine)

```
EIGN(W,0,12)      'Another way to disable Travel Limits
ZS               'Clear faults
ITR(0,0,0,0,0)   'Set Int 0 for: stat word 0, bit 0,
                 'shift to 0, to call C0
EITR(0)         'Enable Interrupt 0
ITRE            'Global Interrupt Enable
PAUSE           'Pause to prevent "END" from disabling
                 'Interrupt, no change to stack

END

C0              'Fault handler
MTB:0           'Motor will turn off with Dynamic
                 'braking, tell other motors to stop.
US(0):0        'Set User Status Bit 0 to 1 (Status
                 'Word 12 bit zero)
US(ADDR):0     'Set User Status Bit "address" to 1
                 '(Status Word 12 Bit "address")

RETURNI
```

EXAMPLE: (Routine pulses output on a given position)

```

EIGN(W,0)           'Disable limits
ZS                  'Clear faults
ITR(0,4,0,0,1)     'ITR(int#,sw,bit,state,lbl)
ITRE                'Enable all interrupts
EITR(0)             'Enable interrupt 0
OUT(1)=1            'Set I(0)/O B to output, high
ADT=100             'Set maximum accel/decel
VT=100000           'Set maximum velocity
MP                  'Set Position mode
'****Main Program Body****
WHILE 1>0
  O=0                'Reset origin for move
  PT=40000           'Set final position
  G                  'Start motion
  WHILE PA<20000    'Loop while motion continues
  LOOP              'Wait for desired position to pass
  OUT(1)=0           'Set output low
  TMR(0,400)         'Use timer 0 for pulse width
  TWAIT
  WAIT=1000          'Wait 1 second
LOOP
END
'****Interrupt Subroutine****
C1
  OUT(1)=1           'Set output high again
RETURNI

```

RELATED COMMANDS:

DITR(int) *Disable Interrupts (see page 394)*
 EISM(x) *E-Configure Input as Sync Controller (see page 423)*
 EITR(int) *Enable Interrupts (see page 424)*
 END *End Program Code Execution (see page 439)*
 ITR(Int#,StatusWord,Bit#,BitState,Label#) *Interrupt Setup (see page 517)*
 ITRD *Interrupt Disable, Global (see page 520)*
 ITRE *Enable Interrupts, Global (see page 522)*
 PAUSE *Pause Program Execution (see page 648)*
 RESUME *Resume Program Execution (see page 704)*
 RETURNI *Return Interrupt (see page 708)*
 RUN *Run Program (see page 714)*



APPLICATION:	Program execution and flow control
DESCRIPTION:	Disable the global interrupt scanner
EXECUTION:	Immediate; remains in effect until otherwise commanded
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	Motor defaults to ITRD (interrupts disabled)
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	ITRD:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The ITRD command is used to disable the global interrupt scanner. ITRD disables the interrupt handler and clears Interrupt Status Bit 15.

For an interrupt to work, it must be enabled at two levels: first, enable *individual* interrupts with the EITR() command using the interrupt number from 0 to 7 in the parentheses; second, enable *all* interrupts with the ITRE command. Similarly, *individual* interrupts can be disabled with the DITR() command, and *all* interrupts can be disabled with the ITRD command. For more details, see the corresponding command-description pages.

NOTE: The user program must also be running for interrupts to take effect, the END and RUN commands will reset the state of the interrupts to defaults.

For more details, see Interrupt Programming on page 195.

EXAMPLE:

```

ITR(0,4,0,0,10) 'Set interrupt 0 for: status word 16, bit 3,
                'Shift to 0, to call C10
TMR(0,2000)     'Timer bit set for 2 seconds
EITR(0)
ITRE           'Global enable interrupts
MP            'Set position mode
VT=10000      'Set velocity target
ADT=50        'Set accel/decel target
PT=10000      'Set position target
G            'Start motion
TWAIT        'Wait for move to complete
'Use ITRD to disable all interrupts after move
ITRD         'Global disable interrupts
END          'Ending the program will also disable interrupts

C10          'Interrupt subroutine
    PRINT("Move exceeded two seconds.",#13)
RETURNI

```

Program output is:

```
Move exceeded two seconds.
```

RELATED COMMANDS:

DITR(int) *Disable Interrupts (see page 394)*
EISM(x) *E-Configure Input as Sync Controller (see page 423)*
EITR(int) *Enable Interrupts (see page 424)*
END *End Program Code Execution (see page 439)*
ITR(Int#,StatusWord,Bit#,BitState,Label#) *Interrupt Setup (see page 517)*
ITRD *Interrupt Disable, Global (see page 520)*
ITRE *Enable Interrupts, Global (see page 522)*
PAUSE *Pause Program Execution (see page 648)*
RESUME *Resume Program Execution (see page 704)*
RETURNI *Return Interrupt (see page 708)*
RUN *Run Program (see page 714)*



APPLICATION:	Program execution and flow control
DESCRIPTION:	Enable the global interrupt scanner
EXECUTION:	Immediate; remains in effect until otherwise commanded
CONDITIONAL TO:	Configuration of interrupt (ITR), program running, individual interrupt enabled (EITR)
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	Motor defaults to ITRD (interrupts disabled)
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	ITRE:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The ITRE command is used to enable the global interrupt scanner.

For an interrupt to work, it must be enabled at two levels: first, enable *individual* interrupts with the EITR() command using the interrupt number from 0 to 7 in the parentheses; second, enable *all* interrupts with the ITRE command. Similarly, *individual* interrupts can be disabled with the DITR() command, and *all* interrupts can be disabled with the ITRD command. For more details, see the corresponding command-description pages.

NOTE: The user program must also be running for interrupts to take effect, the END and RUN commands will reset the state of the interrupts to defaults.

For more details, see Interrupt Programming on page 195.

EXAMPLE: (Routine pulses output on a given position)

```

EIGN(W,0)           'Disable limits
ZS                  'Clear faults
ITR(0,4,0,0,1)     'ITR(int#,sw,bit,state,lbl)
ITRE                'Enable all interrupts
EITR(0)            'Enable interrupt 0
OUT(1)=1           'Set I(0)/O B to output, high
ADT=100            'Set maximum accel/decel
VT=100000          'Set maximum velocity
MP                 'Set Position mode
'****Main Program Body****
WHILE 1>0
  O=0               'Reset origin for move
  PT=40000          'Set final position
  G                 'Start motion
  WHILE PA<20000   'Loop while motion continues
  LOOP              'Wait for desired position to pass
  OUT(1)=0          'Set output low
  TMR(0,400)        'Use timer 0 for pulse width
  TWAIT
  WAIT=1000         'Wait 1 second
LOOP
END
'****Interrupt Subroutine****
C1
  OUT(1)=1          'Set output high again
RETURNI

```

RELATED COMMANDS:

DITR(int) *Disable Interrupts (see page 394)*
EISM(x) *E-Configure Input as Sync Controller (see page 423)*
EITR(int) *Enable Interrupts (see page 424)*
END *End Program Code Execution (see page 439)*
ITR(Int#,StatusWord,Bit#,BitState,Label#) *Interrupt Setup (see page 517)*
ITRD *Interrupt Disable, Global (see page 520)*
PAUSE *Pause Program Execution (see page 648)*
RESUME *Resume Program Execution (see page 704)*
RETURNI *Return Interrupt (see page 708)*
RUN *Run Program (see page 714)*



APPLICATION:	I/O control
DESCRIPTION:	Encoder value latched by falling-edge, hardware-index capture
EXECUTION:	Immediate
CONDITIONAL TO:	Index previously captured
LIMITATIONS:	N/A
READ/REPORT:	RJ(enc)
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Encoder counts
RANGE OF VALUES:	Input: 0 or 1 Output: -2147483648 to 2147483647
TYPICAL VALUES:	Input: 0 or 1 Output: -2147483648 to 2147483647
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	RJ(0):3, x=J(0):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

NOTE: This command is affected by the SCALEP command for ENC1 only. For details, see SCALEP (m,d) on page 726. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

J is the function that stores the last hardware-latched, falling-edge encoder index position. It can be read from a host with the RJ(enc) command, or it can be read by the program with a line such as a=J(enc). The value of enc determines which encoder is being referred to:

- J(0) reads the internal encoder captured count
- J(1) reads the external encoder captured count

The index capture must first be armed with the Aj, Aij or Aji command before a capture will occur. The Bj(enc) command can be used to detect the capture event (due to encoder index or input signal). These capture events can also be detected by status bits in status word 1.

The index is a physical reference mark on the encoder. It is also referred to as a Z pulse, marker pulse, and sometimes a combination of those names. It is typically used in homing sequences requiring a high degree of repeatability.

Class 5.x and later firmware has the ability to redirect port 6 to the Index register input trigger, which allows high-speed position capture through port 6. The internal or external encoder can use this source

through the EIRI and EIRE commands, respectively. When using this method, the previously-stated rules for arming and clearing the index still apply.

EXAMPLE: (homing against a hard stop with Index reference)

NOTE: This method of referencing against a hard stop can eliminate an additional switch and cable.

```

AMPS=100      'Current limit 10%
O=0           'Declare this home
MP            'Set Mode Position
ADT=100      'Set accel/decel
VT=100000    'Set velocity
PT=-1000000  'Move negative
Aj(0)
G             'Start motion
WHILE Bt     'Wait for motion fault
  IF Bj(0)   'If rising-edge index pulse seen
    a=J(0)   'Record falling-edge index position
  ENDIF
LOOP         'Loop back to wait
O=-a        'Last index is home
PT=0        'Move to new home
G           'Start motion
AMPS=1023   'Restore power

```

RELATED COMMANDS:

Ai(enc) *Arm Index Rising Edge (see page 270)*

Aij(enc) *Arm Index Rising Edge Then Falling Edge (see page 272)*

Aj(enc) *Arm Index Falling Edge (see page 274)*

Aji(enc) *Arm Index Falling Edge Then Rising Edge (see page 276)*

^R Bi(enc) *Bit, Index Capture, Rising (see page 309)*

^R Bx(enc) *Bit, Index Input, Real-Time (see page 351)*

EIRE *Enable Index Register, Encoder Capture (see page 419)*

EIRI *Enable Index Register, Input Capture (see page 421)*



APPLICATION:	Motion control
DESCRIPTION:	Acceleration feed forward
EXECUTION:	Buffered until an F command is issued
CONDITIONAL TO:	N/A
LIMITATIONS:	Must be positive
READ/REPORT:	RKA
WRITE:	Read/write
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	0 to 65535
TYPICAL VALUES:	0 to 3000
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	KA:3=1234, a=KA:3, RKA:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

KA sets the buffered acceleration feed-forward gain. The acceleration feed-forward term helps the PID filter cope with the predictable effects of acceleration and inertia.

NOTE: The motion or servo modifications from this command must be applied by the F function. For details, see F on page 457.

The KA gain factor is only applied in position (MP) and velocity (MV) moves. The default value for KA is 0, and acceptable values range from 0 to 65535.

It is difficult or impossible to tune KA in low-inertia systems. Even in high-inertia systems, it can be a challenge to observe the benefit during brief acceleration periods. Therefore, if you think that modifying KA could be useful, use the SMI software Tuner tool for assistance.

EXAMPLE:

```
KA=200    'Set buffered acceleration feed forward
F         'Update PID filter
```

RELATED COMMANDS:

F Force Into PID Filter (see page 457)

R KD=formula Constant, Derivative Coefficient (see page 528)

R KG=formula Constant, Gravitational Offset (see page 530)

R KI=formula Constant, Integral Coefficient (see page 532)

R KL=formula Constant, Integral Limit (see page 535)

R KP=formula Constant, Proportional Coefficient (see page 537)

R KS=formula Constant, Velocity Filter Option (for KD) (see page 540)

R KV=formula Constant, Velocity Feed Forward (see page 542)



APPLICATION:	Motion control; supports the DS2020 Combitronic system
DESCRIPTION:	Derivative coefficient
EXECUTION:	Value of buffered derivative gain
CONDITIONAL TO:	Buffered until an F command is issued
LIMITATIONS:	Must be positive
READ/REPORT:	RKD
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	0 to 65535 DS2020 Combitronic system: 0 to 2147483647
TYPICAL VALUES:	400 to 2000 DS2020 Combitronic system: 50 to 150
DEFAULT VALUE:	Motor-size dependent
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	KD:3=1234, a=KD:3, RKD:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

KD sets the value of the PID filter's derivative gain. If the PID filter gives stable performance, KD is usually the vibration absorbing or damping term.

NOTE: The motion or servo modifications from this command must be applied by the F function. For details, see F on page 457.

For any stable KP value, there is an optimum KD value, and any KD value outside of that causes the motor to be unstable. Therefore, an effective way to tune the filter is to repetitively raise the KP value, and then run the KD term up and down to find the optimum setting. The point at which the KD term cannot stabilize the servo is the point where KP has gone too far.

To test each setting, twist the shaft of the motor and let it go while looking for an abrupt and firm response. Typically, a KD value of approximately ten times KP is a good starting point when $KP < 300$. The SMI software Tuner tool can be useful in finding the optimum setting. However, it does not provide tuning information for the DS2020 Combitronic system.

EXAMPLE:

```
KD=2000      'Set buffered derivative gain
F           'Update PID filter
```


EXAMPLE: (Routine homes motor against a hard stop)

```

MDS                'Using Sine mode commutation
KP=3200            'Increase stiffness from default
KD=10200           'Increase damping from default
F                  'Activate new tuning parameters
AMPS=100           'Lower current limit to 10%
VT=-10000         'Set maximum velocity
ADT=100           'Set maximum accel/decel
MV                 'Set Velocity mode
G                  'Start motion
WHILE EA>-100     'Loop while position error is small
LOOP               'Loop back to WHILE
O=-100            'While pressed, declare home offset
S                  'Abruptly stop trajectory
MP                 'Switch to Position mode
VT=20000          'Set higher maximum velocity
PT=0               'Set target position to be home
G                  'Start motion
TWAIT             'Wait for motion to complete
AMPS=1023         'Restore current limit to maximum
END                'End program

```

RELATED COMMANDS:

R KA=formula *Constant, Acceleration Feed Forward (see page 526)*
R KG=formula *Constant, Gravitational Offset (see page 530)*
R KI=formula *Constant, Integral Coefficient (see page 532)*
R KL=formula *Constant, Integral Limit (see page 535)*
R KP=formula *Constant, Proportional Coefficient (see page 537)*
R KS=formula *Constant, Velocity Filter Option (for KD) (see page 540)*
R KV=formula *Constant, Velocity Feed Forward (see page 542)*



APPLICATION:	Motion control
DESCRIPTION:	Gravitational offset
EXECUTION:	Buffered until an F command is issued
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RKG
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	-16777216 to 16777215
TYPICAL VALUES:	0
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	KG:3=1234, a=KG:3, RKG:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

KG sets the gravity compensation term of the PID filter.

NOTE: The motion or servo modifications from this command must be applied by the F function. For details, see F on page 457.

Simple PID filters are not equipped for a constant force asserted on the system. An example of a constant force is that induced by gravity acting on a vertically moving axis. The KG term exists to offset the PID filter output in a way that removes the effect of these constant forces.

To set KG, set KP and KI to zero and servo in place. The load will want to fall, so you will need to hold it in place. Increase or decrease KG until the load barely holds. Record that value and then continue increasing the parameter until the load begins to move upward. Now record that value. The optimum KG value is the average of the two recorded values.

Valid values for KG are integers from -16777216 to 16777215; the default value is 0. As a result, you may not see much of an effect until KG has a magnitude greater than one million. However, extremely high values will cause rapid pulse-width modulation (PWM) saturation, which results in uncontrollable servo behavior.

EXAMPLE:

```
KG=1000000    'Set buffered gravity term
F             'Update PID filter
```

RELATED COMMANDS:

^R KA=formula *Constant, Acceleration Feed Forward (see page 526)*

R KD=formula *Constant, Derivative Coefficient (see page 528)*

R KI=formula *Constant, Integral Coefficient (see page 532)*

R KL=formula *Constant, Integral Limit (see page 535)*

R KP=formula *Constant, Proportional Coefficient (see page 537)*

R KS=formula *Constant, Velocity Filter Option (for KD) (see page 540)*

R KV=formula *Constant, Velocity Feed Forward (see page 542)*



APPLICATION:	Motion control; supports the DS2020 Combitronic system
DESCRIPTION:	Value of buffered integral gain
EXECUTION:	Buffered until an F command is issued
CONDITIONAL TO:	N/A
LIMITATIONS:	Must be positive; total integral limited by KL (does not apply to DS2020 Combitronic system)
READ/REPORT:	RKI
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	0 to 32767 DS2020 Combitronic system: 0 to 2147483647
TYPICAL VALUES:	0 to 1000 DS2020 Combitronic system: 0
DEFAULT VALUE:	Motor-size dependent DS2020 Combitronic system: 0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	KI:3=1234, a=KI:3, RKI:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The KI term sets the integral gain of the PID filter. The integral compensator is not for stability. Raising it too far will cause the motor to become unstable.

NOTE: The motion or servo modifications from this command must be applied by the F function. For details, see F on page 457.

The KI command is designed to compensate for constant offsets, such as friction in the system or other constant forces. The amount of effort sent to the motor from the KP term of the PID is proportional to the distance it is from its target position. Therefore, as the target gets close, the small position error results in a torque that is too small to allow the motor to reach the final target. The KI term helps to overcome this limitation by adding up, over a long period of time, this small but constant error. This ensures that the servo can eventually reach a position error of 0 at a steady-state speed.

The integral term of the PID filter is generated by taking the sum of the position error of every sample and then multiplying by KI. By doing this, it creates a force that increases over time until the error is corrected. This correction occurs at a rate set by the KI parameter. Therefore, when tuning your motor for stability, it is a good idea to set KI to zero and then increase it until you see that it reliably compensates your system.

NOTE: For the SmartMotor, KL, the protective upper limit, must be set high enough to allow KI to do its job.

NOTE: For the DS2020 Combitronic system, KI can compensate static position error if KV and KP are not sufficient.

EXAMPLE:

```
KI=250      'Set buffered integral gain
F           'Update PID filter
```

RELATED COMMANDS:

R KA=formula *Constant, Acceleration Feed Forward (see page 526)*

R KD=formula *Constant, Derivative Coefficient (see page 528)*

R KG=formula *Constant, Gravitational Offset (see page 530)*

R KL=formula *Constant, Integral Limit (see page 535)*

R KP=formula *Constant, Proportional Coefficient (see page 537)*

R KS=formula *Constant, Velocity Filter Option (for KD) (see page 540)*

R KV=formula *Constant, Velocity Feed Forward (see page 542)*

KII=formula

Current Control Loop: Integrator

APPLICATION:	Motion control
DESCRIPTION:	Current control loop integral gain
EXECUTION:	Immediate
CONDITIONAL TO:	MDC commutation active
LIMITATIONS:	D-style motor does not support this command
READ/REPORT:	RKII
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	0 to 32767
TYPICAL VALUES:	0 to 16000
DEFAULT VALUE:	From factory settings in EEPROM
FIRMWARE VERSION:	5.97.x / 5.98.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

NOTE: The D-style motor does not support this command.

The KII command sets or reports the current-loop integral gain when operating in field-oriented, current-control mode (i.e., MDC).

There are two PI current-control loops when operating in field-oriented control: one loop controls the torque-producing current "Iq", and the other loop nullifies currents that do not produce torque "Id". Both loops use the parameters KPI and KII to control proportional and integral response.

The default (factory) settings for KPI and KII will work in most applications. These should only be changed if necessary and if the effects on the application are understood.

EXAMPLE: (Shows use of KII and KPI)

```
KPI=2000      'Set proportional gain for MDC mode
KII=1500      'Set integral gain for MDC mode
F            'Initiate gains
MDC          'Change to Current mode commutation
```

RELATED COMMANDS:

^R KPI=formula *Current Control Loop: Proportional (see page 539)*



KL=formula

Constant, Integral Limit

APPLICATION:	Motion control
DESCRIPTION:	Integral gain limit
EXECUTION:	Buffered until an F command is issued
CONDITIONAL TO:	N/A
LIMITATIONS:	Must be positive
READ/REPORT:	RKL
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	0 to 32767
TYPICAL VALUES:	0 to 32767
DEFAULT VALUE:	32767
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	KL:3=1234, a=KL:3, RKL:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The KL term sets a limit on the effects of the KI term. Because KI integrates the position error over time, it can eventually dominate the PID equation. To prevent this, KL sets an upper limit on the KI term.

NOTE: The motion or servo modifications from this command must be applied by the F function. For details, see F on page 457.

The KI term will raise the power to the servo as a function of time. If there is something other than friction blocking the servo and it is unable to move, the amount of torque given to the motor can quickly become extremely large. Therefore, KL may be an option for dynamic (changing) loads, or for overshoot due to KI "wind-up".

Note that KL restricts the ability of the PID to compensate for speed when using voltage commutation modes like MDT, MDE and MDS. Therefore, the position error (EA) may become larger as speed increases.

EXAMPLE:

```
KL=1500      'Set buffered integral limit
F           'Update PID filter
```

RELATED COMMANDS:

R KA=formula *Constant, Acceleration Feed Forward (see page 526)*

R KD=formula *Constant, Derivative Coefficient (see page 528)*

R KG=formula *Constant, Gravitational Offset (see page 530)*

R $KI=$ formula *Constant, Integral Coefficient (see page 532)*

R $KP=$ formula *Constant, Proportional Coefficient (see page 537)*

R $KS=$ formula *Constant, Velocity Filter Option (for KD) (see page 540)*

R $KV=$ formula *Constant, Velocity Feed Forward (see page 542)*



APPLICATION:	Motion control; supports the DS2020 Combitronic system
DESCRIPTION:	Proportional coefficient
EXECUTION:	Buffered until an F command is issued
CONDITIONAL TO:	N/A
LIMITATIONS:	Must be positive
READ/REPORT:	RKP
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	0 to 65535 DS2020 Combitronic system: 0 to 2147483647
TYPICAL VALUES:	40 to 8000 DS2020 Combitronic system: 20000 to 60000
DEFAULT VALUE:	Motor-size dependent
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	KP:3=1234, a=KP:3, RKP:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The KP command is used to set the gain of the proportional parameter of the PID filter.

NOTE: The motion or servo modifications from this command must be applied by the F function. For details, see F on page 457.

This creates a force from the PID in direct proportion to how far the motor is pushed away from the calculated trajectory. This force is like a spring that is being stretched — the more it is stretched, the further it resists. While this gives a predictable force to maintain the desired position, it has diminishing results as the error gets smaller. Therefore, to zero the position error in the long term, use the KI term of the PID filter. For details, see KI=formula on page 532.

The higher the KP value, the stiffer the motor will be. At some point, the added stiffness will cause the motor to become unstable. This can sometimes be stabilized by adjusting the KD value (for details, see KD=formula on page 528). However, if moving the KD value up or down does not stabilize the servo, then the KP value is too high and must be reduced.

EXAMPLE:

```
KP=250      'Set buffered proportional gain
F           'Update PID filter
```

EXAMPLE: (Routine homes motor against a hard stop)

```

MDS           'Using Sine mode commutation
KP=3200       'Increase stiffness from default
KD=10200      'Increase damping from default
F            'Activate new tuning parameters
AMPS=100      'Lower current limit to 10%
VT=-10000     'Set maximum velocity
ADT=100       'Set maximum accel/decel
MV           'Set Velocity mode
G            'Start motion
WHILE EA>-100 'Loop while position error is small
LOOP         'Loop back to WHILE
O=-100       'While pressed, declare home offset
S            'Abruptly stop trajectory
MP           'Switch to Position mode
VT=20000     'Set higher maximum velocity
PT=0         'Set target position to be home
G            'Start motion
TWAIT        'Wait for motion to complete
AMPS=1023    'Restore current limit to maximum
END          'End program

```

RELATED COMMANDS:

R KA=formula *Constant, Acceleration Feed Forward (see page 526)*
R KD=formula *Constant, Derivative Coefficient (see page 528)*
R KG=formula *Constant, Gravitational Offset (see page 530)*
R KI=formula *Constant, Integral Coefficient (see page 532)*
R KL=formula *Constant, Integral Limit (see page 535)*
R KS=formula *Constant, Velocity Filter Option (for KD) (see page 540)*
R KV=formula *Constant, Velocity Feed Forward (see page 542)*

KPI=formula

Current Control Loop: Proportional

APPLICATION:	Motion control
DESCRIPTION:	Current control loop proportional gain
EXECUTION:	Immediate
CONDITIONAL TO:	MDC commutation active
LIMITATIONS:	D-style motor does not support this command
READ/REPORT:	RKPI
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	0 to 32767
TYPICAL VALUES:	0 to 16000
DEFAULT VALUE:	From factory settings in EEPROM
FIRMWARE VERSION:	5.97.x / 5.98.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

NOTE: The D-style motor does not support this command.

The KPI command sets or reports the current-loop proportional gain when operating in field-oriented, current-control mode (i.e., MDC).

There are two PI current-control loops when operating in field-oriented control: one loop controls the torque-producing current "Iq", and the other loop nullifies currents that do not produce torque "Id". Both loops use the parameters KPI and KII to control proportional and integral response.

The default (factory) settings for KPI and KII will work in most applications. These should only be changed if necessary and if the effects on the application are understood.

EXAMPLE: (Shows use of KII and KPI)

```
KPI=2000      'Set proportional gain for MDC mode
KII=1500      'Set integral gain for MDC mode
F            'Initiate gains
MDC          'Change to Current mode commutation
```

RELATED COMMANDS:

^R KII=formula *Current Control Loop: Integrator (see page 534)*



APPLICATION:	Motion control
DESCRIPTION:	Velocity filter option (for KD)
EXECUTION:	Buffered until an F command is issued
CONDITIONAL TO:	N/A
LIMITATIONS:	Must be positive
READ/REPORT:	RKS
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	0 to 3 (larger number = longer filter time)
TYPICAL VALUES:	1
DEFAULT VALUE:	1
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	KS:3=1234, a=KS:3, RKS:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The PID filter's KS term is used to adjust the filtering of the KD term.

NOTE: The motion or servo modifications from this command must be applied by the F function. For details, see F on page 457.

The KD term requires a filter because the velocity error signal is inherently noisy and quantized. This is because the measurement of velocity is based on encoder counts per PID sample. It is often a difference of just a few counts.

For example, a speed of VT=65536 is only 1 count per PID sample. This means that in real time, the number jumps between whole numbers. To smooth this out, the quantity can be averaged. The KS value controls the amount of filtering applied. Refer to the next table.

KS Value	Filter Sample Length (PID Samples)	Notes
0	1	No filtering
1	4	Default
2	8	
3	16	

Adjusting the filter will sometimes allow the SmartMotor™ to handle inertial ratios in excess of the traditional 5:1 or 10:1 ratios. This "reflected load to rotor inertia" ratio is often cited as a traditional limit for dependable servo motor applications.

A KS value of 0 produces the least latency, which results in better stability. However, it also produces the most noise (it may produce an audible white noise).

By increasing the KS value, the latency of the PID differential term is increased. Note that this can have a negative impact on the damping ability of the KD term. In other words, the tuning may be less stable with larger values of KS.

EXAMPLE:

```
KS=3      'Set buffered differential sample rate
F         'Update PID filter
```

RELATED COMMANDS:

R KA=formula *Constant, Acceleration Feed Forward (see page 526)*

R KD=formula *Constant, Derivative Coefficient (see page 528)*

R KG=formula *Constant, Gravitational Offset (see page 530)*

R KI=formula *Constant, Integral Coefficient (see page 532)*

R KL=formula *Constant, Integral Limit (see page 535)*

R KP=formula *Constant, Proportional Coefficient (see page 537)*

R KV=formula *Constant, Velocity Feed Forward (see page 542)*



APPLICATION:	Motion control; supports the DS2020 Combitronic system
DESCRIPTION:	Velocity feed forward
EXECUTION:	Buffered until an F command is issued
CONDITIONAL TO:	N/A
LIMITATIONS:	Must be positive
READ/REPORT:	RKV
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	0 to 65535 DS2020 Combitronic system: 0 to 2147483647
TYPICAL VALUES:	0 to 10000 DS2020 Combitronic system: 1000
DEFAULT VALUE:	Motor-size dependent DS2020 Combitronic system: 1000
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	KV:3=1234, a=KV:3, RKV:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

KV sets the gain for the velocity feed-forward element of the extended PID filter.

NOTE: The motion or servo modifications from this command must be applied by the F function. For details, see F on page 457.

The velocity feed-forward element can be thought of as a force that the PID outputs based on the expected speed from the trajectory calculation. This reduces the lag time of the PID waiting for position feedback. The KV term begins the PID response as soon as the trajectory calls for more speed. KP and KI are still required, but the KV term will help improve responsiveness.

If you put the SmartMotor™ into a high-velocity move with KI=0 and monitor the position error with the Motor View tool's Status tab in the SMI software, then you will see a constant position error. To reduce the error to zero, issue a series of successively larger KV parameters (each KV requires an F command to update the PID filter — see the example).

Acceptable values range from 0 to 65535. Typically, useful values range from 0 to 2000. Current values can be reported with RKV.

In the DS2020 Combitronic system, KV sets the feed-forward gain: how much of the velocity computed by the trajectory generator feeds the velocity loop. Usually, if KV=1000 (which means unity gain), the best performances are obtained.

EXAMPLE:

```
KV=1000      'Set buffered velocity feed forward
F           'Update PID filter
```

RELATED COMMANDS:

- R KA=formula *Constant, Acceleration Feed Forward (see page 526)*
- R KD=formula *Constant, Derivative Coefficient (see page 528)*
- R KG=formula *Constant, Gravitational Offset (see page 530)*
- R KI=formula *Constant, Integral Coefficient (see page 532)*
- R KL=formula *Constant, Integral Limit (see page 535)*
- R KP=formula *Constant, Proportional Coefficient (see page 537)*
- R KS=formula *Constant, Velocity Filter Option (for KD) (see page 540)*

Length of Character Count in Communications Port 0

APPLICATION:	Communications control
DESCRIPTION:	Number of characters in channel 0 receive buffer
EXECUTION:	Immediate
CONDITIONAL TO:	Communications channel 0 must be open in data mode
LIMITATIONS:	Maximum buffer length is 31 characters
READ/REPORT:	RLEN
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Number of available characters
RANGE OF VALUES:	0 to 31
TYPICAL VALUES:	0 to 31
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The LEN command checks the receive buffer of serial communications channel 0 and returns the number of characters that are waiting to be processed. Testing the value of LEN is a good way to see if there is a character available for retrieval with the GETCHR command (see the next example).

EXAMPLE:

```

i=0
IF LEN>0           'Any data received?
  GOSUB5           'If so, process data
ENDIF
END
C5
  ab[i]=GETCHR     'Read and store in data
                  'Process incoming data
  i=i+1           'Maintain reference index
RETURN

```

In the previous example, "i" will be equal to LEN.

RELATED COMMANDS:

R GETCHR *Next Character from Communications Port 0 (see page 476)*
R GETCHR1 *Next Character from Communications Port 1 (see page 478)*
R LEN1 *Length of Character Count in Communications Port 1 (see page 545)*
OCHN(...) *Open Channel (see page 632)*

Length of Character Count in Communications Port 1

APPLICATION:	Communications control
DESCRIPTION:	Number of characters in channel 1 receive buffer
EXECUTION:	Immediate
CONDITIONAL TO:	Communications channel 1 must be open in data mode-
LIMITATIONS:	Maximum buffer length is 31 characters
READ/REPORT:	RLEN1
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Number of available characters
RANGE OF VALUES:	0 to 31
TYPICAL VALUES:	0 to 31
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.0.x, 5.16.x or 5.32.x series (D); 6.4.2.x (D) RLEN1 requires: 5.x (D/M); 6.4.2.x (D)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The LEN1 command checks the receive buffer of serial communications channel 1 and returns the number of characters that are waiting to be processed. Testing the value of LEN1 is a good way to see if there is a character waiting for retrieval with GETCHR1 command (see the next example).

NOTE: M-style motors do not have the second communications port (COM 1) needed to support the LEN1 and GETCHR1 commands.

EXAMPLE:

```

i=0
IF LEN1>0      'Any data received?
  GOSUB5      'If so, process data
ENDIF
END
C5
  ab[i]=GETCHR1 'Read and store in data
                'Process incoming data
  i=i+1      'Maintain reference index
RETURN

```

From the above example, "i" will be equal to LEN1.

RELATED COMMANDS:

R GETCHR *Next Character from Communications Port 0 (see page 476)*

R GETCHR1 *Next Character from Communications Port 1 (see page 478)*

R LEN *Length of Character Count in Communications Port 0 (see page 544)*

OCHN(...) *Open Channel (see page 632)*

APPLICATION:	Data conversion
DESCRIPTION:	Get float value from 32-bit IEEE format
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RLFS(value)
WRITE:	N/A
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	Input: 32-bit integer, -2147483648 to 2147483647 Output: any floating-point value within 32-bit float range: $\pm 1 \times 10^{38}$
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The LFS command is used to get the float value from 32-bit IEEE-754 format.

It allows the import of floating-point values from external systems — this format may be needed for interchange.

The 32-bit value input by this function is *not* a normal integer value. It is an encoded number that includes the exponent of the floating-point value.

NOTE: The input of this function does not have any usefulness within the SmartMotor programming language.

The output of this function is a floating-point value that can be used in a formula. The output of LFS() is directed to DFS(). For details on DFS(), see DFS(value) on page 393.

EXAMPLE:

```
a1[0]=1162934955      '4 byte value in IEEE-754 format
af[0]=LFS(a1[0])     'Convert from IEEE-754 to float

Raf[0]                'Report value of float
```

Program output is:

```
3343.666748046
```

RELATED COMMANDS:

R af[index]=formula *Array Float [index] (see page 267)*

R DFS(value) *Dump Float, Single (see page 393)*

LOAD**Download Compiled User Program to Motor**

APPLICATION:	Program access
DESCRIPTION:	Download and store executable SmartMotor™ program to motor
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	EEPROM is read/write unless "locked"
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

NOTE: This command is intended to be used in custom terminal software for PLCs, HMIs or similar devices.

The LOAD command is used by a terminal to download a compiled program file and store it within the user-program EEPROM in the SmartMotor. The LOAD command causes a SmartMotor to load all incoming host communications into program memory up to the first occurrence of ASCII character 255. This command is mainly used by host utilities, which also compile the program before download.

User programs are stored in the SmartMotor's EEPROM memory. The maximum program size depends on the motor class you are using:

- For Class 5 motors, the maximum program size is 32767 bytes.
- For Class 6 motors, the maximum program size is 64150 bytes.

LOAD terminates the current motion mode or trajectory. However, it does not change motion parameters such as EL, ADT, VT, KP, etc., or alter the current value of the user variables.

If the motor does not receive the ASCII 255 byte after the LOAD command is issued, it will continue to store incoming serial bytes directly to the user-program EEPROM. During this time, the motor will not respond to your commands. The only way to terminate this condition is to transmit ASCII 255 bytes or to reset the power.

NOTE: The SMI (SmartMotor Interface) software package is adjusted to take care of this automatically.

By using the LOAD command, you can download the file from any controller, HMI, PLC or PC-based program capable of storing an ASCII text file. For any given motor that is actively addressed (i.e., you are talking to it and it responds), if you issue the LOAD command to the motor, it immediately goes into

a memory-write mode while checking all incoming data. After issuing the LOAD command, every ASCII character that is received goes directly into the user-program EEPROM. To terminate the LOAD command, the last characters sent must be two hex FF characters and one hex 20 character. This tells the motor that it is the end of the file and to revert to regular command mode.

Details on the downloadable file:

When you compile an SMS file with the SMI software, it creates an SMX file extension with the same name in the same directory. That is the file you need to download to the motor.

Perform these steps to complete the download operation:

1. Establish serial communications with the motor.
2. Issue the LOAD command.
3. If ECHO is enabled, you should see the LOAD command and then an ECHOed hex 20.
4. Hex value 06 should be transmitted by the motor. Read and verify this byte.
5. Transmit the first 32 characters from the SMX file. If the SMX file is shorter than 32 bytes, just send what is available and skip to step 8 (sending FF FF 20).
6. Hex value 06 should be transmitted by the motor. Read and verify this byte.
7. Repeat step 5 using the next 32 bytes.
8. When the last character is read from the file and sent to the motor, then send two hex FF characters and one hex 20 character to the motor.
9. Issue another RCKS command. If it returns a success status (with the P character at the end), then the download was successful.
10. RUN the program (no reboot required).

These are reasons for unsuccessful download:

- Noise on the serial port
- Loss of connection during download
- Failure to send the two hex FF characters and one hex 20 character before power down
- The SMI-compiled SMX file was altered in some way

NOTE: Do not alter the SMX file from its originally-compiled version. For example, if you open an SMX file in Windows Notepad to view and save it, Notepad automatically adds a carriage return character to the end of each line. Therefore, the saved file will not work, and the carriage returns must be removed before downloading the file to the motor.

EXAMPLE:

This procedure is an example of the steps needed for an outside program to send an SMX file to the SmartMotor.

1. Write the LOAD command.
2. Read a 0x06 character back from the motor.
3. Write 32 bytes of the .SMX file to the motor at a time.
4. Read the 0x06 character back from the motor.
5. Write any remaining bytes of the .SMX file (< 32).
6. Write 0xff 0xff 0x20.

RELATED COMMANDS:

LOCKP *Lock Program (see page 551)*

RCKS *Report Checksum (see page 701)*

RUN *Run Program (see page 714)*

RUN? *Halt Program Execution Until RUN Received (see page 716)*

UP *Upload Compiled Program and Header (see page 797)*

UPLOAD *Upload Standard User Program (see page 799)*

LOCKP

Lock Program

APPLICATION:	Program access
DESCRIPTION:	Prevents function of UP and UPLOAD
EXECUTION:	N/A
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The LOCKP command modifies the downloaded program in the motor's EEPROM to prevent it from being uploaded. That is, the commands UP and UPLOAD will not be able to upload the program body or contents.

NOTE: LOCKP does not prevent the download of another program.

The LOCKP command should be used after program development and testing is complete.

LOCKP can be used as a serial command or incorporated in a user program for Class 5 and later motors.

- If it is used as a serial command, it should be issued from the Terminal window.
- If it is used in a program, be sure that part of the program with the LOCKP command is run to ensure LOCKP protection is effective.



CAUTION: It is the developer's responsibility to make sure the part of the program with the LOCKP command is run to ensure LOCKP protection—the LOCKP command itself has to execute—just having it in the program doesn't take effect until that part of the program is actually executed. It should be verified by trying to upload after running the program.

After LOCKP is issued, issuing UP or UPLOAD will no longer produce results.

EXAMPLE:

```
LOCKP    'Requires a motor reboot to take effect.'
```

RELATED COMMANDS:

UP *Upload Compiled Program and Header (see page 797)*

UPLOAD *Upload Standard User Program (see page 799)*

LOOP

Loop Back to WHILE Formula

APPLICATION:	Program execution and flow control
DESCRIPTION:	Terminator for WHILE formula
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

LOOP is the statement terminator for the WHILE control block. Each WHILE must have only one corresponding LOOP. Each time LOOP is encountered, program execution branches back to reevaluate the WHILE formula.

The WHILE formula...LOOP control block creates a program loop that repeatedly executes for as long as the formula value is true or nonzero. The formula is evaluated when WHILE is first encountered, and each time program execution is sent back to the WHILE by the corresponding terminating LOOP statement. If the formula value is zero or false, program execution continues on the line of code just below the LOOP command.

```
WHILE formula...LOOP
```

It is legal to jump from an external program location to a label within a WHILE control loop. However, this method may not be the best practice.

LOOP is not a valid terminal command. It is only valid within a user program.

BREAK can be used to exit the WHILE loop.

EXAMPLE:

```
b=1
WHILE b<5
    PRINT (#13, "b=", b)
    b=b+1
LOOP PRINT (#13, "Exit Loop")
END
```

The previous code outputs:

```
b=1  
b=2  
b=3  
b=4  
b=5  
Exit Loop
```

RELATED COMMANDS:

BREAK *Break from CASE or WHILE Loop (see page 331)*

WHILE *formula While Condition Program Flow Control (see page 841)*



Mode Cam (Electronic Camming)

APPLICATION:	Motion control
DESCRIPTION:	Request Cam mode (electronic camming)
EXECUTION:	Buffered until a G command is issued
CONDITIONAL TO:	Cam table created
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	MC:3 or MC(0):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The MC command puts the SmartMotor™ into Cam mode, which causes the motor to follow a predetermined profile according to an external encoder source. To set up a Cam operation, you must also specify position data and initialize it to the controller source (either an external or internal timer). The camming motion is started by issuing a G command. The next example shows a complete command sequence.

NOTE: Refer to the Related Commands section, as there are several important commands to use when creating a Cam table and configuring the cam motion.

In Cam mode, each value of the external encoder defines a required corresponding motor position. Cams typically define a periodic motion profile or trajectory.

NOTE: The MCW(table, point) command is *mandatory* when using Cam mode (MC) It should be called after the creation of cam tables but before the G command. If cam tables already exist in memory from a power-on, then MCW should be called at least once before the G command to enter Cam mode. Refer to the next example.

Cam tables may be stored in EEPROM or in user variables ab, aw or al.

The controller source can be selected with the SRC command. Also, the MFA, MFD, MFMUL, MFDIV, MFSLEW and MFSDC commands are in effect with Cam mode. This allows sophisticated traversal control over the Cam table. For instance, a motion can be defined where a continuously running controller can be eased into and out of using the MFA and MFD ramps. Also, using the MFSLEW command, a predefined number of controller counts can be accepted before the Cam table halts.

The MCMUL and MCDIV commands allow the output (follower position) to be scaled without rewriting the Cam table.

EXAMPLE: (Routine exercises each Cam User Bit during the programmed cam profile)

```

EIGN (w, 0)
ZS
CTA (7, 0, 0)          'Add table into RAM al[0]-al[8].
CTW (0, 0, 1)         'Add 1st point, Cam User Bit 0 ON.
CTW (1000, 4000, 1)   'Add 2nd point, Cam User Bit 0 ON.
CTW (3000, 8000, 2)   'Add 3rd point, Cam User Bit 1 ON.
CTW (4000, 12000, 132) 'Add 4th, Spline Mode, Cam Bit 2 ON.
CTW (1000, 16000, 136) 'Add 5th, Spline Mode, Cam Bit 3 ON.
CTW (-2000, 20000, 16) 'Add 6th point, Cam Bit 4 ON.
CTW (0, 24000, 32)    'Add 7th point, Cam Bit 5 ON.
MC                    'Select Cam Mode.
SRC (2)               'Use the virtual controller encoder.
MCE (0)               'Force Linear interpolation.
MCW (0, 0)            'Use table 0 in RAM from point 0.
MFMUL=1               'Simple 1:1 ratio from virtual enc.
MFDIV=1               'Simple 1:1 ratio from virtual enc.
MFA (0) MFD (0)      'Disable virtual enc. ramp-up/ramp-
                      'down sections.
MFSLEW (24000, 1)    'Table is 6 segments * 4000 encoder
                      'counts each. Specify 1 for the second
                      'argument, which forces this number as
                      'the output total of the virtual controller
                      'encoder into the cam.
MFSDC (-1, 0)        'Disable virtual controller (gearing) repeat.
G                     'Begin move.
END                   'Required END.

```

RELATED COMMANDS:

R MCDIV=formula *Mode Cam Divisor (see page 557)*

MFO *Mode Follow, Zero External Counter (see page 578)*

MFA(distance[,m/s]) *Mode Follow Ascend (see page 580)*

MFD(distance[,m/s]) *Mode Follow Descend (see page 585)*

R MFDIV=formula *Mode Follow Divisor (see page 588)*

R MFMUL=formula *Mode Follow Multiplier (see page 598)*

MFSDC(distance,mode) *Mode Follow, Stall-Dwell-Continue (see page 603)*

MFSLEW(distance[,m/s]) *Mode Follow Slew (see page 605)*

MSO *Mode Step, Zero External Counter (see page 616)*

R MCMUL=formula *Mode Cam Multiplier (see page 560)*

MCE(arg) *Mode Cam Enable () (see page 558)*

MCW(table,point) *Mode Cam Where (Start Point) (see page 562)*

ECS(counts) *Encoder Count Shift (see page 410)*

CTA(points,seglen[,location]) *Cam Table Attribute (see page 376)*

CTW(pos[,seglen][,user]) *Cam Table Write Data Points (see page 383)*

CTE(table) *Cam Table Erase (see page 378)*

SRC(enc_src) *Source, Follow and/or Cam Encoder (see page 759)*



MCDIV=formula

Mode Cam Divisor

APPLICATION:	Motion control
DESCRIPTION:	Cam mode ratio divisor
EXECUTION:	Buffered at start of cam motion and at restarts due to MFSLEW length
CONDITIONAL TO:	Cam mode active
LIMITATIONS:	N/A
READ/REPORT:	RMCDIV
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	-32767 to -1 and 1 to 32767 (0 excluded)
TYPICAL VALUES:	-32767 to -1 and 1 to 32767 (0 excluded)
DEFAULT VALUE:	1
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	MCDIV:3=1234, a=MCDIV:3, RMCDIV:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The MCDIV command provides the Cam mode ratio divisor. It works in combination with the MCMUL command, which provides the Cam mode ratio multiplier. For usage details, see MCMUL=formula on page 560.

NOTE: MCDIV cannot be set to 0.

EXAMPLE:

For examples, see MCMUL=formula on page 560.

RELATED COMMANDS:

MC *Mode Cam (Electronic Camming)* (see page 555)

R MCMUL=formula *Mode Cam Multiplier* (see page 560)

MFSDC(distance,mode) *Mode Follow, Stall-Dwell-Continue* (see page 603)

MFSLEW(distance[,m/s]) *Mode Follow Slew* (see page 605)

APPLICATION:	Motion control
DESCRIPTION:	Enable Cam mode operation type
EXECUTION:	Buffered until a G command is issued
CONDITIONAL TO:	MC mode selected
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	0,1,2
TYPICAL VALUES:	0,1,2
DEFAULT VALUE:	1
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The MCE(arg) command specifies the Cam mode operation type. For details on Cam mode, see Cam Mode (Electronic Camming) on page 156.

The arg parameter must be one of these values:

arg	Description
0	Force linear interpolation. This overrides any per-segment, linear versus spline option.
1	(Default) Allow spline mode unless a segment has a linear specification. The shape of the spline assumes that both the beginning and end of the table have a slope of 0.
2	Allow spline mode unless a segment has a linear specification. The shape (slope) of the spline takes into consideration wrapping around the end of the table.

EXAMPLE: (Routine exercises each Cam User Bit during the programmed cam profile)

```

EIGN (w, 0)
ZS
CTA (7, 0, 0)           'Add table into RAM al[0]-al[8].
CTW (0, 0, 1)           'Add 1st point, Cam User Bit 0 ON.
CTW (1000, 4000, 1)     'Add 2nd point, Cam User Bit 0 ON.
CTW (3000, 8000, 2)     'Add 3rd point, Cam User Bit 1 ON.
CTW (4000, 12000, 132)  'Add 4th, Spline Mode, Cam Bit 2 ON.
CTW (1000, 16000, 136)  'Add 5th, Spline Mode, Cam Bit 3 ON.
CTW (-2000, 20000, 16)  'Add 6th point, Cam Bit 4 ON.
CTW (0, 24000, 32)      'Add 7th point, Cam Bit 5 ON.
MC                       'Select Cam Mode.
SRC (2)                  'Use the virtual controller encoder.
MCE (0)                  'Force Linear interpolation.
MCW (0, 0)               'Use table 0 in RAM from point 0.
MFMUL=1                  'Simple 1:1 ratio from virtual enc.
MFDIV=1                  'Simple 1:1 ratio from virtual enc.
MFA (0) MFD (0)          'Disable virtual enc. ramp-up/ramp-
                          'down sections.
MFSLEW (24000, 1)       'Table is 6 segments * 4000 encoder
                          'counts each. Specify 1 for the second
                          'argument, which forces this number as
                          'the output total of the virtual controller
                          'encoder into the cam.
MFSDC (-1, 0)           'Disable virtual controller (gearing) repeat.
G                         'Begin move.
END                       'Required END.

```

RELATED COMMANDS:

MC *Mode Cam (Electronic Camming) (see page 555)*



MCMUL=formula

Mode Cam Multiplier

APPLICATION:	Motion control
DESCRIPTION:	Cam mode ratio multiplier
EXECUTION:	Buffered at start of cam motion and at restarts due to MFSLEW length
CONDITIONAL TO:	Cam mode active
LIMITATIONS:	N/A
READ/REPORT:	RMCMUL
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	-4194304 to 4194304
TYPICAL VALUES:	-4194304 to 4194304
DEFAULT VALUE:	1
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	MCMUL:3=1234, a=MCMUL:3, RMCMUL:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The MCMUL command provides the Cam mode ratio multiplier. It is used in combination with MCDIV to scale the output of the follower position in Cam mode. This allows a Cam table's amplitude to be rescaled without rewriting the table.

Choose a ratio of MCMUL/MCDIV that provides sufficient resolution while allowing the desired range. For example, to allow scaling from 10% to 1000% in 0.1% increments, choose MCDIV=1000. For this case, a setting of MCMUL=1000 would provide a 100% (1:1) scaling of the Cam table.

If the ratio of MCMUL/MCDIV produces a negative ratio, then the amplitude of the Cam table is also negative.

Typically, the scaling should be configured before initiating cam motion with the G command. However, under some circumstance, it is possible to change the scaling while moving.

To change scaling while moving, the changes to MCMUL are only accepted if a restart of the cam motion occurs. A G command will force the restart of the cam motion. However, issuing a G command during cam motion will cause a major disruption in speed and position. It will move the starting follower position of the table to home, which is usually detrimental to machine operations. Therefore, issuing a G command will only work if the cam is at the starting follower position and remains there long enough to reliably issue that command. It may be difficult to correctly time this in a program.

Instead, the controller profile can be used to accurately time the restart of the cam motion. The Follow mode commands are still active in Cam mode, and are used to feed the controller into the Cam table.

The MFSLEW and MFSDC commands can be used to create a controller profile that repeats to correspond to the start of the Cam table.

The controller profile can be programmed to restart by carefully selecting an MFSLEW(length,1) command value to align with the length of the Cam table (i.e., the number of segments times the segment length, or the sum of all variable-length segment lengths). Note that the MFSLEW command specifies the length in "follower" units because the output (follower) of the follow commands is fed into the cam. The MFSDC(0,0) command is also required if a repetitive cam motion is desired. The input source (specified by the SRC command) can be any source. For details, see SRC(enc_src) on page 759.

EXAMPLE: (Subroutine from Cam program)

C41

```
MP PT=0 G TWAIT
SRC (2)
MCE (1)          'Spline
MFA (0)
MFD (0)
MFMUL=1
MFDIV=1
MCMUL=1
MCDIV=1
MFSLEW (112000,1)
MFSDC (100,0)    'Set dwell for "c" counts, auto reverse after dwell
MC
G
```

RETURN

RELATED COMMANDS:

MC *Mode Cam (Electronic Camming) (see page 555)*

R MCDIV=formula *Mode Cam Divisor (see page 557)*

R MCMUL=formula *Mode Cam Multiplier (see page 560)*

MFSDC(distance,mode) *Mode Follow, Stall-Dwell-Continue (see page 603)*

MFSLEW(distance[,m/s]) *Mode Follow Slew (see page 605)*

SRC(enc_src) *Source, Follow and/or Cam Encoder (see page 759)*

MCW(table,point) Mode Cam Where (Start Point)

APPLICATION:	Motion control
DESCRIPTION:	Specifies the Cam mode start point
EXECUTION:	Buffered until a G command is issued
CONDITIONAL TO:	Cam table created; MC (Cam mode) selected
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	Input: table: 0-10 point: 0-65535
TYPICAL VALUES:	Input: table: 0-10 point: 0-750
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The MCW command selects the Cam table and specifies the starting point used when the cam motion is initiated with a G command. Typically, the starting point is point 0, but values from 0 to 65535 are valid. For the table input parameter, valid values and their meanings are:

Table Selection	Description
0	RAM table
1-10	Tables stored sequentially in EEPROM

NOTE: The MCW(table, point) command is *mandatory* when using Cam mode (MC) It should be called after the creation of cam tables but before the G command. If cam tables already exist in memory from a power-on, then MCW should be called at least once before the G command to enter Cam mode. Refer to the next example.

EXAMPLE: (Routine exercises each Cam User Bit during the programmed cam profile)

```

EIGN (w, 0)
ZS
CTA (7, 0, 0)           'Add table into RAM a1[0]-a1[8].
CTW (0, 0, 1)          'Add 1st point, Cam User Bit 0 ON.
CTW (1000, 4000, 1)    'Add 2nd point, Cam User Bit 0 ON.
CTW (3000, 8000, 2)    'Add 3rd point, Cam User Bit 1 ON.
CTW (4000, 12000, 132) 'Add 4th, Spline Mode, Cam Bit 2 ON.
CTW (1000, 16000, 136) 'Add 5th, Spline Mode, Cam Bit 3 ON.
CTW (-2000, 20000, 16) 'Add 6th point, Cam Bit 4 ON.
CTW (0, 24000, 32)     'Add 7th point, Cam Bit 5 ON.
MC                     'Select Cam Mode.
SRC (2)                'Use the virtual controller encoder.
MCE (0)                'Force Linear interpolation.
MCW (0, 0)             'Use table 0 in RAM from point 0.
MFMUL=1                'Simple 1:1 ratio from virtual enc.
MFDIV=1                'Simple 1:1 ratio from virtual enc.
MFA (0) MFD (0)        'Disable virtual enc. ramp-up/ramp-
                        'down sections.
MFSLEW (24000, 1)     'Table is 6 segments * 4000 encoder
                        'counts each. Specify 1 for the second
                        'argument, which forces this number as
                        'the output total of the virtual controller
                        'encoder into the cam.
MFSDC (-1, 0)         'Disable virtual controller (gearing) repeat.
G                      'Begin move.
END                    'Required END.

```

RELATED COMMANDS:

CTA(points,seglen[,location]) *Cam Table Attribute (see page 376)*
 CTE(table) *Cam Table Erase (see page 378)*
 R CTT *Cam Table Total in EEPROM (see page 382)*
 CTW(pos[,seglen][,user]) *Cam Table Write Data Points (see page 383)*
 G *Start Motion (GO) (see page 473)*
 MC *Mode Cam (Electronic Camming) (see page 555)*



Enable TOB Feature (Commutation Mode)

APPLICATION:	Motion control
DESCRIPTION:	Enable the Trajectory Overshoot Braking (TOB) option
EXECUTION:	Immediate; remains in effect until otherwise commanded
CONDITIONAL TO:	Trapezoidal (6-step) commutation mode active: MDT or MDE
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	By default, this mode is not enabled
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	MDB:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The MDB command enables the Trajectory Overshoot Braking (TOB) option.

This command should be used to enable TOB after using the MDT or MDE commands. Note that:

- This option reverts to off when one of these commutation mode commands are used.
- This option is off by default. Status Word 6, Bit 9 indicates if this mode is active.

NOTE: MDE, MDS and MDC require angle match before they will take effect. This means the SmartMotor's factory calibration is valid and the index mark of the internal encoder has been seen after startup. The default commutation mode for D-style motors is MDT (see MDT on page 576); the default commutation mode for M-style motors is MDC (see MDC on page 566).

EXAMPLE: (Shows the use of MDB and MDE)

```
'NOTE: MDE and MDB can help with handling high inertia loads.
EIGN(W,0)           'Make all onboard I/O inputs.
ZS                 'Clear errors.
MP VT=20000 ADT=100 O=0 'Mode Position, Velocity, accel/decel,
                    'zero encoder.
MDE 'Switch to Enhanced Trap Commutation Mode (default is MDT).
MDB 'Turn on Trajectory Overshoot Braking (MDE mode is required for MDB).
PT=8000 G TWAIT    'Move to Absolute Position 8000.
PT=0 G TWAIT       'Move to Absolute Position 0.
END
```

RELATED COMMANDS:

MDC *Mode Current (Commutation Mode) (see page 566)*

MDE *Mode Enhanced (Commutation Mode) (see page 568)*

MDH *Mode Hybrid (Commutation Mode) (see page 570)*

MDHV *Mode Hybrid Velocity (Commutation Mode) (see page 572)*

MDS *Mode Sine (Commutation Mode) (see page 574)*

MDT *Mode Trap (Commutation Mode) (see page 576)*



Mode Current (Commutation Mode)

APPLICATION:	Motion control
DESCRIPTION:	Sine current commutation mode
EXECUTION:	Immediate; remains in effect until otherwise commanded
CONDITIONAL TO:	M-style motors only, not supported in D-style motors
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	D-style motors default to MDT commutation mode; M-style motors default to MDC commutation mode
FIRMWARE VERSION:	5.97.x / 5.98.x (D/M); 6.x (D/M)
COMBITRONIC:	MDC:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The MDC command enables the motor's sinusoidal (sine) commutation mode augmented with digital current control. Available only in the M-style SmartMotor, this method offers optimum performance without sacrificing quiet operation. It is the best choice for an application when this capability is available.

Because MDC uses the encoder, it requires angle match (the first sighting of the encoder index) before it will engage.

Use status word 6 to see the active commutation mode.

NOTE: MDE, MDS and MDC require angle match before they will take effect. This means the SmartMotor's factory calibration is valid and the index mark of the internal encoder has been seen after startup. The default commutation mode for D-style motors is MDT (see MDT on page 576); the default commutation mode for M-style motors is MDC (see MDC on page 566).

EXAMPLE:

```
KPI=2000      'Set proportional gain for MDC mode
KII=1500      'Set integral gain for MDC mode
F             'Initiate gains

MDC           'Change to Current Mode commutation

MV            'Set velocity move
VT=200000    'Set velocity target
ADT=50        'Set accel/decel target
G            'Start motion
```

RELATED COMMANDS:

MDB *Enable TOB Feature (Commutation Mode) (see page 564)*
MDE *Mode Enhanced (Commutation Mode) (see page 568)*
MDH *Mode Hybrid (Commutation Mode) (see page 570)*
MDHV *Mode Hybrid Velocity (Commutation Mode) (see page 572)*
MDS *Mode Sine (Commutation Mode) (see page 574)*
MDT *Mode Trap (Commutation Mode) (see page 576)*



Mode Enhanced (Commutation Mode)

APPLICATION:	Motion control
DESCRIPTION:	Enables enhanced trapezoidal (6-step) mode commutation using the encoder
EXECUTION:	Immediate; remains in effect until otherwise commanded
CONDITIONAL TO:	Motor rotates past internal index to initialize commutation angle
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	D-style motors default to MDT commutation mode; M-style motors default to MDC commutation mode
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	MDE:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The MDE command enables the motor's enhanced trapezoidal commutation mode using the encoder. This driving method is the same as basic trapezoidal commutation using Hall sensors. However, it also uses the internal encoder to add accuracy to the commutation trigger points. This idealized trapezoidal commutation mode offers the greatest motor torque and speed, but it can exhibit minor ticking sounds at low rates, which are created as the current shifts abruptly from one coil to the next.

Because MDE uses the encoder, it requires angle match (the first sighting of the encoder index) before it will engage.

Use status word 6 to see the active commutation mode.

NOTE: MDE, MDS and MDC require angle match before they will take effect. This means the SmartMotor's factory calibration is valid and the index mark of the internal encoder has been seen after startup. The default commutation mode for D-style motors is MDT (see MDT on page 576); the default commutation mode for M-style motors is MDC (see MDC on page 566).

EXAMPLE: (Shows the use of MDB and MDE)

```
'NOTE: MDE and MDB can help with handling high inertia loads.
EIGN(W,0)           'Make all onboard I/O inputs.
ZS                 'Clear errors.
MP VT=20000 ADT=100 O=0 'Mode Position, Velocity, accel/decel,
                    'zero encoder.
MDE 'Switch to Enhanced Trap Commutation Mode (default is MDT).
MDB 'Turn on Trajectory Overshoot Braking (MDE mode is required for MDB).
PT=8000 G TWAIT    'Move to Absolute Position 8000.
PT=0 G TWAIT       'Move to Absolute Position 0.
END
```

RELATED COMMANDS:

MDB *Enable TOB Feature (Commutation Mode) (see page 564)*
MDC *Mode Current (Commutation Mode) (see page 566)*
MDH *Mode Hybrid (Commutation Mode) (see page 570)*
MDHV *Mode Hybrid Velocity (Commutation Mode) (see page 572)*
MDS *Mode Sine (Commutation Mode) (see page 574)*
MDT *Mode Trap (Commutation Mode) (see page 576)*



Mode Hybrid (Commutation Mode)

APPLICATION:	Motion control
DESCRIPTION:	Enables hybrid commutation mode
EXECUTION:	Immediate; remains in effect until otherwise commanded
CONDITIONAL TO:	Motor rotates past internal index to initialize commutation angle
LIMITATIONS:	Command is not available for any Class 6 motors
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	D-style motors default to MDT commutation mode; M-style motors default to MDC commutation mode
FIRMWARE VERSION:	5.x.4.31 and later (NOTE: 5.0.x, 5.16.x or 5.32.x series only); no Class 6
COMBITRONIC:	MDH:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The MDH command enables the motor's hybrid commutation mode. The purpose of this mode is to overcome the lack of MDC mode support in Class 5 D-style motors running specific firmware (i.e., firmware 5.x.4.31 and later in the 5.0.x, 5.16.x or 5.32.x series only). For details on MDC mode, see MDC on page 566.

The operation of MDH mode will apply either MDS or MDE commutation mode based on an operating speed specified by the MDHV command:

- below that speed, MDS commutation is applied for smooth rotation
- above that speed, MDE commutation is applied for high-speed efficiency

NOTE: At higher speeds, the motor tends to average out any of the low-speed smoothness issues. At lower speeds, the loss of efficiency is minor.

To determine the selection of either MDS or MDE commutation, the motor's actual measured speed as an absolute (positive) value is compared to the value set by the MDHV command based on the criteria described above.

EXAMPLE: (shows MDH and MDHV)

```
MDHV=500000  'Set hybrid transition velocity
MDH          'Set hybrid mode
             'Will remain in hybrid mode until commanded otherwise
MV          'Set velocity move
VT=600000   'Set velocity target
ADT=10      'Set accel/decel target
G           'Start motion
END
```

RELATED COMMANDS:

MDB *Enable TOB Feature (Commutation Mode) (see page 564)*
MDC *Mode Current (Commutation Mode) (see page 566)*
MDE *Mode Enhanced (Commutation Mode) (see page 568)*
MDHV *Mode Hybrid Velocity (Commutation Mode) (see page 572)*
MDS *Mode Sine (Commutation Mode) (see page 574)*
MDT *Mode Trap (Commutation Mode) (see page 576)*



Mode Hybrid Velocity (Commutation Mode)

APPLICATION:	Motion control
DESCRIPTION:	Hybrid velocity commutation mode transition speed
EXECUTION:	Immediate; remains in effect until otherwise commanded
CONDITIONAL TO:	MDH mode active
LIMITATIONS:	Command is not available for any Class 6 motors
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	Assignment
UNITS:	N/A
RANGE OF VALUES:	0 to 2147483647
TYPICAL VALUES:	100000 to 1000000
DEFAULT VALUE:	546133
FIRMWARE VERSION:	5.x.4.31 and later (NOTE: 5.0.x, 5.16.x or 5.32.x series only); no Class 6
COMBITRONIC:	MDHV:3=500000 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The MDHV command is a setting in relation to the motor's hybrid velocity commutation mode, MDH, for Class 5 D-style motors running specific firmware (i.e., firmware 5.x.4.31 and later in the 5.0.x, 5.16.x or 5.32.x series only). The command is used as:

MDHV=

which sets the motor actual speed (a positive, absolute value) where the transition from MDS to MDE commutation mode occurs. Refer to the next code example. For more details, see MDH on page 570.

EXAMPLE: (shows MDH and MDHV)

```
MDHV=500000  'Set hybrid transition velocity
MDH          'Set hybrid mode
             'Will remain in hybrid mode until commanded otherwise
MV          'Set velocity move
VT=600000   'Set velocity target
ADT=10      'Set accel/decel target
G           'Start motion
END
```

RELATED COMMANDS:

MDB *Enable TOB Feature (Commutation Mode) (see page 564)*

MDC *Mode Current (Commutation Mode) (see page 566)*

MDE *Mode Enhanced (Commutation Mode) (see page 568)*

MDH *Mode Hybrid (Commutation Mode) (see page 570)*

MDS *Mode Sine (Commutation Mode) (see page 574)*

MDT *Mode Trap (Commutation Mode) (see page 576)*

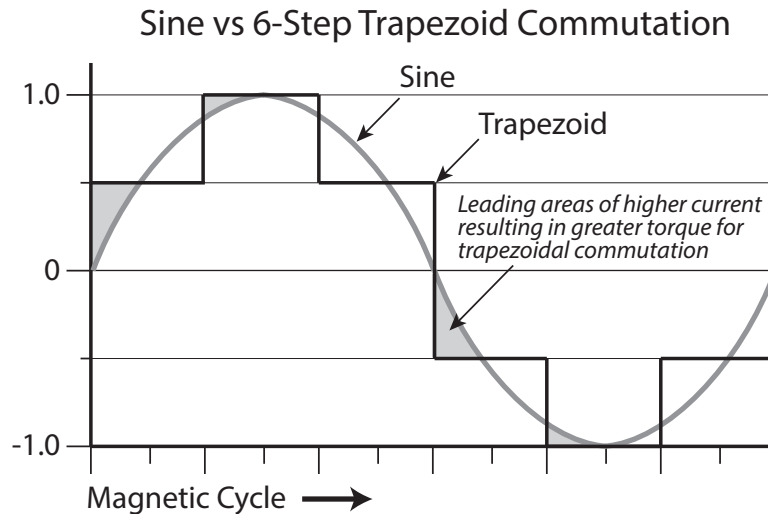


APPLICATION:	Motion control
DESCRIPTION:	Enable sine mode commutation, voltage mode
EXECUTION:	Immediate; remains in effect until otherwise commanded
CONDITIONAL TO:	Motor rotates past internal index to initialize commutation angle
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	D-style motors default to MDT commutation mode; M-style motors default to MDC commutation mode
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	MDS:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The MDS command enables the motor's sine commutation mode (voltage mode). This mode offers smoother commutation when compared to trapezoidal modes. It accomplishes this by shifting current more gradually from one coil to the next. Whereas, trapezoidal modes exhibit higher torque due to the longer application of current to the windings, which overrides losses.

Refer to the next figure. It shows the differences between sine and 6-step trapezoidal commutation modes, used by the SmartMotor, for one magnetic cycle. (Commutation ensures firing of switches on the proper magnetic cycle.) The shaded leading areas indicate the application of current sooner in trapezoidal commutation mode, which results in greater torque.



Because MDS uses the encoder, for motors with incremental encoders, it requires angle match (the first sighting of the encoder index) before it will engage. MDS is the best choice for applications that require extremely smooth and quiet rotation at low speeds.

Use status word 6 to see the active commutation mode.

NOTE: MDE, MDS and MDC require angle match before they will take effect. This means the SmartMotor's factory calibration is valid and the index mark of the internal encoder has been seen after startup. The default commutation mode for D-style motors is MDT (see MDT on page 576); the default commutation mode for M-style motors is MDC (see MDC on page 566).

EXAMPLE:

```
MDS      'Set sine mode
          'Will remain in sine mode until commanded otherwise
MV       'Set velocity move
VT=50000 'Set velocity target
ADT=10   'Set accel/decel target
G        'Start motion
END
```

RELATED COMMANDS:

MDB *Enable TOB Feature (Commutation Mode) (see page 564)*
MDC *Mode Current (Commutation Mode) (see page 566)*
MDE *Mode Enhanced (Commutation Mode) (see page 568)*
MDH *Mode Hybrid (Commutation Mode) (see page 570)*
MDHV *Mode Hybrid Velocity (Commutation Mode) (see page 572)*
MDT *Mode Trap (Commutation Mode) (see page 576)*



Mode Trap (Commutation Mode)

APPLICATION:	Motion control
DESCRIPTION:	Enable trapezoidal (6-step) mode commutation using Hall sensors
EXECUTION:	Immediate; remains in effect until otherwise commanded
CONDITIONAL TO:	N/A
LIMITATIONS:	Command is not available for any Class 6 motors
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	D-style motors default to MDT commutation mode; M-style motors default to MDC commutation mode
FIRMWARE VERSION:	5.x (D/M); no Class 6
COMBITRONIC:	MDT:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The MDT command enables the motor's trapezoidal commutation mode using Hall sensors (the default mode for D-style motors). Despite the minor inaccuracies that are typical in the mechanical placement of the sensors, this is the most simple and effective method, and it is ready on boot up.

Use status word 6 to see the active commutation mode.

NOTE: MDE, MDS and MDC require angle match before they will take effect. This means the SmartMotor's factory calibration is valid and the index mark of the internal encoder has been seen after startup. The default commutation mode for D-style motors is MDT (see MDT on page 576); the default commutation mode for M-style motors is MDC (see MDC on page 566).

EXAMPLE:

```
MDT          'Set trap mode
             'Will remain in trap mode until commanded otherwise
MV          'Set velocity move
VT=1500000  'Set velocity target
ADT=100     'Set accel/decel target
G           'Start motion
END
```

RELATED COMMANDS:

MDB *Enable TOB Feature (Commutation Mode) (see page 564)*

MDC *Mode Current (Commutation Mode) (see page 566)*
MDE *Mode Enhanced (Commutation Mode) (see page 568)*
MDH *Mode Hybrid (Commutation Mode) (see page 570)*
MDHV *Mode Hybrid Velocity (Commutation Mode) (see page 572)*
MDS *Mode Sine (Commutation Mode) (see page 574)*



Mode Follow, Zero External Counter

APPLICATION:	Motion control
DESCRIPTION:	Reset external encoder to zero, select quadrature mode input
EXECUTION:	Immediate; remains in effect until otherwise commanded
CONDITIONAL TO:	External encoder inputs available, connected to an encoder
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	This is not the motor default; it is best practice to issue MFO before beginning a program that uses quadrature inputs
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	MFO:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:



WARNING: For Class 5 D-series motors, certain special features may override the brake function. In particular, the MFR, MSR, MFO, MSO commands, or any similar feature from a network interface (including CANopen modes of operation: -1, -3, -11), may interfere with a brake assignment to I/O 0 or 1). Therefore, use of I/O 0 or 1 *is not* recommended for the brake in the Class 5 D-series if follow or step modes are used, regardless of SRC setting. For a programming example, refer to MFO on page 578.

The MFO command zeroes the second encoder register (see CTR(enc) on page 380) without changing the current motion mode of the SmartMotor™.

Following MFO, the A and B pins will be interpreted as a quadrature encoder signal. MSO is the opposite input mode.

- On the D-style motor, these are inputs 0 and 1.
- On the M-style motor, these encoder inputs are differential and are labeled separately from general I/O signals.

If the Mode Follow with Ratio (MFR) or the Cam mode does not meet your requirements, you can write your own loop and define a unique relationship between the incoming secondary encoder signal and the motor's position.

MFO is also typically used to take input from a quadrature output selector switch, especially in the context of a user interface. It is not necessary to use the inputs for motion.

EXAMPLE: (Shows use of MFO, MFDIV and MFMUL)

```
EIGN(W,0)    'Make all onboard I/O inputs
ZS           'Clear errors
MFO          'Reset CTR(1)
MFMUL=21     'Multiplier = 21
MFDIV=-10    'Divisor = -10
MFR          'Example: input 100 external encoder counts,
             'resulting motion is -210 counts
G            'Start following external encoder
END
```

RELATED COMMANDS:

R CTR(enc) Counter, Encoder, Step and Direction (see page 380)

MS0 Mode Step, Zero External Counter (see page 616)

MSR Mode Step Ratio (see page 618)



MFA(distance[,m/s])

Mode Follow Ascend

APPLICATION:	Motion control
DESCRIPTION:	Follow mode, ascend ramp;
EXECUTION:	Buffered until a G command is issued or a profiled move restarts through MFSDC
CONDITIONAL TO:	MFR, MSR or MC modes
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	Either encoder counts from encoder input, or internal encoder counts (selectable)
RANGE OF VALUES:	Input: value: 0 to 2147483647 [m/s]: 0 or 1
TYPICAL VALUES:	Input: value: 0 to 2147483647 [m/s]: 0 or 1
DEFAULT VALUE:	MFA(0,0) (no ramp — Follow mode immediately jumps to ratio when G issued)
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	MFA(1000,1):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

NOTE: This command is affected by the SCALEP command. For details, see SCALEP(m,d) on page 726. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

The MFA command sets the ascend ramp to the specified sync ratio from a ratio of zero. It uses the format:

MFA(exp1[,exp2])

where:

- Exp1
Specifies counts — valid values are from 0 to 2147483647. Set to 0 (default) to disable.
- Exp2
(Optional) Specifies the meaning of exp1. Values are: 0 specifies input units (controller units); 1 specifies distance-traveled units (follower units).

When operating in MFR or MSR mode, it is possible to create a motion profile where the ratio of the incoming encoder signal to the motor output motion can be gradually ramped up. This ramp starts when motion is commanded to start with a G command, and will end when the programmed ratio of MFMUL/MFDIV is reached. The rate of increase is controlled by the two parameters to the MFA function. The first argument is a distance (in encoder counts), and the second argument specifies if that distance is in terms of the encoder input or the motors output motion.

In MC mode MFA still functions. However it does not directly affect the motor's output motion. It is a front-end profile between the encoder input and the operation of the Cam table. MFA allows for gradual increase of the rate at which cam points are followed.

For a figure showing use examples of this command, see MFSDC Modes on page 148.

EXAMPLE: (profile driven by an incoming encoder signal)

```
MFMUL=300
MFDIV=100
MFA (300,1)      'Follower moves 300 counts over ascend
MFD (600,1)      'Follower moves 600 counts over descend
MFSLEW (200,1)   'Follower maintains sync ratio for 200 counts
MFR
G
```

EXAMPLE: (Cam program example; uses virtual encoder)

```
CTE (1)          'Erase all EEPROM tables.
CTA (7,4000)     'Create 7-point table at each 4K encoder increment.
CTW (0)          'Add 1st point.
CTW (1000)       'Add 2nd point; go to point 1000 from start.
CTW (3000)       'Add 3rd point; go to point 3000 from start.
CTW (4000)       'Add 4th point; go to point 4000 from start.
CTW (1000)       'Add 5th point; go to point 1000 from start.
CTW (-2000)      'Add 6th point; go to point -2000 from start.
CTW (0)          'Add 7th point; return to starting point.
                 'Table has now been written to EEPROM.
SRC (2)          'Use the virtual encoder.
MCE (0)          'Force linear interpolation.
MCW (1,0)        'Use table 1 from point 0.
MFMUL=1          'Simple 1:1 ratio from virtual encoder.
MFDIV=1          'Simple 1:1 ratio from virtual encoder.
MFA (0) MFD (0)  'Disable virtual encoder ramp-up/
                 'ramp-down sections.
MFSLEW (24000,1) 'Table is 6 segments * 4000 encoder
                 'counts each.
                 'Specify the second argument as a 1 to
                 'force this number as the output total of
                 'the virtual encoder into the cam.
MFSDC (-1,0)     'Disable virtual encoder profile repeat.
MC               'Enter Cam mode.
G               'Begin move.
END
```

RELATED COMMANDS:

MFD(distance[,m/s]) *Mode Follow Descend (see page 585)*

R MFDIV=formula *Mode Follow Divisor (see page 588)*

R MFMUL=formula *Mode Follow Multiplier (see page 598)*

MFR *Mode Follow Ratio (see page 600)*

MFSDC(distance,mode) *Mode Follow, Stall-Dwell-Continue (see page 603)*

MFSLEW(distance[,m/s]) *Mode Follow Slew (see page 605)*

MSR *Mode Step Ratio (see page 618)*

SRC(enc_src) *Source, Follow and/or Cam Encoder (see page 759)*



MFCTP(arg1,arg2)

Mode Follow Control Traverse Point

APPLICATION:	Motion control
DESCRIPTION:	Control information for traverse mode
EXECUTION:	Buffered until a G command is issued
CONDITIONAL TO:	MFR or MSR mode; MFSDC(x,2) mode selected
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	Input: arg1: -1,0,1 arg2: 0,1
TYPICAL VALUES:	Input: arg1: -1,0,1 arg2: 0,1
DEFAULT VALUE:	MFCTP(0,0)
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	MFCTP(0,0):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The MFCTP command provides control information for traverse mode. It allows a several mode selections to be made. It uses the format:

MFCTP(exp1,exp2)

Refer to the next tables for exp1 and exp2 values and descriptions.

- Exp1

Value	Description
-1	When G is issued, continue in the same direction from when the previous traverse move was ended. This direction is indicated by status word 7, bit 13. This state is not reset by an X or an OFF.
0	(Default at power up) When G is issued, initially traverse toward higher bound.
1	When G is issued, initially traverse toward lower bound.

- Exp2

Value	Description
0	(Default at power up) The RPC(2) frame of reference is frozen when the servo is off (OFF, MTB, MT).
1	The RPC(2) frame of reference will be updated with shaft motion when the servo is off (OFF, MTB, MT). This is a special setting to ensure backward compatibility with existing applications that may use the RPC(2) frame of reference.

EXAMPLE: (Single-trajectory traverse winding application)

```

          ' *** User does some type of homing before this. ***
SRC(2)      '*** For demo controller signal. ***
'Typical applications would use SRC(1) for encoder input.

MFCTP(0,1)  'Start traverse state in "normal" direction;
           'activate update of RCP(2) when servo is off.

MFL(1000,1) 'Lower-end ramp.
MFH(1000,1) 'Higher-end ramp.
MFLTP=-1000 'Lower traverse point.
MFHTP=1000  'Higher traverse point.
MFMUL=1     'Ratio (default is 1).
MFDIV=1     'Ratio (default is 1).
MFSDC(4000,2) 'Dwell for 4000 counts, 2 is active traverse mode.
MFR         'Enable Follow mode at specified ratio.
G           'Begin move.

```

RELATED COMMANDS:

R MFDIV=formula *Mode Follow Divisor (see page 588)*
MFH(distance[,m/s]) *Mode Follow, High Ascend/Descend Rate (see page 590)*
R MFHTP=formula *Mode Follow, High Traverse Point (see page 592)*
MFL(distance[,m/s]) *Mode Follow, Low Ascend/Descend Rate (see page 594)*
R MFLTP=formula *Mode Follow, Low Traverse Point (see page 596)*
R MFMUL=formula *Mode Follow Multiplier (see page 598)*
MFR *Mode Follow Ratio (see page 600)*
MFSDC(distance,mode) *Mode Follow, Stall-Dwell-Continue (see page 603)*
MSR *Mode Step Ratio (see page 618)*



MFD(distance[,m/s])

Mode Follow Descend

APPLICATION:	Motion control
DESCRIPTION:	Follow mode, descend ramp; default is zero (off)
EXECUTION:	Buffered until a G command is issued, or a profiled move restarts through MFSDC
CONDITIONAL TO:	MFR, MSR or MC modes
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	Encoder counts from encoder input or internal counter (selectable)
RANGE OF VALUES:	Input: value: 0 to 2147483647 [m/s]: 0 or 1
TYPICAL VALUES:	Input: value: 0 to 2147483647 [m/s]: 0 or 1
DEFAULT VALUE:	MFD(0,0) (no ramp — immediately change to zero ratio at end of slew or X command)
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	MFD(1000,1):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

NOTE: This command is affected by the SCALEP command. For details, see SCALEP(m,d) on page 726. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

The MFD command specifies the descend ramp from the current ratio to a ratio of 0. It uses the format:

MFD(exp1[,exp2])

where:

- Exp1
Specifies counts — valid values from 0 to 2147483647. Set to 0 (default) to disable.
- Exp2
(Optional) Specifies the meaning of exp1. Values are: 0 specifies input units (controller units); 1 specifies distance-traveled units (follower units).

When operating in MFR or MSR mode, it is possible to create a motion profile where the ratio of the incoming encoder signal to the motor output motion can be gradually ramped down. This ramp starts when an X (decelerate to stop) is commanded, or a pre-programmed distance was traveled as specified by MFSLEW. The ramp will end when the ratio reaches 0. The rate of decrease is controlled by the two parameters to the MFD function. The first argument is a distance (in encoder counts), and the second argument specifies if that distance is in terms of the encoder input or the motors output motion.

In MC mode, MFD still functions. However, it does not directly affect the output motor motion. It is a front-end profile between the encoder input and the operation of the Cam table. MFD can be used to gradually decrease the rate at which cam points are followed.

For a figure showing use examples of this command, see MFSDC Modes on page 148.

EXAMPLE: (profile driven by an incoming encoder signal)

```
MFMUL=300
MFDIV=100
MFA(300,1)      'Follower moves 300 counts over ascend
MFD(600,1)      'Follower moves 600 counts over descend
MFSLEW(200,1)   'Follower maintains sync ratio for 200 counts
MFR
G
```

EXAMPLE: (Cam program example; uses virtual encoder)

```
CTE(1)          'Erase all EEPROM tables.
CTA(7,4000)     'Create 7-point table at each 4K encoder increment.
CTW(0)         'Add 1st point.
CTW(1000)      'Add 2nd point; go to point 1000 from start.
CTW(3000)      'Add 3rd point; go to point 3000 from start.
CTW(4000)      'Add 4th point; go to point 4000 from start.
CTW(1000)      'Add 5th point; go to point 1000 from start.
CTW(-2000)     'Add 6th point; go to point -2000 from start.
CTW(0)         'Add 7th point; return to starting point.
               'Table has now been written to EEPROM.
SRC(2)         'Use the virtual encoder.
MCE(0)         'Force linear interpolation.
MCW(1,0)       'Use table 1 from point 0.
MFMUL=1        'Simple 1:1 ratio from virtual encoder.
MFDIV=1        'Simple 1:1 ratio from virtual encoder.
MFA(0) MFD(0)  'Disable virtual encoder ramp-up/
               'ramp-down sections.
MFSLEW(24000,1) 'Table is 6 segments * 4000 encoder
               'counts each.
               'Specify the second argument as a 1 to
               'force this number as the output total of
               'the virtual encoder into the cam.
MFSDC(-1,0)    'Disable virtual encoder profile repeat.
MC             'Enter Cam mode.
G             'Begin move.
END
```

RELATED COMMANDS:

MFA(distance[,m/s]) *Mode Follow Ascend (see page 580)*

R MFDIV=formula *Mode Follow Divisor (see page 588)*

R MFMUL=formula *Mode Follow Multiplier (see page 598)*

MFR *Mode Follow Ratio (see page 600)*

MFSDC(distance,mode) *Mode Follow, Stall-Dwell-Continue (see page 603)*

MFSLEW(distance[,m/s]) *Mode Follow Slew (see page 605)*

MSR *Mode Step Ratio (see page 618)*



MFDIV=formula

Mode Follow Divisor

APPLICATION:	Motion control
DESCRIPTION:	Divisor for external encoder mode follow with ratio MFMUL/MFDIV
EXECUTION:	Buffered until a G command is issued, or a profiled move restarts through MFSDC
CONDITIONAL TO:	MFR, MSR or MC modes
LIMITATIONS:	Class 6 D-style cannot output an encoder signal
READ/REPORT:	RMFDIV
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	-32767 to -1 and 1 to 32767 (0 excluded)
TYPICAL VALUES:	-32767 to -1 and 1 to 32767 (0 excluded)
DEFAULT VALUE:	1
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	MFDIV:3=1234, a=MFDIV:3, RMFDIV:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The MFDIV command specifies the divisor for use with the MFR, MSR or MC command. For more details about how this ratio is applied, see MFR on page 600.

MFMUL/MFDIV specifies the ratio for the MFR or MSR mode. The MC mode is also affected. To use the MFR or MSR command, you will need to define the specific relationship (ratio) of the encoder count input to outgoing requested encoder counts of motion.

Both MFMUL and MFDIV may be positive or negative; this controls the resulting direction of shaft rotation.

NOTE: MFMUL and MFDIV are each set to 1 by default (1 to 1 ratio). Therefore, it is only necessary to specify either or both if you want to change the default ratio.

NOTE: MFR and MSR are used to enable the desired Electronic Gearing mode (Mode Follow or Mode Step, respectively). They are not needed to enable a change to the ratio—the "G" command will do that.

For Class 6 M-style motors, when the internal encoder is directed as an output and received into another motor for the purposes of Follow mode, the resolution at the receiving motor will be 4096 instead of 4000. MFMUL and MFDIV will need to compensate accordingly. For more notes, see "Encoder Output" in ENCD(in_out) on page 437.

EXAMPLE: (Shows use of MF0, MFDIV and MFMUL)

```
EIGN(W,0)    'Make all onboard I/O inputs
ZS           'Clear errors
MF0          'Reset CTR(1)
MFMUL=21     'Multiplier = 21
MFDIV=-10    'Divisor = -10
MFR          'Example: input 100 external encoder counts,
             'resulting motion is -210 counts
G            'Start following external encoder
END
```

RELATED COMMANDS:

MC *Mode Cam (Electronic Camming) (see page 555)*
R MFMUL=formula *Mode Follow Multiplier (see page 598)*
MFR *Mode Follow Ratio (see page 600)*
MSR *Mode Step Ratio (see page 618)*



MFH(distance[,m/s])

Mode Follow, High Ascend/Descend Rate

APPLICATION:	Motion control
DESCRIPTION:	Sets the ramp at the high end of the traverse
EXECUTION:	Buffered until a G command is issued or either traverse point after reversal
CONDITIONAL TO:	MFR or MSR mode; MFSDC(x,2) mode selected
LIMITATIONS:	If the combined ramp follower distances of MFL and MFH exceed the traverse distance, then the ramp rate(s) are increased as required.
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	Either encoder counts from encoder input, or internal encoder counts (selectable)
RANGE OF VALUES:	Input: value: 0 to 2147483647 [m/s]: 0 or 1
TYPICAL VALUES:	Input: value: 0 to 2147483647 [m/s]: 0 or 1
DEFAULT VALUE:	MFH(0,0) (immediately change ratio at high end of traverse)
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	MFH(1000,1):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

NOTE: This command is affected by the SCALEP command. For details, see SCALEP(m,d) on page 726. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

The MFH command sets the ramp at the high end of the traverse. It uses the format:

```
MFH(exp1[,exp2])
```

where:

- Exp1
Specifies counts — valid values from 0 to 2147483647. Set to 0 (default) to disable.
- Exp2
(Optional) Specifies the meaning of exp1. Values of exp2: 0 specifies input units (controller units); 1 specifies distance-traveled units (follower units).

MFH behaves similar to MFA and MFD, where a ramp is defined during the follow profile. However, MFH has a slightly different application. It is only used in absolute traverse mode, MFSDC(x,2). MFH

defines the descent and ascent ramps at the high end of the absolute traverse. This is an important difference because the other ramps, MFA and MFD, are associated with increasing and decreasing ramps, respectively. Whereas, MFH is associated with the higher physical position and orientation of the ramp in the absolute traverse profile.

The MFH command can be set at any time, but the value is buffered and accepted when the absolute traverse mode reaches a speed of 0. Typically, this is when starting, or when a traverse point is reached and the motor has ramped down to a stop (using the previous ramp rate).

NOTE: This can cause an asymmetry because the descent and ascent ramps at a particular traverse point will not be the same.

Depending on the application, this could be a problem. The best way to avoid an asymmetry is to set the ramp when heading away from the traverse point it affects. In other words, only set MFH when the motor is traveling in a negative direction away from the high traverse point.

NOTE: MFA is not applied in the absolute traverse mode. MFD is only applied in the absolute traverse mode if an X (decelerate to stop) command is issued.

For a figure showing use examples of this command, see MFSDC Modes on page 148.

EXAMPLE: (Single-trajectory traverse winding application)

```

SRC(2)      ' *** User does some type of homing before this. ***
            '*** For demo controller signal. ***
'Typical applications would use SRC(1) for encoder input.

MFCTP(0,1)  'Start traverse state in "normal" direction;
            'activate update of RCP(2) when servo is off.
MFL(1000,1) 'Lower-end ramp.
MFH(1000,1) 'Higher-end ramp.
MFLTP=-1000 'Lower traverse point.
MFHTP=1000  'Higher traverse point.
MFMUL=1     'Ratio (default is 1).
MFDIV=1     'Ratio (default is 1).
MFSDC(4000,2) 'Dwell for 4000 counts, 2 is active traverse mode.
MFR         'Enable Follow mode at specified ratio.
G           'Begin move.

```

RELATED COMMANDS:

MFCTP(arg1,arg2) *Mode Follow Control Traverse Point (see page 583)*

R MFDIV=formula *Mode Follow Divisor (see page 588)*

R MFHTP=formula *Mode Follow, High Traverse Point (see page 592)*

MFL(distance[,m/s]) *Mode Follow, Low Ascend/Descend Rate (see page 594)*

R MFLTP=formula *Mode Follow, Low Traverse Point (see page 596)*

R MFMUL=formula *Mode Follow Multiplier (see page 598)*

MFR *Mode Follow Ratio (see page 600)*

MSR *Mode Step Ratio (see page 618)*

MFSDC(distance,mode) *Mode Follow, Stall-Dwell-Continue (see page 603)*



MFHTP=formula

Mode Follow, High Traverse Point

APPLICATION:	Motion control
DESCRIPTION:	High traverse point
EXECUTION:	Buffered until a G command is issued, or the opposite traverse point
CONDITIONAL TO:	MFR or MSR mode; MFSDC(x,2) mode selected
LIMITATIONS:	MFHTP-MFLTP must be in the range 0 to 2147483647
READ/REPORT:	RMFHTP
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	Followed encoder counts
RANGE OF VALUES:	-2147483648 to 2147483647
TYPICAL VALUES:	0 to 1000000
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	MFHTP:3=1234, a=MFHTP:3, RMFHTP:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

NOTE: This command is affected by the SCALEP command. For details, see SCALEP(m,d) on page 726. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

When in absolute traverse mode, MFHTP=formula sets the high traverse point in terms of the follower motor's position. The value may be anywhere in the range from -2147483648 to 2147483647. However, it should be higher than the low traverse point specified by MFLTP.

NOTE: The distance between MFLTP and MFHTP must be in the range from 0 to 2147483647; MFHTP must be the higher value.

The MFHTP traverse point can be set at any time. However, it is buffered and accepted into the motion profile when the motion profile reaches the opposite traverse point (MFLTP).

For a figure showing use examples of this command, see MFSDC Modes on page 148.

EXAMPLE: (Single-trajectory traverse winding application)

```

          ' *** User does some type of homing before this. ***
SRC(2)    '*** For demo controller signal. ***
'Typical applications would use SRC(1) for encoder input.

MFCTP(0,1) 'Start traverse state in "normal" direction;
          'activate update of RCP(2) when servo is off.
MFL(1000,1) 'Lower-end ramp.
MFH(1000,1) 'Higher-end ramp.
MFLTP=-1000 'Lower traverse point.
MFHTP=1000 'Higher traverse point.
MFMUL=1 'Ratio (default is 1).
MFDIV=1 'Ratio (default is 1).
MFSDC(4000,2) 'Dwell for 4000 counts, 2 is active traverse mode.
MFR 'Enable Follow mode at specified ratio.
G 'Begin move.

```

RELATED COMMANDS:

MFCTP(arg1,arg2) *Mode Follow Control Traverse Point (see page 583)*
 R MFDIV=formula *Mode Follow Divisor (see page 588)*
 MFH(distance[,m/s]) *Mode Follow, High Ascend/Descend Rate (see page 590)*
 MFL(distance[,m/s]) *Mode Follow, Low Ascend/Descend Rate (see page 594)*
 R MFLTP=formula *Mode Follow, Low Traverse Point (see page 596)*
 R MFMUL=formula *Mode Follow Multiplier (see page 598)*
 MFR *Mode Follow Ratio (see page 600)*
 MFSDC(distance,mode) *Mode Follow, Stall-Dwell-Continue (see page 603)*
 MSR *Mode Step Ratio (see page 618)*



MFL(distance[,m/s])

Mode Follow, Low Ascend/Descend Rate

APPLICATION:	Motion control
DESCRIPTION:	Sets the ramp at the low end of the traverse
EXECUTION:	Buffered until a G command is issued or either traverse point after reversal
CONDITIONAL TO:	MFR or MSR mode; MFSDC(x,2) mode selected
LIMITATIONS:	If the combined ramp follower distances of MFL and MFH exceed the traverse distance, then the ramp rate(s) are increased as required.
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	Encoder counts from encoder input or internal counter (selectable)
RANGE OF VALUES:	Input: value: 0 to 2147483647 [m/s]: 0 or 1
TYPICAL VALUES:	Input: value: 0 to 2147483647 [m/s]: 0 or 1
DEFAULT VALUE:	MFL(0,0) (immediately change ratio at low end of traverse)
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	MFL(1000,1):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

NOTE: This command is affected by the SCALEP command. For details, see SCALEP(m,d) on page 726. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

The MFL command specifies the ramp at the low end of the traverse. It uses the format:

```
MFL(exp1[,exp2])
```

where:

- Exp1
Specifies counts — valid value from 0 to 2147483647. Set to 0 (default) to disable.
- Exp2
(Optional) specifies the meaning of exp1. Values are: 0 specifies input units (controller units); 1 specifies distance-traveled units (follower units).

MFL behaves similar to MFA and MFD — a ramp is defined during the follow profile. However, MFL is only used in absolute traverse mode, MFSDC(x,2). MFL defines the descent and ascent ramps at the

lower end of the absolute traverse. This is an important difference because the other ramps, MFA and MFD, are associated with increasing and decreasing ramps, respectively. Whereas, MFL is associated with the lower physical position and orientation of the ramp in the absolute traverse profile.

The MFL command can be set at any time, but the value is buffered and accepted when the absolute traverse mode reaches a speed of 0. Typically, this is when starting, or when a traverse point is reached and the motor has ramped down to a stop (using the previous ramp rate).

NOTE: This can cause an asymmetry because the descent and ascent ramps at a particular traverse point will not be the same.

Depending on the application, this could be a problem. The best way to avoid an asymmetry is to set the ramp when heading away from the traverse point it affects. In other words, only set MFL when the motor is traveling in a positive direction away from the low traverse point.

NOTE: MFA is not applied in the absolute traverse mode. MFD is only applied in the absolute traverse mode if an X (decelerate to stop) command is issued.

For a figure showing use examples of this command, see MFSDC Modes on page 148.

EXAMPLE: (Single-trajectory traverse winding application)

```

SRC(2)          ' *** User does some type of homing before this. ***
                '*** For demo controller signal. ***
                'Typical applications would use SRC(1) for encoder input.

MFCTP(0,1)      'Start traverse state in "normal" direction;
                'activate update of RCP(2) when servo is off.

MFL(1000,1)     'Lower-end ramp.
MFH(1000,1)     'Higher-end ramp.
MFLTP=-1000     'Lower traverse point.
MFHTP=1000     'Higher traverse point.
MFMUL=1         'Ratio (default is 1).
MFDIV=1         'Ratio (default is 1).
MFSDC(4000,2)  'Dwell for 4000 counts, 2 is active traverse mode.
MFR             'Enable Follow mode at specified ratio.
G              'Begin move.

```

RELATED COMMANDS:

MFCTP(arg1,arg2) *Mode Follow Control Traverse Point (see page 583)*

R MFDIV=formula *Mode Follow Divisor (see page 588)*

MFH(distance[,m/s]) *Mode Follow, High Ascend/Descend Rate (see page 590)*

R MFHTP=formula *Mode Follow, High Traverse Point (see page 592)*

R MFLTP=formula *Mode Follow, Low Traverse Point (see page 596)*

R MFMUL=formula *Mode Follow Multiplier (see page 598)*

MFR *Mode Follow Ratio (see page 600)*

MFSDC(distance,mode) *Mode Follow, Stall-Dwell-Continue (see page 603)*

MSR *Mode Step Ratio (see page 618)*



APPLICATION:	Motion control
DESCRIPTION:	Low traverse point
EXECUTION:	Buffered until a G command is issued, or the opposite traverse point
CONDITIONAL TO:	MFR or MSR mode; MFSDC(x,2) mode selected
LIMITATIONS:	MFHTP-MFLTP must be in the range 0 to 2147483647
READ/REPORT:	RMFLTP
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	Followed encoder counts
RANGE OF VALUES:	-2147483648 to 2147483647
TYPICAL VALUES:	0 to 1000000
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	MFLTP:3=1234, a=MFLTP:3, RMFLTP:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

NOTE: This command is affected by the SCALEP command. For details, see SCALEP(m,d) on page 726. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

When in absolute traverse mode, MFLTP=formula sets the lower traverse point in terms of the follower motor's position. The value may be anywhere in the range from -2147483648 to 2147483647. However, it should be lower than the high traverse point specified by MFHTP.

NOTE: The distance between MFLTP and MFHTP must be in the range from 0 to 2147483647; MFHTP must be the higher value.

The MFLTP traverse point can be set at any time. However, it is buffered and accepted into the motion profile when the motion profile reaches the opposite traverse point (MFHTP).

For a figure showing use examples of this command, see MFSDC Modes on page 148.

EXAMPLE: (Single-trajectory traverse winding application)

```

          ' *** User does some type of homing before this. ***
SRC(2)    '*** For demo controller signal. ***
'Typical applications would use SRC(1) for encoder input.

MFCTP(0,1) 'Start traverse state in "normal" direction;
          'activate update of RCP(2) when servo is off.
MFL(1000,1) 'Lower-end ramp.
MFH(1000,1) 'Higher-end ramp.
MFLTP=-1000 'Lower traverse point.
MFHTP=1000 'Higher traverse point.
MFMUL=1 'Ratio (default is 1).
MFDIV=1 'Ratio (default is 1).
MFSDC(4000,2) 'Dwell for 4000 counts, 2 is active traverse mode.
MFR 'Enable Follow mode at specified ratio.
G 'Begin move.

```

RELATED COMMANDS:

MFCTP(arg1,arg2) *Mode Follow Control Traverse Point (see page 583)*
 R MFDIV=formula *Mode Follow Divisor (see page 588)*
 MFH(distance[,m/s]) *Mode Follow, High Ascend/Descend Rate (see page 590)*
 R MFHTP=formula *Mode Follow, High Traverse Point (see page 592)*
 MFL(distance[,m/s]) *Mode Follow, Low Ascend/Descend Rate (see page 594)*
 R MFMUL=formula *Mode Follow Multiplier (see page 598)*
 MFR *Mode Follow Ratio (see page 600)*
 MFSDC(distance,mode) *Mode Follow, Stall-Dwell-Continue (see page 603)*
 MSR *Mode Step Ratio (see page 618)*



MFMUL=formula

Mode Follow Multiplier

APPLICATION:	Motion control
DESCRIPTION:	Multiplier for external encoder mode follow with ratio MFMUL/MFDIV
EXECUTION:	Buffered until a G command is issued or a profiled move restarts through MFSDC
CONDITIONAL TO:	MFR, MSR or MC mode
LIMITATIONS:	Class 6 D-style cannot output an encoder signal
READ/REPORT:	RMFMUL
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	-32767 to 32767
TYPICAL VALUES:	-32767 to 32767
DEFAULT VALUE:	1
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	MFMUL:3=1234, a=MFMUL:3, RMFMUL:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The MFMUL command specifies the multiplier for use with the MFR, MSR or MC command. For more details about how this ratio is applied, see the MSR on page 618.

MFMUL/MFDIV specifies the ratio for the MFR or MSR mode. The MC mode is also affected. To use the MFR or MSR command, you will need to define the specific relationship (ratio) of the encoder count input to outgoing requested encoder counts of motion.

Both MFMUL and MFDIV may be positive or negative; this controls the resulting direction of shaft rotation.

NOTE: MFMUL and MFDIV are each set to 1 by default (1 to 1 ratio). Therefore, it is only necessary to specify either or both if you want to change the default ratio.

NOTE: MFR and MSR are used to enable the desired Electronic Gearing mode (Mode Follow or Mode Step, respectively). They are not needed to enable a change to the ratio—the "G" command will do that.

For Class 6 M-style motors, when the internal encoder is directed as an output and received into another motor for the purposes of Follow mode, the resolution at the receiving motor will be 4096 instead of 4000. MFMUL and MFDIV will need to compensate accordingly. For more notes, see "Encoder Output" in ENCD(in_out) on page 437.

EXAMPLE: (Shows use of MF0, MFDIV and MFMUL)

```
EIGN(W,0)    'Make all onboard I/O inputs
ZS           'Clear errors
MF0          'Reset CTR(1)
MFMUL=21     'Multiplier = 21
MFDIV=-10    'Divisor = -10
MFR          'Example: input 100 external encoder counts,
             'resulting motion is -210 counts
G            'Start following external encoder
END
```

RELATED COMMANDS:

MC *Mode Cam (Electronic Camming) (see page 555)*
R MFDIV=formula *Mode Follow Divisor (see page 588)*
MFR *Mode Follow Ratio (see page 600)*
MSR *Mode Step Ratio (see page 618)*



APPLICATION:	Motion control
DESCRIPTION:	Request mode follow with ratio
EXECUTION:	Buffered until a G command is issued
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	MP
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	MFR:3 or MFR(0):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:



WARNING: For Class 5 D-series motors, certain special features may override the brake function. In particular, the MFR, MSR, MFO, MSO commands, or any similar feature from a network interface (including CANopen modes of operation: -1, -3, -11), may interfere with a brake assignment to I/O 0 or 1). Therefore, use of I/O 0 or 1 *is not* recommended for the brake in the Class 5 D-series if follow or step modes are used, regardless of SRC setting. For a programming example, refer to MFR on page 600.

The MFR command is used to implement a fractional relationship between an incoming secondary encoder signal and the SmartMotor™ internal shaft position, which is represented by the primary internal encoder count. The fractional relationship is defined by the user-set ratio of MFMUL to MFDIV. The motor will move in proportion to the incoming encoder signal.

NOTE: MFMUL and MFDIV are each set to 1 by default (1 to 1 ratio). Therefore, it is only necessary to specify either or both if you want to change the default ratio.

NOTE: MFR and MSR are used to enable the desired Electronic Gearing mode (Mode Follow or Mode Step, respectively). They are not needed to enable a change to the ratio—the "G" command will do that.

The encoder input is configured as a quadrature input. If a step-and-direction operation is desired, then select MSR instead of MFR. Both commands operate the same with the exception of the input signal type.

Either the external encoder can be selected, or an internal time-base can be selected. For details, see SRC(enc_src) on page 759.

The motion can be gradually started and stopped even if the encoder source is constantly running. For instance, if a conveyor is constantly running, the motor can be ramped up to speed and then ramped down from speed by command or programmed distance. This allows for operations on moving products, such as labeling, stamping or transfers from one conveyor to another. For more information, see MFA (distance[,m/s]) on page 580, MFD(distance[,m/s]) on page 585, and MFSLEW(distance[,m/s]) on page 605.

There are automatic modes that allow for repeating start-stop cycles or for traversing applications like winders and cut-to-length material. For more information, see MFSDC(distance,mode) on page 603.

To use MFR, if something other than the default 1:1 ratio is desired, you will need to define the relationship (ratio) of the external encoder input to shaft position, which is represented by the primary internal encoder count. Both MFMUL and MFDIV may be positive or negative — this controls the resulting direction of shaft rotation.

The MFR command and then a G command will immediately turn on the servo. The servo-off flag (Bo) is set to 0; the trajectory flag (Bt) is set to 1. The motion is restricted by the current value of EL. The motion is also subject to the currently defined activity of the limit switches.

The fractional ratio of input encoder to motor motion is maintained with a continuing remainder. This means that the ratio can run continuously without concern about loss of accuracy. Any ratio that can be expressed with the integers MFMUL and MFDIV, and within their range limits, can be maintained accurately. For example, the ratio 1:10 or 2:3 can be maintained accurately. Keep in mind that the values of MFMUL and MFDIV can be creatively selected to give the desired ratio. For instance, MFMUL=10 and MFDIV=395 gives a reduction ratio of 39.5.

NOTE: The only method of inputting a ratio to the SmartMotor is with the MFMUL and MFDIV commands, which require integers within a certain range.

Ratios that are *irrational* — a mathematical definition that means it cannot be represented by a ratio of integers — cannot be tracked continuously. This is a fundamental mathematical principal and not a limitation of the SmartMotor. For example, converting degrees to radians, or diameter to circumference, is considered irrational because an integer ratio cannot express the precision required to operate continuously. Such a ratio would only be accurate in a limited range of motion. Depending on the application, this could be a problem. Therefore, it is the responsibility of the system designer to be aware of this.

For a figure showing use examples of this command, see MFSDC Modes on page 148.

Programming Note

NOTE: When using the EOBK command in programs with the MFR command, be aware of the information below.

In situations where EOBK(0) is used before MFR, note that MFR interferes with I/O 0 and 1. This defeats, for example, EOBK(0), from working properly when it is placed before MFR.

To program this correctly:

- Choose an output value for EOBK that is something other than 0 or 1, e.g., EOBK(2):

```
EOBK (2)
...
MFR
G
```

OR

- If EOBK(0) (or EOBK(1)) must be used, be sure to reissue EOBK *after* MFR but *before* the G command:

```
MFR
EOBK (0)
G
```

EXAMPLE: (phase offset adjustment)

In some applications, it may be necessary to introduce a phase shift to achieve proper alignment during MFR following. To perform this shift, configure trajectory 1 to execute a position move.

```
EIGN(W,0)      'Make all onboard I/O inputs
ZS            'Clear errors
MF0          'Reset CTR(1)
MFMUL=21     'Multiplier = 21
MFDIV=-10    'Divisor = -10
MFR          'Example: input 100 external encoder counts,
             'resulting motion is -210 counts
MP(1)        'Position mode in trajectory 1
             'Also, keep MFR active
PRT=0        'No phase shift
G(2)         'Start following
             'Implementing phase adjust:
PRT=500      ' Set relative distance
VT=5000     ' Set velocity target
ADT=100     ' Set accel/decel
G(1)         ' Start phase adjust
END
```

RELATED COMMANDS:

R CTR(enc) *Counter, Encoder, Step and Direction (see page 380)*
 G *Start Motion (GO) (see page 473)*
 MFA(distance[,m/s]) *Mode Follow Ascend (see page 580)*
 MFCTP(arg1,arg2) *Mode Follow Control Traverse Point (see page 583)*
 MFD(distance[,m/s]) *Mode Follow Descend (see page 585)*
 R MFDIV=formula *Mode Follow Divisor (see page 588)*
 MFH(distance[,m/s]) *Mode Follow, High Ascend/Descend Rate (see page 590)*
 R MFHTP=formula *Mode Follow, High Traverse Point (see page 592)*
 MFL(distance[,m/s]) *Mode Follow, Low Ascend/Descend Rate (see page 594)*
 R MFLTP=formula *Mode Follow, Low Traverse Point (see page 596)*
 R MFMUL=formula *Mode Follow Multiplier (see page 598)*
 MFSDC(distance,mode) *Mode Follow, Stall-Dwell-Continue (see page 603)*
 MFSLEW(distance[,m/s]) *Mode Follow Slew (see page 605)*
 MSR *Mode Step Ratio (see page 618)*
 SRC(enc_src) *Source, Follow and/or Cam Encoder (see page 759)*



MFSDC(distance,mode)

Mode Follow, Stall-Dwell-Continue

APPLICATION:	Motion control
DESCRIPTION:	Follow mode stall-dwell-continue; controls dwell and repeat of the follow profile
EXECUTION:	Buffered until a G command is issued, or a profiled move restarts through MFSDC
CONDITIONAL TO:	MFR, MSR, or MC modes; MFSLEW must not be disabled
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	Encoder counts from encoder input
RANGE OF VALUES:	Input: value: -1 to 2147483647 mode: 0, 1, 2
TYPICAL VALUES:	Input: value: -1 to 2147483647 mode: 0, 1, 2
DEFAULT VALUE:	MFSDC(-1,0) (disabled — does not repeat)
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	MFSDC(1000,1):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The MFSDC command sets the Follow mode stall-dwell-continue, which is used to control the dwell and repeat of the follow profile. It uses the format:

```
MFSDC(exp1,exp2)
```

where:

- Exp1
Values from 0 to 2147483647 to specify the number of controller counts the follower dwells at zero ratio. Set to -1 (default) to disable. When disabled, Follow Mode will not restart automatically.
- Exp2
Values are: 0 to repeat the gearing profile in the same direction; 1 to repeat the gearing profile in the opposite direction; 2 for the improved traverse mode that uses absolute position targets. A setting of 0 is typical for feeding labels in label applications, and a setting of 1 or 2 is typical for traverse-and-takeup spool winding applications.

In MFR, MSR or MC mode, MFSDC is useful for creating a delay and restarting automatically after the MFSLEW distance. The controller encoder input will continue forward while the motor's output motion remains stationary. After the specified number of encoder input counts, the motor will continue motion again.

For additional details and figures, see MFSDC(distance,mode) on page 147.

EXAMPLE: (Cam program example; uses virtual encoder)

```

CTE (1)           'Erase all EEPROM tables.
CTA (7,4000)      'Create 7-point table at each 4K encoder increment.
CTW (0)           'Add 1st point.
CTW (1000)        'Add 2nd point; go to point 1000 from start.
CTW (3000)        'Add 3rd point; go to point 3000 from start.
CTW (4000)        'Add 4th point; go to point 4000 from start.
CTW (1000)        'Add 5th point; go to point 1000 from start.
CTW (-2000)       'Add 6th point; go to point -2000 from start.
CTW (0)           'Add 7th point; return to starting point.
                  'Table has now been written to EEPROM.
SRC (2)           'Use the virtual encoder.
MCE (0)           'Force linear interpolation.
MCW (1,0)         'Use table 1 from point 0.
MFMUL=1           'Simple 1:1 ratio from virtual encoder.
MFDIV=1           'Simple 1:1 ratio from virtual encoder.
MFA (0) MFD (0)   'Disable virtual encoder ramp-up/
                  'ramp-down sections.
MFSLEW (24000,1) 'Table is 6 segments * 4000 encoder
                  'counts each.
                  'Specify the second argument as a 1 to
                  'force this number as the output total of
                  'the virtual encoder into the cam.
MFSDC (-1,0)      'Disable virtual encoder profile repeat.
MC                'Enter Cam mode.
G                'Begin move.
END

```

RELATED COMMANDS:

MFA(distance[,m/s]) *Mode Follow Ascend (see page 580)*
MFD(distance[,m/s]) *Mode Follow Descend (see page 585)*
R MFDIV=formula *Mode Follow Divisor (see page 588)*
R MFMUL=formula *Mode Follow Multiplier (see page 598)*
MFR *Mode Follow Ratio (see page 600)*
MFSLEW(distance[,m/s]) *Mode Follow Slew (see page 605)*
MSR *Mode Step Ratio (see page 618)*



MFSLEW(distance[,m/s])

Mode Follow Slew

APPLICATION:	Motion control
DESCRIPTION:	Follow mode slew at ratio for a fixed distance
EXECUTION:	Buffered until a G command is issued, or a profiled move restarts through MFSDC
CONDITIONAL TO:	MFR, MSR or MC modes
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	Encoder counts from encoder input or internal counter (selectable)
RANGE OF VALUES:	Input: value: -1 to 2147483647 [m/s]: 0 or 1
TYPICAL VALUES:	Input: value: -1 to 2147483647 [m/s]: 0 or 1
DEFAULT VALUE:	MFSLEW(-1,0) (disabled — runs endlessly when slew section is reached.)
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	MFSLEW(1000,1):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

NOTE: This command is affected by the SCALEP command. For details, see SCALEP(m,d) on page 726. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

The MFSLEW command sets the Follow mode slew at ratio for a fixed distance. It uses the format:

```
MFSLEW(distance[,m/s])
```

where:

- distance
Specifies counts — valid values from -1 to 2147483647. Set to -1 (default) to disable. When disabled, Follow mode continuously runs at ratio. A distance of 0 is acceptable. It produces a motion where only the MFA and MFD ramps form a triangular profile.
- m/s
(Optional) Specifies the meaning of exp1. Values are: 0 for designating input units (controller units); 1 for designating distance traveled (follower units).

When operating in MFR or MSR mode, it is possible to create a motion profile where the motor follows the encoder only for a specified distance — even if the encoder continues on. MFSLEW specifies this distance in terms of the incoming encoder or the motor's outgoing motion.

In MC mode, this has special importance. It is possible to set the length of MFSLEW equal to the controller length of the Cam table. For example, a cam of 10 points (9 segments) with a segment length of 100 will have a total length of 900 counts. In this example, by setting MFSLEW(900,1), exactly one pass through the Cam table can be selected. In combination with the MFSDC(0,0) command, an automatic repeat is selected. This allows for certain parameters like MCMUL/MCDIV, MFSLEW and MFMUL/MFDIV to be updated synchronously with the end of the Cam table.

For a figure showing use examples of this command, see MFSDC Modes on page 148.

EXAMPLE: (profile driven by an incoming encoder signal)

```
MFMUL=300
MFDIV=100
MFA(300,1)      'Follower moves 300 counts over ascend
MFD(600,1)      'Follower moves 600 counts over descend
MFSLEW(200,1)   'Follower maintains sync ratio for 200 counts
MFR
G
```

RELATED COMMANDS:

MFA(distance[,m/s]) *Mode Follow Ascend (see page 580)*

MFD(distance[,m/s]) *Mode Follow Descend (see page 585)*

R MFDIV=formula *Mode Follow Divisor (see page 588)*

R MFMUL=formula *Mode Follow Multiplier (see page 598)*

MFR *Mode Follow Ratio (see page 600)*

MFSDC(distance,mode) *Mode Follow, Stall-Dwell-Continue (see page 603)*

MSR *Mode Step Ratio (see page 618)*



APPLICATION:	Motion control
DESCRIPTION:	Request homing mode
EXECUTION:	Buffered until a G command is issued
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	6.x (D/M); no Class 5
COMBITRONIC:	MH:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The MH (Mode, Homing) command is used to set the SmartMotor to homing mode. MH can be sent directly to the motor as a serial command or entered as part of a user program.

There are no parameters for this command — it simply requests the homing mode for the motor on the next G command.

For more details on SmartMotor homing operations, see the *SmartMotor Homing Procedures and Methods Application Note*.

EXAMPLE:

Enter these commands in the SMI software Terminal window:

```
MH      'Set the motor mode to homing
G      'Start the homing operation
```

RELATED COMMANDS:

R HM_ADT=formula *Homing Accel/Decel Target (see page 491)*

R HM_MTHD=formula *Homing Method (see page 492)*

R HM_OSET=formula *Homing Offset (see page 496)*

R HM_VTS=formula *Homing Velocity Target to Switch (see page 498)*

R HM_VTZ=formula *Homing Velocity Target to Zero (see page 500)*

R MODE *Mode Operating (see page 610)*



MINV(arg)

Mode Inverse (Commutation Inverse)

APPLICATION:	Motion control
DESCRIPTION:	Inverts (reverses, negates, turns around) the direction convention of the motor
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	Do not apply while motor drive is enabled
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	Input: 0 or 1
TYPICAL VALUES:	0
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	MINV(0):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The MINV (Mode Inverse) command inverts the direction convention (orientation) of the SmartMotor:

- MINV(0) restores the default direction convention
- MINV(1) inverts the direction convention

When looking at the end of the motor shaft, the default direction convention of the SmartMotor is clockwise shaft rotation = positive encoder counts, and counterclockwise shaft rotation = negative encoder counts. Issuing the MINV(1) command inverts (flips) this direction convention.

EXAMPLE:

Enter these commands in the SMI software Terminal window:

```
ENCO      'Read position from internal encoder
O=1234    'Set origin to 1234
RPA       'Responds with 1234
'Manually rotate the motor shaft a few turns clockwise
RPA       'Responds with a higher count, like 9936
MINV(1)   'Inverts the direction convention
RPA       'Responds with same higher count, like 9936
'Manually rotate the motor shaft a few turns clockwise
RPA       'Responds with a lower count, like 694
MINV(0)   'Restores the default direction convention
'Manually rotate the motor shaft a few turns clockwise
RPA       'Responds with a higher count, like 8723
```

RELATED COMMANDS:

R CTR(enc) *Counter, Encoder, Step and Direction (see page 380)*

R PA *Position, Actual (see page 646)*



APPLICATION:	Motion control
DESCRIPTION:	Get operating mode, specific trajectory
EXECUTION:	Read only
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RMODE ; supports the DS2020 Combitronic system
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	-5 to 7
TYPICAL VALUES:	-5 to 7
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	RMODE(1):3, x=MODE(1):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The MODE command gets (reads) the operating mode:

- MODE gets the active operating mode
- MODE(...) gets the active operating mode, specific trajectory
For example x=MODE(1) will report trajectory 1 (MP, MV, MT)

The next table describes the possible meaning for the returned value:

Meaning	Value from MODE or MODE(0)	Trajectory 1 MODE(1)	Trajectory 2 Mode(2)
Cyclic Synchronous Torque (CST)	10	10	0
Cyclic Synchronous Velocity (CSV)	9	9	0
Cyclic Synchronous Position (CSP)	8	8	0
CANopen Interpolation	7	7	0
Homing	6	6	0
Torque (MT)	4	4	0

Meaning	Value from MODE or MODE(0)	Trajectory 1 MODE(1)	Trajectory 2 Mode(2)
Velocity (MV)	3	3	0
Position (MP)	1	1	0
Null (move generator inactive)	0	0	0
Quadrature Follow (MFR)	-2	0	-2
Step/Direction Follow (MSR)	-3	0	-3
Cam (MC)	-4	0	-4
Mixed: MP and MFR	-5	1	-2
Mixed: MP and MSR	-5	1	-3
Mixed: MP and MC	-5	1	-4
Mixed: MV and MFR	-5	3	-2
Mixed: MV and MSR	-5	3	-3
Mixed: MV and MC	-5	3	-4
JLS*	-13	-13	0

*Applies only to the JLS firmware version.

EXAMPLE:

In the SMI editor, create this program, download it to a SmartMotor and then run it.

```
EIGN (W, 0)      'Disable hardware limits
ZS              'Clear status bits
MP             'Set position mode
AT=500         'Preset acceleration.
VT=1000000    'Preset velocity.
PT=0          'Zero out position.
O=0           'Declare origin
G             'Servo in place
END           'Required END of program command
```

At the SMI software Terminal window, type these commands:

NOTE: Do not enter the comments — those are for your information and to show what is returned by the commands.

```
RMODE          'Reports 1 for position mode
PRINT (MODE, #13) 'Prints 1 for position mode
```

Edit the program and substitute MV (velocity mode), download it to a SmartMotor and then run it.

At the SMI software Terminal window, type these commands:

NOTE: Do not enter the comments — those are for your information and to show what is returned by the commands.

```
RMODE          'Reports 3 for velocity mode
PRINT (MODE, #13) 'Prints 3 for velocity mode
```

RELATED COMMANDS:

MC *Mode Cam (Electronic Camming) (see page 555)*

MFR *Mode Follow Ratio (see page 600)*

MP *Mode Position (see page 613)*

MSR *Mode Step Ratio (see page 618)*

MT *Mode Torque (see page 620)*

MV *Mode Velocity (see page 624)*



APPLICATION:	Motion control; supports the DS2020 Combitronic system
DESCRIPTION:	Request position mode
EXECUTION:	Buffered until a G command is issued
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	This is the default motion mode at power up
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	MP:3 or MP(0):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

Mode Position (MP), or position mode, is the default mode of the motor. If you ever change modes, you can return to position mode by issuing the MP command. The mode request is buffered until a G command is issued.

NOTE: For a standard position-mode move, the SmartMotor™ requires, at a minimum, a position target (PT), nonzero trajectory velocity (VT) and a nonzero positive acceleration/deceleration (ADT).

MP calculates the trajectory to the target position when the G command is issued. The preceding PT=formula or PRT=formula determines if the move is to be absolute (destination target set equal to buffered PT value) or relative (destination target set equal to current trajectory position plus the buffered PRT offset value).

The PID (servo) will be active. The MP mode calculates a trapezoidal velocity profile as a function of time. This profile is calculated to accelerate, reach a slow speed, and decelerate in a way so the speed is exactly 0 at the target position. The PID uses this calculated ideal position (PC) and compares it to the actual position (PA). The PID will apply torque to the motor to follow this profile with as little error (EA) as possible. Position error is due to basic physics of friction, inertia, gravity or any other force on the motor.

The G command may be issued at any time, and it may be repeated (particularly in the case of relative modes with PRT=offset). When repeating the G command in the middle of a move, the result will depend on the absolute versus relative mode:

- Absolute mode (initiated with a PT command) will always target the most recent value of PT. If the PT value is the same as before, that is acceptable and can be useful for changing the speed while keeping the target the same. If the application has newer information about the desired target, the PT value can be different than previous values in order to correct the target position.
- Relative mode (initiated with a PRT command) will always calculate a new ending position by adding PRT to the current position. This is important to understand because a G command issued while in motion will not remember the previous relative target. In indexing applications like rotary tables, this could lead to an offset from the original indexed locations. Therefore, this mode must be used carefully.

Assuming there are no faults, an MP command immediately followed by a G command turns on the servo. The servo-off flag (Bo) is set to 0; the trajectory flag (Bt) is set to 1. The motion is restricted by the current value of EL. The motion is also subject to the currently defined activity of the limit switches. RMODE responds with a P.

EXAMPLE:

```
MV           'Velocity mode
ADT=1000     'Set accel/decel
VT=50000    'Set velocity
G           'Start motion
WAIT=6000   'Wait 6000 samples
MP          'Position mode
ADT=50      'Set accel/decel
VT=40000   'Set velocity
PT=1000    'Set position
G          'Start (change) motion
WAIT=200   'Wait 200 samples
VT=45000   'Change velocity
PT=0       'Update position
G          'Start motion
```

EXAMPLE: (Routine homes motor against a hard stop)

```
MDS           'Using Sine mode commutation
KP=3200      'Increase stiffness from default
KD=10200    'Increase damping from default
F           'Activate new tuning parameters
AMPS=100     'Lower current limit to 10%
VT=-10000   'Set maximum velocity
ADT=100     'Set maximum accel/decel
MV          'Set Velocity mode
G           'Start motion
WHILE EA>-100 'Loop while position error is small
LOOP        'Loop back to WHILE
O=-100      'While pressed, declare home offset
S           'Abruptly stop trajectory
MP          'Switch to Position mode
VT=20000    'Set higher maximum velocity
PT=0        'Set target position to be home
G           'Start motion
TWAIT       'Wait for motion to complete
AMPS=1023   'Restore current limit to maximum
END         'End program
```

RELATED COMMANDS:

ADT=formula *Acceleration/Deceleration Target (see page 263)*

R EL=formula *Error Limit (see page 426)*

G *Start Motion (GO) (see page 473)*

MV *Mode Velocity (see page 624)*

R PRT=formula *Position, Relative Target (see page 683)*

R PT=formula *Position, (Absolute) Target (see page 690)*

R VT=formula *Velocity Target (see page 828)*



Mode Step, Zero External Counter

APPLICATION:	Motion control
DESCRIPTION:	Request step-and-direction counter mode; zero the external counter
EXECUTION:	Immediate; remains in effect until otherwise commanded
CONDITIONAL TO:	Step-and-direction input available
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	Motor default is MS0; however, it is best practice to issue MS0 before beginning program that uses step and direction
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	MS0:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:



WARNING: For Class 5 D-series motors, certain special features may override the brake function. In particular, the MFR, MSR, MFO, MS0 commands, or any similar feature from a network interface (including CANopen modes of operation: -1, -3, -11), may interfere with a brake assignment to I/O 0 or 1). Therefore, use of I/O 0 or 1 *is not* recommended for the brake in the Class 5 D-series if follow or step modes are used, regardless of SRC setting. For a programming example, refer to MS0 on page 616.

The MS0 command zeroes the second encoder register (see CTR(enc) on page 380) without changing the current motion mode of the SmartMotor™.

Following MS0, the A and B pins will be interpreted as a step-and-direction signal. MFO is the opposite input mode.

- On the D-style motor, these are inputs 0 and 1.
- On the M-style motor, these encoder inputs are differential and are labeled separately from general I/O signals.

If the Mode Step with Ratio (MSR) or the Cam mode does not meet your requirements, you can write your own loop and define a unique relationship between the incoming secondary encoder signal and the motor's position.

It is not necessary to use the inputs for motion. Any application that needs to count an input signal can use this feature.

Step-and-direction inputs are most commonly used to emulate a simple stepper motor drive. In MSR mode, the MFMUL and MFDIV commands can be used to allow a specific travel distance with each incoming pulse, just as a stepper drive would do.

NOTE: As with most stepping systems, opto-isolation modules are recommended to assure robust step-and-direction operation.

EXAMPLE:

```
MS0          'Reset CTR to zero
             'CTR value follows step-and-direction inputs
```

RELATED COMMANDS:

R CTR(enc) *Counter, Encoder, Step and Direction (see page 380)*

MFO *Mode Follow, Zero External Counter (see page 578)*

R MFDIV=formula *Mode Follow Divisor (see page 588)*

R MFMUL=formula *Mode Follow Multiplier (see page 598)*

MFR *Mode Follow Ratio (see page 600)*

MSR *Mode Step Ratio (see page 618)*



APPLICATION:	Motion control
DESCRIPTION:	Request step mode with ratio
EXECUTION:	Buffered until a G command is issued
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	MP
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	MSR:3 or MSR(0):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:



WARNING: For Class 5 D-series motors, certain special features may override the brake function. In particular, the MFR, MSR, MFO, MSO commands, or any similar feature from a network interface (including CANopen modes of operation: -1, -3, -11), may interfere with a brake assignment to I/O 0 or 1). Therefore, use of I/O 0 or 1 *is not* recommended for the brake in the Class 5 D-series if follow or step modes are used, regardless of SRC setting. For a programming example, refer to MSR on page 618.

The MSR command operation is nearly identical to the MFR command. The only difference is that the encoder input is configured as a step-and-direction signal. For a more detailed description of how the following ratio is set and how to use the profile commands such as MFA, MFSLEW, etc., see MFR on page 600. Those commands also apply in the MSR motion.

NOTE: MFMUL and MFDIV are each set to 1 by default (1 to 1 ratio). Therefore, it is only necessary to specify either or both if you want to change the default ratio.

NOTE: MFR and MSR are used to enable the desired Electronic Gearing mode (Mode Follow or Mode Step, respectively). They are not needed to enable a change to the ratio—the "G" command will do that.

MSR is typically used in applications where the input signal is from a controller using stepper motors. The SmartMotor™ can be used in place of a stepper drive and stepper motor. By carefully applying MFMUL and MFDIV, you can select a wide range of motion per input step.

NOTE: As with most stepping systems, opto-isolation modules are recommended to assure robust step-and-direction operation.

EXAMPLE:

```
MS0           'Reset CTR
MFMUL=21     'Numerator = 21
MFDIV=-10    'Denominator = -10
MSR          'Example: input 100 external encoder counts,
             'resulting motion is -210 counts
MP (1)       'Position mode in trajectory 1 while also
             'keeping MSR active
PRT=0        'No phase shift
G (2)        'Start following
             'Implementing phase adjust:
PRT=500      ' Set relative position target
VT=5000     ' Set velocity target
ADT=100     ' Set accel/decel target
G (1)        ' Start phase adjust
```

RELATED COMMANDS:

R CTR(enc) *Counter, Encoder, Step and Direction (see page 380)*
 G *Start Motion (GO) (see page 473)*
 MFA(distance[,m/s]) *Mode Follow Ascend (see page 580)*
 MFCTP(arg1,arg2) *Mode Follow Control Traverse Point (see page 583)*
 MFD(distance[,m/s]) *Mode Follow Descend (see page 585)*
 R MFDIV=formula *Mode Follow Divisor (see page 588)*
 MFH(distance[,m/s]) *Mode Follow, High Ascend/Descend Rate (see page 590)*
 R MFHTP=formula *Mode Follow, High Traverse Point (see page 592)*
 MFL(distance[,m/s]) *Mode Follow, Low Ascend/Descend Rate (see page 594)*
 R MFLTP=formula *Mode Follow, Low Traverse Point (see page 596)*
 R MFMUL=formula *Mode Follow Multiplier (see page 598)*
 MFR *Mode Follow Ratio (see page 600)*
 MFSDC(distance,mode) *Mode Follow, Stall-Dwell-Continue (see page 603)*
 MFSLEW(distance[,m/s]) *Mode Follow Slew (see page 605)*
 SRC(enc_src) *Source, Follow and/or Cam Encoder (see page 759)*



APPLICATION:	Motion control
DESCRIPTION:	Request torque mode
EXECUTION:	Buffered until a G command is issued
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	MP
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	MT:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The MT (Mode Torque) command enables Torque mode. In this mode, the motor is commanded to develop a specific output effort, set by T =formula.

- $T=32767$ results in 100% PWM in the positive direction
- $T=-32767$ results in 100% PWM in the negative direction

While in this mode, the encoder still tracks position and can still be read with the PA variable. However, the PID loop is off, and the motor is not servoing or running a trajectory.

For any given torque and no applied load, there will be a velocity at which the Back EMF (BEMF) of the motor stops acceleration and holds an almost-constant velocity. Therefore, under the no-load condition, the T command will control velocity. As the delivered torque increases, the velocity decreases.

NOTE: MT does not regulate torque in the D-style motor. Instead, it delivers a fixed amount of voltage (PWM) to the motor coils.

Assuming there are no faults, an MT command and then a G command immediately activates the servo. The servo off flag (Bo) is set to 0; the trajectory flag (Bt) will indicate if a TS ramp has been set. The motion is not restricted by the current EL value. For instance, issuing EL=0 would have no effect on the current motion. The motion is subject to the currently defined activity of the limit switches.

The ramp-up rate for the T value can be controlled with the TS command. For details, see TS=formula on page 786.



CAUTION: Do not attempt to regulate speed with Torque mode. It is not designed for that and will give poor results. Likewise, it is difficult to place a speed limit on Torque mode. If the load decreases, it causes the motor shaft speed to increase to a new equilibrium because power must remain constant.

EXAMPLE: (Increases torque, one unit every PID sample period, up to 8000 units)

```
MT          'Select torque mode
T=8000      'Final torque after the TS ramp that we want
TS=65536    'Increase the torque by 1 unit of T per PID sample
G          'Begin move
```

RELATED COMMANDS:

R Bt *Bit, Trajectory In Progress (see page 345)*

R T=formula *Torque, Open-Loop Commanded (see page 769)*

R TS=formula *Torque Slope (see page 786)*



APPLICATION:	Motion control
DESCRIPTION:	Enables torque brake mode, which dynamically brakes the motor
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	MP
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	MTB:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The MTB (Mode Torque Brake) command places the SmartMotor™ into dynamic brake mode. In this mode, the motor coils are shorted together. Any motion of the shaft would normally produce Back EMF (BEMF) that is proportional to speed. However, having the windings shorted out causes this BEMF to be dissipated immediately. The result is a magnetic-damping counterforce to any attempted motion of the shaft from an external source.

If MTB is issued while moving at a given speed, the shaft will come to a stop at a rate proportional to the BEMF that was being generated when the MTB command was issued. The shaft doesn't stop at any predetermined or commanded position, and its trajectory is uncontrolled.

While in MTB, the motor will not produce any external DC-bus voltage rise if the shaft is rotated because all windings are shorted together. As a result, the DC bus is protected against bus overvoltage to within the drive stage current limits.

MTB automatically engages when the motor is off. The only way to prevent that automatic action is to manually "freewheel" the motor by issuing a BRKRLS command and then an OFF command (in that order). Those two commands do not need to be in immediate sequence—i.e., other commands, except MTB, can be between them. To re-enable the automatic MTB function, issue an MTB command.

NOTE: To ensure the motor remains in "freewheel" state, issue the FSA command (with action 1, servo off / freewheel) before issuing the BRKRLS OFF command sequence. For details, see FSA (cause,action) on page 465.

Also, by default, MTB is automatically issued when the motor faults on overtemperature, position errors or travel limits. For information on changing this action, refer to FSA(cause,action) on page 465.

EXAMPLE: (Fault-handler subroutine, shows use of MTB and US)

```
C0          'Fault handler
MTB:0      'Motor will turn off with Dynamic
           'braking, tell other motors to stop.
US (0) :0  'Set User Status Bit 0 to 1 (Status
           'Word 12 bit zero)
US (ADDR) :0 'Set User Status Bit "address" to 1
           '(Status Word 12 Bit "address")
```

RETURNI

RELATED COMMANDS:

BRKRLS *Brake Release (see page 335)*

FSA(cause,action) *Fault Stop Action (see page 465)*

G *Start Motion (GO) (see page 473)*

OFF *Off (Drive Stage Power) (see page 636)*



APPLICATION:	Motion control; supports the DS2020 Combitronic system
DESCRIPTION:	Request velocity mode
EXECUTION:	Buffered until a G command is issued
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	MP
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	MV:3 or MV(0):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The MV (Mode Velocity) command enables velocity mode. In this mode, the value of VT, the target velocity, can be negative or positive. In contrast, position mode (MP) only uses the magnitude of the velocity parameter. Acceleration and velocity can be changed at any time, even during motion. The G command will initiate "on the fly" changes to any of the parameters.

If the actual velocity is greater than the value defined by VT, then on reception of the next G command, the motor shaft will decelerate at the rate set by ADT until the excess velocity is removed. Conversely, if the actual velocity is less than VT when the G command is entered, then the motor shaft motion will accelerate at the rate set by ADT until the requested velocity is reached. Similarly, if the actual velocity is in the opposite direction of VT when the G command is entered, then the motor shaft motion will first decelerate and then accelerate at the rate set by ADT until the requested velocity is reached.

When the commanded velocity VT is reached, motion continues at that rate (i.e., maintains uniform velocity until the commanded velocity is changed or the mode is otherwise terminated). The encoder count may "wrap around" during this mode, but no position error will be declared during the wrap.

The PID (servo) will be active. The MV mode calculates a ramp up to the specified velocity based on the specified acceleration (ADT). The profile will stay at the velocity until commanded to stop with an X command, which will decelerate the motor to a stop. The velocity mode calculates position as a function of time. This is different than simpler velocity controls that do not track position but only track velocity error. The PID uses this ideal calculated position (PC) and compares it to the actual position (PA). This allows any accumulated speed errors to be corrected on average with a high degree of accuracy. Therefore, any loading that slowed the motor will be "caught up". The PID will apply torque to

the motor to follow this profile with minimal actual error (EA). Position error is due to basic physics of friction, inertia, gravity or any other force on the motor.

A velocity-error based system can be emulated by commanding the motor to ignore position error limits.



WARNING: This method is not appropriate where a position-error limit fault is required for safety protection.

Refer to EL=formula on page 426 and the KI=formula on page 532. By disabling these two features and possibly reducing KP, it is possible to have a "softer" velocity controller that will not attempt to strictly adhere to a position error of 0.

Assuming there are no faults, an MV command and then a G command immediately turns on the servo. The servo-off flag (Bo) is set to 0; the trajectory flag (Bt) is set to 1. The motion is restricted by the current value of EL. The motion is also subject to the currently defined activity of the limit switches. RMODE responds with a V.

EXAMPLE: (Routine homes motor against a hard stop)

```

MDS           'Using Sine mode commutation
KP=3200       'Increase stiffness from default
KD=10200     'Increase damping from default
F            'Activate new tuning parameters
AMPS=100     'Lower current limit to 10%
VT=-10000   'Set maximum velocity
ADT=100      'Set maximum accel/decel
MV           'Set Velocity mode
G            'Start motion
WHILE EA>-100 'Loop while position error is small
LOOP         'Loop back to WHILE
O=-100      'While pressed, declare home offset
S           'Abruptly stop trajectory
MP          'Switch to Position mode
VT=20000    'Set higher maximum velocity
PT=0        'Set target position to be home
G           'Start motion
TWAIT       'Wait for motion to complete
AMPS=1023   'Restore current limit to maximum
END         'End program

```

RELATED COMMANDS:

ADT=formula *Acceleration/Deceleration Target (see page 263)*

R EL=formula *Error Limit (see page 426)*

G *Start Motion (GO) (see page 473)*

MP *Mode Position (see page 613)*

R VT=formula *Velocity Target (see page 828)*

APPLICATION:	Communications control
DESCRIPTION:	Send NMT state (broadcast or to a specific node)
EXECUTION:	Immediate
CONDITIONAL TO:	Enabled through CANCTL(17,value), see CANCTL(function,value) on page 359.
LIMITATIONS:	Does not apply to Class 6 systems
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x.4.30 (D/M) requires CAN option; 6.4.2.x (D)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The NMT command transmits an NMT message to the network; it can command a either a specific or all follower devices to enter the commanded state. To do this, use:

```
NMT(follower addr, state)
```

where:

follower addr	is a variable such as: x, ab[x], aw[x], al[x], or a constant such as: 2 (a motor with CAN address 2). Range is 1 to 127; use 0 for a broadcast. If follower addr is to itself, then internal loopback is used instead.
state	is the desired NMT state from one of these values: <ul style="list-style-type: none"> 1 Operational state (PDO and SDO communications are functional) 2 Stopped state (no SDO or PDO communications) 128 Pre-operational state (SDO communications are allowed; no PDO communications) DEFAULT state at power up 129 Reset application (resets the whole motor) 130 Reset communications (resets the CANopen stack mappings, etc., but motor program, variables, etc retains current state)

EXAMPLE:

```
NMT(0,1) 'Tell everyone to go operational.  
NMT(2,128) 'Tell motor 2 to go pre-operational.  
x=CAN(4)  
IF x!=0  
    ' NMT command failed.  
ENDIF
```

RELATED COMMANDS:

R CAN, CAN(arg) *CAN Bus Status (see page 357)*
CANCTL(function,value) *CAN Control (see page 359)*
SDORD(...) *SDO Read (see page 730)*
SDOWR(...) *SDO Write (see page 732)*



O=formula, O
(trj#)=formula
Origin

APPLICATION:	Motion control; supports the DS2020 Combitronic system
DESCRIPTION:	Set the SmartMotor origin
EXECUTION:	Immediate
CONDITIONAL TO:	Present trajectory position
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	Assignment
UNITS:	Encoder counts DS2020 Combitronic system: user increments, see FD=expression on page 461
RANGE OF VALUES:	-2147483648 to 2147483647
TYPICAL VALUES:	-2147483648 to 2147483647
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	O:3=1234 or O(0):3=1234 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

NOTE: This command is affected by the SCALEP command. For details, see SCALEP(m,d) on page 726. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.



CAUTION: For motors with absolute encoders (Class 5M-style with -FB01 option; Class 6 D-style), use of the O= or OSH= command will confuse the absolute position information. Therefore, do not use these commands. Instead, use the ENCCTL command. For details, see ENCCTL(function,value).

The O (Origin) command allows the current commanded (trajectory) position to be set to any value. The actual position is also updated by the same difference. However, the position error remains the same before and after executing this command. You may declare the current trajectory position as zero by entering O=0 (the capital letter "O" = the number zero). Similarly, you may declare the current position to be 1234 by entering O=1234.

NOTE: Using the O=formula does not modify previously entered PT or PRT registers.

NOTE: The DS2020 Combitronic system supports the O=formula format only.

Instead, the O(trj#)= form of the command changes the *virtual* position of trajectory 1 and 2. There are no actual positions to change in those cases. Positions PA and PC are not affected by O(1)= or O(2)=.

The O command shifts the position counters, as shown in the next table:

Command	Trajectory Position	Actual Position
O=formula	PC set to 'formula value'	PA set to PC - EA
O(0)=formula	PC set to 'formula value'	PA set to PC - EA
O(1)=formula	PC(1) set to 'formula value'	N/A
O(2)=formula	PC(2) set to 'formula value'	N/A

NOTE: Consider using the OSH command when the amount of position shift is already known. In other words, if you know that you want to remove 1000 counts from the present position, then use the command OSH=-1000 instead. For details, see OSH=formula, OSH(trj#)=formula on page 642.

EXAMPLE: (Reassigning origin does not modify buffered PT and PRT values)

```

ADT=20      'Set accel/decel target
VT=100000  'Set velocity target
PT=5000    'Set position target
MP         'Set Position mode
O=-1000    'Current position set to negative 10000
GOSUB5
O=12345    'Current position set to 12345
GOSUB5
PRT=5000
O=3000     'Current position set to 3000
GOSUB5
END
C5
  PRINT(#13,"Move origin is ",PA)
  G
  WHILE Bt LOOP
  WAIT=4000
  PRINT(#13,"Position is ")
  RPA
RETURN

```

Program output is:

```

Move origin is   -1000
Position is      5000
Move origin is   12345
Position is      5000
Move origin is   3000
Position is      8000

```

RELATED COMMANDS:

R EA *Error Actual (see page 401)*

R FD=expression *Resolution to Set Units of Position/Velocity/Acceleration (see page 461)*

OSH=formula, OSH(trj#)=formula *Origin Shift (see page 642)*

R PA *Position, Actual (see page 646)*

R PC, PC(axis) *Position, Commanded (see page 650)*

APPLICATION:	I/O control
DESCRIPTION:	Read current output driving state of 24 Volt I/O
EXECUTION:	Immediate
CONDITIONAL TO:	Class 5 M-style, or D-style with AD1 option
LIMITATIONS:	Command is not available for any Class 6 motors
READ/REPORT:	ROC(...)
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	Output: Depends on command format and motor model (see details)
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); no Class 6
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

NOTE: The OC command applies to only Class 5 M-style motors or D-style motors with the AD1 option. It is designed for 24V I/O, and therefore, does not apply to the D-style 5V I/O. In that case, refer to the IN/RIN commands (with bitmask), see IN(...) on page 509.

The OC command reads the specified output or a block of outputs. This applies to the 24 volt sourcing I/O. When the output is set high (24 volts), the value is represented by this command as a 1. It can be used in these ways:

- =OC(IO)
 - Individual output status of I/O number; result is 1 if output is ON and 0 if it is OFF.
 - D-style: IO is 16-25
 - M-style: IO is 0-10
- =OC(W,word)
 - Get output status for a whole word of I/O bits; result is a bitfield (e.g., bits 0-10 are represented with a range of numbers from 0 to 2047).
 - D-style: word is 1; result is 0-1023
 - M-style: word is 0; result is 0-2047

NOTE: This does not represent the state of the inputs. Only the commanded outputs are represented.

EXAMPLE: (Subroutine reports the status of the 24V expansion IO)

```
'This code reports the status of the 24V expansion IO
a=0          'Set loop start point for first IO
WHILE a<10  'While less than number of IO
  b=a+16     'Set b to IO number for 24V expansion
  c=OC(b)    'Set c to IO condition status
  d=OF(b)    'Set d to IO fault status
  PRINT("Output ",b," on pin ",a)
  IF d==1    'If d represents overcurrent
    PRINT(" is FAULTED overcurrent.",#13)
  ELSEIF d==2 'If d represents a possible short
    PRINT(" is FAULTED short.",#13)
  ELSEIF c    'If c is true
    PRINT(" is HIGH.",#13)
  ELSE      'If c is not true
    PRINT(" is LOW.",#13)
  ENDIF
  a=a+1     'Increment loop counter
LOOP
```

Program output is:

```
Output 16 on pin 0 is HIGH.
Output 17 on pin 1 is HIGH.
Output 18 on pin 2 is LOW.
Output 19 on pin 3 is LOW.
Output 20 on pin 4 is LOW.
Output 21 on pin 5 is LOW.
Output 22 on pin 6 is LOW.
Output 23 on pin 7 is LOW.
Output 24 on pin 8 is LOW.
Output 25 on pin 9 is LOW.
```

RELATED COMMANDS:

R OF(...) *Output Fault (see page 634)*

OR(value) *Output, Reset (see page 638)*

OS(...) *Output, Set (see page 640)*

APPLICATION:	Communications control; supports the DS2020 Combitronic system
DESCRIPTION:	Open a communications channel
EXECUTION:	Immediate
CONDITIONAL TO:	External communication I/O connections
LIMITATIONS:	Hardware capabilities
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	See detailed description
TYPICAL VALUES:	See detailed description
DEFAULT VALUE:	D-style: OCHN (RS2, 0, N, 9600, 1, 8, C) M-style: OCHN (RS4, 0, N, 9600, 1, 8, C)
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020) OCHN channel 1 requires: 5.x (D/M); 6.4.2.x (D)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

CAUTION: The OCHN command will cause the SmartMotor to ignore incoming commands and can lock you out. Therefore, during development, prevent this by using the RUN? command at the start of each program.

NOTE: If you get locked out and are unable to communicate with the SmartMotor, you may be able to recover communications using the SMI software's Communication Lockup Wizard. For more details, see Communication Lockup Wizard on page 31.

OCHN(Type,Channel,Parity,Baud,StopBits,DataBits,Mode[,Timeout]) opens a serial channel with these specifications:

NOTE: Not all combinations of values are permitted; see the next tables showing the allowed combinations.

Allowed Values for D-style Motors								
Type	Chnl	Parity	Baud	Stop bits	Data bits	Mode	Timeout (optional)	
RS2	0	E, O, N	100, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200	1, 2	8	C, D	Relevant in C (command) mode	
RS4	0, 1	E, O, N	100, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400,	1, 2	8	C, D	Relevant in C (command) mode	

Allowed Values for D-style Motors							
Type	Chnl	Parity	Baud	Stop bits	Data bits	Mode	Timeout (optional)
			57600, 115200				
MB4	1	E, O, N	100, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200	1, 2	8	D	N/A
DMX	1	N	250000	2	8	D	N/A
IIC	1	N/A	60000 to 1000000	N/A	N/A	D	N/A

Parity: O=odd, E=even, N=none; Mode: C=command, D=data

NOTE: For the D-style motor, opening channel 0 as an RS-485 port dedicates I/O pin 6 to the RS-485 control function. This is required for use with Moog Animatics RS-232 to RS-485 converters like the RS485 and RS485-ISO.

Allowed Values for M-style Motors							
Type	Chnl	Parity	Baud	Stop bits	Data bits	Mode	Timeout (optional)
RS4	0	E, O, N	100, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200	1, 2	8	C, D	Relevant in C (command) mode
MB4	0	E, O, N	100, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200	1, 2	8	D	N/A
DMX	0	N	250000	2	8	D	N/A

Parity: O=odd, E=even, N=none; Mode: C=command, D=data

EXAMPLE: (Shows use of ECHO_OFF1 and OCHN)

```
EIGN (w, 0)    'Make all onboard I/O inputs
ZS            'Clear errors
OCHN (RS4, 1, N, 9600, 1, 8, C) 'Open aux communications channel
ECHO_OFF1    'Turn echo off for aux communications channel
END
```

RELATED COMMANDS:

CCHN(type,channel) *Close Communications Channel (RS-232 or RS-485) (see page 365)*

^R CHN(channel) *Communications Error Flag (see page 367)*

APPLICATION:	I/O control
DESCRIPTION:	Get output faults (24 volt I/O)
EXECUTION:	Immediate
CONDITIONAL TO:	M-style, or D-style with AD1 option
LIMITATIONS:	Command is not available for any Class 6 motors
READ/REPORT:	ROF(...)
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	Depends on command format and motor model (see details)
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); no Class 6
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The OF command reads the specified output or a block of outputs. It can be used in these ways:

- =OF(IO)

Returns the current fault state for that I/O number, where the returned value: 0 = no fault , 1 = overcurrent, 2 = possible shorted

 - D-style: IO is 16-25
 - M-style: IO is 0-10
- =OF(S,word)

Returns the bit mask of current faulted I/O points, where word is the 16-bit word number, 0 is Controller I/O Status Word 16.
If any of these bits are set due to a fault, then the I/O fault status flag (Motor Status Word 3, bit 7) is also set.

 - D-style: word is 1; result is 0-1023
 - M-style: word is 0; result is 0-2047
- =OF(L,word)

Returns the bit mask of Latched Faulted I/O points, where word is the 16-bit word number.
Reading a 16-bit word will attempt to clear the I/O word latch.

 - D-style: word is 1; result is 0-1023
 - M-style: word is 0; result is 0-2047

- =OF(D,word)
 - Returns an error code from the controller associated with this I/O word.
 - D-style: word is 1
 - M-style: word is 0

EXAMPLE: (Subroutine reports the status of the 24V expansion IO)

```
'This code reports the status of the 24V expansion IO
a=0          'Set loop start point for first IO
WHILE a<10   'While less than number of IO
  b=a+16     'Set b to IO number for 24V expansion
  c=OC(b)    'Set c to IO condition status
  d=OF(b)    'Set d to IO fault status
  PRINT("Output ",b," on pin ",a)
  IF d==1    'If d represents overcurrent
    PRINT(" is FAULTED overcurrent.",#13)
  ELSEIF d==2 'If d represents a possible short
    PRINT(" is FAULTED short.",#13)
  ELSEIF c    'If c is true
    PRINT(" is HIGH.",#13)
  ELSE       'If c is not true
    PRINT(" is LOW.",#13)
  ENDIF
  a=a+1      'Increment loop counter
LOOP
```

Program output is:

```
Output 16 on pin 0 is HIGH.
Output 17 on pin 1 is HIGH.
Output 18 on pin 2 is LOW.
Output 19 on pin 3 is LOW.
Output 20 on pin 4 is LOW.
Output 21 on pin 5 is LOW.
Output 22 on pin 6 is LOW.
Output 23 on pin 7 is LOW.
Output 24 on pin 8 is LOW.
Output 25 on pin 9 is LOW.
```

RELATED COMMANDS:

R OC(...) *Output Condition (see page 630)*
 OR(value) *Output, Reset (see page 638)*
 OS(...) *Output, Set (see page 640)*



OFF

Off (Drive Stage Power)

APPLICATION:	Motion control; supports the DS2020 Combitronic system
DESCRIPTION:	Turn servo off
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	OFF
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	OFF:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The OFF command turns off the power to the motor coils and terminates the activity of the current motion mode. The motor off system flag, Bo, will be set to 1. The shaft will be free to coast to a stop or rotated by other external means. The trajectory in progress system flag, Bt, will be set to zero.

NOTE: When commanded OFF, the motor will still track any shaft movement and continue to update the current encoder position.

By default, the motor will activate MTB. To prevent that automatic action, manually "freewheel" the motor by issuing a BRKRLS command and then an OFF command (in that order). Those two commands do not need to be in immediate sequence—i.e., other commands, except MTB, can be between them. To re-enable the automatic MTB function, issue an MTB command. For more details on MTB, see MTB on page 622.

NOTE: To ensure the motor remains in "freewheel" state, issue the FSA command (with action 1, servo off / freewheel) before issuing the BRKRLS OFF command sequence. For details, see FSA (cause,action) on page 465.

EXAMPLE:

```

'Set an interrupt on input 6 to freewheel motor
ITR(0,16,6,0,100)      'Setup interrupt 0 to run C100
EITR(0)                'Enable interrupt 0
ITRE                   'Global enable interrupts
PAUSE                  'Pause program to keep interrupts running
END                    'End of program

C100                   'Interrupt subroutine
  BRKRLS                'Turn off MTB or brake
  OFF                   'Freewheel the motor
  'Remain here while input 6 is low
  WHILE IN(6)==0 LOOP
RETURNI                'Return from interrupt

```

EXAMPLE: (Shows use of "freewheel")

```

PT=PA      'Set Target position to current position
G          'Holds position
OFF        'Drive stage off, but MTB (dynamic braking) active
G          'Holds again
BRKRLS    'No change seen yet
OFF        'Now motor freewheels
G          'Motor holds in place again
OFF        'Motor freewheels
MTB        'Motor has dynamic braking
G          'Motor holds position
OFF        'Motor off but WITH dynamic braking
BRKRLS    'No change seen yet
G          'Motor holds position
OFF        'Motor freewheels
G          'Motor holds position
OFF        'Motor freewheels
MTB        'Returns back to default of dynamic braking when OFF is issued

```

RELATED COMMANDS:

R B(word,bit) *Status Byte (see page 297)*

BRKRLS *Brake Release (see page 335)*

FSA(cause,action) *Fault Stop Action (see page 465)*

G *Start Motion (GO) (see page 473)*

MTB *Mode Torque Brake (see page 622)*

R W(word) *Report Specified Status Word (see page 833)*



OR(value) Output, Reset

APPLICATION:	I/O control
DESCRIPTION:	Reset (turn off) the specified output
EXECUTION:	Immediate
CONDITIONAL TO:	I/O available for general output (not assigned to special function)
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	Depends on command format and motor model (see details)
TYPICAL VALUES:	Depends on command format and motor model (see details)
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	OR(0):3 or OR(W,0):3 or OR(W,0,7):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The OR (Output Reset) command resets (turns off) the output specified by *value*:

- OR(IO)
Reset a single output to logic 0 (off).
- OR(W,word)
Simultaneously resets all outputs in the specified word.
- OR(W,word[,mask])
Reset outputs in the specified word if those bits are also a "1" in the bitmask.

Motor Type	word Allowed Values	IO Allowed Range	Logic 0 Voltage	Logic 1 Voltage	Bitmask Range
D-style	0	0-6	0	5	0 to 255
		7 (virtual only, not connected)	N/A	N/A	
D-style with AD1 option	0	0-6	0	5	0 to 255
		7 (virtual only, not connected)	N/A	N/A	
	1	16-25	0	24	0 to 1023

Motor Type	word Allowed Values	IO Allowed Range	Logic 0 Voltage	Logic 1 Voltage	Bitmask Range
M-style	0	0-10	0	24	0 to 2047

EXAMPLE:

```

WHILE 1           '1 is always true
  OS(0)           'Set output to 1
  OR(0)           'Set output to 0
LOOP              'Will loop forever

```

RELATED COMMANDS:

EIGN(...) *Enable as Input for General-Use (see page 412)*
R IN(...) *Specified Input (see page 509)*
R OC(...) *Output Condition (see page 630)*
R OF(...) *Output Fault (see page 634)*
OS(...) *Output, Set (see page 640)*
OUT(...)=formula *Output, Activate/Deactivate (see page 644)*



APPLICATION:	I/O control
DESCRIPTION:	Set (turn on) the specified output
EXECUTION:	Immediate
CONDITIONAL TO:	I/O available for general output (not assigned to a special function)
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	Depends on command format and motor model (see details)
TYPICAL VALUES:	Depends on command format and motor model (see details)
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	OS(0):3 or OS(W,0):3 or OS(W,0,7):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The OS (Output Set) command sets (turns on) the output specified by *value*:

- OS(I0)
Sets a single output to logic 1 (on).
- OS(W,word)
Simultaneously sets all outputs in the specified word.
- OS(W,word[,mask])
Sets outputs in the specified word if those bits are also a "1" in the bitmask.

Motor Type	word Allowed Values	IO Allowed Range	Logic 0 Voltage	Logic 1 Voltage	Bitmask Range
D-style	0	0-6	0	5	0 to 255
		7 (virtual only, not connected)	N/A	N/A	
D-style with AD1 option	0	0-6	0	5	0 to 255
		7 (virtual only, not connected)	N/A	N/A	
	1	16-25	0	24	0 to 1023

Motor Type	word Allowed Values	IO Allowed Range	Logic 0 Voltage	Logic 1 Voltage	Bitmask Range
M-style	0	0-10	0	24	0 to 2047

EXAMPLE:

```

WHILE 1          '1 is always true
  OS(0)         'Set output to 1
  OR(0)         'Set output to 0
LOOP            'Will loop forever

```

EXAMPLE: (turn on multiple ports)

```

WHILE a<4
  OS(a+4)       'turn ON I/O ports 4 thru 7.
  a=a+1
LOOP

```

EXAMPLE: (set all I/O to 5V)

```

i=0
WHILE i<=6      'Program loops until i = 6
  OS(i)         'Each output is enabled (set to 5V) as program loops
  i=i+1         'Increment i by 1 to enable the next input on next loop
LOOP           'Loop back to WHILE

```

RELATED COMMANDS:

EIGN(...) *Enable as Input for General-Use (see page 412)*

R IN(...) *Specified Input (see page 509)*

R OC(...) *Output Condition (see page 630)*

R OF(...) *Output Fault (see page 634)*

OR(value) *Output, Reset (see page 638)*

OUT(...)=formula *Output, Activate/Deactivate (see page 644)*



OSH=formula, OSH (trj#)=formula Origin Shift

APPLICATION:	Motion control
DESCRIPTION:	Shifts the origin of the position counter during motion
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	Assignment
UNITS:	Encoder counts
RANGE OF VALUES:	-2147483648 to 2147483647
TYPICAL VALUES:	-2147483648 to 2147483647
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	OSH:3=1234 or OSH(0):3=1234 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

NOTE: This command is affected by the SCALEP command. For details, see SCALEP(m,d) on page 726. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.



CAUTION: For motors with absolute encoders (Class 5M-style with -FB01 option; Class 6 D-style), use of the O= or OSH= command will confuse the absolute position information. Therefore, do not use these commands. Instead, use the ENCCTL command. For details, see ENCCTL(function,value).

The OSH (origin shift) command allows the current commanded (trajectory) position to be shifted. The shift is relative, which can be useful in applications where the origin needs to be shifted during motion without losing any position counts. Additionally, the actual position is updated by the same difference. However, the position error remains the same before and after executing this command.

Instead, the OSH(trj#)= form of the command changes the *virtual* position of trajectory 1 and 2. There are no actual positions to change in those cases. Positions PA and PC are not affected by OSH(1)=, or OSH(2)=.

The OSH command shifts the position counters, as shown in the next table:

Command	Trajectory Position	Actual Position
OSH=formula	PC=PC+'formula value'	PA=PA+'formula value'
OSH(0)=formula	PC=PC+'formula value'	PA=PA+'formula value'
OSH(1)=formula	PC(1)=PC(1)+'formula value'	N/A
OSH(2)=formula	PC(2)=PC(2)+'formula value'	N/A

EXAMPLE:

'Patterning a move can be done using the same routine
'by simply shifting the origin between moves.

```
PT=0           'Move to the origin.
G TWAIT
GOSUB (30)     'Run a subroutine to perform a set of absolute
               'position moves.
OSH=40000     'Shift the origin.
PT=0           'Move to the origin.
G TWAIT
GOSUB (30)     'Run the same subroutine with shifted origin.
OFF           'Turn off motor.
END           'End of program.

C30
      'Absolute position motion profile.
RETURN
```

RELATED COMMANDS:

R EA *Error Actual (see page 401)*
O=formula, O(trj#)=formula *Origin (see page 628)*
R PA *Position, Actual (see page 646)*
R PC, PC(axis) *Position, Commanded (see page 650)*



OUT(...)=formula

Output, Activate/Deactivate

APPLICATION:	I/O control; supports the DS2020 Combitronic system
DESCRIPTION:	Set or reset outputs according to assigned value
EXECUTION:	Immediate
CONDITIONAL TO:	I/O available for general output (not assigned to a special function)
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	Depends on command format and motor model (see details)
TYPICAL VALUES:	Depends on command format and motor model (see details)
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	OUT(0):3=1 or OUT(W,0):3=32 or OUT(W,0,7)=32 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The OUT command activates (turns on) or deactivates (turns off) the output specified by *IO*. If the *formula* least-significant bit = 1, then it's true (on); otherwise, it's false (off).

- OUT(*IO*)=*formula*
If bit 0 in the formula to the right of "=" is 1, then set I/O ON; otherwise, when it is even or zero, turn it OFF.
- OUT(W,*word*)=*formula*
Set the group of bits in the specified I/O word to the bitwise value from the formula.
- OUT(W,*word*[,*mask*])=*formula*
Set the group of bits in the specified I/O word to the bitwise value from the formula. However, leave bits as-is if they are bitwise set to 0 in the bitmask value.

Motor Type	<i>word</i> Allowed Values	IO Allowed Range	Logic 0 Voltage	Logic 1 Voltage	Formula Value Range	Bitmask Range
D-style	0	0-6	0	5	0 to 255	0 to 255
		7 (virtual only, not connected)	N/A	N/A		
D-style with	0	0-6	0	5	0 to 255	0 to 255

Motor Type	word Allowed Values	IO Allowed Range	Logic 0 Voltage	Logic 1 Voltage	Formula Value Range	Bitmask Range
AD1 option	0	7 (virtual only, not connected)	N/A	N/A		
	1	16-25	0	24	0 to 1023	0 to 1023
M-style	0	0-10	0	24	0 to 2047	0 to 2047
DS2020 Com-bitronic system	N/A	5	0	24		

EXAMPLE: (For pulse width)

```

. . .
WHILE 1>0
    O=0           'Reset origin for move
    PT=40000     'Set final position
    G           'Start motion
    WHILE PA<20000 'Loop while motion continues
    LOOP       'Wait for desired position to pass
    OUT(1)=0   'Set output lo
    TMR(0,400) 'Use timer 0 for pulse width
    TWAIT WAIT=1000 'wait 1 second
LOOP
. . .

```

EXAMPLE: (Set all I/O to outputs, and set their level to the value of x)

```

x=1           'x can be 1 (ON) or 0 (OFF)
i=0
WHILE i<=6   'Loops until i=6
    OUT(i)=x  'Set to output and turn on or off based on value of x
    i=i+1    'Increment i by 1
LOOP        'Loop back to WHILE

```

RELATED COMMANDS:

EIGN(...) *Enable as Input for General-Use (see page 412)*

R IN(...) *Specified Input (see page 509)*

R OC(...) *Output Condition (see page 630)*

R OF(...) *Output Fault (see page 634)*

OR(value) *Output, Reset (see page 638)*

OS(...) *Output, Set (see page 640)*



APPLICATION:	Motion control
DESCRIPTION:	Actual absolute position
EXECUTION:	Next PID sample
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RPA ; supports the DS2020 Combitronic system
WRITE:	Read only
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	Encoder counts Report for DS2020 Combitronic system: user increments, see FD=e-expression on page 461
RANGE OF VALUES:	-2147483648 to 2147483647
TYPICAL VALUES:	-2147483648 to 2147483647
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	RPA:3, x=PA:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

NOTE: This command is affected by the SCALEP command. For details, see SCALEP(m,d) on page 726. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

The PA (Position Actual) command is used to access the value of the main axis encoder. This number may be called the current position or actual position.

The main axis is assigned with the ENCO or ENC1 command. If the motor shaft moves, the value of PA will be changed by the net number of encoder counts occurring during the shaft motion. The primary encoder is tracked at all times — it is independent of the operation mode of the SmartMotor™ or any error condition.

For details on adjusting the value of PA, see the commands O=formula, O(trj#)=formula on page 628 and OSH=formula, OSH(trj#)=formula on page 642.

EXAMPLE:

```
ADT=100           'Set buffered accel/decel
VT=40000         'Set buffered velocity
MV               'Set to Mode Velocity
G                'GO, start motion trajectory
WHILE PA<=5000   'Wait until real-time position
LOOP             'Exceeds 5000 counts
PRINT("Position is above 5000",#13)
```

NOTE: PA follows the primary encoder that is used to close the loop. For example, if you issue ENC1, then it will follow an external encoder. For more details, see ENCO on page 432 and ENC1 on page 433.

RELATED COMMANDS:

R CTR(enc) *Counter, Encoder, Step and Direction (see page 380)*
ENC0 *Encoder Zero (Close Loop on Internal Encoder) (see page 432)*
ENC1 *Encoder Zero (Close Loop on External Encoder) (see page 433)*
R FD=expression *Resolution to Set Units of Position/Velocity/Acceleration (see page 461)*
R PC, PC(axis) *Position, Commanded (see page 650)*
R PMA *Position, Modulo Actual (see page 657)*
R PT=formula *Position, (Absolute) Target (see page 690)*



APPLICATION:	Program execution and flow control
DESCRIPTION:	Pause program execution; used for interrupts
EXECUTION:	Immediate
CONDITIONAL TO:	User program running
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	PAUSE:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

When executed, the PAUSE command suspends program execution until the RESUME command is received. It will not affect the current state of the Interrupt Handler (i.e., if the Interrupt Handler is enabled, it will still be enabled after a PAUSE), and its execution has no effect on the interrupt/subroutine stack.

PAUSE is primarily used to put the main part of a program to sleep when a program is 100% driven by interrupt events. Additionally, PAUSE is very useful for debugging. For instance, you may wish to pause a program at key locations when trying to isolate a problem:

```
PRINT("Debug pause, type RESUME",#13)
PAUSE
PRINT("Resumed",#13)
```

There is a separate stack for PAUSE, which will restore the state of PAUSE (that existed before a GOSUB from a terminal or an interrupt) after a RETURN or RETURNI. Any RESUME that occurred during the time the GOSUB or interrupt routine was executing will not impact the PAUSE in the previous context.

EXAMPLE:

```
EIGN(W,0,12)    'Another way to disable Travel Limits
ZS             'Clear faults
ITR(0,0,0,0,0) 'Set Int 0 for: stat word 0, bit 0,
              'shift to 0, to call C0
EITR(0)       'Enable Interrupt 0
ITRE         'Global Interrupt Enable
PAUSE        'Pause to prevent "END" from disabling
              'Interrupt, no change to stack
'RESUME must be issued externally over communications;
'it is not allowed to be compiled within a program.
END
C0           'Fault handler
  MTB:0      'Motor will turn off with Dynamic
              'breaking, tell other motors to stop.
  US(0):0    'Set User Status Bit 0 to 1 (Status
              'Word 12 bit zero)
  US(ADDR):0 'Set User Status Bit "address" to 1
              '(Status Word 12 Bit "address")
```

RETURNI**RELATED COMMANDS:**

RESUME *Resume Program Execution (see page 704)*



PC, PC(axis) Position, Commanded

APPLICATION:	Motion control
DESCRIPTION:	Commanded absolute position
EXECUTION:	Next PID sample
CONDITIONAL TO:	Mode of motion control, or drive OFF status
LIMITATIONS:	N/A
READ/REPORT:	RPC, RPC(axis)
WRITE:	Read only
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	Encoder counts
RANGE OF VALUES:	-2147483648 to 2147483647
TYPICAL VALUES:	-2147483648 to 2147483647
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	RPC(0):3, x=PC(0):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

NOTE: This command is affected by the SCALEP command. For details, see SCALEP(m,d) on page 726. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

The PC (Position, Commanded) command gets (reads) the commanded absolute position:

- =PC
Commanded position of the motor shaft as a result of motion trajectory generation. This may include a sum of concurrent moves such as a Follow mode move combined with a position move.
- =PC(0)
Equivalent to: a=PC
- =PC(1)
Reports the commanded position in the frame of reference for trajectory generator one.
- =PC(2)
Reports the commanded position in the frame of reference for trajectory generator two.

The commanded position is the calculated trajectory position. It is the reference position where the motor would ideally be positioned. However, the motor's actual position (PA) may vary — it depends on how closely the PID tuning brings the position error (EA) to 0.

NOTE: If the drive is off, then PC and PC(0) simply follow the actual position PA.

PC(1) and PC(2) are more virtually calculated values. They do not have a direct effect on the commanded position. However, they can be used to individually keep track of the relative changes in those two calculated trajectories.

EXAMPLE: (Shows the use of PC, PRA and PRC)

```
'NOTE: This example requires an external encoder.
EIGN(W,0,12) ZS      'Disable overtravel limits and clear errors
O(0)=0 O(1)=0 O(2)=0 'Zero PC(0),PC(1) and PC(2)
MF0 MFDIV=1 MFMUL=1 'Reset CTR(1) Divisor=1 Multiplier=1
MFR      'Enable Follow mode at specified ratio
MP(1)    'Mode Position in trajectory 1 while keeping Mode Follow active
PRT=0 G(2) 'Set PRT=0 and start following with phase adjust active
PRINT("Adjust external encoder to ~1000 counts.",#13)
PRINT("Then type GOSUB10",#13)
END
C10
  x=CTR(1) PRINT("The external encoder CTR(1)=",x,#13)
  x=PC(0) PRINT("PC(0)=",x,#13)
  x=PC(1) PRINT("PC(1)=",x,#13)
  x=PC(2) PRINT("PC(2)=",x,#13)
  PRINT("PRA=",PRA,#13) 'PRA will be zero because PRT=0
  PRINT("PRC=",PRC,#13) 'PRC will be zero because PRT=0
  'Set relative distance, velocity target, and Accel/Decel values
  PRT=2000 VT=5000 ADT=100 G(1) TWAIT(1) 'Start Position Relative phase adjust
  PRINT("After relative move, the values are...",#13)
  x=CTR(1) PRINT("The external encoder CTR(1)=",x,#13)
  x=PC(0) PRINT("PC(0)=",x,#13)
  x=PC(1) PRINT("PC(1)=",x,#13)
  x=PC(2) PRINT("PC(2)=",x,#13)
  PRINT("PRA=",PRA,#13) 'PRA=PRC-PositionError; note default PID tuning values
  PRINT("PRC=",PRC,#13) 'PRC=2000 because PRT=2000
  PRINT("Position Error=",EA,#13)
RETURN
```

Program output is:

```
RUN
Adjust external encoder to ~1000 counts.
Then type GOSUB10
GOSUB10
The external encoder CTR(1)=1000
PC(0)=1000
PC(1)=0
PC(2)=1000
PRA=0
PRC=0
After relative move, the values are...
The external encoder CTR(1)=1000
PC(0)=3000
PC(1)=2000
PC(2)=1000
PRA=1991
PRC=2000
Position Error=9
```

RELATED COMMANDS:

R PA *Position, Actual (see page 646)*

R PT=formula *Position, (Absolute) Target (see page 690)*

APPLICATION:	Math function
DESCRIPTION:	Gets the mathematical value pi
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	Accurate to six digits (single-precision float)
READ/REPORT:	RPI
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	3.141592
TYPICAL VALUES:	3.141592
DEFAULT VALUE:	3.141592
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The PI (pi constant) command gets the mathematical value of pi (3.141592).

The trigonometric functions SIN, COS and TAN require degrees as input. The functions ASIN, ACOS and ATAN return values in units of degrees. Therefore, this constant is useful, especially in cases where you need to convert radians to degrees.

EXAMPLE:

```
af[7]=ATAN(af[6])*180/PI 'Set af[7] to arctan result converted to radians
```

RELATED COMMANDS:

R af[index]=formula *Array Float [index] (see page 267)*

R ACOS(value) *Arccosine (see page 259)*

R ASIN(value) *Arcsine (see page 284)*

R ATAN(value) *Arctangent (see page 289)*

R COS(value) *Cosine (see page 372)*

R SIN(value) *Sine (see page 738)*

R TAN(value) *Tangent (see page 775)*

Proportional-Integral-Differential Filter Rate

APPLICATION:	Motion control
DESCRIPTION:	Set PID sample rate to basic rate
EXECUTION:	Next PID update
CONDITIONAL TO:	N/A
LIMITATIONS:	Command is not available for any Class 6 motors
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	PID Modulo samples
RANGE OF VALUES:	Valid values: 1, 2 (default), 4 and 8
TYPICAL VALUES:	N/A
DEFAULT VALUE:	PID2
FIRMWARE VERSION:	5.x (D/M); no Class 6
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

NOTE: The motor will turn off (freewheel) when this command is issued.

The PID parameter sets the PID sample rate. Valid values are PID1, PID2, PID4 and PID8. PID2 (8000 samples per second) is the default value. For details on determining the actual sample rate of your SmartMotor™, see the RSP on page 710.

During each PID sample period, the motor firmware scans and updates the encoder position, trajectory generator and serial communications ports. It uses position error to perform the PID calculation to control the servo drive stage. The user program code, if any, is executed when the microprocessor is not involved in these activities.

Both velocity and acceleration, SRC(2) and SRC(-2), are impacted by the PID setting. However, there are no effects from the PID setting on CLK and WAIT.

The values of 1, 2, 4 and 8 mean the PID filter will react and update on position error to correct drive power at different rates (refer to the next table). This does not change how code is executed, but it does change how much time is given to that execution. As a result, a program run at PID8 will typically run faster than one run at PID1. However, because the frequency of PID updates to the drive stage are changed and samples of position error are done at different intervals, PID8 will result in a more coarse or abrasive motion than PID1. Therefore, special care should be taken when using the PID command, as improper usage could result in very sporadic motion.

The current PID rate can be reported through the SAMP command.

Command	PID / Trajectory Update Rate	Period (µsec)	SAMP Report	RSP Report
PID1	16 kHz	62.5	16000	06250/5...

Command	PID / Trajectory Update Rate	Period (μsec)	SAMP Report	RSP Report
PID2	8 kHz (default)	125	8000	12500/5...
PID4	4 kHz	250	4000	25000/5...
PID8	2 kHz	500	2000	50000/5...

EXAMPLE: (comparison of the different PID values)

```

'For a 4000 count encoder SmartMotor:
'Using three fixed values under each of the PID settings
v=655360      'use to Set commanded Velocity (4000 count encoder)
a=256         'use to Set commanded Acceleration
w=1000       'use to set Wait time

PID1         'Default PID updates every servo sample
WAIT=w      'Wait time      = 1 second
VT=v        'Velocity      = 2400 RPM
ADT=a       'Accel/Decel   = 250 RPS^2

PID2         'PID updates every 2 servo samples
WAIT=w      'Wait time      = 1 second
VT=v        'Velocity      = 1200 RPM
ADT=a       'Accel/Decel   = 62.5 RPS^2

PID4         'PID updates every 4 servo samples
WAIT=w      'Wait time      = 1 second
VT=v        'Velocity      = 600 RPM
ADT=a       'Accel/Decel   = 15.625 RPS^2

PID8         'PID updates every 8 servo samples
WAIT=w      'Wait time      = 1 second
VT=v        'Velocity      = 300 RPM
ADT=a       'Accel/Decel   = 3.9063 RPS^2

PID2         'Return to Default PID
WAIT=w      'Wait time      = 1 second

END

```

In the previous example, although the values used for Velocity, Acceleration/Deceleration, and Wait times remained the same, their effect was changed by the PID setting. As a result, much care should be taken if changes are made in the middle of a program.

While the motor is motionless, the PID parameter can be changed from PID1 to PID8, to increase I/O scanning efficiency or other code execution, and then returned to PID1 just before the next move. This is a technique used to increase response time for input triggers or mathematical calculations when there is no trajectory in progress.

RELATED COMMANDS:

ADT=formula *Acceleration/Deceleration Target (see page 263)*

R AT=formula *Acceleration Target (see page 286)*

R DT=formula *Deceleration Target (see page 396)*

RSP *Report Sampling Rate and Firmware Revision (see page 710)*

R SAMP *Sampling Rate (see page 722)*

SRC(enc_src) *Source, Follow and/or Cam Encoder (see page 759)*

R VT=formula *Velocity Target (see page 828)*



APPLICATION:	Motion control
DESCRIPTION:	Gets (reads) the actual position in modulo counts
EXECUTION:	Next PID sample
CONDITIONAL TO:	PML limit
LIMITATIONS:	N/A
READ/REPORT:	RPMA
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Encoder counts
RANGE OF VALUES:	0 to 2147483646
TYPICAL VALUES:	0 to 1000000
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	RPMA:3, x=PMA:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

NOTE: This command is affected by the SCALEP command. For details, see SCALEP(m,d) on page 726. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

The PMA (Position Modulo Actual) command reads the actual motor position in modulo counts. This counter is affected by the O= and OSH= commands.

NOTE: The value of PMA is always positive.

The PML command is used to configure the modulo-rollover point. When the encoder travels in a positive direction, the modulo count will increase. When the count equals or exceeds the value set by PML, then the counter rolls over to 0. If the encoder travels in the negative direction, the count decreases until it is less than 0. At that point, it will be automatically rolled to PML-1.

The PML command will reset the PMA counter to 0.

The RPMA (report PMA) command updates according to the active encoder selected by ENC0 or ENC1.

EXAMPLE: (Shows the use of the PMA, PML and PMT commands)

```

EIGN(W,0)           'Make all onboard I/O inputs
ZS                 'Clear errors
MP VT=20000 ADT=100 O=0 'Mode Position, velocity, accel/decel, zero enc.
PML=RES           'RES on NEMA 23 is 4000, NEMA 34 is 8000
GOSUB(10)         'Print positions
PT=6000 G TWAIT   'Absolute move
WAIT=1000         'Wait 1 second
GOSUB(10)         'Print positions
PMT=3000 G TWAIT  'Modulo move
WAIT=1000         'Wait 1 second
GOSUB(10)         'Print positions
PMT=1000 G TWAIT  'Modulo move
WAIT=1000         'Wait 1 second
GOSUB(10)         'Print positions
END
C10               'Subroutine 10
PRINT("Actual absolute position: ",PA,#13)
PRINT("Actual modulo position: ",PMA,#13)
RETURN           'Return to command after GOSUB

```

RELATED COMMANDS:

ENC0 *Encoder Zero (Close Loop on Internal Encoder) (see page 432)*
 ENC1 *Encoder Zero (Close Loop on External Encoder) (see page 433)*
 R PML=formula *Modulo Position Limit (see page 659)*
 R PMT=formula *Position, Modulo Target (see page 661)*



PML=formula

Modulo Position Limit

APPLICATION:	Motion control
DESCRIPTION:	Get/set the modulo position limit
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	PML minimum and maximum values are speed dependent (see details)
READ/REPORT:	RPML
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	Encoder counts
RANGE OF VALUES:	1 - 2147483647 PML minimum and maximum values are speed dependent (see details)
TYPICAL VALUES:	100 - 1000000 PML minimum and maximum values are speed dependent (see details)
DEFAULT VALUE:	1000
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	PML:3=1234, a=PML:3, RPML:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

NOTE: This command is affected by the SCALEP command. For details, see SCALEP(m,d) on page 726. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

The PML command is used to get (read) or set the modulo limit. The modulo counter (PMA) can have a range of values where the lowest value is 0 and the highest value is PML-1. For more details, see PMA on page 657.

NOTE: PML resets PMA to 0.

The PML value must be greater than the motor speed in terms of encoder counts per PID sample. However, the PML value must be smaller than 2147483647 encoder counts per sample — this prevents overflow.

EXAMPLE: (Shows the use of the PMA, PML and PMT commands)

```

EIGN(W,0)           'Make all onboard I/O inputs
ZS                 'Clear errors
MP VT=20000 ADT=100 O=0 'Mode Position, velocity, accel/decel, zero enc.
PML=RES           'RES on NEMA 23 is 4000, NEMA 34 is 8000
GOSUB(10)         'Print positions
PT=6000 G TWAIT   'Absolute move
WAIT=1000        'Wait 1 second
GOSUB(10)         'Print positions
PMT=3000 G TWAIT  'Modulo move
WAIT=1000        'Wait 1 second
GOSUB(10)         'Print positions
PMT=1000 G TWAIT  'Modulo move
WAIT=1000        'Wait 1 second
GOSUB(10)         'Print positions
END
C10               'Subroutine 10
PRINT("Actual absolute position: ",PA,#13)
PRINT("Actual modulo position: ",PMA,#13)
RETURN           'Return to command after GOSUB

```

RELATED COMMANDS:

ENC0 *Encoder Zero (Close Loop on Internal Encoder) (see page 432)*
 ENC1 *Encoder Zero (Close Loop on External Encoder) (see page 433)*
 R PMA *Position, Modulo Actual (see page 657)*
 R PMT=formula *Position, Modulo Target (see page 661)*



PMT=formula

Position, Modulo Target

APPLICATION:	Motion control
DESCRIPTION:	Gets/sets the modulo target
EXECUTION:	Buffered until a G command is issued
CONDITIONAL TO:	PML for allowed range
LIMITATIONS:	N/A
READ/REPORT:	RPMT
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	Encoder counts
RANGE OF VALUES:	0 to 2147483646
TYPICAL VALUES:	0 to 1000000
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	PMT:3=1234, a=PMT:3, RPMT:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

NOTE: This command is affected by the SCALEP command. For details, see SCALEP(m,d) on page 726. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

The PMT command is used to get (read) or set the modulo target. When in position mode (MP), the trajectory will compute the shortest distance to get from the current position to the requested position with respect to the PML limit. Therefore, the motor may move in either a clockwise or counterclockwise direction based on the one that produces the shortest motion in modulo terms.

For example, assume PML=4000 and the current commanded position (PC) is 0. If PMT=3000, then the motor will actually move counterclockwise by 1000 counts. At the end of this move, PC will be -1000 and PMA will be 3000.

Because PMA is an actual encoder position and not given in terms of commanded trajectory, the PMT target will take into account the current position error. This helps to prevent accumulated error in the target position based on the position error at the time the move is started. Note that the PT, PRT and PMT targets all act on the ideal currently commanded position instead of the actual position, which may be subject to position error (EA).

EXAMPLE: (Shows the use of the PMA, PML and PMT commands)

```

EIGN(W,0)           'Make all onboard I/O inputs
ZS                 'Clear errors
MP VT=20000 ADT=100 O=0 'Mode Position, velocity, accel/decel, zero enc.
PML=RES           'RES on NEMA 23 is 4000, NEMA 34 is 8000
GOSUB(10)         'Print positions
PT=6000 G TWAIT   'Absolute move
WAIT=1000        'Wait 1 second
GOSUB(10)         'Print positions
PMT=3000 G TWAIT  'Modulo move
WAIT=1000        'Wait 1 second
GOSUB(10)         'Print positions
PMT=1000 G TWAIT  'Modulo move
WAIT=1000        'Wait 1 second
GOSUB(10)         'Print positions
END
C10               'Subroutine 10
PRINT("Actual absolute position: ",PA,#13)
PRINT("Actual modulo position: ",PMA,#13)
RETURN           'Return to command after GOSUB

```

RELATED COMMANDS:

ENC0 *Encoder Zero (Close Loop on Internal Encoder) (see page 432)*
 ENC1 *Encoder Zero (Close Loop on External Encoder) (see page 433)*
 R PMA *Position, Modulo Actual (see page 657)*
 R PML=formula *Modulo Position Limit (see page 659)*

PRA Position, Relative Actual

APPLICATION:	Motion control
DESCRIPTION:	Get actual position relative to move start
EXECUTION:	Next PID sample
CONDITIONAL TO:	MV or MP mode
LIMITATIONS:	Origin change affects the value reported from PRA during the next move
READ/REPORT:	RPRA
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Encoder counts
RANGE OF VALUES:	-2147483648 to 2147483647
TYPICAL VALUES:	-2147483648 to 2147483647
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

NOTE: This command is affected by the SCALEP command. For details, see SCALEP(m,d) on page 726. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

The PRA (Position Relative Actual) command is used to get (read) the actual position relative to the point where the move started. This includes position error in addition to the commanded relative move.

NOTE: This command is primarily for use in MP mode but will also work in MV mode. Other modes of motion do not support this command.

The value reported from PRA is calculated using the value of PC(1) at the time the move began. Therefore, for accurate PRA calculation, if the origin is changed (i.e., O=0), then also correct trajectory 1 to the same value at that time: O(1)=0.

EXAMPLE: (Shows the use of PC, PRA and PRC)

```
'NOTE: This example requires an external encoder.
EIGN(W,0,12) ZS      'Disable overtravel limits and clear errors
O(0)=0 O(1)=0 O(2)=0 'Zero PC(0),PC(1) and PC(2)
MF0 MFDIV=1 MFMUL=1 'Reset CTR(1) Divisor=1 Multiplier=1
MFR      'Enable Follow mode at specified ratio
MP(1)    'Mode Position in trajectory 1 while keeping Mode Follow active
PRT=0 G(2) 'Set PRT=0 and start following with phase adjust active
PRINT("Adjust external encoder to ~1000 counts.",#13)
PRINT("Then type GOSUB10",#13)
END
C10
  x=CTR(1) PRINT("The external encoder CTR(1)=",x,#13)
  x=PC(0) PRINT("PC(0)=",x,#13)
  x=PC(1) PRINT("PC(1)=",x,#13)
  x=PC(2) PRINT("PC(2)=",x,#13)
  PRINT("PRA=",PRA,#13) 'PRA will be zero because PRT=0
  PRINT("PRC=",PRC,#13) 'PRC will be zero because PRT=0
  'Set relative distance, velocity target, and Accel/Decel values
  PRT=2000 VT=5000 ADT=100 G(1) TWAIT(1) 'Start Position Relative phase adjust
  PRINT("After relative move, the values are...",#13)
  x=CTR(1) PRINT("The external encoder CTR(1)=",x,#13)
  x=PC(0) PRINT("PC(0)=",x,#13)
  x=PC(1) PRINT("PC(1)=",x,#13)
  x=PC(2) PRINT("PC(2)=",x,#13)
  PRINT("PRA=",PRA,#13) 'PRA=PRC-PositionError; note default PID tuning values
  PRINT("PRC=",PRC,#13) 'PRC=2000 because PRT=2000
  PRINT("Position Error=",EA,#13)
RETURN
```

Program output is:

```
RUN
Adjust external encoder to ~1000 counts.
Then type GOSUB10
GOSUB10
The external encoder CTR(1)=1000
PC(0)=1000
PC(1)=0
PC(2)=1000
PRA=0
PRC=0
After relative move, the values are...
The external encoder CTR(1)=1000
PC(0)=3000
PC(1)=2000
PC(2)=1000
PRA=1991
PRC=2000
Position Error=9
```


RELATED COMMANDS:

R PA *Position, Actual (see page 646)*

R PC, PC(axis) *Position, Commanded (see page 650)*

R PRC *Position, Relative Commanded (see page 666)*

PRC Position, Relative Commanded

APPLICATION:	Motion control
DESCRIPTION:	Get commanded position relative to move start
EXECUTION:	Next PID sample
CONDITIONAL TO:	MV or MP mode
LIMITATIONS:	Origin change affects the value reported from PRA during the next move
READ/REPORT:	RPRC
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Encoder counts
RANGE OF VALUES:	-2147483648 to 2147483647
TYPICAL VALUES:	-2147483648 to 2147483647
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

NOTE: This command is affected by the SCALEP command. For details, see SCALEP(m,d) on page 726. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

The PRC (Position Relative Commanded) command is used to get (read) the commanded position relative to the point where the move started. This mode does not consider position error; it only reports the change in the trajectory-calculated position from the start of the move.

NOTE: This is primarily for use in MP mode but will also work in MV mode. Other motion modes do not support this command.

The value reported from PRC is calculated using the value of PC(1) at the time the move began. Therefore, for accurate PRC calculations, if the origin is changed (i.e., O=0), then also correct trajectory 1 to the same value at that time: O(1)=0.

EXAMPLE: (Shows the use of PC, PRA and PRC)

```
'NOTE: This example requires an external encoder.
EIGN(W,0,12) ZS      'Disable overtravel limits and clear errors
O(0)=0 O(1)=0 O(2)=0 'Zero PC(0),PC(1) and PC(2)
MF0 MFDIV=1 MFMUL=1 'Reset CTR(1) Divisor=1 Multiplier=1
MFR      'Enable Follow mode at specified ratio
MP(1)    'Mode Position in trajectory 1 while keeping Mode Follow active
PRT=0 G(2) 'Set PRT=0 and start following with phase adjust active
PRINT("Adjust external encoder to ~1000 counts.",#13)
PRINT("Then type GOSUB10",#13)
END
C10
  x=CTR(1) PRINT("The external encoder CTR(1)=",x,#13)
  x=PC(0) PRINT("PC(0)=",x,#13)
  x=PC(1) PRINT("PC(1)=",x,#13)
  x=PC(2) PRINT("PC(2)=",x,#13)
  PRINT("PRA=",PRA,#13) 'PRA will be zero because PRT=0
  PRINT("PRC=",PRC,#13) 'PRC will be zero because PRT=0
  'Set relative distance, velocity target, and Accel/Decel values
  PRT=2000 VT=5000 ADT=100 G(1) TWAIT(1) 'Start Position Relative phase adjust
  PRINT("After relative move, the values are...",#13)
  x=CTR(1) PRINT("The external encoder CTR(1)=",x,#13)
  x=PC(0) PRINT("PC(0)=",x,#13)
  x=PC(1) PRINT("PC(1)=",x,#13)
  x=PC(2) PRINT("PC(2)=",x,#13)
  PRINT("PRA=",PRA,#13) 'PRA=PRC-PositionError; note default PID tuning values
  PRINT("PRC=",PRC,#13) 'PRC=2000 because PRT=2000
  PRINT("Position Error=",EA,#13)
RETURN
```

Program output is:

```
RUN
Adjust external encoder to ~1000 counts.
Then type GOSUB10
GOSUB10
The external encoder CTR(1)=1000
PC(0)=1000
PC(1)=0
PC(2)=1000
PRA=0
PRC=0
After relative move, the values are...
The external encoder CTR(1)=1000
PC(0)=3000
PC(1)=2000
PC(2)=1000
PRA=1991
PRC=2000
Position Error=9
```

RELATED COMMANDS:

R PA *Position, Actual (see page 646)*

R PC, PC(axis) *Position, Commanded (see page 650)*

R PRA *Position, Relative Actual (see page 663)*

PRINT(...)

Print Data to Communications Port

APPLICATION:	Data conversion
DESCRIPTION:	Serial communications PRINT function
EXECUTION:	Immediate, at current baud rate
CONDITIONAL TO:	Appropriate communications port open (see details)
LIMITATIONS:	Maximum command length: 63 characters (includes the PRINT statement) PRINT command not executed until transmit buffer is cleared from previous PRINT statements
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	See details
TYPICAL VALUES:	See details
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

For Class 6 motors, the PRINT command corresponds with the value of STDOUT, which is 8 (USB port) by default for M-style and 0 (COM 0/RS-232 port) by default for D-style. For STDOUT details, see `STDOUT=formula` on page 764.

- To explicitly output to the Class 6 SmartMotor's communication port 0, use the PRINT0 command. For details, see PRINT0(...) on page 673.
- To explicitly output to the Class 6 SmartMotor's USB port, use the PRINT8 command. For details, see PRINT8(...) on page 680.

For Class 5 motors, the PRINT command outputs only to communications port 0.

One or more items (see the next list) may be transmitted using the PRINT command. The items are separated with commas. Therefore, the number of arguments to the PRINT command is flexible.

- Strings of ASCII text can be specified with quotes: PRINT("Hello world")
- Variables may be specified: PRINT(a), PRINT(al[0]), PRINT(af[0])
These values will print in decimal form (with ASCII character encoding).
- System variables may be printed as long as they do not require () parenthesis: PRINT(VA)
- Literal values for characters may be specified (one 8-bit character at a time): PRINT(#32)

- Binary data may be specified (one 8-bit character at a time): PRINT(#ab[0], #ab[1])

This is useful in data mode when communicating with non-ASCII mode systems.

Combinations of these methods are permitted:

```
PRINT("The value of a is: ",a,#13)
```

This prints the text, the actual value of variable "a" (as a decimal value in ASCII text), and the newline with the #13.

PRINT() commands are typically entered in a user program to send output to a terminal for display, communicate with third party devices, or send commands to other motors.

Raw ASCII code values are prefixed by the # sign, as shown in the next table:

Character	Format
space	#32
tab	#9
carriage return	#13
line feed	#10



CAUTION: Do not use a comment marker (!) within PRINT(). It will cause a compiler error.

PRINT() will wait to begin execution until previous commands have completed transmission from the transmit buffer.

There is a practical difference between PRINT(a,b,c) and the sequence PRINT(a) PRINT(b) PRINT(c). Executing from within a program PRINT(a,b,c) will output the values of a, b, and c without the possibility of another command from the terminal interfering. However, executing PRINT(a) PRINT(b) PRINT(c) from within a program while the host terminal is transmitting GOSUB5 to the motor could lead to the GOSUB5 routine executing between the PRINT commands, which would result in the PRINT sequence not outputting as desired.

The PRINT buffer size is 31 bytes. However, that does not impact the timing (unless a single PRINT statement is generating a very long numeric output from variables—then it should be broken apart into multiple PRINT statements). Additionally,

- PRINT waits until the buffer is totally empty before starting (so that each PRINT can buffer the same amount of data)
- The buffer is there to ensure the PRINT can move on to the next line of code while those characters are transmitting
- The next PRINT that is encountered will wait until the current PRINT finishes. Report commands in a program like RPA will act the same as a PRINT in terms of timing (but x=PA is not actually printing anything, so there is no delay).

Also, see the EXAMPLE: (Print time delay), later in this topic.

EXAMPLE:

```

OFF
KP=100      'Set Proportional Gain
O=1234      'Set origin to 1234
a=1
b=2
PRINT("Demonstration:",#13)
PRINT("a=",a)
PRINT(" and b=",b,#13)
PRINT("a+b=",a+b,#13)
PRINT("Position: ",PA,#13)
WAIT=10      'Allow time for serial buffer processing
PRINT("KP=",KP,#13)
PRINT("Hello World",#13,#13)
PRINT("Run Subroutines",#13)
WAIT=10
PRINT(#128,"GOSUB5 ",#13)  'Tell all motors to run subroutine
C5
  WAIT=10
  PRINT(#129,"GOSUB10",#13) 'Tell Motor-1 to run subroutine
C10
  WAIT=10
  PRINT(#130,"GOSUB20",#13) 'Tell Motor-2 to run subroutine
C20
  WAIT=10
  PRINT(#131,"GOSUB30",#13) 'Tell Motor-3 to run subroutine
C30
  x=123
  PRINT(#132,"GOSUB",x,#13) 'Tell Motor-4 to run subroutine
C123
  v=100000
  a=100
  p=2000
  PRINT(#130,"ADT=",a," VT=",v,#13) 'Set speed and accel/decel in motor 2
  WAIT=10
  PRINT(#130,"MP PT=",p," G",#13)  'Command Motor-2 to position 2000
  WAIT=10
  PRINT(#13,#13,"End of Demonstration.",#13)
END

```

Program output is:

```

Demonstration:
a=1 and b=2
a+b=3
Position: 1234
KP=100
Hello World

```

```

Run Subroutines
GOSUB5
GOSUB10
GOSUB20
GOSUB30
GOSUB123
ADT=100 VT=100000
MP PT=2000 G

```

End of Demonstration.

EXAMPLE: (Print time delay)

```

RUN?
WAIT=1000      'Make sure anything previous has finished printing
' O=1000000000 'For RPA test below
CLK=0
a=CLK
PRINT("asdfasdklfjasldkfjaslkdj",#13)
b=CLK
' x=PA          'Does not delay timing
' RPA           'Delays the same as PRINT
PRINT("01234567890123456789012345678",#13)
c=CLK
PRINT("tyuityuityuityuityuity",#13)
d=CLK

PRINT("a time: ",a,#13)
PRINT("b time: ",b,#13)
PRINT("c time: ",c,#13)
PRINT("d time: ",d,#13)

END

```

RELATED COMMANDS:

R BAUD(channel)=formula *Set BAUD Rate (RS-232 and RS-485) (see page 303)*
CCHN(type,channel) *Close Communications Channel (RS-232 or RS-485) (see page 365)*
OCHN(...) *Open Channel (see page 632)*
PRINT0(...) *Print Data to Communications Port 0 (see page 673)*
PRINT1(...) *Print Data to Communications Port 1 (see page 677)*
PRINT8(...) *Print Data to USB Port (see page 680)*

PRINTO(...)

Print Data to Communications Port 0

APPLICATION:	Data conversion
DESCRIPTION:	Serial communications channel 0 PRINT function
EXECUTION:	Immediate, at current baud rate
CONDITIONAL TO:	Channel 0 serial port open
LIMITATIONS:	Maximum command length: 63 characters (includes the PRINTO statement) PRINTO command not executed until transmit buffer is cleared from previous PRINT statements
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	See details
TYPICAL VALUES:	See details
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	6.x (D/M); no Class 5
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

NOTE: For firmware versions earlier than version 6.0, refer to the PRINT command. For details, see PRINT(...) on page 669.

One or more items (see the next list) may be transmitted from COM 0 using the PRINTO command. The items are separated with commas. Therefore, the number of arguments to the PRINTO command is flexible.

- Strings of ASCII text can be specified with quotes: PRINTO("Hello world")
- Variables may be specified: PRINTO(a), PRINTO(a[0]), PRINTO(af[0])
These values will print in decimal form (with ASCII character encoding).
- System variables may be printed as long as they do not require () parenthesis: PRINTO(VA)
- Literal values for characters may be specified (one 8-bit character at a time): PRINTO(#32)
- Binary data may be specified (one 8-bit character at a time): PRINTO(#ab[0], #ab[1])

This is useful in data mode when communicating with non-ASCII mode systems.

Combinations of these methods are permitted:

```
PRINTO("The value of a is: ",a,#13)
```

This prints the text, the actual value of variable "a" (as a decimal value in ASCII text), and the newline with the #13.

PRINTO() commands are typically entered in a user program to send output to a terminal for display, communicate with third party devices, or send commands to other motors.

Raw ASCII code values are prefixed by the # sign, as shown in the next table:

Character	Format
space	#32
tab	#9
carriage return	#13
line feed	#10



CAUTION: Do not use a comment marker (') within PRINTO(). It will cause a compiler error.

PRINTO() will wait to begin execution until previous commands have completed transmission from the transmit buffer.

There is a practical difference between PRINTO(a,b,c) and the sequence PRINTO(a) PRINTO(b) PRINTO(c). Executing from within a program PRINTO(a,b,c) will output the values of a, b, and c without the possibility of another command from the terminal interfering. However, executing PRINTO(a) PRINTO(b) PRINTO(c) from within a program while the host terminal is transmitting GOSUB5 to the motor could lead to the GOSUB5 routine executing between the PRINTO commands, which would result in the PRINTO sequence not outputting as desired.

EXAMPLE:

```

OFF
KP=100      'Set Proportional Gain
O=1234      'Set origin to 1234
a=1
b=2
PRINTO ("Demonstration:", #13)
PRINTO ("a=", a)
PRINTO (" and b=", b, #13)
PRINTO ("a+b=", a+b, #13)
PRINTO ("Position: ", PA, #13)
WAIT=10                                'Allow time for serial buffer processing
PRINTO ("KP=", KP, #13)
PRINTO ("Hello World", #13, #13)
PRINTO ("Run Subroutines", #13)
WAIT=10
PRINTO (#128, "GOSUB5 ", #13)           'tell all motors to run subroutine
C5
  WAIT=10
  PRINTO (#129, "GOSUB10", #13)         'Tell Motor-1 to run subroutine
C10
  WAIT=10
  PRINTO (#130, "GOSUB20", #13)        'Tell Motor-2 to run subroutine
C20
  WAIT=10
  PRINTO (#131, "GOSUB30", #13)        'Tell Motor-3 to run subroutine
C30
  x=123
  PRINTO (#132, "GOSUB", x, #13)        'Tell Motor-4 to run subroutine
C123
  v=100000
  a=100
  p=2000
  PRINTO (#130, "ADT=", a, " VT=", v, #13) 'Set speed and accel/decel in motor 2
  WAIT=10
  PRINTO (#130, "MP PT=", p, " G", #13)  'Command Motor-2 to position 2000
  WAIT=10
  PRINTO (#13, #13, "End of Demonstration.", #13)
END

```

Program output is:

Demonstration:

a=1 and b=2

a+b=3

Position: 1234

KP=100

Hello World

Run Subroutines

GOSUB5

GOSUB10

GOSUB20

GOSUB30

GOSUB123

ADT=100 VT=100000

MP PT=2000 G

End of Demonstration.

RELATED COMMANDS:

R BAUD(channel)=formula *Set BAUD Rate (RS-232 and RS-485) (see page 303)*

CCHN(type,channel) *Close Communications Channel (RS-232 or RS-485) (see page 365)*

OCHN(...) *Open Channel (see page 632)*

PRINT(...) *Print Data to Communications Port (see page 669)*

PRINT1(...) *Print Data to Communications Port 1 (see page 677)*

PRINT8(...) *Print Data to USB Port (see page 680)*

PRINT1(...)

Print Data to Communications Port 1

APPLICATION:	Data conversion
DESCRIPTION:	Serial communications channel 1 PRINT function
EXECUTION:	Immediate, at current baud rate
CONDITIONAL TO:	Channel 1 serial port open
LIMITATIONS:	Maximum command length: 63 characters (includes the PRINT1 statement) PRINT1 command not executed until transmit buffer is cleared from previous PRINT1 statements
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	See details
TYPICAL VALUES:	See details
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.0.x, 5.16.x or 5.32.x series (D); 6.4.2.x (D)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

NOTE: The PRINT1 command is not available for version 6.0 or later firmware.

The PRINT1() command is used to transmit (output) data to serial communications channel 1. On a Class 5 D-style motor, I/O pins 4 and 5 are used, which is also known as the secondary serial channel; on a Class 6 D-style motor, I/O pins 19 and 20 are used. This option is not available for the M-style motor.

NOTE: The proper OCHN command is required before using PRINT1().

PRINT1 explicitly outputs to COM 1(where available) and is not influenced by the STDOUT setting. Refer to PRINT(...) on page 669 for more PRINT/PRINT1 details.

EXAMPLE:

```

OFF
KP=100      'Set Proportional Gain
O=1234      'Set origin to 1234
a=1
b=2
PRINT1 ("Demonstration:",#13)
PRINT1 ("a=",a)
PRINT1 (" and b=",b,#13)
PRINT1 ("a+b=",a+b,#13)
PRINT1 ("Position: ",PA,#13)
WAIT=10     'Allow time for serial buffer processing
PRINT1 ("KP=",KP,#13)
PRINT1 ("Hello World",#13,#13)
PRINT1 ("Run Subroutines",#13)
WAIT=10
PRINT1 (#128,"GOSUB5 ",#13)      'Tell all motors to run subroutine
C5
  WAIT=10
  PRINT1 (#129,"GOSUB10",#13)    'Tell Motor-1 to run subroutine
C10
  WAIT=10
  PRINT1 (#130,"GOSUB20",#13)   'Tell Motor-2 to run subroutine
C20
  WAIT=10
  PRINT1 (#131,"GOSUB30",#13)   'Tell Motor-3 to run subroutine
C30
  x=123
  PRINT1 (#132,"GOSUB",x,#13)    'Tell Motor-4 to run subroutine
C123
  v=100000
  a=100
  p=2000
  PRINT1 (#130,"ADT=",a," VT=",v,#13) 'Set speed and accel/decel in motor 2
  WAIT=10
  PRINT1 (#130,"MP PT=",p," G",#13) 'Command Motor-2 to position 2000
  WAIT=10
  PRINT1 (#13,#13,"End of Demonstration.",#13)
END

```

Program output is:

Demonstration:

a=1 and b=2

a+b=3

Position: 1234

KP=100

Hello World

Run Subroutines

GOSUB5

GOSUB10

GOSUB20

GOSUB30

GOSUB123

ADT=100 VT=100000

MP PT=2000 G

End of Demonstration.

RELATED COMMANDS:

R BAUD(channel)=formula *Set BAUD Rate (RS-232 and RS-485) (see page 303)*

CCHN(type,channel) *Close Communications Channel (RS-232 or RS-485) (see page 365)*

OCHN(...) *Open Channel (see page 632)*

PRINT(...) *Print Data to Communications Port (see page 669)*

PRINT0(...) *Print Data to Communications Port 0 (see page 673)*

PRINT8(...) *Print Data to USB Port (see page 680)*

PRINT8(...)

Print Data to USB Port

APPLICATION:	Data conversion
DESCRIPTION:	USB port PRINT function
EXECUTION:	Immediate, at current baud rate
CONDITIONAL TO:	USB port open
LIMITATIONS:	Maximum command length: 63 characters (includes the PRINT8 statement) PRINT8 command not executed until transmit buffer is cleared from previous PRINT8 statements
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	See details
TYPICAL VALUES:	See details
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	6.x (D/M); no Class 5
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The PRINT8() command is used to transmit (output) data to the Class 6 SmartMotor's USB port.

The only difference between PRINT0 and PRINT8 is the destination (output): PRINT0 explicitly outputs to COM 0, PRINT8 explicitly outputs to the USB port. Refer to PRINT0(...) on page 673 for more PRINT details.

EXAMPLE:

```
OFF
KP=100      'Set Proportional Gain
O=1234      'Set origin to 1234
a=1
b=2
PRINT8 ("Demonstration:",#13)
PRINT8 ("a=",a)
PRINT8 (" and b=",b,#13)
PRINT8 ("a+b=",a+b,#13)
PRINT8 ("Position: ",PA,#13)
WAIT=10     'Allow time for serial buffer processing
PRINT8 ("KP=",KP,#13)
PRINT8 ("Hello World",#13,#13)
PRINT8 ("Run Subroutines",#13)
WAIT=10
PRINT8 (#128,"GOSUB5 ",#13)      'Tell all motors to run subroutine
C5
  WAIT=10
  PRINT8 (#129,"GOSUB10",#13)    'Tell Motor-1 to run subroutine
C10
  WAIT=10
  PRINT8 (#130,"GOSUB20",#13)   'Tell Motor-2 to run subroutine
C20
  WAIT=10
  PRINT8 (#131,"GOSUB30",#13)   'Tell Motor-3 to run subroutine
C30
  x=123
  PRINT8 (#132,"GOSUB",x,#13)   'Tell Motor-4 to run subroutine
C123
  v=100000
  a=100
  p=2000
  PRINT8 (#130,"ADT=",a," VT=",v,#13) 'Set speed and accel/decel in motor 2
  WAIT=10
  PRINT8 (#130,"MP PT=",p," G",#13)  'Command Motor-2 to position 2000
  WAIT=10
  PRINT8 (#13,#13,"End of Demonstration.",#13)
END
```

Program output is:

Demonstration:

a=1 and b=2

a+b=3

Position: 1234

KP=100

Hello World

Run Subroutines

GOSUB5

GOSUB10

GOSUB20

GOSUB30

GOSUB123

ADT=100 VT=100000

MP PT=2000 G

End of Demonstration.

RELATED COMMANDS:

R BAUD(channel)=formula Set BAUD Rate (RS-232 and RS-485) (see page 303)

CCHN(type,channel) Close Communications Channel (RS-232 or RS-485) (see page 365)

OCHN(...) Open Channel (see page 632)

PRINT(...) Print Data to Communications Port (see page 669)

PRINT0(...) Print Data to Communications Port 0 (see page 673)

PRINT1(...) Print Data to Communications Port 1 (see page 677)



PRT=formula

Position, Relative Target

APPLICATION:	Motion control
DESCRIPTION:	Gets/sets the relative target position
EXECUTION:	Buffered until a G command is issued
CONDITIONAL TO:	MP (position mode)
LIMITATIONS:	Wait until previous relative move is complete before commanding G
READ/REPORT:	RPRT
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	Encoder counts
RANGE OF VALUES:	-2147483648 to 2147483647
TYPICAL VALUES:	-2147483648 to 2147483647
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	PRT:3=1234, a=PRT:3, RPRT:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

NOTE: This command is affected by the SCALEP command. For details, see SCALEP(m,d) on page 726. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

The PRT command is used to get (read) or set the relative target position. It allows a relative-distance move to be specified when the motor is in Position mode.

- PRT=formula
Sets the relative target position.
- x=PRT
Gets the relative target position and assigns it to the variable x.

The target is in terms of encoder counts to travel in the range -2147483648 to +2147483647. Either during or after a move, the relative distance will be added to the current trajectory position and not the actual position. Therefore, if a previous move is still in progress, then the relative distance will be added to the current trajectory position at the time that G is commanded.

NOTE: If the total distance traveled needs to directly correspond to the number of moves made, then make sure a move has finished before commanding G again.

Status word 3, bit 8 reports 1 when acting on a PRT target (relative position).

PRT acts on the ideal currently commanded position and not the actual position, which may be subject to position error (EA).

EXAMPLE: (Dual-trajectory spool winding program)

```

SRC (2)           'Set signal source to internal 8K counts/sec
MFMUL=100        'Default is 1
MFDIV=100        'Default is 1
MFA (500,1)      'Set ascend ratio distance of 500 follower counts
MFD (500,1)      'Set descend ratio distance of 500 follower counts
MFR (2)          'Enable Follow mode at specified ratio for SECOND TRAJECTORY
MFSLEW (8000,1)  'Stay at slew ratio for 8000 counts of the follower
MFSDC (100,1)    'Dwell for 100 counts, auto repeat in reverse direction
G (2)           'Begin to follow controller signal in SECOND trajectory
MP (1)          'Set FIRST TRAJECTORY mode to Position mode
VT=100000       'Set velocity to run over top of gearing
ADT=100         'Set accel/decel to run over gearing
PRT=1000        'Set relative move
G (1)           'Shift all motion 1000 counts in positive direction

```

RELATED COMMANDS:

ADT=formula *Acceleration/Deceleration Target (see page 263)*
R EL=formula *Error Limit (see page 426)*
G *Start Motion (GO) (see page 473)*
MP *Mode Position (see page 613)*
R PRA *Position, Relative Actual (see page 663)*
R PRC *Position, Relative Commanded (see page 666)*
R PT=formula *Position, (Absolute) Target (see page 690)*
R VT=formula *Velocity Target (see page 828)*

PRTS(...) Position, Relative Target, Synchronized

APPLICATION:	Motion control
DESCRIPTION:	Sets the synchronized relative target position
EXECUTION:	Buffered until a GS is issued
CONDITIONAL TO:	Motors in the sync group are positioned at the their target position: PT=PC
LIMITATIONS:	ADTS and VTS commands must be set before issuing this command Up to three axes of motion; must be orthogonal (Cartesian) coordinates, not radial or polar
READ/REPORT:	None
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	Encoder counts
RANGE OF VALUES:	Input: Position: -2147483648 to 2147483647 Axis: 1-127
TYPICAL VALUES:	Input: Position: -2147483648 to 2147483647 Axis: 1-127
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M), not ver. 5.32.x.x; 6.x (D/M), M requires EIP option
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

NOTE: This command requires a Combitronic-supported motor. Although this command does not support Combitronic syntax, it does use Combitronic communication to pass information between other motors.

NOTE: This command is affected by the SCALEP command. For details, see SCALEP(m,d) on page 726. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

The PRTS command is used to set the synchronized relative target position. It allows you to identify two or three axis positions (posn) and their associated CAN addresses (axisn) to cause a synchronized, relative, multi-axis move where the combined path velocity is controlled as shown:

```
PRTS (pos1;axis1, pos2;axis2 [, pos3;axis3])
```

NOTE: There is a three-axis limitation for this command.

Additional axes can be synchronized using the PTSS and PRTSS commands.

The synchronized motion is initiated with a GS command. For more details, see Synchronized Motion on page 179.

Some gantry-type, multiple-axis machines have two motors operating the same axis of motion (see the next figure). Below is the full syntax for the PTS command, which shows additional/optional parameters (enclosed in braces "{ }") for support of two motors operating the same axis. The optional parameter contains the motor address for the second motor of the axis. (For the PRTS command, replace PTS with PRTS.)

```
PTS(pos1;addr1{;addr1'},pos2;addr2{;addr2'}[,pos3;addr3{;axis3'}])
```

This is illustrated in the next example. (If you are using the PRTS command, substitute PRTS in place of PTS below.)

Position target X = 2000

Position target Y = 1000

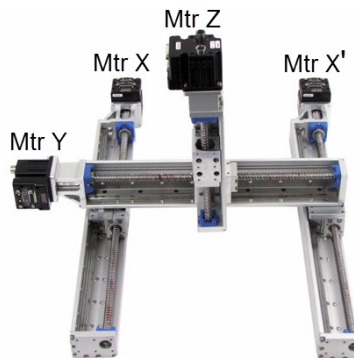
Position target Z = 500

Motor address X = 5

Motor address X' = 6

Motor address Y = 7

Motor address Z = 8



```
PTS (2000;5;6,1000;7) 'Two-motor X axis (X, X'), plus Y axis
```

```
PTS (2000;5;6,1000;7,500;8) 'Two-motor X axis (X, X'), plus Y & Z axes
```

In these cases, the same position, velocity and acceleration data sent to motor address 5 is also sent to motor address 6, with both motors driving the gantry's X axis.

EXAMPLE: (3-axis synchronized relative move to position x:y:z for motors 1, 2 and 3)

This sample code may be executed by any motor sharing the same CAN bus network with the motors being commanded to move.

NOTE: Ensure no motor drive faults exist prior to calling this subroutine.

```
. . .
C20
OFF:0 MP:0 PRT:0=0 G TWAIT          'Initialize to stationary in position mode
PT:1=PC:1 PT:2=PC:2 PT:3=PC:3      'Set target and commanded positions equal
WAIT=50
VTS=v                               'Set target path velocity
ADTS=a                              'Set target path accel/decel
PRTS (x;1, y;2, z;3)                'Use Position Target Synchronized moves
PRTSS (a;4)                          'Supplemental synchronized relative target
IF PTSD!=0                           'Prevent 0-length (divide by zero) move
  GS                                  'Go Synchronized
  TSWAIT                             'Wait until path move time is complete
ENDIF
RETURN
```

For additional examples, see A Note About PTS and PRTS on page 181.

RELATED COMMANDS:

ADTS=formula *Acceleration/Deceleration Target, Synchronized (see page 265)*

GS *Start Synchronized Motion (GO Synchronized) (see page 487)*

PRTSS(...) *Position, Relative Target, Synchronized, Supplemental (see page 688)*

PTS(...) *Position Target, Synchronized (see page 692)*

PTSS(...) *Position Target, Synchronized Supplemental (see page 696)*

TSWAIT *Trajectory Synchronized Wait (see page 788)*

VTS=formula *Velocity Target, Synchronized Move (see page 831)*

Position, Relative Target, Synchronized, Supplemental

APPLICATION:	Motion control
DESCRIPTION:	Sets the supplemental synchronized relative-target position
EXECUTION:	Buffered until a GS is issued
CONDITIONAL TO:	Motors in the sync group are positioned at the their target position: PT=PC Must be issued after PRTS and before GS
LIMITATIONS:	N/A
READ/REPORT:	None
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	Encoder counts
RANGE OF VALUES:	Input: Position: -2147483648 to 2147483647 Axis: 1-127
TYPICAL VALUES:	Input: Position: -2147483648 to 2147483647 Axis: 1-127
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M), not ver. 5.32.x.x; 6.x (D/M), M requires EIP option
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

NOTE: This command requires a Combitronic-supported motor. Although this command does not support Combitronic syntax, it does use Combitronic communication to pass information between other motors.

NOTE: This command is affected by the SCALEP command. For details, see SCALEP(m,d) on page 726. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

The PRTSS command allows supplemental axis moves to be added and synchronized with the previous motion commanded by PTS() or PRTS(). Issue the additional axis commands after a PTS() or PRTS() command but before the next GS command.

The PRTSS command allows you to specify an axis position (posn) and its associated CAN address (axisn):

```
PRTSS (posn;axisn)
```

By the time the PRTSS or PTSS command is issued, the move time has already been determined by the PTS or PRTS command. The command may be issued as many times as desired. There are no additional resources consumed by adding more axes.

The supplemental axis motions will start with the next GS at exactly the same time as the main PTS() or PRTS() motion. Further, they will transition from their accelerations to their slew velocities at exactly the same time, and they will decelerate and stop at exactly the same time.

It is important to ensure that the target position in each motor is equal to the motor's current position. The best way to ensure this is to use an absolute position move (using PT=) in all participating motors before issuing the PTS command.

For more details, see Synchronized Motion on page 179.

EXAMPLE: (3-axis synchronized relative move to position x:y:z for motors 1, 2 and 3)

This sample code may be executed by any motor sharing the same CAN bus network with the motors being commanded to move.

NOTE: Ensure no motor drive faults exist prior to calling this subroutine.

```

. . .
C20
  OFF:0 MP:0 PRT:0=0 G TWAIT      'Initialize to stationary in position mode
  PT:1=PC:1  PT:2=PC:2  PT:3=PC:3 'Set target and commanded positions equal
  WAIT=50
  VTS=v      'Set target path velocity
  ADTS=a     'Set target path accel/decel
  PRTS(x;1,y;2,z;3) 'Use Position Target Synchronized moves
  PRTSS(a;4)  'Supplemental synchronized relative target
  IF PTSD!=0  'Prevent 0-length (divide by zero) move
    GS        'Go Synchronized
    TSWAIT    'Wait until path move time is complete
  ENDIF
RETURN

```

For additional examples, see A Note About PTS and PRTS on page 181.

RELATED COMMANDS:

ADTS=formula *Acceleration/Deceleration Target, Synchronized (see page 265)*

GS *Start Synchronized Motion (GO Synchronized) (see page 487)*

R PRT=formula *Position, Relative Target (see page 683)*

PRTS(...) *Position, Relative Target, Synchronized (see page 685)*

PTS(...) *Position Target, Synchronized (see page 692)*

PTSS(...) *Position Target, Synchronized Supplemental (see page 696)*

TSWAIT *Trajectory Synchronized Wait (see page 788)*

VTS=formula *Velocity Target, Synchronized Move (see page 831)*



PT=formula Position, (Absolute) Target

APPLICATION:	Motion control; supports the DS2020 Combitronic system
DESCRIPTION:	Get/set absolute target position
EXECUTION:	Buffered until a G command is issued
CONDITIONAL TO:	MP (position mode)
LIMITATIONS:	N/A
READ/REPORT:	RPT
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	Encoder counts DS2020 Combitronic system: user increments, see FD=expression on page 461
RANGE OF VALUES:	-2147483648 to 2147483647 (see NOTE in Detailed Description)
TYPICAL VALUES:	-1000000000 to 1000000000
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	PT:3=1234, a=PT:3, RPT:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

NOTE: This command is affected by the SCALEP command. For details, see SCALEP(m,d) on page 726. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

The PT command is used to get (read) or set absolute target position.

To specify an absolute target position to the SmartMotor's position origin, set PT=target position (either positive or negative) and then issue a G command.

PT=formula sets the target position in Position mode.

NOTE: While PT= allows a range of -2,147,483,648 to +2,147,483,647, at these extremes, the relative distance from one end to the other is greater than 32-bits. Therefore, the calculated move within the motor will overflow and may move opposite of the expected direction. To avoid this problem, a best practice is to keep the motor target position within the "typical values" range: -1000000000 to +1000000000.

For the DS2020 Combitronic system, PT=I(0) command (available only via RS-232) will set the target position to the last captured index position of the feedback sensor.

If the appropriate trajectory parameters ADT and VT are specified, then the motor will move, when the G command is issued, to the position specified by the last PT value requested.

Status word 3, bit 8 reports 0 when acting on a PT target (absolute position).

RPT will report the actual position. However, if you set a variable equal to PT, such as a=PT, that variable will be loaded with the last-entered target position rather than the actual position. If you want to use the actual position in your program, then use a PA variable such as a=PA.

EXAMPLE: (Shows use of ADT, PT and VT)

```
MP           'Set mode position
ADT=5000    'Set target accel/decel
PT=20000   'Set absolute position
VT=10000   'Set velocity
G           'Start motion
END         'End program
```

EXAMPLE: (Routine homes motor against a hard stop)

```
MDS           'Using Sine mode commutation
KP=3200      'Increase stiffness from default
KD=10200     'Increase damping from default
F            'Activate new tuning parameters
AMPS=100     'Lower current limit to 10%
VT=-10000   'Set maximum velocity
ADT=100      'Set maximum accel/decel
MV           'Set Velocity mode
G            'Start motion
WHILE EA>-100 'Loop while position error is small
LOOP         'Loop back to WHILE
O=-100       'While pressed, declare home offset
S            'Abruptly stop trajectory
MP           'Switch to Position mode
VT=20000    'Set higher maximum velocity
PT=0         'Set target position to be home
G            'Start motion
TWAIT       'Wait for motion to complete
AMPS=1023   'Restore current limit to maximum
END         'End program
```

RELATED COMMANDS:

ADT=formula *Acceleration/Deceleration Target (see page 263)*

R EL=formula *Error Limit (see page 426)*

R FD=expression *Resolution to Set Units of Position/Velocity/Acceleration (see page 461)*

G *Start Motion (GO) (see page 473)*

MP *Mode Position (see page 613)*

R PA *Position, Actual (see page 646)*

R PRT=formula *Position, Relative Target (see page 683)*

R VT=formula *Velocity Target (see page 828)*

PTS(...) Position Target, Synchronized

APPLICATION:	Motion control
DESCRIPTION:	Sets the synchronized target position
EXECUTION:	Buffered until a GS is issued
CONDITIONAL TO:	Motors in the sync group are positioned at the their target position: PT=PC
LIMITATIONS:	ADTS and VTS commands must be set before issuing this command Up to three axes of motion; must be orthogonal (Cartesian) coordinates, not radial or polar.
READ/REPORT:	None
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	Encoder counts
RANGE OF VALUES:	Input: Position: -2147483648 to 2147483647 Axis: 1-127
TYPICAL VALUES:	Input: Position: -2147483648 to 2147483647 Axis: 1-127
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M), not ver. 5.32.x.x; 6.x (D/M), M requires EIP option
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

NOTE: This command requires a Combitronic-supported motor. Although this command does not support Combitronic syntax, it does use Combitronic communication to pass information between other motors.

NOTE: This command is affected by the SCALEP command. For details, see SCALEP(m,d) on page 726. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

The PTS command is used to set the synchronized target position. It allows you to identify two or three axis positions (posn) and their associated CAN axis addresses (axisn) to cause a synchronized-relative, multi-axis move. For multiple-axis machines that are not using two motors to drive an axis, the combined path velocity is controlled as shown:

```
PTS (pos1; axis1, pos2; axis2 [, pos3; axis3])
```

NOTE: There is a three-axis limitation for this command.

Additional axes can be synchronized using the PTSS and PRTSS commands.

The command is illustrated in the next example:

Position target X = 1000

Position target Y = 2000

Motor address X = 5

Motor address Y = 7

```
PTS(1000;5,2000;7) 'X axis and Y axis
```

The synchronized motion is initiated with a GS command.

It is important to ensure that the target position in each motor is equal to the motor's current position. The best way to ensure this is to use an absolute position move (using PT=) in all participating motors before issuing the PTS command. For more details, see Synchronized Motion on page 179.

Some gantry-type, multiple-axis machines have two motors operating the same axis of motion (see the next figure). Below is the full syntax for the PTS command, which shows additional/optional parameters (enclosed in braces "{ }") for support of two motors operating the same axis. The optional parameter contains the motor address for the second motor of the axis. (For the PRTS command, replace PTS with PRTS.)

```
PTS(pos1;addr1{;addr1'},pos2;addr2{;addr2'}[,pos3;addr3{;axis3'}])
```

This is illustrated in the next example. (If you are using the PRTS command, substitute PRTS in place of PTS below.)

Position target X = 2000

Position target Y = 1000

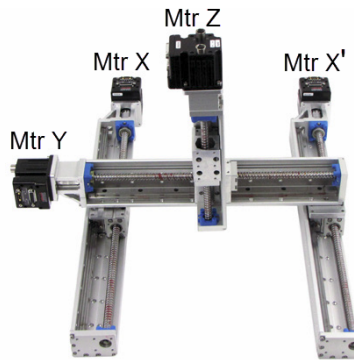
Position target Z = 500

Motor address X = 5

Motor address X' = 6

Motor address Y = 7

Motor address Z = 8



```
PTS(2000;5;6,1000;7) 'Two-motor X axis (X, X'), plus Y axis
```

```
PTS(2000;5;6,1000;7,500;8) 'Two-motor X axis (X, X'), plus Y & Z axes
```

In these cases, the same position, velocity and acceleration data sent to motor address 5 is also sent to motor address 6, with both motors driving the gantry's X axis.

EXAMPLE: (2-axis synchronized absolute move to position x:y for motors 1 and 2)

This sample code may be executed by any motor sharing the same CAN bus network with the motors being commanded to move.

NOTE: Ensure no motor drive faults exist prior to calling this subroutine.

```

. . .
C20
  OFF:0 MP:0 PRT:0=0 G TWAIT      'Initialize to stationary in position mode
  PT:1=PC:1  PT:2=PC:2          'Set target and commanded positions equal
  WAIT=50
  VTS=v                          'Set target path velocity
  ADTS=a                          'Set target path accel/decel
  PTS (x;1,y;2)                  'Use Position Target Synchronized moves
  PTSS (a;3)                     'Supplemental synchronized target
  IF PTSD!=0                     'Prevent 0-length (divide by zero) move
    GS                            'Go Synchronized
    TSWAIT                        'Wait until path move time is complete
  ENDIF
RETURN

```

For additional examples, see A Note About PTS and PRTS on page 181.

RELATED COMMANDS:

ADTS=formula *Acceleration/Deceleration Target, Synchronized (see page 265)*

GS *Start Synchronized Motion (GO Synchronized) (see page 487)*

PRTSS(...) *Position, Relative Target, Synchronized, Supplemental (see page 688)*

PTSS(...) *Position Target, Synchronized Supplemental (see page 696)*

TSWAIT *Trajectory Synchronized Wait (see page 788)*

VTS=formula *Velocity Target, Synchronized Move (see page 831)*

PTSD

Position Target, Synchronized Distance

APPLICATION:	Motion control
DESCRIPTION:	Gets the synchronized target move linear distance
EXECUTION:	Immediate
CONDITIONAL TO:	PTS or PRTS command
LIMITATIONS:	N/A
READ/REPORT:	RPTSD
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Encoder counts
RANGE OF VALUES:	-2147483648 to 2147483647
TYPICAL VALUES:	-2147483648 to 2147483647
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M), not ver. 5.32.x.x; 6.x (D/M), M requires EIP option
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The PTSD command reports the linear distance of the synchronized target move. This distance represents the vector distance of 2- or 3-dimensional moves computed by the PTS() or PRTS() command.

After a PTS() or PRTS() command, the combined distance is stored in the PTSD variable so that it may be read by the programmer.

For more details, see Synchronized Motion on page 179.

EXAMPLE: (Assign a value for a frame of reference)

```
d=PTSD      'Assign to "d" the vector sum distance in encoder counts
            'of a predefined synchronized move
```

RELATED COMMANDS:

ADTS=formula *Acceleration/Deceleration Target, Synchronized (see page 265)*

GS *Start Synchronized Motion (GO Synchronized) (see page 487)*

PRTS(...) *Position, Relative Target, Synchronized (see page 685)*

PTS(...) *Position Target, Synchronized (see page 692)*

RPTST *Position Target, Synchronized Time (see page 698)*

TSWAIT *Trajectory Synchronized Wait (see page 788)*

VTS=formula *Velocity Target, Synchronized Move (see page 831)*

APPLICATION:	Motion control
DESCRIPTION:	Sets the supplemental synchronized target position
EXECUTION:	Buffered until a GS is issued
CONDITIONAL TO:	Motors in the sync group are positioned at the their target position: PT=PC Must be issued after PTS and before GS
LIMITATIONS:	N/A
READ/REPORT:	None
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	Encoder counts
RANGE OF VALUES:	Input: Position: -2147483648 to 2147483647 Axis: 1-127
TYPICAL VALUES:	Input: Position: -2147483648 to 2147483647 Axis: 1-127
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M), not ver. 5.32.x.x; 6.x (D/M), M requires EIP option
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

NOTE: This command requires a Combitronic-supported motor. Although this command does not support Combitronic syntax, it does use Combitronic communication to pass information between other motors.

NOTE: This command is affected by the SCALEP command. For details, see SCALEP(m,d) on page 726. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

The PTSS command allows supplemental axis moves to be added and synchronized with the previous PTS() or PRTS() commanded motion.

Issue the additional axis commands after a PTS() or PRTS() command but before the next GS command.

The PTSS command allows you to identify one axis position (posn) and its associated axis CAN address (axisn):

```
PTSS (posn;axisn)
```

By the time the PRTSS or PTSS command is issued, the move time has already been determined by the PTS or PRTS command. The command may be issued as many times as desired. There are no additional resources consumed by adding more axes.

The supplemental axis motions will start with the next GS at exactly the same time as the main PTS() or PRTS() motion. Further, they will transition from their accelerations to their slew velocities at exactly the same time, and they will decelerate and stop at exactly the same time.

It is important to ensure that the target position in each motor is equal to the motor's current position. The best way to ensure this is to use an absolute position move (using PT=) in all participating motors before issuing the PTS command.

For more details, see Synchronized Motion on page 179.

EXAMPLE: (2-axis synchronized absolute move to position x;y for motors 1 and 2)

This sample code may be executed by any motor sharing the same CAN bus network with the motors being commanded to move.

NOTE: Ensure no motor drive faults exist prior to calling this subroutine.

```

. . .
C20
  OFF:0 MP:0 PRT:0=0 G TWAIT      'Initialize to stationary in position mode
  PT:1=PC:1  PT:2=PC:2          'Set target and commanded positions equal
  WAIT=50
  VTS=v                          'Set target path velocity
  ADTS=a                          'Set target path accel/decel
  PTS (x;1,y;2)                  'Use Position Target Synchronized moves
  PTSS (a;3)                      'Supplemental synchronized target
  IF PTSD!=0                      'Prevent 0-length (divide by zero) move
    GS                            'Go Synchronized
    TSWAIT                        'Wait until path move time is complete
  ENDIF
RETURN

```

For additional examples, see A Note About PTS and PRTS on page 181.

RELATED COMMANDS:

ADTS=formula *Acceleration/Deceleration Target, Synchronized (see page 265)*

GS *Start Synchronized Motion (GO Synchronized) (see page 487)*

PRTSS(...) *Position, Relative Target, Synchronized, Supplemental (see page 688)*

PTS(...) *Position Target, Synchronized (see page 692)*

TSWAIT *Trajectory Synchronized Wait (see page 788)*

VTS=formula *Velocity Target, Synchronized Move (see page 831)*

PTST

Position Target, Synchronized Time

APPLICATION:	Motion control
DESCRIPTION:	Stores the time for a synchronized move to the target position
EXECUTION:	Immediate
CONDITIONAL TO:	PTS or PRTS command
LIMITATIONS:	N/A
READ/REPORT:	RPTST
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	milliseconds
RANGE OF VALUES:	-2147483648 to 2147483647
TYPICAL VALUES:	-2147483648 to 2147483647
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M), not ver. 5.32.x.x; 6.x (D/M), M requires EIP option
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The PTST command stores the time (in milliseconds) for a synchronized move to the target position. After a PTS() or PRTS() command, the combined distance is stored in the PTST variable (in milliseconds) so that it may be used by the programmer.

For more details, see Synchronized Motion on page 179.

EXAMPLE:

```
t=PTST/2      'Calculate time in milliseconds to complete half of
              'a predefined synchronized move.
TMR(0,t)     'Start timer that will timeout halfway through the move.
GS           'Start synchronized move.
```

RELATED COMMANDS:

ADTS=formula *Acceleration/Deceleration Target, Synchronized (see page 265)*
 GS *Start Synchronized Motion (GO Synchronized) (see page 487)*
 PRTS(...) *Position, Relative Target, Synchronized (see page 685)*
 PTS(...) *Position Target, Synchronized (see page 692)*
 R PTSD *Position Target, Synchronized Distance (see page 695)*
 TSWAIT *Trajectory Synchronized Wait (see page 788)*
 VTS=formula *Velocity Target, Synchronized Move (see page 831)*

RANDOM=formula

Random Number

APPLICATION:	Math function
DESCRIPTION:	Seeds the random number generator; gets (reads) the next value from the random number generator
EXECUTION:	Immediate
CONDITIONAL TO:	Previous calls to RANDOM
LIMITATIONS:	N/A
READ/REPORT:	RRANDOM
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	Output: 0 to 2147483647 Input: -2147483648 to 2147483647
TYPICAL VALUES:	Output: 0 to 2147483647 Input: -2147483648 to 2147483647
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The RANDOM command is used to get (read) the next value from the pseudorandom number generator. RANDOM can also be assigned to seed the random-number generator.

- **RANDOM=formula**
Sets the random seed value. Any 32-bit value is accepted.
- **x=RANDOM**
Gets the next value from the random number generator and assigns it to the variable x.

The output of the random number generator is an integer (whole number) in the range from 0 to 2147483647.

The number generated is not truly random— it does use a predictable sequence if the starting point (seed) and algorithm are known. This allows a test sequence to be repeated and use the random generator to exercise a wide range of values. However, if this were used in programs expecting unpredictability (e.g., a game or secret number generator), the results may be disappointing.

If an unpredictable seed is desired, then a more creative approach must be used to initialize the value based on events in the real world. For example, you could measure the time between some real-world inputs that are somewhat random.

Typically, an application will need this number reduced to a more useful range. The modulo operator can be used to accomplish this. For instance, if you need numbers in the range from -100 to +100, then this formula may be helpful:

$a=(\text{RANDOM}\%201)-100$

EXAMPLE:

```
x=200000      'Max distance (full stroke) of actuator
               'in encoder counts.
PT=RANDOM%x   'Use Modulo function and RANDOM function
               'to create random position targets within max stroke.
```

RELATED COMMANDS:

N/A

APPLICATION:	Program access
DESCRIPTION:	Reports the program checksum
EXECUTION:	Immediate
CONDITIONAL TO:	User program downloaded.
LIMITATIONS:	No CKS or x=CKS form of this command
READ/REPORT:	RCKS
WRITE:	Read only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	ASCII alphanumeric string, see description
TYPICAL VALUES:	N/A
DEFAULT VALUE:	000000 0000E1 P ("END" program)
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The RCKS command reports the program checksum. Typically, this is automatically used by the SMI software to verify the integrity of the program that is being downloaded. For example, if the download process is interrupted and only part of the program is loaded, then the RCKS value will not match the value expected by the host.

When the user program is erased with the "Clear EEPROM" feature of SMI (technically, a simple program with an END command is loaded), then the returned value from RCKS is:

```
000000 0000E1 P
```

The first group of six hex digits are related to program labels. For example, C100 is a target for GOSUB. The second group of six hex digits are the program itself. The last character is either P or F, representing pass or fail, respectively.

NOTE: There is no CKS or x=CKS form of the command, as this command is not meant for use in a user program. Typically, only SMI or other serial hosts would use this command to verify download of the user program to the motor.

EXAMPLE: (Commanded from the terminal window)

```
RCKS
```

The command reports:

```
000000 0000E1 P
```

RELATED COMMANDS:

R Bk Bit, Program EEPROM Data Status (see page 315)



APPLICATION:	System
DESCRIPTION:	Gets (reads) the encoder resolution
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RRES ; supports the DS2020 Combitronic system
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Encoder counts Report for DS2020 Combitronic system: user increments, see FD=e-expression on page 461
RANGE OF VALUES:	N/A Report for DS2020 Combitronic system: 0 to 4294967295
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	RRES:3, x=RES:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The RES command is used to get the resolution of the encoder (for example, a=RES). This command is the preferred method for obtaining the encoder resolution. It is simple to include it in user programs, and it takes into account any special scaling or compensation.

NOTE: Any program that requires the encoder resolution should use this command instead of placing a hard-coded value in the program.

This command can also serve as a check at the beginning of a program to ensure that the program is running on a motor with the expected encoder type. Refer to the next example.

EXAMPLE: (Check for proper encoder type)

```

      IF RES!=4000
OFF PRINT("Wrong encoder.",#13) END
ENDIF

```

EXAMPLE: (Scale Factor Multipliers)

```
' SAMP: motor sample time in samples per second
' RES: encoder resolution in counts per rev
' 65536: internal fractional constant for motor unit calculations
' 60: conversion between seconds and minutes
' *1.0: way to typecast integer values out to full floating-point values
' af[2] thru af[7]: 32-bit, floating-point array variables

'These abbreviations are used in this code:
' NatAD (Native Accel{decel})
' NatVel (Native Velocity)
' RPM (revolutions per minute)
' RPS (revolutions per second)

'Calculating multipliers
af[2]=(((SAMP*1.0)/RES)*60)/65536
af[3]=(((RES*1.0)/SAMP)/60)*65536
af[4]=((SAMP*1.0)/RES)/65536
af[5]=((RES*1.0)/SAMP)*65536
af[6]=af[5]/SAMP
af[7]=af[4]*SAMP

'Input      Output
'NatVel -> RPM      Multiplier
'RPM     -> NatVel   Multiplier
'NatVel -> RPS      Multiplier
'RPS     -> NatVel   Multiplier
'RPS^2   -> NatAD    Multiplier
'NatAD   -> RPS^2    Multiplier

'Examples
'Suppose you wish to set a Velocity of 3000 RPM:
'(This method simply uses the above multiplier)
VT=3000*af[3]  '3000 RPM desired speed multiplied by af[3]
               '(RPM to Native Velocity multiplier)

'Suppose you wish to read real time velocity in units of RPS:
s=VA*af[5]    'Converts native VA (actual velocity) into RPS
               'and assigns it to the variable "s"
```

END**RELATED COMMANDS:**R FD=expression *Resolution to Set Units of Position/Velocity/Acceleration (see page 461)*R FW *Firmware Version (see page 471)*R SAMP *Sampling Rate (see page 722)*



APPLICATION:	Program execution and flow control
DESCRIPTION:	Resume program execution
EXECUTION:	Immediate
CONDITIONAL TO:	A user program started and PAUSEd
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	RESUME:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

When executed, the RESUME command starts program execution from the location after the PAUSE command (where the user program is currently paused). It is designed to be issued externally over communications and should not be compiled within a program.

NOTE: RESUME is intended to manually resume a program from a command terminal for debugging purposes — the program must be at a PAUSE and not executing an interrupt or subroutine.

The RESUME command only operates on PAUSE in the current context of the program. Examples of different contexts are the main program versus a subroutine or interrupt routine. PAUSE used outside the context of the currently executed command will continue to PAUSE (e.g., if there is a PAUSE in the main program and a PAUSE in an interrupt, then the PAUSE that is active at the time will be resumed). If a main program is at a PAUSE but other commands are executing in the interrupt, then a RESUME at that time will not affect the PAUSE in the main program— it will remain paused.

EXAMPLE:

```
EIGN(W,0,12)    'Another way to disable travel limits.
ZS              'Clear faults.
ITR(0,0,0,0,0) 'Set Int 0 for: stat word 0, bit 0,
                'shift to 0, to call C0.
EITR(0)        'Enable Interrupt 0.
ITRE           'Global Interrupt Enable.
PAUSE          'Pause to prevent "END" from disabling
                'Interrupt, no change to stack.
'RESUME must be issued externally over communications;
'it is not allowed to be compiled within a program.
END
C0             'Fault handler.
  MTB:0        'Motor will turn off with Dynamic
                'breaking, tell other motors to stop.
  US(0):0      'Set User Status Bit 0 to 1 (Status
                'Word 12 bit zero).
  US(ADDR):0   'Set User Status Bit "address" to 1
                '(Status Word 12 Bit "address").
```

RETURNI**RELATED COMMANDS:**

PAUSE *Pause Program Execution (see page 648)*

RETURN

Return From Subroutine

APPLICATION:	Program execution and flow control
DESCRIPTION:	Return program execution to next statement after current subroutine call
EXECUTION:	Immediate
CONDITIONAL TO:	A previous GOSUBn program statement was performed
LIMITATIONS:	Up to nine nested GOSUB or interrupt subroutines may occur at one time
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The RETURN command is used to terminate a subroutine within a user program.

NOTE: Subroutines present a great opportunity to partition and organize your code.

When the RETURN command is invoked, program execution continues immediately after the GOSUB that initiated the subroutine call. RETURN is normally executed from within a program, but with care, the HOST terminal may also be used to issue a RETURN instruction.

In order to execute the RETURN program statement, the processor needs to be able to recall (from the stack) the program address point where it should return. The "stack" is a memory region where these addresses are stored. A maximum of nine address locations can be stored within the stack. Therefore, do not use more than nine nested subroutines; otherwise, it may cause a stack overflow and crash the program.

NOTE: RETURNI must be used to return from interrupt subroutines; RETURN must always be used to return from GOSUB subroutines.

EXAMPLE:

```
PRINT("WAIT FOR HOST TERMINAL COMMANDS",#13)
GOSUB10          'Start of subroutine 10.
PRINT("PROGRAM RECEIVED EXTERNAL RETURN")
END
C10              'Start of subroutine 10.
WHILE 1          'Wait for terminal commands.
    WAIT=100     'Report terminal errors.
    IF Bs
        PRINT(#13,"SCAN ERROR",#13)
        Zs
    ENDIF
LOOP
RETURN          'Return to line just below GOSUB10 command.
```

RELATED COMMANDS:

C{number} *Command Label (see page 353)*
END *End Program Code Execution (see page 439)*
GOSUB(label) *Subroutine Call (see page 480)*
RETURNI *Return Interrupt (see page 708)*
RUN *Run Program (see page 714)*
RUN? *Halt Program Execution Until RUN Received (see page 716)*

RETURNI

Return Interrupt

APPLICATION:	Program execution and flow control
DESCRIPTION:	Return program execution to next statement after current interrupt subroutine call
EXECUTION:	Immediate
CONDITIONAL TO:	Interrupts configured and have caused an interrupt subroutine to execute
LIMITATIONS:	Up to nine nested GOSUB or interrupt subroutines may occur at one time
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The RETURNI command is used to terminate an interrupt subroutine within a user program. This is different than terminating a subroutine with the RETURN command. For details, see RETURN on page 706.

NOTE: Subroutines present a great opportunity to partition and organize your code.

Interrupt subroutines end with the RETURNI command to distinguish them from ordinary subroutines. After the interrupt code execution reaches the RETURNI command, it will return to the program at exactly the point where it was interrupted. An interrupt subroutine must not be called directly with a GOSUB command.

NOTE: RETURNI must be used to return from interrupt subroutines; RETURN must always be used to return from GOSUB subroutines.

For more details, see Interrupt Programming on page 195.

EXAMPLE:

```

EIGN(W,0,12)      'Another way to disable travel limits.
ZS                'Clear faults.
ITR(0,0,0,0,0)   'Set interrupt 0 for Status Word 0, Bit 0,
                  'Shift to 0, to call C0.
EITR(0)          'Enable interrupt 0.
ITRE             'Global interrupt enable.
PAUSE           'Pause to prevent "END" from disabling
                  'interrupt; no change to stack.
'RESUME must be issued externally over communications;
'it is not allowed to be compiled within a program.
END
C0              'Fault handler.
  MTB:0         'Motor will turn off with dynamic
                  'braking; tell other motors to stop.
  US(0):0      'Set User Status Bit 0 to 1 (Status
                  'Word 12 bit zero).
  US(ADDR):0   'Set User Status Bit "address" to 1
                  '(Status Word 12 Bit "address").

```

RETURNI**RELATED COMMANDS:**

C{number} *Command Label (see page 353)*
DITR(int) *Disable Interrupts (see page 394)*
EITR(int) *Enable Interrupts (see page 424)*
END *End Program Code Execution (see page 439)*
GOSUB(label) *Subroutine Call (see page 480)*
ITR(Int#,StatusWord,Bit#,BitState,Label#) *Interrupt Setup (see page 517)*
ITRD *Interrupt Disable, Global (see page 520)*
ITRE *Enable Interrupts, Global (see page 522)*
RETURN *Return From Subroutine (see page 706)*
RUN *Run Program (see page 714)*
RUN? *Halt Program Execution Until RUN Received (see page 716)*

Report Sampling Rate and Firmware Revision

APPLICATION:	System; supports the DS2020 Combitronic system
DESCRIPTION:	Report PID sample period and firmware revision
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	No SP or x=SP form of this command
READ/REPORT:	RSP
WRITE:	Read only
LANGUAGE ACCESS:	N/A
UNITS:	See details
RANGE OF VALUES:	ASCII alphanumeric string, see description
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The RSP report command returns a five-digit value of the PID sample period followed by an ASCII string code representing the firmware version. For the DS2020 Combitronic system, it reports the firmware version in the format "xxx...xxx/DS2020".

NOTE: There is no SP or x=SP form of the command, as this command is not meant for use in a user program.

The next table shows PID rates and RSP responses.

Command	PID / Trajectory Update Rate	Period (µsec)	SAMP Report	RSP Report
PID1	16 kHz	62.5	16000	06250/5...
PID2	8 kHz (default)	125	8000	12500/5...
PID4	4 kHz	250	4000	25000/5...
PID8	2 kHz	500	2000	50000/5...

The PID sample period, in microseconds, is the five-digit number reported/100.

The sample period is followed by a "/" character and the firmware version string.

EXAMPLE: (Terminal command sent to an SM23165D SmartMotor with 5.0.3.44 firmware)

RSP

The command reports:

12500/5.0.3.44

RELATED COMMANDS:

PID# *Proportional-Integral-Differential Filter Rate (see page 654)*

RSP1 *Report Firmware Compile Date (see page 712)*

R SP2 *Bootloader Version (see page 755)*

RSP5 *Report Network Card Firmware Version (see page 713)*

R SP6 *Serial Number (see page 756)*

R SAMP *Sampling Rate (see page 722)*

RSP1 Report Firmware Compile Date

APPLICATION:	System
DESCRIPTION:	Report the firmware compile date and time
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	No SP1 or x=SP1 form of this command
READ/REPORT:	RSP1
WRITE:	Read only
LANGUAGE ACCESS:	N/A
UNITS:	See description
RANGE OF VALUES:	ASCII alphanumeric string, see description
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The RSP1 report command returns the date and time that the motor firmware was compiled.

NOTE: There is no SP1 or x=SP1 form of the command, as this command is not meant for use in a user program.

All version 5.xx series motors respond in the form of:

Month Day, Year HH:MM:SS

EXAMPLE: (Terminal command sent to an SM23165D SmartMotor with 5.0.3.44 firmware)

[RSP1](#)

The command reports:

Dec 20 2012 13:08:28

RELATED COMMANDS:

R FW Firmware Version (see page 471)

RSP Report Sampling Rate and Firmware Revision (see page 710)

R SP2 Bootloader Version (see page 755)

RSP5 Report Network Card Firmware Version (see page 713)

R SP6 Serial Number (see page 756)

RSP5

Report Network Card Firmware Version

APPLICATION:	System
DESCRIPTION:	Reports network interface card firmware version
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	No SP5 or x=SP5 form of this command
READ/REPORT:	RSP5
WRITE:	Read only
LANGUAGE ACCESS:	N/A
UNITS:	See description
RANGE OF VALUES:	ASCII alphanumeric string, see description
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	6.x (D/M); no Class 5
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The RSP5 command reports the network interface card firmware version.

NOTE: There is no SP5 or x=SP5 form of the command, as this command is not meant for use in a user program.

All version 6.x series SmartMotors respond in the form of:

```
n.n.n.n
```

Refer to the next example.

EXAMPLE: (Terminal command sent to a SmartMotor)

[RSP5](#)

The command reports the motor's firmware version, for example:

```
2.5.28.0
```

RELATED COMMANDS:

R FW *Firmware Version* (see page 471)

RSP *Report Sampling Rate and Firmware Revision* (see page 710)

RSP1 *Report Firmware Compile Date* (see page 712)

R SP2 *Bootloader Version* (see page 755)

R SP6 *Serial Number* (see page 756)



APPLICATION:	Program execution and flow control
DESCRIPTION:	Execute user EEPROM program beginning at initial command
EXECUTION:	Immediate
CONDITIONAL TO:	No effect if no EEPROM program exists
LIMITATIONS:	Valid EEPROM stored program commands
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	RUN at power recycle or software reset
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	RUN:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The RUN command starts a stored (downloaded) user EEPROM program. Issuing a RUN command does not reset any motion, variable or I/O state. However, it does reset the program-execution pointer (stack pointer) to zero, and resets the internal GOSUB stack.

NOTE: To test your program with a truly fresh start, use the Z command to completely reset the motor as if it were just powered up. For details, see Z on page 846.

If a program exists within the SmartMotor™ user EEPROM, it will automatically run every time the motor is turned on. To prevent this, make RUN? the first command of your user program. Or, if you wish, place a RUN? command further down in your program. At power up, the program will automatically execute only down to the RUN? statement. The program execution will stop at that point.

NOTE: Programs that are deliberately started with the RUN command (usually from the serial terminal), will start from the top of the program and skip over the RUN? command.

Because user programs start automatically, it is possible to write a program that prevents SMI communications with the motor. For instance, a Z (reset) or CCHN command at the beginning of the program can make it difficult to connect to the motor.

NOTE: If you get locked out and are unable to communicate with the SmartMotor, you may be able to recover communications using the SMI software's Communication Lockup Wizard. For more details, see Communication Lockup Wizard on page 31.

EXAMPLE: (user program with possible halt)

In Class 5, issuing the RUN command causes top-down execution through the entire program no matter where the RUN? command is placed.

```
PRINT(" Enter RUN to start",#13)   'Prompt user for RUN
RUN?                               'Run command requested; stop program
                                   'execution until "RUN" command is received
PRINT(" LOADING TRAJECTORY",#13)
ADT=100                            'Set target accel/decel
VT=1000000                         'Set target velocity
PT=1000000                         'Set target position
MP                                  'Mode Position
PRINT(" EXECUTING TRAJECTORY",#13)
G                                   'Begin motion.
END                                 'Required END
```

Program output is:

```
Enter RUN to start
LOADING TRAJECTORY
EXECUTING TRAJECTORY
```

RELATED COMMANDS:

END *End Program Code Execution (see page 439)*
 GOSUB(label) *Subroutine Call (see page 480)*
 GOTO(label) *Branch Program Flow to a Label (see page 482)*
 LOAD *Download Compiled User Program to Motor (see page 548)*
 LOCKP *Lock Program (see page 551)*
 PAUSE *Pause Program Execution (see page 648)*
 RESUME *Resume Program Execution (see page 704)*
 RUN? *Halt Program Execution Until RUN Received (see page 716)*
 UP *Upload Compiled Program and Header (see page 797)*
 UPLOAD *Upload Standard User Program (see page 799)*

RUN?**Halt Program Execution Until RUN Received**

APPLICATION:	Program execution and flow control
DESCRIPTION:	Halt execution of user program started without RUN
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

If a program exists within the SmartMotor™ user EEPROM, it will automatically run every time the motor is turned on. To prevent this, make RUN? the first command of your user program. Or, if you wish, place a RUN? command further down in your program. At power up, the program will automatically execute only down to the RUN? statement. The program execution will stop at that point.

NOTE: Programs that are deliberately started with the RUN command (usually from the serial terminal), will start from the top of the program and skip over the RUN? command.

RUN? does not terminate the current motion mode or trajectory, change motion parameters such as EL, ADT, VT or KP, or alter the current value of the user variables.

EXAMPLE: (user program with possible halt)

In Class 5, issuing the RUN command causes top-down execution through the entire program no matter where the RUN? command is placed.

```
PRINT(" Enter RUN to start",#13)   'Prompt user for RUN
RUN?                               'Run command requested; stop program
                                   'execution until "RUN" command is received
PRINT(" LOADING TRAJECTORY",#13)
ADT=100                            'Set target accel/decel
VT=1000000                          'Set target velocity
PT=1000000                          'Set target position
MP                                  'Mode Position
PRINT(" EXECUTING TRAJECTORY",#13)
G                                   'Begin motion.
END                                 'Required END
```

Program output is:

```
Enter RUN to start
LOADING TRAJECTORY
EXECUTING TRAJECTORY
```

RELATED COMMANDS:

END *End Program Code Execution (see page 439)*
 GOSUB(label) *Subroutine Call (see page 480)*
 GOTO(label) *Branch Program Flow to a Label (see page 482)*
 LOAD *Download Compiled User Program to Motor (see page 548)*
 LOCKP *Lock Program (see page 551)*
 PAUSE *Pause Program Execution (see page 648)*
 RESUME *Resume Program Execution (see page 704)*
 RUN? *Halt Program Execution Until RUN Received (see page 716)*
 UP *Upload Compiled Program and Header (see page 797)*
 UPLOAD *Upload Standard User Program (see page 799)*



S (as command)

Stop Motion

APPLICATION:	Motion control; supports the DS2020 Combitronic system
DESCRIPTION:	Abruptly stop motor motion
EXECUTION:	Immediate
CONDITIONAL TO:	EL value
LIMITATIONS:	If position error exceeds EL, motor will shut off and coast to a stop
REPORT VALUE:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	S:3 or S(0):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:



CAUTION: Careful use of the S command is vital! It causes an emergency stop, and the resulting motion is very abrupt.

The S command causes an emergency stop. It does not turn the motor off; it sets the target position to the current position. The resulting commanded motion is very abrupt. In some cases, it will be so abrupt that the amplifier can overcurrent or the servo error can exceed the maximum error set by the EL command. This will, in turn, cause the motor to turn off and coast. Consequently, careful use of the S command is vital.

For the DS2020 Combitronic system, the S command uses the velocity control system to stop the motor by passing a zero velocity reference. If needed, the maximum current is used. This command cannot be issued to stop the motor if the system becomes unstable when tuning KP, KI, KD, KV, because the control loops are used by the S command. In that case, only an OFF command will effectively disable the drive (and engage the brake if available and configured).

The S command also halts the homing operation. For more details on SmartMotor homing operations, see the *SmartMotor Homing Procedures and Methods Application Note*.

EXAMPLE:

```

EIGN(W,0)
ZS
ADT=100
VT=1000000
PT=5000000
G
WHILE Bt      'While trajectory is active
IF PA>80000  'Set a position to look for
    S        'Stop abruptly
    PRINT("Emergency Stop")
ENDIF
LOOP

```

Program output is:

Emergency Stop

EXAMPLE: (Routine homes motor against a hard stop)

```

MDS          'Using Sine mode commutation
KP=3200      'Increase stiffness from default
KD=10200     'Increase damping from default
F           'Activate new tuning parameters
AMPS=100     'Lower current limit to 10%
VT=-10000   'Set maximum velocity
ADT=100      'Set maximum accel/decel
MV          'Set Velocity mode
G           'Start motion
WHILE EA>-100 'Loop while position error is small
LOOP        'Loop back to WHILE
O=-100      'While pressed, declare home offset
S           'Abruptly stop trajectory
MP         'Switch to Position mode
VT=20000    'Set higher maximum velocity
PT=0        'Set target position to be home
G          'Start motion
TWAIT      'Wait for motion to complete
AMPS=1023  'Restore current limit to maximum
END        'End program

```

RELATED COMMANDS:

ADT=formula *Acceleration/Deceleration Target (see page 263)*

R EL=formula *Error Limit (see page 426)*

G *Start Motion (GO) (see page 473)*

MP *Mode Position (see page 613)*

MV *Mode Velocity (see page 624)*

R PRT=formula *Position, Relative Target (see page 683)*

R PT=formula *Position, (Absolute) Target (see page 690)*

X *Decelerate to Stop (see page 844)*

SADDR# Set Address

APPLICATION:	Communications control; supports the DS2020 Combitronic system
DESCRIPTION:	Set motor address
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT VALUE:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	Formulas and conditional testing through ADDR
UNITS:	Number
RANGE OF VALUES:	1 to 120
TYPICAL VALUES:	1 to 4
DEFAULT VALUE:	0= global address
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The SADDR{value} command is used to set the unit address of a SmartMotor™, where "value" is an integer between 0 and 120. Separate addresses allow multiple SmartMotors to share a common communication channel and still differentiate themselves.

The SADDR command is typically one of the first commands in a downloaded program. In an RS-485 network, where all communications go over the same two parallel wires, the SADDR command must be in the program. Whereas, in an RS-232 network, where communications travel from one motor to the next, addressing can be accomplished from a host or controller motor.

The address can be from 0 to 120. Address 0 is the global address (the motor has no unique address); it is used to talk to all motors on a network at once.

EXAMPLE:

```
SADDR1      'Set address to 1
```

When given a nonzero address, a SmartMotor begins to listen to commands after it receives its own unique address or the global address byte from the network. There is no need to repeat the address byte with subsequent commands intended for the same motor. The particular SmartMotor will continue to listen to commands until it receives a different address byte, after which commands are ignored. The echo function of the SmartMotor is not affected by the addressed state. That is, if told to echo, then a SmartMotor will echo regardless of whether it is listening to commands or not.

EXAMPLE:

```
'Example auto-addressing for four SmartMotors with SADDR command
'on an RS-232 daisy chain.
'This program code would be run at the same time
'in all motors on the chain at power-up.
ECHO          'Enable ECHO mode.
a=1           'User variable "a" to set address.
WAIT=2000    'Wait about 1/2 second to allow
              'power-up to each motor.
PRINT(#128,"a=a+1 ",#13) 'Print downstream to each motor.
WAIT=2000    'Wait about 1/2 second for each
              'motor to ECHO through the same
              'string to the next motor.

'NOTE: At this point, each motor will have run the exact same code
'causing successive motors downstream to receive the same command
'string from the number of motors upstream.
SWITCH a      'Check the value of "a"
  CASE 1
    SADDR1    'Set address to 1
  GOSUB10
  BREAK
  CASE 2
    SADDR2    'Set address to 2
  GOSUB20
  BREAK
  CASE 3
    SADDR3    'Set address to 3
  GOSUB30
  BREAK
  CASE 4
    SADDR4    'Set address to 4
  GOSUB40
  BREAK
ENDS
END
C10  'MOTOR 1 CODE
RETURN
C20  'MOTOR 2 CODE
RETURN
C30  'MOTOR 3 CODE
RETURN
C40  'MOTOR 4 CODE
RETURN
```

RELATED COMMANDS:

R ADDR=formula *Address (for RS-232 and RS-485) (see page 261)*



SAMP

Sampling Rate

APPLICATION:	System
DESCRIPTION:	Gets (reads) the sample rate in Hertz (Hz)
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RSAMP
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Hz
RANGE OF VALUES:	2000 to 16000
TYPICAL VALUES:	2000 to 16000
DEFAULT VALUE:	8000
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	RSAMP:3, x=SAMP:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The SAMP command is used to get (read) the sample rate in Hertz (Hz). You can issue SAMP to read sample rate at any time. This represents the update rate of the motion trajectory and PID loop. Therefore, knowing this rate can aid the choice of PID tuning parameters, or acceleration and velocity values.

This command is the preferred method for obtaining the sample rate. It is simple to include it in user programs.

NOTE: Instead of placing a hard-coded value in the program, any program that requires the sample rate should use this command.

The next table shows the possible SAMP report values.

Command	PID / Trajectory Update Rate	Period (µsec)	SAMP Report	RSP Report
PID1	16 kHz	62.5	16000	06250/5...
PID2	8 kHz (default)	125	8000	12500/5...
PID4	4 kHz	250	4000	25000/5...
PID8	2 kHz	500	2000	50000/5...

EXAMPLE:

```
a=SAMP 'Assign sample rate to variable a.
PRINT("The current sample rate is: ",a) 'Print info to terminal.
END
```

Program output is:

The current sample rate is: 8000

EXAMPLE: (Scale Factor Multipliers)

```
' SAMP: motor sample time in samples per second
' RES: encoder resolution in counts per rev
' 65536: internal fractional constant for motor unit calculations
' 60: conversion between seconds and minutes
' *1.0: way to typecast integer values out to full floating-point values
' af[2] thru af[7]: 32-bit, floating-point array variables

'These abbreviations are used in this code:
' NatAD (Native Accel{decel})
' NatVel (Native Velocity)
' RPM (revolutions per minute)
' RPS (revolutions per second)

'Calculating multipliers
af[2]=(((SAMP*1.0)/RES)*60)/65536
af[3]=(((RES*1.0)/SAMP)/60)*65536
af[4]=((SAMP*1.0)/RES)/65536
af[5]=((RES*1.0)/SAMP)*65536
af[6]=af[5]/SAMP
af[7]=af[4]*SAMP

'Input      Output
'NatVel -> RPM      Multiplier
'RPM      -> NatVel  Multiplier
'NatVel -> RPS      Multiplier
'RPS      -> NatVel  Multiplier
'RPS^2    -> NatAD   Multiplier
'NatAD    -> RPS^2   Multiplier

'Examples
'Suppose you wish to set a Velocity of 3000 RPM:
'(This method simply uses the above multiplier)
VT=3000*af[3] '3000 RPM desired speed multiplied by af[3]
              '(RPM to Native Velocity multiplier)

'Suppose you wish to read real time velocity in units of RPS:
s=VA*af[5]    'Converts native VA (actual velocity) into RPS
              'and assigns it to the variable "s"
```

END

RELATED COMMANDS:

R FW *Firmware Version (see page 471)*

R RES *Resolution (see page 702)*



APPLICATION:	Motion control
DESCRIPTION:	Applies the specified scale factor to subsequent acceleration (deceleration) values.
EXECUTION:	Immediate; remains in effect until otherwise commanded
CONDITIONAL TO:	Acceleration (deceleration) value is required
LIMITATIONS:	Command is not available for any Class 6 motors
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	For both arg1 and arg2, range is 0 to 2147483647
TYPICAL VALUES:	Typical range: 0 to 1000000
DEFAULT VALUE:	Default: 0 (feature disabled)
FIRMWARE VERSION:	5.x.4.58 (D/M); no Class 6
COMBITRONIC:	SCALEP(m,d):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The SCALEA(m,d) command applies a scale factor, which is calculated by m/d, to the acceleration (deceleration) value in any subsequent commands. The command syntax is SCALEA(m,d), where:

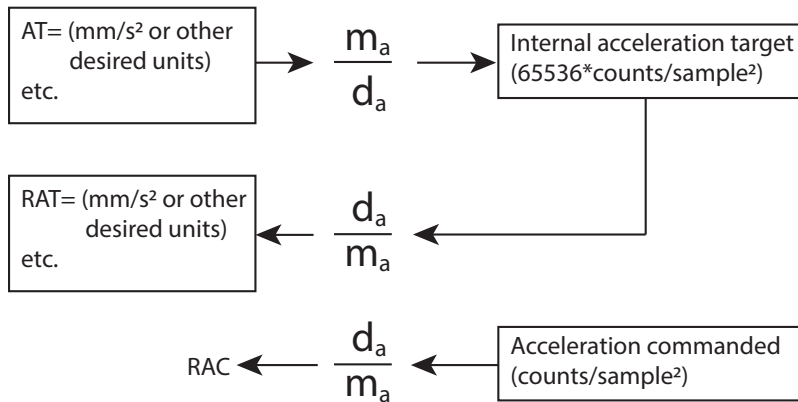
- m = multiplier
- d = divisor

The acceleration (deceleration) value scale factor remains in effect until SCALEA(0,0) is issued.

NOTE: SCALEA(0,0) deactivates acceleration (deceleration) scaling.

For example SCALEA(10,1) AT=1 internally sets the acceleration (deceleration) target to 10 (i.e., multiplies the actual setting by 10). Reported acceleration (deceleration) values will do the reverse (i.e., it takes the actual acceleration (deceleration) and divides by 10).

To determine the correct values for m and d, use the Motor Scaling tool in SMI. For details, see the SMI software help.

SCALEA(m_a,d_a) Diagram

For a table listing the commands that are affected by the SCALE commands, see Commands Affected by SCALE on page 903.

EXAMPLE:

```
SCALEA(10,1)  'Sets the acceleration (deceleration) scale factor to 10x.
' All subsequent acceleration (deceleration) values will be affected by this
scaling.
AT=100       'Acceleration target of 100 is actually 1000 due to 10x
              'scale factor.
SCALEA(0,0)  'Deactivates the acceleration (deceleration) scale factor.
```

RELATED COMMANDS:

SCALEP(m,d) *Scale Position Value (see page 726)*

SCALEV(m,d) *Scale Velocity Value (see page 728)*

Also, see Commands Affected by SCALE on page 903



SCALEP(m,d)

Scale Position Value

APPLICATION:	Motion control
DESCRIPTION:	Applies the specified scale factor to subsequent position values.
EXECUTION:	Immediate; remains in effect until otherwise commanded
CONDITIONAL TO:	Position value is required
LIMITATIONS:	Command is not available for any Class 6 motors
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	For both arg1 and arg2, range is 0 to 2147483647
TYPICAL VALUES:	Typical range: 0 to 1000000
DEFAULT VALUE:	Default: 0 (feature disabled)
FIRMWARE VERSION:	5.x.4.58 (D/M); no Class 6
COMBITRONIC:	SCALEP(m,d):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The SCALEP(m,d) command applies a scale factor, which is calculated by m/d, to the position value in any subsequent commands. The command syntax is SCALEP(m,d), where:

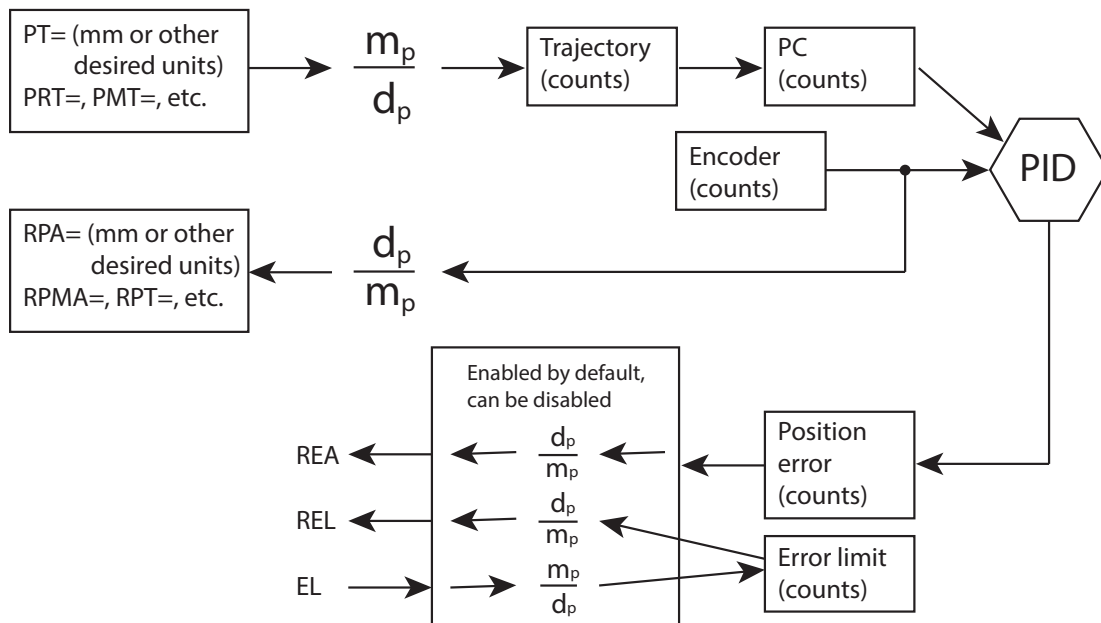
- m = multiplier
- d = divisor

The position value scale factor remains in effect until SCALEP(0,0) is issued.

NOTE: SCALEP(0,0) deactivates position scaling.

For example SCALEP(10,1) PT=1 internally sets the position target to 10 actual encoder counts. Reported position values will do the reverse (i.e., it takes the encoder position and divides by 10).

To determine the correct values for m and d, use the Motor Scaling tool in SMI. For details, see the SMI software help.

SCALEP(m_p,d_p) Diagram

For a table listing the commands that are affected by the SCALE commands, see Commands Affected by SCALE on page 903.

EXAMPLE:

```
SCALEP (10,1)    'Sets the position scale factor to 10x.
                  ' All subsequent position values will be affected by this scaling.
PT=1             'Position target of 1 is actually 10 due to 10x position scale
factor.
SCALEP (0,0)    'Deactivates the position scale factor.
```

RELATED COMMANDS:

SCALEA(m,d) *Scale Acceleration Value (see page 724)*

SCALEV(m,d) *Scale Velocity Value (see page 728)*

Also, see *Commands Affected by SCALE* on page 903



SCALEV(m,d)

Scale Velocity Value

APPLICATION:	Motion control
DESCRIPTION:	Applies the specified scale factor to subsequent velocity values.
EXECUTION:	Immediate; remains in effect until otherwise commanded
CONDITIONAL TO:	Position value is required
LIMITATIONS:	Command is not available for any Class 6 motors
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	For both arg1 and arg2, range is 0 to 2147483647
TYPICAL VALUES:	Typical range: 0 to 1000000
DEFAULT VALUE:	Default: 0 (feature disabled)
FIRMWARE VERSION:	5.x.4.58 (D/M); no Class 6
COMBITRONIC:	SCALEP(m,d):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The SCALEV(m,d) command applies a scale factor, which is calculated by m/d, to the velocity value in any subsequent commands. The command syntax is SCALEV(m,d), where:

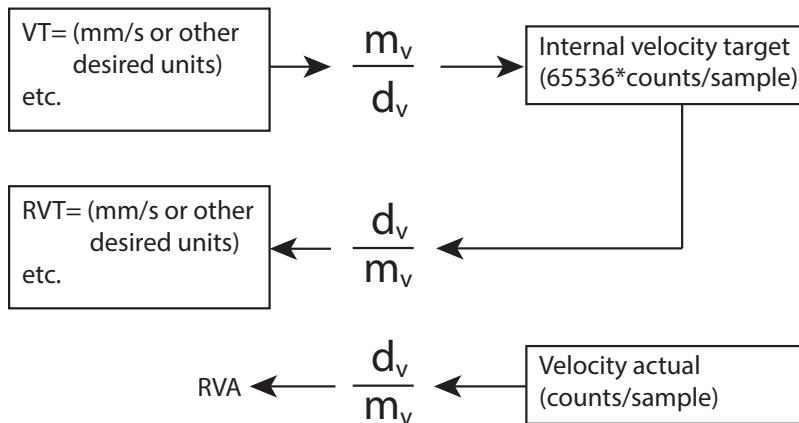
- m = multiplier
- d = divisor

The velocity value scale factor remains in effect until SCALEV(0,0) is issued.

NOTE: SCALEV(0,0) deactivates velocity scaling.

For example SCALEV(10,1) VT=1 internally scales the velocity target to 10 (i.e., multiplies the actual setting by 10). Reported velocity values will do the reverse (i.e., it takes the actual velocity and divides by 10).

To determine the correct values for m and d, use the Motor Scaling tool in SMI. For details, see the SMI software help.

SCALEV(m_v,d_v) Diagram

For a table listing the commands that are affected by the SCALE commands, see Commands Affected by SCALE on page 903.

EXAMPLE:

```
SCALEV(10,1)   'Sets the velocity scale factor to 10x.
' All subsequent velocity values will be affected by this scaling.
VT=1000        'Velocity target of 1000 is actually 10000 due to 10x
               'velocity scale factor.
SCALEP(0,0)    'Deactivates the velocity scale factor.
```

RELATED COMMANDS:

SCALEA(m,d) *Scale Acceleration Value (see page 724)*

SCALEP(m,d) *Scale Position Value (see page 726)*

Also, see Commands Affected by SCALE on page 903

APPLICATION:	Communications control
DESCRIPTION:	Reads the value from SDO; assigns value read to a variable
EXECUTION:	Immediate
CONDITIONAL TO:	Enabled through CANCTL(17,value), see CANCTL(function,value) on page 359.
LIMITATIONS:	Does not apply to Class 6 MT/MT2 systems
READ/REPORT:	RSDORD
WRITE:	Read only
LANGUAGE ACCESS:	Communications control
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x.4.30 (D/M) requires CAN option; 6.4.2.x (D)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The SDORD command gets (reads) the value from the specified SDO on a specified device. It can assign that value to a variable. To do this, use:

```
x=SDORD(follower addr, index, sub-index, length)
```

where:

follower addr	is a variable such as: x, ab[x], aw[x], al[x], or a constant such as: 2 (a motor with CAN address 2). The valid range is from 1 to 127. SDO's cannot be broadcast. If follower addr is to itself, then internal loopback is used instead.
index	is a variable such as: x, ab[x], aw[x], al[x], or a constant such as: 24640 (for object 6040hex).
sub-index	is a variable such as: x, ab[x], aw[x], al[x], or a constant such as: 0 (for sub-index 0).
length	is a variable such as: x, ab[x], aw[x], al[x], or a constant such as: 4. This is the number of bytes in the data; 1, 2, and 4 are the only valid values.

RSDORD(follower addr, index, sub-index, length) is the report version of the command. See above for a description of the parameters.

The function will pause (not proceed) a user program until a confirmation is received or a timeout occurs. In the event of a timeout, the value 0 is returned and an error code will be indicated. The program is also responsible for checking the error status. Refer to the next section on error handling.

The value returned will be interpreted as a signed value. In other words, if the length is 1 or 2 and if reporting or assigning it to a longer data type such as variable x, then the data is sign-extended.

Note that user interrupt events, ITR(...), can occur when waiting for an SDO operation to complete. However, do not call an SDO operation from multiple interrupt levels concurrently. For example, if an SDORD command is called in the main loop of a program, then do not call an SDORD or SDOWR in an interrupt routine.

ERROR HANDLING:

Errors are handled in this manner:

- Errors during a read or write to CANopen objects through SDO may *not* specifically cause a CAN error bit in status word 2. RCAN(4) command should always be inspected to verify success of the SDORD, SDOWR, and NMT commands.
- After each SDO read or write or NMT the specific code returned from the device will be readable using command RCAN(4)
- In the case of successful SDO read or write or NMT command, then the RCAN(4) shall report 0 immediately after such command. The user should always inspect for value 0 to know that an operation was successful.
- The user program is responsible for implementing any strategy for retry and/or giving up after a certain number of tries. The firmware simply reports the status of each attempt and will not automatically retry.

EXAMPLE: Read an SDO

```
x=SDORD(1, 24592,0,2)  ' Read 2 bytes from address 1,
                        ' object 0x6010, sub-index 0.
e=CAN(4)              ' Get any error information

y=SDORD(1, 24608,0,2)  ' Read 2 bytes from address 1,
                        ' object 0x6020, sub-index 0.
ee=CAN(4)             ' Get any error information

IF (e|ee)==0          ' Confirm the status of both SDO operations.
                      ' Success
    b=x               ' Set some example variable according
    c=y               ' to the data received.
    GOSUB(3)          ' Some routine to take action when this data is valid.
ELSE
    GOSUB(8)          ' Go do something to deal with error when read fails.
ENDIF
```

RELATED COMMANDS:

R CAN, CAN(arg) *CAN Bus Status (see page 357)*

CANCTL(function,value) *CAN Control (see page 359)*

NMT *Send NMT State (see page 626)*

SDOWR(...) *SDO Write (see page 732)*

APPLICATION:	Communications control
DESCRIPTION:	Writes a value to the specified SDO on a specified device
EXECUTION:	Immediate
CONDITIONAL TO:	Enabled through CANCTL(17,value), see CANCTL(function,value) on page 359.
LIMITATIONS:	Does not apply to Class 6 MT/MT2 systems
READ/REPORT:	Use SDORD and RSDORD
WRITE:	Write only
LANGUAGE ACCESS:	Communications control
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x.4.30 (D/M) requires CAN option; 6.4.2.x (D)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The SDOWR command writes a value to the specified SDO on a specified device. To do this, use:

```
SDOWR(follower addr, index, sub-index, length, value)
```

where:

follower addr	is a variable such as: x, ab[x], aw[x], al[x], or a constant such as: 2 (a motor with CAN address 2). The valid range is from 1 to 127. SDO's cannot be broadcast. If follower addr is to itself, then internal loopback is used instead.
index	is a variable such as: x, ab[x], aw[x], al[x], or a constant such as: 24640 (for object 6040hex).
sub-index	is a variable such as: x, ab[x], aw[x], al[x], or a constant such as: 0 (for sub-index 0).
length	is a variable such as: x, ab[x], aw[x], al[x], or a constant such as: 4. This is the number of bytes in the data; 1, 2, and 4 are the only valid values.
value	is a variable such as: x, ab[x], aw[x], al[x], or a constant such as: 1234 (max 32-bits of data). If the length is shorter than the variable or constant given, then the value is truncated regardless of sign.

The function will pause (not proceed) a user program until a confirmation is received or a timeout occurs. Refer to the next section on error handling.

Note that user interrupt events, ITR(...), can occur when waiting for an SDO operation to complete. However, do not call an SDO operation from multiple interrupt levels concurrently. For example, if an

SDOWR command is called in the main loop of a program, then do not call an SDORD or SDOWR in an interrupt routine.

ERROR HANDLING:

Errors are handled in this manner:

- Errors during a read or write to CANopen objects through SDO may *not* specifically cause a CAN error bit in status word 2. RCAN(4) command should always be inspected to verify success of the SDORD, SDOWR, and NMT commands.
- After each SDO read or write or NMT the specific code returned from the device will be readable using command RCAN(4)
- In the case of successful SDO read or write or NMT command, then the RCAN(4) shall report 0 immediately after such command. The user should always inspect for value 0 to know that an operation was successful.
- The user program is responsible for implementing any strategy for retry and/or giving up after a certain number of tries. The firmware simply reports the status of each attempt and will not automatically retry.

EXAMPLE: Write an SDO

```
a=1234
SDOWR(1,9029,0,4,a) ' Write 4 bytes to address 1,
IF CAN(4)==0      ' Confirm the status of the most recent SDO operation.
                  ' Success
      GOSUB(4)    ' Some routine to take action when the write succeeds.
ELSE
      GOSUB(9)    ' Go do something to deal with error when write fails.
ENDIF
```

RELATED COMMANDS:

R CAN, CAN(arg) *CAN Bus Status (see page 357)*

CANCTL(function,value) *CAN Control (see page 359)*

NMT *Send NMT State (see page 626)*

SDORD(...) *SDO Read (see page 730)*

Silence Outgoing Communications on Communications Port 0

APPLICATION:	Communications control
DESCRIPTION:	Motor prevented from printing to channel 0
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	TALK state
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The SILENT command causes the SmartMotor™ to suppress all PRINT messages from being transmitted on channel 0. Report commands originating from a user program are also suppressed if they are configured to transmit from COM channel 0 (see STDOUT=formula on page 764).

This command is typically used when a program has PRINT statements that may be interfering with debugging efforts. For instance, when opening a polling window or issuing serial commands during debugging, it may become necessary to suppress PRINT statements in a program.

The SILENT command does not prevent the SmartMotor from sending messages in response to incoming serial report commands from the host. Also, it does not interfere with ECHOing received serial communication over channel 0.

The TALK command negates the effect of SILENT and restores the motor's COM 0 port to its default operating state. For details, see TALK on page 771.

SILENT may be issued from the terminal or within a user program. However, the command is typically sent from a host.

EXAMPLE: (Shows the use of SILENT and TALK)

```

RUN?           'Wait here for the RUN command.
               'Set a=1 in the Terminal window to
               'allow print statements.
WHILE 1        'Endless loop
IF a==1 TALK ENDIF 'If variable a is set to 1, allow
               'PRINT statements on channel 0.
IF a==0 SILENT ENDIF 'If variable a is set to 0, suppress
                  'PRINT statements on channel 0.
PRINT("Position=", PA, #13) 'Print the Actual Position.
WAIT=3000      'Wait 3 seconds.
LOOP           'Loop back to WHILE 1 command.
END

```

RELATED COMMANDS:

ECHO *Echo Incoming Data on Communications Port 0 (see page 403)*

ECHO_OFF *Turn Off Echo on Communications Port 0 (see page 407)*

PRINT(...) *Print Data to Communications Port (see page 669)*

STDOUT=formula *Set Device Output (see page 764)*

SILENT1 *Silence Outgoing Communications on Communications Port 1 (see page 736)*

TALK *Talk on Communications Port 0 (see page 771)*

TALK1 *Talk on Communications Port 1 (see page 773)*

SILENT1

Silence Outgoing Communications on Communications Port 1

APPLICATION:	Communications control
DESCRIPTION:	Motor prevented from printing to channel 1
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	TALK1 state
FIRMWARE VERSION:	5.0.x, 5.16.x or 5.32.x series (D); 6.4.2.x (D)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The SILENT1 command causes the SmartMotor™ to suppress all PRINT1 messages from being transmitted on channel 1. Report commands originating from a user program are also suppressed if they are configured to transmit from COM channel 1 (see STDOUT=formula on page 764).

This command is typically used when a program has PRINT1 statements that may be interfering with debugging efforts. For instance, when opening a polling window or issuing serial commands during debugging, it may become necessary to suppress PRINT1 statements in a program.

The SILENT1 command does not prevent the SmartMotor from sending messages in response to incoming serial report commands from the host. Also, it does not interfere with ECHOing received serial communication over channel 1.

The TALK1 command negates the effect of SILENT1 and restores the motor's COM 1 port to its default operating state.

NOTE: These commands are typically sent from a host rather than existing within a program.

EXAMPLE: (Shows use of SILENT1 and TALK1)

```
RUN?          'Wait here for the RUN command.
              'Set a=1 in the Terminal window to allow print statements.
OCHN(RS4,1,N,9600,1,8,C) 'Open ports 4 and 5 as RS-485 channel 1.
WHILE 1       'Endless loop.
IF a==1 TALK1 ENDIF 'If variable a is set to 1, allow
                'PRINT statements on channel 1.
IF a==0 SILENT1 ENDIF 'If variable a is set to 0, suppress
                    'PRINT statements on channel 1.
PRINT1("Position=",PA,#13) 'Print the actual position.
WAIT=3000 'Wait 3 seconds.
LOOP       'Loop back to WHILE 1 command.
END
```

RELATED COMMANDS:

ECHO1 *Echo Incoming Data on Communications Port 1 (see page 406)*

ECHO_OFF1 *Turn Off Echo on Communications Port 1 (see page 409)*

PRINT1(...) *Print Data to Communications Port 1 (see page 677)*

STDOUT=formula *Set Device Output (see page 764)*

SILENT *Silence Outgoing Communications on Communications Port 0 (see page 734)*

TALK *Talk on Communications Port 0 (see page 771)*

TALK1 *Talk on Communications Port 1 (see page 773)*

SIN(value)

Sine

APPLICATION:	Math function
DESCRIPTION:	Gets the sine of the input value
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RSIN(value)
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Degrees input
RANGE OF VALUES:	Input in degrees (floating-point): 0.0 to 360.0 (larger values can be used, but it is not recommended; user should keep range within modulo 360) Output (floating-point): ± 1.0
TYPICAL VALUES:	Input in degrees (floating-point): 0.0 to 360.0 (larger values can be used, but it is not recommended; user should keep range within modulo 360) Output (floating-point): ± 1.0
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The SIN command takes an input angle in degrees and returns a floating-point sine:

$$af[1]=SIN(arg)$$

where *arg* is in degrees, and may be an integer (i.e., *a*, *aw*[0]) or floating-point variable (i.e., *af*[0]). Integer or floating-point constants may also be used (i.e., 23 or 23.7, respectively).

This command cannot have within the parenthesis: math operators, other parenthetical functions, or a Combitronic request from another motor. For example, *x=FABS(PA)* is allowed, but *x=FABS(PA:3)* is not allowed.

The result of this function is a floating-point type. If used in an equation, the operations in the equation that are processed after this function are automatically promoted to a float. This is dependent on the mathematical order of operations in the equation. As with other equations (e.g., *x=a+b*), the variable to the left of "=" may be an integer variable to accept the result. However, the value will be truncated to fit to that integer type. For example, the assignment "*aw*[0]=" will drop any fractional amount and truncate the result to the range -32768 to 32767 (*aw*[0]=100.5 will report as 100, and *aw*[0]=40000.0 will report as -25536).

Although the floating-point variables and their standard binary operations conform to IEEE-754 double precision, the floating-point square root and trigonometric functions only produce IEEE-754 single-precision results. For more details, see Variables and Math on page 198.

EXAMPLE:

```
af[0]=SIN(57.3)      'Set array variable = SIN(57.3)
Raf[0]              'Report value of af[0] variable
RSIN(57.3)          'Report SIN(57.3)
af[1]=42.3          '42.3 degrees
af[0]=SIN(af[1])    'Variables may be put in the parenthesis
Raf[0]
END
```

Program output is:

```
0.841510772
0.841510772
0.673012495
```

RELATED COMMANDS:

R ACOS(value) *Arccosine (see page 259)*
R ASIN(value) *Arcsine (see page 284)*
R ATAN(value) *Arctangent (see page 289)*
R COS(value) *Cosine (see page 372)*
R TAN(value) *Tangent (see page 775)*



APPLICATION:	Motion control
DESCRIPTION:	Disables the software limits
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	None
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	By default, software limits are disabled
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	SLD:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The SLD command is used to disable the software limits. The limits are enabled using the SLE command.

As an alternative to hardware limits connected to the limit inputs of the SmartMotor™, software limits are "virtual" limit switches that offer distinct advantages. For example, in the event the actual position of the motor strays beyond the desired region of operation, software limits can interrupt motion with a fault. Further, the limit fault is directionally sensitive, so it will cause a fault if motion is commanded beyond a limit that has been reached. For more details, see Limits and Fault Handling on page 207.

EXAMPLE: (Shows use of SLD, SLE, SLM, SLN and SLP)

```

EIGN(W,0)      'Make all onboard I/O inputs
ZS            'Clear errors
SLD          'Disable software limits
O=0          'Zero the encoder position CTR(0)
SLM(1)       'SLM(1) will make a soft limit trigger the flag AND
              'cause a fault
              'SLM(0) will make a soft limit trigger the flag AND
              'will NOT cause a fault
SLN=-8000    'Set the negative software limit to -8000 encoder counts
SLP=8000     'Set the positive software limit to 8000 encoder counts
SLE         'Enable software limits
MP          'Set the SmartMotor to position mode
ADT=100     'Set a value for accel/decel
VT=20000    'Set a value for velocity target
PT=7000 G TWAIT 'Move to absolute position 7000 (no fault)
PT=-7000 G TWAIT 'Move to absolute position -7000 (no fault)
PT=9000 G TWAIT 'Move to absolute position 9000
              'The motor will fault at position 8000 and set
              'these bits:
              '   Bo(Motor is off) in Status Word 0
              '   Brs(Historical positive S/W limit) in Status Word 1
WAIT=2000   'Wait two seconds
ZS         'Clear errors
PT=0 G TWAIT 'Move to absolute position 0 (no fault)
END

```

RELATED COMMANDS:

SLE *Software Limits, Enable (see page 742)*
R SLM(mode) *Software Limit Mode (see page 748)*
R SLN=formula *Software Limit, Negative (see page 750)*
R SLP=formula *Software Limit, Positive (see page 752)*



APPLICATION:	Motion control
DESCRIPTION:	Enables the software limits
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	None
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	By default, software limits are disabled (SLD)
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	SLE:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The SLE command is used to enable the software limits. The software limits are disabled using the SLD command. For details, see SLD on page 740.

As an alternative to hardware limits connected to the limit inputs of the SmartMotor™, software limits are "virtual" limit switches that offer distinct advantages. For example, in the event the actual position of the motor strays beyond the desired region of operation, software limits can interrupt motion with a fault. Further, the limit fault is directionally sensitive, so it will cause a fault if motion is commanded beyond a limit that has been reached. For more details, see Limits and Fault Handling on page 207.

EXAMPLE: (Shows use of SLD, SLE, SLM, SLN and SLP)

```

EIGN(W,0)      'Make all onboard I/O inputs
ZS             'Clear errors
SLD           'Disable software limits
O=0           'Zero the encoder position CTR(0)
SLM(1)        'SLM(1) will make a soft limit trigger the flag AND
              'cause a fault
              'SLM(0) will make a soft limit trigger the flag AND
              'will NOT cause a fault
SLN=-8000     'Set the negative software limit to -8000 encoder counts
SLP=8000      'Set the positive software limit to 8000 encoder counts
SLE           'Enable software limits
MP            'Set the SmartMotor to position mode
ADT=100       'Set a value for accel/decel
VT=20000      'Set a value for velocity target
PT=7000 G TWAIT 'Move to absolute position 7000 (no fault)
PT=-7000 G TWAIT 'Move to absolute position -7000 (no fault)
PT=9000 G TWAIT 'Move to absolute position 9000
              'The motor will fault at position 8000 and set
              'these bits:
              '   Bo(Motor is off) in Status Word 0
              '   Brs(Historical positive S/W limit) in Status Word 1
WAIT=2000     'Wait two seconds
ZS            'Clear errors
PT=0 G TWAIT  'Move to absolute position 0 (no fault)
END

```

RELATED COMMANDS:

SLD *Software Limits, Disable (see page 740)*
R SLM(mode) *Software Limit Mode (see page 748)*
R SLN=formula *Software Limit, Negative (see page 750)*
R SLP=formula *Software Limit, Positive (see page 752)*

SLEEP**Ignore Incoming Commands on Communications Port 0**

APPLICATION:	Communications control; supports the DS2020 Combitronic system
DESCRIPTION:	Prevents motor from executing channel 0 commands
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	WAKE state
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The SLEEP command is used to put a SmartMotor™ into sleep mode with respect to channel 0 serial commands. While in sleep mode, a SmartMotor will continue to echo (if in ECHO mode) all characters received over the network, but it will ignore all commands other than a WAKE command.

The most common use of the SLEEP command is to keep daisy-chained SmartMotors from responding to commands in a program that is being downloaded to another SmartMotor in the same chain.

If a program is running when a SmartMotor receives the SLEEP command, that program will continue to run. Messages originating from within the running program of a sleeping SmartMotor will be transmitted unless the motor is also in SILENT mode. For details, see SILENT on page 734.

SLEEP may be issued from the terminal or within a user program. SLEEP mode is terminated by the WAKE command.

EXAMPLE: (Shows use of SLEEP, SLEEP1, WAKE and WAKE1)

```
'These commands can be sent from the SMI software Terminal
>window to address three SmartMotors:
'0SADDR1
'1ECHO
'1SLEEP
'0SADDR2
'2ECHO
'2SLEEP
'0SADDR3
'3ECHO
'0WAKE
'A host program other than SMI can send the same commands, but the
'prefixed addressing is different. The 0, 1, 2 and 3 are actually
'0x80, 0x81, 0x82 and 0x83, respectively.
'The decimal equivalent of the hex values are 128, 129, 130 and 131.
'The next commands can be sent from a program in motor 1 to
'Motor 2:
PRINT(#130,"SLEEP",#13) 'Cause channel 0 (RS-232) of motor 2 to SLEEP.
PRINT(#130,"WAKE",#13) 'Cause channel 0 (RS-232) of motor 2 to WAKE.
PRINT(#130,"SLEEP1",#13) 'Cause channel 1 (RS-485) of motor 2 to SLEEP.
'through channel 0 (RS-232).
PRINT(#130,"WAKE1",#13) 'Cause channel 1 (RS-485) of motor 2 to WAKE
'through channel 0 (RS-232).

'Assuming channel 1 (RS-485) is open on all motors with the
'OCHN command, the same commands can be sent with the PRINT1
'command:
OCHN(RS4,1,N,9600,1,8,C) 'Open ports 4 and 5 as RS-485 channel 1.
PRINT1(#130,"SLEEP",#13) 'Cause channel 0 (RS-232) of motor 2 to SLEEP.
PRINT1(#130,"WAKE",#13) 'Cause channel 0 (RS-232) of motor 2 to WAKE.
PRINT1(#130,"SLEEP1",#13) 'Cause channel 1 (RS-485) of motor 2 to SLEEP.
PRINT1(#130,"WAKE1",#13) 'Cause channel 1 (RS-485) of motor 2 to WAKE.
END
```

RELATED COMMANDS:

ECHO *Echo Incoming Data on Communications Port 0 (see page 403)*

ECHO_OFF *Turn Off Echo on Communications Port 0 (see page 407)*

SLEEP1 *Ignore Incoming Commands on Communications Port 1 (see page 746)*

WAKE *Wake Communications Port 0 (see page 837)*

WAKE1 *Wake Communications Port 1 (see page 839)*

SLEEP1**Ignore Incoming Commands on Communications Port 1**

APPLICATION:	Communications control
DESCRIPTION:	Prevents motor from executing channel 1 commands
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	WAKE1 state
FIRMWARE VERSION:	5.0.x, 5.16.x or 5.32.x series (D); 6.4.2.x (D)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The SLEEP1 command is used to put a SmartMotor™ into sleep mode with respect to channel 1 serial commands. While in sleep mode, a SmartMotor will continue to echo (if in ECHO mode) all characters received over the network, but it will ignore all commands other than a WAKE1 command.

The most common use of the SLEEP1 command is to keep daisy-chained SmartMotors from responding to commands in a program that is being downloaded to another SmartMotor in the same chain.

If a program is running when a SmartMotor receives the SLEEP1 command, that program will continue to run. Messages originating from within the running program of a sleeping SmartMotor will be transmitted unless the motor is also in SILENT1 mode. For details, see SILENT1 on page 736.

SLEEP1 may be issued from the terminal or within a user program. SLEEP1 mode is terminated by the WAKE1 command.

EXAMPLE: (Shows use of SLEEP, SLEEP1, WAKE and WAKE1)

```
'These commands can be sent from the SMI software Terminal
>window to address three SmartMotors:
'0SADDR1
'1ECHO
'1SLEEP
'0SADDR2
'2ECHO
'2SLEEP
'0SADDR3
'3ECHO
'0WAKE
'A host program other than SMI can send the same commands, but the
'prefixed addressing is different. The 0, 1, 2 and 3 are actually
'0x80, 0x81, 0x82 and 0x83, respectively.
'The decimal equivalent of the hex values are 128, 129, 130 and 131.
'The next commands can be sent from a program in motor 1 to
'Motor 2:
PRINT(#130,"SLEEP",#13) 'Cause channel 0 (RS-232) of motor 2 to SLEEP.
PRINT(#130,"WAKE",#13) 'Cause channel 0 (RS-232) of motor 2 to WAKE.
PRINT(#130,"SLEEP1",#13) 'Cause channel 1 (RS-485) of motor 2 to SLEEP.
                          'through channel 0 (RS-232).
PRINT(#130,"WAKE1",#13) 'Cause channel 1 (RS-485) of motor 2 to WAKE
                          'through channel 0 (RS-232).

'Assuming channel 1 (RS-485) is open on all motors with the
'OCHN command, the same commands can be sent with the PRINT1
'command:
OCHN(RS4,1,N,9600,1,8,C) 'Open ports 4 and 5 as RS-485 channel 1.
PRINT1(#130,"SLEEP",#13) 'Cause channel 0 (RS-232) of motor 2 to SLEEP.
PRINT1(#130,"WAKE",#13) 'Cause channel 0 (RS-232) of motor 2 to WAKE.
PRINT1(#130,"SLEEP1",#13) 'Cause channel 1 (RS-485) of motor 2 to SLEEP.
PRINT1(#130,"WAKE1",#13) 'Cause channel 1 (RS-485) of motor 2 to WAKE.
END
```

RELATED COMMANDS:

ECHO1 *Echo Incoming Data on Communications Port 1 (see page 406)*
ECHO_OFF1 *Turn Off Echo on Communications Port 1 (see page 409)*
SLEEP *Ignore Incoming Commands on Communications Port 0 (see page 744)*
WAKE *Wake Communications Port 0 (see page 837)*
WAKE1 *Wake Communications Port 1 (see page 839)*



SLM(mode)

Software Limit Mode

APPLICATION:	Motion control
DESCRIPTION:	Gets or sets the soft limit mode
EXECUTION:	Immediate
CONDITIONAL TO:	Software limits enabled (SLE)
LIMITATIONS:	N/A
READ/REPORT:	RSLM
WRITE:	Read/write
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	0 or 1
TYPICAL VALUES:	0 or 1
DEFAULT VALUE:	1
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	SLM(0):3, a=SLM:3, RSLM:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The SLM command gets (reads) or sets the soft limit mode:

- SLM
Get soft limit mode (e.g., a=SLM)
- SLM(...)
Set soft limit mode

When setting the soft limit, it can be made to trigger a flag only (not cause a fault) or to trigger the flag and cause a fault:

- SLM (0)
Make a soft limit trigger the flag only; will not cause a fault
- SLM (1)
Make a soft limit trigger the flag *and* cause a fault (default mode)

As an alternative to hardware limits connected to the limit inputs of the SmartMotor™, software limits are "virtual" limit switches that offer distinct advantages. For example, in the event the actual position of the motor strays beyond the desired region of operation, software limits can interrupt motion with a fault. Further, the limit fault is directionally sensitive, so it will cause a fault if motion is commanded beyond a limit that has been reached. For more details, see Limits and Fault Handling on page 207.

EXAMPLE: (Shows use of SLD, SLE, SLM, SLN and SLP)

```

EIGN(W,0)      'Make all onboard I/O inputs
ZS            'Clear errors
SLD          'Disable software limits
O=0          'Zero the encoder position CTR(0)
SLM(1)       'SLM(1) will make a soft limit trigger the flag AND
              'cause a fault
              'SLM(0) will make a soft limit trigger the flag AND
              'will NOT cause a fault
SLN=-8000    'Set the negative software limit to -8000 encoder counts
SLP=8000     'Set the positive software limit to 8000 encoder counts
SLE          'Enable software limits
MP           'Set the SmartMotor to position mode
ADT=100     'Set a value for accel/decel
VT=20000    'Set a value for velocity target
PT=7000 G TWAIT 'Move to absolute position 7000 (no fault)
PT=-7000 G TWAIT 'Move to absolute position -7000 (no fault)
PT=9000 G TWAIT 'Move to absolute position 9000
              'The motor will fault at position 8000 and set
              'these bits:
              '   Bo(Motor is off) in Status Word 0
              '   Brs(Historical positive S/W limit) in Status Word 1
WAIT=2000   'Wait two seconds
ZS          'Clear errors
PT=0 G TWAIT 'Move to absolute position 0 (no fault)
END

```

RELATED COMMANDS:

SLD *Software Limits, Disable (see page 740)*
 SLE *Software Limits, Enable (see page 742)*
 R SLM(mode) *Software Limit Mode (see page 748)*
 R SLN=formula *Software Limit, Negative (see page 750)*
 R SLP=formula *Software Limit, Positive (see page 752)*



SLN=formula

Software Limit, Negative

APPLICATION:	Motion control
DESCRIPTION:	Gets (reads) or sets the left/negative software limit
EXECUTION:	Immediate
CONDITIONAL TO:	Software limits enabled (SLE)
LIMITATIONS:	N/A
READ/REPORT:	RSLN
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	Encoder counts
RANGE OF VALUES:	-2147483648 to 2147483647 NOTE: values at this extreme range may not trigger because they are at the wrap point (see details)
TYPICAL VALUES:	-1000000 to 0
DEFAULT VALUE:	-2147483648 (effectively disabled)
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	SLN:3=1234, a=SLN:3, RSLN:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

NOTE: This command is affected by the SCALEP command. For details, see SCALEP(m,d) on page 726. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

The SLN command is used to get (read) or set the left/negative software limit:

- SLN
Get left/negative software limit
- SLN=...
Set left/negative software limit

The range of allowed values is from -2147483648 to 2147483647. However, this extreme range should not be used because the soft limits may not trigger at the wrap point due to the mathematical sign change. Further, it is not recommended to operate absolute-position applications near the wrap point. This limitation is partly related to speed (encoder counts per PID sample).

Therefore, as a rough estimate, the range from -2147400000 to 2147400000 is more realistic.

NOTE: SLP should typically be set to a higher value than SLN.

As an alternative to hardware limits connected to the limit inputs of the SmartMotor™, software limits are "virtual" limit switches that offer distinct advantages. For example, in the event the actual position of the motor strays beyond the desired region of operation, software limits can interrupt motion with a

fault. Further, the limit fault is directionally sensitive, so it will cause a fault if motion is commanded beyond a limit that has been reached. For more details, see Limits and Fault Handling on page 207.

EXAMPLE: (Shows use of SLD, SLE, SLM, SLN and SLP)

```

EIGN(W,0)      'Make all onboard I/O inputs
ZS             'Clear errors
SLD           'Disable software limits
O=0           'Zero the encoder position CTR(0)
SLM(1)        'SLM(1) will make a soft limit trigger the flag AND
              'cause a fault
              'SLM(0) will make a soft limit trigger the flag AND
              'will NOT cause a fault
SLN=-8000     'Set the negative software limit to -8000 encoder counts
SLP=8000      'Set the positive software limit to 8000 encoder counts
SLE           'Enable software limits
MP           'Set the SmartMotor to position mode
ADT=100       'Set a value for accel/decel
VT=20000      'Set a value for velocity target
PT=7000 G TWAIT 'Move to absolute position 7000 (no fault)
PT=-7000 G TWAIT 'Move to absolute position -7000 (no fault)
PT=9000 G TWAIT 'Move to absolute position 9000
              'The motor will fault at position 8000 and set
              'these bits:
              '   Bo(Motor is off) in Status Word 0
              '   Brs(Historical positive S/W limit) in Status Word 1
WAIT=2000     'Wait two seconds
ZS           'Clear errors
PT=0 G TWAIT  'Move to absolute position 0 (no fault)
END

```

RELATED COMMANDS:

SLD *Software Limits, Disable (see page 740)*
 SLE *Software Limits, Enable (see page 742)*
 R SLM(mode) *Software Limit Mode (see page 748)*
 R SLP=formula *Software Limit, Positive (see page 752)*



SLP=formula

Software Limit, Positive

APPLICATION:	Motion control
DESCRIPTION:	Gets (reads) or sets the right/positive software limit
EXECUTION:	Immediate
CONDITIONAL TO:	Software limits enabled (SLE)
LIMITATIONS:	N/A
READ/REPORT:	RSPL
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	Encoder counts.
RANGE OF VALUES:	-2147483648 to 2147483647 NOTE: values at this extreme range may not trigger because they are at the wrap point. See details.
TYPICAL VALUES:	0 to 1000000
DEFAULT VALUE:	2147483647 (effectively disabled)
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	SLP:3=1234, a=SLP:3, RSPL:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

NOTE: This command is affected by the SCALEP command. For details, see SCALEP(m,d) on page 726. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

The SLP command is used to get (read) or set the right/positive software limit:

- SLP
Get right/positive software limit
- SLP=...
Set right/positive software limit

The range of allowed values is -2147483648 to 2147483647. However, this extreme range should not be used because the soft limits may not trigger at the wrap point due to the mathematical sign change. It is not recommended to operate absolute position applications near the wrap point. This limitation is partly related to speed (encoder counts per PID sample). Therefore, as a rough estimate, the range -2147400000 to 2147400000 is more realistic.

NOTE: SLP should typically be set to a higher value than SLN.

As an alternative to hardware limits connected to the limit inputs of the SmartMotor™, software limits are "virtual" limit switches that offer distinct advantages. For example, in the event the actual position of the motor strays beyond the desired region of operation, software limits can interrupt motion with a

fault. Further, the limit fault is directionally sensitive, so it will cause a fault if motion is commanded beyond a limit that has been reached. For more details, see Limits and Fault Handling on page 207.

EXAMPLE: (Shows use of SLD, SLE, SLM, SLN and SLP)

```

EIGN(W,0)      'Make all onboard I/O inputs
ZS             'Clear errors
SLD           'Disable software limits
O=0           'Zero the encoder position CTR(0)
SLM(1)        'SLM(1) will make a soft limit trigger the flag AND
              'cause a fault
              'SLM(0) will make a soft limit trigger the flag AND
              'will NOT cause a fault
SLN=-8000     'Set the negative software limit to -8000 encoder counts
SLP=8000      'Set the positive software limit to 8000 encoder counts
SLE           'Enable software limits
MP            'Set the SmartMotor to position mode
ADT=100       'Set a value for accel/decel
VT=20000      'Set a value for velocity target
PT=7000 G TWAIT 'Move to absolute position 7000 (no fault)
PT=-7000 G TWAIT 'Move to absolute position -7000 (no fault)
PT=9000 G TWAIT 'Move to absolute position 9000
              'The motor will fault at position 8000 and set
              'these bits:
              '   Bo(Motor is off) in Status Word 0
              '   Brs(Historical positive S/W limit) in Status Word 1
WAIT=2000     'Wait two seconds
ZS            'Clear errors
PT=0 G TWAIT  'Move to absolute position 0 (no fault)
END

```

RELATED COMMANDS:

SLD *Software Limits, Disable (see page 740)*
 SLE *Software Limits, Enable (see page 742)*
 R SLM(mode) *Software Limit Mode (see page 748)*
 R SLN=formula *Software Limit, Negative (see page 750)*

SNAME("string") Set PROFINET Station Name

APPLICATION:	Communications control
DESCRIPTION:	Sets the PROFINET station name
EXECUTION:	Immediate
CONDITIONAL TO:	PROFINET versions of motors only.
LIMITATIONS:	SmartMotor command length limit restricts the SNAME to 54 characters. There are several important limitations imposed per PROFINET standards, see below.
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	String
RANGE OF VALUES:	Characters: 0 through 9, lowercase a through z; period (".") and hyphen ("-") with restrictions. See the <i>Class 6 SmartMotor™ PROFINET Guide</i> .
TYPICAL VALUES:	axis1
DEFAULT VALUE:	smc6dev01
FIRMWARE VERSION:	6.x (D/M) requires EPN option; no Class 5
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The SNAME command sets a unique PROFINET station name. For proper PROFINET operation, each SmartMotor must have a unique station name set with the SNAME instruction.

The command setting is nonvolatile. Therefore, it will be remembered between power cycles.

There are specific limitations to the SNAME conventions imposed by PROFINET standards.

For more details, see the *Class 6 SmartMotor™ PROFINET Guide*.

EXAMPLE: (Change the nonvolatile station name for PROFINET within a user program)

```
...
SNAME ("mymotor1")
a=ETH (0)
IF (a&2)
    Z      'Execute reset if Station Name changed
ENDIF
...
```

RELATED COMMANDS:

ETHCTL(function,value) *Control Industrial Ethernet Network Features (see page 456)*

IPCTL(function,"string") *Set IP Address, Subnet Mask or Gateway (see page 515)*



APPLICATION:	System
DESCRIPTION:	The bootloader revision
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RSP2
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	-1, 0-127
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	a=SP2:3, RSP2:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The SP2 command returns a single value of the bootloader revision. This number can be used in a program to inspect the firmware version. For example:

```
x=SP2
```

assigns the value of the bootloader revision to the variable x.

A report version of the command, RSP2, is also available.

EXAMPLE:

```
RSP2    5
```

RELATED COMMANDS:

R FW *Firmware Version (see page 471)*

RSP *Report Sampling Rate and Firmware Revision (see page 710)*

RSP1 *Report Firmware Compile Date (see page 712)*

RSP5 *Report Network Card Firmware Version (see page 713)*



APPLICATION:	System
DESCRIPTION:	Reports the serial number of the motor
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RSP6
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x.4.58 (D/M); 6.x.2.37 (M), 6.4.2.x (D)
COMBITRONIC:	a=SP6:3, RSP6:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The SP6 command returns a single value of the serial number. This number can be used in a program to inspect the serial number. For example:

```
x=SP6
```

assigns the value of the serial number to the variable x.

A report version of the command, RSP6, is also available.

EXAMPLE:

```
RSP6    123456
```

RELATED COMMANDS:

R FW *Firmware Version (see page 471)*

RSP *Report Sampling Rate and Firmware Revision (see page 710)*

R SP2 *Bootloader Version (see page 755)*

RSP1 *Report Firmware Compile Date (see page 712)*

RSP5 *Report Network Card Firmware Version (see page 713)*

SQRT(value)

Integer Square Root

APPLICATION:	Math function
DESCRIPTION:	Gets (reads) the integer square root
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RSQRT(value)
WRITE:	Read only
LANGUAGE ACCESS:	N/A
UNITS:	Positive integer
RANGE OF VALUES:	Input: Positive integer 0 - 2147483647 Output: Positive integer 0 - 46340
TYPICAL VALUES:	Input: Positive integer 0 - 2147483647 Output: Positive integer 0 - 46340
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

NOTE: For a floating-point version, see FSQRT(value) on page 469.

The SQRT command gets (reads) the integer square root of a variable or value:

```
=SQRT(x)
```

where x = any positive integer ≥ 0 .

The integer square root of x is the greatest integer \leq the square root of x. For example, if x is 6, the integer square root is 2 because 2 is the greatest integer \leq the square root of 6. Therefore, all inputs for x from 4 through 8 will give the result 2; when x is 9 through 15, the result changes to 3.

EXAMPLE:

```
a=9           'Set variable a = 9
r=SQRT(4)     'Set variable r = SQRT(4)
RSQRT(4)
s=SQRT(6)     'Set variable s = SQRT(6)
RSQRT(6)
t=SQRT(8)     'Set variable t = SQRT(8)
RSQRT(8)
u=SQRT(a)     'Set variable u = SQRT(a)
RSQRT(a)
PRINT(r," ",s," ",t," ",u,#13) 'Print value of each variable
END
```

Program output is:

2
2
2
3
2, 2, 2, 3

RELATED COMMANDS:

^R FSQRT(value) *Floating-Point Square Root (see page 469)*

SRC(enc_src)

Source, Follow and/or Cam Encoder

APPLICATION:	Motion control
DESCRIPTION:	Set the input source for Follow and Cam modes
EXECUTION:	Buffered until a G command is issued
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	-2,-1,0,1,2
TYPICAL VALUES:	-2,-1,0,1,2
DEFAULT VALUE:	1
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	SRC(2):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The SRC(enc_src) command is used to select the input source used in Follow and Cam modes.

The SRC command allows the SmartMotor to use the many advanced following and camming functions even without an external encoder input. For example, through the use of the SRC command either the external encoder or a fixed-rate "virtual encoder" can be used as the input source to the cam. This fixed-rate encoder also works through the Follow mode, so the actual rate into the cam can be set. For more details, see Follow Mode with Ratio (Electronic Gearing) on page 140.

Refer to the next table for valid enc_src values.

Value of enc_src	Result
-2	-1 * internal time base at PID rate
-1	-1 * external encoder: MF0 or MS0
0	Null (no counts, standstill)
1	External encoder: MF0 or MS0
2	Internal time base at PID rate

EXAMPLE: (Cam program example; uses virtual encoder)

```

CTE (1)           'Erase all EEPROM tables.
CTA (7,4000)      'Create 7-point table at each 4K encoder increment.
CTW (0)           'Add 1st point.
CTW (1000)        'Add 2nd point; go to point 1000 from start.
CTW (3000)        'Add 3rd point; go to point 3000 from start.
CTW (4000)        'Add 4th point; go to point 4000 from start.
CTW (1000)        'Add 5th point; go to point 1000 from start.
CTW (-2000)       'Add 6th point; go to point -2000 from start.
CTW (0)           'Add 7th point; return to starting point.
                  'Table has now been written to EEPROM.

SRC (2)           'Use the virtual encoder.
MCE (0)           'Force linear interpolation.
MCW (1,0)         'Use table 1 from point 0.
MFMUL=1          'Simple 1:1 ratio from virtual encoder.
MFDIV=1          'Simple 1:1 ratio from virtual encoder.
MFA (0) MFD (0)  'Disable virtual encoder ramp-up/
                  'ramp-down sections.
MFSLEW (24000,1) 'Table is 6 segments * 4000 encoder
                  'counts each.
                  'Specify the second argument as a 1 to
                  'force this number as the output total of
                  'the virtual encoder into the cam.
MFSDC (-1,0)     'Disable virtual encoder profile repeat.
MC               'Enter Cam mode.
G               'Begin move.
END

```

RELATED COMMANDS:

G *Start Motion (GO)* (see page 473)
MFR *Mode Follow Ratio* (see page 600)
R MFDIV=formula *Mode Follow Divisor* (see page 588)
R MFMUL=formula *Mode Follow Multiplier* (see page 598)
MSR *Mode Step Ratio* (see page 618)

STACK

Stack Pointer Register, Clear

APPLICATION:	Program execution and flow control
DESCRIPTION:	Reset user program subroutine return stack
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT VALUE:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The STACK command empties the queue of pending (GOSUB) RETURN addresses, clears active interrupts from the stack (but new interrupt events remain pending), and resets any PAUSE statements.

NOTE: Use DITR() or EITR() before the STACK command to stop any pending interrupt events from reoccurring. Additionally, DITR() will prevent future calls.

In order to execute the RETURN program statement, the processor needs to be able to recall the program address point where it should return. The "stack" is a region where these addresses are stored.

A maximum of nine address locations can be stored within the stack. This means that if a tenth GOSUB is called before any intervening RETURN statements, the stack will overflow and the program execution may fail. The stack region is managed using a pointer to the currently effective return address storage location. The STACK command directly resets this pointer to its initial (starting) condition. By doing this, the STACK command clears all RETURN addresses in the stack queue.

NOTE: Care should be taken when the STACK command is used. Issuing STACK will cause any subsequent RETURN command to be ignored. Therefore, proper program flow, with GOTO commands or otherwise, should be used to prevent a memory mapping error.

Because the GOSUB command may be issued serially to the SmartMotor, it may be possible to overflow the stack regardless of the downloaded program code. The STACK command could also be issued through serial communications to clear the stack and prevent overflow. However, that method is not recommended because it would be difficult to know what line of code the motor may be running at that time.

EXAMPLE:

```

x=0    'Set variable x equal to zero
GOTO0  'Go directly to the C0 label
C10
PRINT(#13, "NO PROGRAM CRASH")
RETURN
END
'These commands intentionally call subroutines without RETURN commands
C0 x=x+1 PRINT("x=",x,#13) GOSUB1 'First GOSUB without return.
C1 x=x+1 PRINT("x=",x,#13) GOSUB2 'Second GOSUB without return.
C2 x=x+1 PRINT("x=",x,#13) GOSUB3 'Third GOSUB without return.
C3 x=x+1 PRINT("x=",x,#13) GOSUB4 'Fourth GOSUB without return.
C4 x=x+1 PRINT("x=",x,#13) GOSUB5 'Fifth GOSUB without return.
C5 x=x+1 PRINT("x=",x,#13) GOSUB6 'Sixth GOSUB without return.
C6 x=x+1 PRINT("x=",x,#13) GOSUB7 'Seventh GOSUB without return.
C7 x=x+1 PRINT("x=",x,#13) GOSUB8 'Eighth GOSUB without return.
C8 x=x+1 PRINT("x=",x,#13) GOSUB9 'Ninth GOSUB without return.
C9 x=x+1 PRINT("x=",x,#13) 'GOSUB10 'if this GOSUB is called,
    'the program WILL crash!
STACK  'Reset internal stack, which
GOSUB10 'allows this GOSUB to execute without crashing the program.
PRINT(#13,"RETURN FROM GOSUB10 OK",#13)
END

```

Program output is:

```

x=1
x=2
x=3
x=4
x=5
x=6
x=7
x=8
x=9
x=10

```

```

NO PROGRAM CRASH
RETURN FROM GOSUB10 OK

```

The previous example does not show the preferred way to write code. It is provided to show where the STACK command would be used to prevent program crashes.

Often, the STACK command is used after an error or motor-protection fault is detected. Then, immediately after the STACK command, a RUN, END or GOTO command (located near the top of the program) is issued to recover.

RELATED COMMANDS:

END *End Program Code Execution (see page 439)*

GOSUB(label) *Subroutine Call (see page 480)*

GOTO(label) *Branch Program Flow to a Label (see page 482)*

PAUSE *Pause Program Execution (see page 648)*

RUN *Run Program (see page 714)*

RUN? *Halt Program Execution Until RUN Received (see page 716)*

STDOUT=formula Set Device Output

APPLICATION:	Communications control
DESCRIPTION:	Specify where report commands are printed
EXECUTION:	Immediate
CONDITIONAL TO:	Available serial ports and/or fieldbus options; see the next table
LIMITATIONS:	For Class 5 motors, does not redirect PRINT, only reports; for Class 6 motors, the PRINT command is redirected; see the next table
READ/REPORT:	N/A
WRITE:	Write only
LANGUAGE ACCESS:	Assignment
UNITS:	Port number
RANGE OF VALUES:	See the next table
TYPICAL VALUES:	0, 1 (Class 5 & 6 D-style) 0 (Class 5 M-style) 0, 8 (Class 6 M-style)
DEFAULT VALUE:	0 (Class 5 & 6 D-style, Class 5 M-style) 8 (Class 6 M-style)
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The STDOUT command is used to select the motor output channel for report commands.

Motor type	Com 0 type	Com 1 type	Com 8 type	STDOUT=default	STDOUT=range of values	Report to	PRINT() to	PRINT0() to	PRINT1() to	PRINT8() to
Class 5 D-style	RS-232	RS-485	N/A	0	0 (Com 0: RS-232 or RS-485 with external adapter, except CDS7), 1 (Com 1: RS-485), 4 (CANopen encapsulation), 5 (Modbus encapsulation on Com 0), 6 (Modbus encapsulation on Com 1)	STDOUT setting	Com 0	N/A	Com 1	N/A
Class 6 D-style	RS-232	RS-485	USB	0	0 (Com 0: RS-232), 1 (Com 1: RS-485), 4 (CANopen encapsulation), 5 (Modbus encapsulation on Com 0), 6 (Modbus encapsulation on Com 1), 8 (USB)	STDOUT setting	STDOUT setting	Com 0	Com 1	USB
Class 5 M-style	RS-485	N/A	N/A	0	0 (Com 0: RS-485), 4 (CANopen encapsulation), 5 (Modbus encapsulation on Com 0)	STDOUT setting	Com 0	N/A	N/A	N/A

Motor type	Com 0 type	Com 1 type	Com 8 type	STDOUT= default	STDOUT= range of values	Report to	PRINT() to	PRINT0() to	PRINT1() to	PRINT8() to
Class 6 M-style	RS-485	N/A	USB	8	0 (Com 0: RS-485), 5 (Modbus encapsulation on Com 0), 8 (USB)	STDOUT setting	STDOUT setting	Com 0	N/A	USB

EXAMPLE:

```
OCHN (RS4,1,N,9600,1,8,C) 'Open ports 4 and 5 as RS-485 channel 1.
STDOUT=0 'Channel 0 is selected for output of report commands.
RPA      'The Absolute Position will be sent out channel 0 (RS-232).
STDOUT=1 'Channel 1 is selected for output of report commands.
RPA      'The Absolute Position will be sent out channel 1 (RS-485).
END
```

RELATED COMMANDS:

(none)

SWITCH formula

Switch, Program Flow Control

APPLICATION:	Program execution and flow control
DESCRIPTION:	Multiple choice branch for program execution
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	Can only be executed from within user program
REPORT VALUE:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The SWITCH command allows program flow control based on specific integer values of a formula, a specific parameter or a variable.

The execution time is similar to the equivalent IF formula control block. Therefore, placing the most likely CASE values at the top of the CASE list will yield faster program execution times.

At execution time, the program interpreter evaluates the SWITCH formula value and then tests the CASE numbers for an equal value in the programmed order.

- If the SWITCH formula value does equal the CASE number, then program execution continues with the command immediately after.
- If the SWITCH formula value does not equal the CASE number, then the next CASE statement is evaluated.
- If the SWITCH formula value does not equal any CASE number, then the DEFAULT entry point is used.
- If the SWITCH formula value does not equal any CASE number and there is no DEFAULT case, then program execution passes through the SWITCH to the ENDS without performing any commands.

If a BREAK is encountered, then program execution branches to the instruction or label after the ENDS of the SWITCH control block. BREAK can be used to isolate CASEs. Without BREAK, the CASE number syntax is transparent and program execution continues at the next instruction. That is, you will run into the next CASE number code sequence.

Each SWITCH control block must have at least one CASE number defined plus one, and only one, ENDS statement. SWITCH is not a valid terminal command — it is only valid within a user program.

EXAMPLE:

Consider this code fragment:

```

SWITCH v
  CASE 1
    PRINT (" v = 1 ",#13)
  BREAK
  CASE 2
    PRINT (" v = 2 ",#13)
  BREAK
  CASE 3
    PRINT (" v = -23 ",#13)
  BREAK
  DEFAULT
    PRINT ("v IS NOT 1, 2 OR -23",#13)
  BREAK
ENDS

```

The first line, SWITCH v, lets the SmartMotor™ know that it is checking the value of the variable v. Each subsequent CASE begins the section of code that tells the SmartMotor what to do if v is equal to that case.

EXAMPLE:

```

a=-3                                     'Assign a value
WHILE a<4
  PRINT (#13,"a=",a," ")
  SWITCH a                               'Test the value
    CASE 3
      PRINT ("MAX VALUE",#13)
    BREAK
    CASE -1                               'Negative test values are valid
    CASE -2                               'Note no BREAK here
    CASE -3
      PRINT ("NEGATIVE")
    BREAK                                 'Note use of BREAK
    CASE 0
      PRINT ("ZERO")                     'Zero test value is valid
      PRINT ("ZERO")                     'Note order is random
      DEFAULT                             'The default case
      PRINT ("NO MATCH VALUE")
    BREAK
  ENDS                                   'Need not be numerical
  a=a+1
LOOP
END

```

Program output is:

```
a=-3 NEGATIVE
a=-2 NEGATIVE
a=-1 NEGATIVE
a=0 ZERO
a=1 NO MATCH VALUE
a=2 NO MATCH VALUE
a=3 MAX VALUE
```

RELATED COMMANDS:

BREAK *Break from CASE or WHILE Loop (see page 331)*
CASE *formula Case Label for SWITCH Block (see page 360)*
DEFAULT *Default Case for SWITCH Structure (see page 388)*
ENDS *End SWITCH Structure (see page 443)*



APPLICATION:	Motion control
DESCRIPTION:	Torque value for Torque mode
EXECUTION:	Buffered until a G command is issued
CONDITIONAL TO:	Torque mode (MT)
LIMITATIONS:	N/A
REPORT VALUE:	RT
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	Fraction of available torque
RANGE OF VALUES:	-32767 to 32767
TYPICAL VALUES:	-10000 and 10000
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	T:3=1234, a=T:3, RT:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The MT command enables Torque mode. In this mode, the motor is commanded to develop a specific output effort, which is set by T=formula. Where the values are from -32767 to 32767, T=-32767 results in full torque in the negative direction. The encoder still tracks position and can still be read with the PA variable, but the PID loop is off and the motor does not servo or run a trajectory.

In voltage commutation modes (MDT, MDE and MDS), MT sets the PWM signal to the drive at a fixed percentage. For any given setting of T and no applied load, there will be a velocity at which the Back EMF (BEMF) of the motor causes the acceleration to stop and the velocity to hold nearly constant. Under the no load or static load conditions, the T command will control velocity. As the load increases, the velocity decreases.

In current-control commutation, MDC, T= sets a request for current, which is proportional to torque.



CAUTION: There is no inherent speed-limiting behavior when using MDC mode.

Any previous faults must be cleared before issuing the G command.

When setting larger values of T, the effect can be an abrupt current spike. This is a result of the motor requiring more current as it moves from a standstill and accelerates to speed. To reduce the impact, the TS= command can be used to gently apply the current.

EXAMPLE: (Increases torque, one unit every PID sample period, up to 8000 units)

```
MT          'Select torque mode
T=8000      'Final torque after the TS ramp that we want
TS=65536    'Increase the torque by 1 unit of T per PID sample
G          'Begin move
```

RELATED COMMANDS:

MT *Mode Torque (see page 620)*

R TS=formula *Torque Slope (see page 786)*

APPLICATION:	Communications control
DESCRIPTION:	Motor restored to print on channel 0
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	TALK state
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

NOTE: This command is typically sent from a host rather than existing within a SmartMotor program.

The TALK command restores the motor's ability to print messages to the serial communication channel 0 if that ability was previously suppressed with the SILENT command. This command is typically used after downloading a user program to a SmartMotor™ within a daisy chain. It could also be used to "un-silence" a debug routine.

TALK may be issued from the terminal or within a user program. However, the command is typically sent from a host.

EXAMPLE: (Shows the use of SILENT and TALK)

```

RUN?           'Wait here for the RUN command.
               'Set a=1 in the Terminal window to
               'allow print statements.
WHILE 1       'Endless loop
IF a==1 TALK ENDIF 'If variable a is set to 1, allow
               'PRINT statements on channel 0.
IF a==0 SILENT ENDIF 'If variable a is set to 0, suppress
               'PRINT statements on channel 0.
PRINT("Position=", PA, #13) 'Print the Actual Position.
WAIT=3000     'Wait 3 seconds.
LOOP         'Loop back to WHILE 1 command.
END

```

RELATED COMMANDS:

ECHO *Echo Incoming Data on Communications Port 0 (see page 403)*

ECHO_OFF *Turn Off Echo on Communications Port 0 (see page 407)*

PRINT(...) *Print Data to Communications Port (see page 669)*

SILENT *Silence Outgoing Communications on Communications Port 0 (see page 734)*

SILENT1 *Silence Outgoing Communications on Communications Port 1 (see page 736)*

STDOUT=formula *Set Device Output (see page 764)*

TALK1 *Talk on Communications Port 1 (see page 773)*

TALK1

Talk on Communications Port 1

APPLICATION:	Communications control
DESCRIPTION:	Motor restored to print on channel 0
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	TALK1 state
FIRMWARE VERSION:	5.0.x, 5.16.x or 5.32.x series (D); 6.4.2.x (D)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

NOTE: This command is typically sent from a host rather than existing within a SmartMotor program.

The TALK1 command restores the motor's ability to print messages to the serial communication channel 1 if that ability was previously suppressed with the SILENT1 command. This command is typically used after downloading a user program to a SmartMotor™ within a daisy chain. It could also be used to "un-silence" a debug routine.

TALK1 may be issued from the terminal or within a user program. However, the command is typically sent from a host.

EXAMPLE: (Shows use of SILENT1 and TALK1)

```

RUN?          'Wait here for the RUN command.
              'Set a=1 in the Terminal window to allow print statements.
OCHN (RS4,1,N,9600,1,8,C) 'Open ports 4 and 5 as RS-485 channel 1.
WHILE 1      'Endless loop.
IF a==1 TALK1 ENDIF 'If variable a is set to 1, allow
                  'PRINT statements on channel 1.
IF a==0 SILENT1 ENDIF 'If variable a is set to 0, suppress
                    'PRINT statements on channel 1.
PRINT1 ("Position=",PA,#13) 'Print the actual position.
WAIT=3000 'Wait 3 seconds.
LOOP      'Loop back to WHILE 1 command.
END

```

RELATED COMMANDS:

ECHO1 *Echo Incoming Data on Communications Port 1 (see page 406)*

ECHO_OFF1 *Turn Off Echo on Communications Port 1 (see page 409)*

PRINT(...) *Print Data to Communications Port (see page 669)*

SILENT *Silence Outgoing Communications on Communications Port 0 (see page 734)*

SILENT1 *Silence Outgoing Communications on Communications Port 1 (see page 736)*

STDOUT=formula *Set Device Output (see page 764)*

TALK *Talk on Communications Port 0 (see page 771)*

APPLICATION:	Math function
DESCRIPTION:	Gets the tangent of the input value
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RTAN(value)
WRITE:	N/A
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	Degrees input
RANGE OF VALUES:	Input in degrees (floating-point): -90.0 to 90.0 (a larger value can be used, but it is not recommended) Output (floating-point): TAN theoretically approaches \pm infinity at \pm 90 degrees
TYPICAL VALUES:	Input in degrees (floating-point): -90.0 to 90.0 (a larger value can be used, but it is not recommended) Output (floating-point): TAN theoretically approaches \pm infinity at \pm 90 degrees
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The TAN command takes an input angle in degrees and returns a floating-point tangent:

$$af[1]=TAN(arg)$$

where *arg* is in degrees, and may be an integer (i.e., *a*, *aw*[0]) or floating-point variable (i.e., *af*[0]). Integer or floating-point constants may also be used (i.e., 23 or 23.7, respectively).

This command cannot have within the parenthesis: math operators, other parenthetical functions, or a Combitronic request from another motor. For example, *x*=FABS(PA) is allowed, but *x*=FABS(PA:3) is not allowed.

The result of this function is a floating-point type. If used in an equation, the operations in the equation that are processed after this function are automatically promoted to a float. This is dependent on the mathematical order of operations in the equation. As with other equations (e.g., *x*=*a*+*b*), the variable to the left of "=" may be an integer variable to accept the result. However, the value will be truncated to fit to that integer type. For example, the assignment "*aw*[0]=" will drop any fractional amount and truncate the result to the range -32768 to 32767 (*aw*[0]=100.5 will report as 100, and *aw*[0]=40000.0 will report as -25536).

Although the floating-point variables and their standard binary operations conform to IEEE-754 double precision, the floating-point square root and trigonometric functions only produce IEEE-754 single-precision results. For more details, see Variables and Math on page 198.

EXAMPLE:

```
af[0]=TAN(45.7)      'Set array variable = TAN(45.7)
Raf[0]              'Report value of af[0] variable
RTAN(45.7)          'Report TAN(45.7)
af[1]=78.3          '78.3 degrees
af[0]=TAN(af[1])    'Variables may be put in the parenthesis
Raf[0]
END
```

Program output is:

```
1.024738192
1.024738192
4.828816413
```

RELATED COMMANDS:

```
R ACOS(value) Arccosine (see page 259)
R ASIN(value) Arcsine (see page 284)
R ATAN(value) Arctangent (see page 289)
R COS(value) Cosine (see page 372)
R SIN(value) Sine (see page 738)
```




TEMP, TEMP(arg)

Temperature, Motor

APPLICATION:	System
DESCRIPTION:	Read motor temperature
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RTEMP, RTEMP(arg) ; supports the DS2020 Combitronic system
WRITE:	Read only
LANGUAGE ACCESS:	N/A
UNITS:	Degrees Celsius
RANGE OF VALUES:	-40 to 100 (approximately) Report for DS2020 Combitronic system: 0 to 170
TYPICAL VALUES:	20 to 60
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	RTEMP(0):3, t=TEMP(0):3, af[0]=TEMP(0):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The TEMP command reads the motor temperature measured on the main drive board close to the motor. The current temperature of the motor can be determined by assigning TEMP to a user variable. The units are in degrees Celsius.

Some motors are equipped with thermistors inside the motor windings. These additional sensors can be read using RTEMP(1), RTEMP(2) and RTEMP(3)

Report Command	Assignment to Integer	Assignment to Float ^a	Meaning
RTEMP, or RTEMP(0)	t=TEMP or t=TEMP(0)	af[0]=TEMP or af[0]=TEMP(0)	Sensor on drive board
RTEMP(1)	t=TEMP(1)	af[0]=TEMP(1)	Sensor in winding
RTEMP(2)	t=TEMP(2)	af[0]=TEMP(2)	Sensor in winding
RTEMP(3)	t=TEMP(3)	af[0]=TEMP(3)	Sensor in winding

a) When the value is assigned to a floating-point variable, resolution improves to 0.1 degrees C.

EXAMPLE:

```
t=TEMP
Rt          'response 30
PRINT (TEMP) 'response 31 - the motor is warming up
```

You can set the overheat temperature trip point with the command:

TH=formula

NOTE: A motor in the overheat condition will not turn on the servo even if commanded to do so.

If the motor were operating in Torque mode at $TEMP > TH$, the motor would shut off. It would not restart until both the condition $TH - TEMP > 5$ was true *and* the ZS command (or Zh) was reissued.

```
a=-5
WHILE a<=10
  TH=TEMP+a
  WAIT=4000
  G
  WAIT=4000
  IF Bt
    BREAK
  ENDIF
  a=a+1
LOOP
PRINT ("MOTOR RESTARTED WHEN TH-TEMP=", a)
END
```

Program output is:

Restart announced at TH - TEMP = 6.

RELATED COMMANDS:

R Bh *Bit, Overheat (see page 307)*

R TH=formula *Temperature, High Limit (see page 779)*

Zh *Reset Temperature Fault (see page 852)*



TH=formula

Temperature, High Limit

APPLICATION:	System; supports the DS2020 Combitronic system
DESCRIPTION:	Set maximum allowable temperature (high limit)
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RTH
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	Degrees Celsius
RANGE OF VALUES:	0 to 85 DS2020 Combitronic system: 0 to 150
TYPICAL VALUES:	40 to 85
DEFAULT VALUE:	85
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	TH:3=60, a=TH:3, RTH:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

TH=formula sets the maximum allowable temperature (high limit) at which the SmartMotor is permitted to continually servo. If the temperature goes above the TH value, the amplifier will turn off, Bh will be set to 1, the motor off bit (Bo) will be set to 1, and the trajectory bit will be cleared to 0.

NOTE: The SmartMotor will reject any command to clear the Bh fault or start motion until the temperature has fallen by 5 degrees Celsius.

EXAMPLE: (Demonstrates relationship between TEMP, TH, and Bh)

```

GOSUB10          'Report TEMP, TH, and Bh
a=5
b=TEMP
WHILE a>-5      'Vary TH about the current TEMP
    TH=b-a
    WAIT=2000
    GOSUB10     'Observe Bh flag change from 0 to 1
    a=a-1       'as TH is reduced to TEMP value and less
LOOP
END
C10
    PRINT(#13,"Read the temperature ",b)
    PRINT(#13,"Read TH overheat value ",TH)
    PRINT(#13,"Read Bh overheat flag ",Bh)
RETURN

```

Program output is:

```

Read the temperature 0
Read TH overheat value 85
Read Bh overheat flag 0
Read the temperature 29
Read TH overheat value 24
Read Bh overheat flag 1
Read the temperature 29
Read TH overheat value 25
Read Bh overheat flag 1
Read the temperature 29
Read TH overheat value 26
Read Bh overheat flag 1
Read the temperature 29
Read TH overheat value 27
Read Bh overheat flag 1
Read the temperature 29
Read TH overheat value 28
Read Bh overheat flag 1
Read the temperature 29
Read TH overheat value 29
Read Bh overheat flag 1
Read the temperature 29
Read TH overheat value 30
Read Bh overheat flag 1
Read the temperature 29
Read TH overheat value 31
Read Bh overheat flag 1
Read the temperature 29
Read TH overheat value 32
Read Bh overheat flag 1
Read the temperature 29
Read TH overheat value 33
Read Bh overheat flag 1

```

RELATED COMMANDS:

R Bh *Bit, Overheat (see page 307)*

R TEMP, TEMP(arg) *Temperature, Motor (see page 777)*



APPLICATION:	Program execution and flow control
DESCRIPTION:	Gets (reads) or sets one of the timers
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	four unique timers available
READ/REPORT:	RTMR(timer)
WRITE:	Read/write
LANGUAGE ACCESS:	N/A
UNITS:	Milliseconds
RANGE OF VALUES:	0 to 2147483647 milliseconds (negative values not recommended)
TYPICAL VALUES:	0 to 2147483647 milliseconds (negative values not recommended)
DEFAULT VALUE:	0 milliseconds
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	TMR(0,1000):3, a=TMR(0):3, RTMR(0):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The TMR command gets (reads) or sets the timer:

- =TMR(timer)
Get a specific timer value, e.g., a=TMR(0) gets the time value from timer #0.
- TMR(timer,time) as a command
Set a timer ID# to a specific time value, e.g., TMR(0,1000) sets timer #0 to 1000 milliseconds.

The range of timer is from 0 to 3, and 8. (Timer 8 is a more recent feature and may not be available in all firmware at this time.)

- Timers 0-3 are single-shot timers which run down to 0 each time the TMR command is called and TMR must be called again for the next event to occur.
- Timer 8 allows for a timer that auto-reloads for interrupt generation on a consistent time-base. TMR needs only to be called one time for endless timer event generation.

The range of time is from 0 to 2147483647. A negative number can be set; however, it is not recommended.

The TMR command allows a count-down timer to be enabled. This is useful for triggering interrupt routines. When a timer is running, the corresponding status bit in Status Word 4 will be set to the value 1. When it reaches zero, the status bit will revert to 0. This bit change can be used to trigger a subroutine through the ITR() function. For more details on ITR(), see ITR (Int#,StatusWord,Bit#,BitState,Label#) on page 517.

EXAMPLE:

```

EIGN (2)           'Disable Left Limit
EIGN (3)           'Disable Right Limit
ZS                'Clear faults
MP                'Set Position Mode
VT=500000         'Set Target Velocity.
AT=300            'Set Target Acceleration.
DT=100            'Set Target Deceleration.
TMR(0,1000)       'Set Timer 0 to 1s
ITR(0,4,0,0,20)  'Set Interrupt
EITR(0)           'Enable Interrupt
ITRE              'Enable all Interrupts
p=0               'Initialize variable p
O=0               'Set commanded and actual pos. to zero
C10               'Place a label
IF PA>47000       'Just before 12 moves
    DITR(0)        'Disable Interrupt
    TWAIT          'Wait till reaches 48000
    p=0            'Reset variable p
    PT=p           'Set Target Position
    G              'Start motion
    TWAIT         'Wait for move to complete
    EITR(0)        'Re-enable Interrupt
    TMR(0,1000)    'Re-start timer
ENDIF GOTO10      'Go back to label
END               'End (never reached)
C20               'Interrupt Subroutine Label
    TMR(0,1000)    'Re-start timer
    p=p+4000       'Increment variable p
    PT=p           'Set Target Position
    G              'Start Motion
RETURNI           'Return to main loop

```

EXAMPLE: (for pulse width)

```

. . .
WHILE 1>0
    O=0            'Reset origin for move
    PT=40000      'Set final position
    G              'Start motion
    WHILE PA<20000 'Loop while motion continues
    LOOP          'Wait for desired position to pass
    OUT(1)=0      'Set output lo
    TMR(0,400)    'Use timer 0 for pulse width
    TWAIT WAIT=1000 'wait 1 second
LOOP
. . .

```

RELATED COMMANDS:

ITR(Int#,StatusWord,Bit#,BitState,Label#) *Interrupt Setup (see page 517)*



APPLICATION:	Motion control
DESCRIPTION:	Gets (reads) the real-time torque of the motor
EXECUTION:	Next PID sample
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RTRQ
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	-32767 to 32767
TYPICAL VALUES:	-32767 to 32767
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	RTRQ:3, x=TRQ:3 where ":3" is the motor address — use the actual address or a variable

NOTE: Combitronic is supported in version 5.x.4.31 and later.

DETAILED DESCRIPTION:

The TRQ command is used to get (read) the real-time torque demand of the PID or MT mode of the motor. In MT mode, the value reported will reflect any applied TS ramp.

NOTE: The value returned by TRQ (and RTRQ) will typically be one less than the T (torque) value due to internal calculations. It may also be reduced in cases where the motor's output is in limitation. TRQ represents the output effort of the motor in both MT (torque mode) and servo modes (MV, MP, etc.). Therefore, it provides a seamless transfer across those modes without causing a ripple or bump in force to the load.

In other modes where the servo is enabled, the value of TRQ reports the demand of the PID loop.

EXAMPLE:

At the SMI terminal prompt, type these commands:

```
MT
T=3000
G
```

NOTE: In Torque mode, the new torque value does not take effect until a G command is issued.

Now use the TRQ command to read the real-time torque:

```
PRINT (TRQ)
```

Program output is:

```
2999
```

RELATED COMMANDS:

MT *Mode Torque (see page 620)*

R T=formula *Torque, Open-Loop Commanded (see page 769)*

R TS=formula *Torque Slope (see page 786)*



APPLICATION:	Motion control
DESCRIPTION:	Gets (reads) or sets the torque slope
EXECUTION:	Buffered until a G command is issued
CONDITIONAL TO:	MT mode
LIMITATIONS:	N/A
READ/REPORT:	RTS
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	((Units of T=) per sample)*65536
RANGE OF VALUES:	-1 to 2147483647
TYPICAL VALUES:	-1 (disable), or from 65536 to 1000000
DEFAULT VALUE:	-1 (disabled)
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	TS:3=1234, a=TS:3, RTS:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The TS command is used to get (read) or set the torque slope:

- =TS
Get torque slope setting
- TS=formula
Set torque slope

In Torque mode (MT), the TS= command allows new torque settings to be reached gradually rather than instantly. Values may be from -1 to +2147483647. A value of -1 disables the slope feature and causes new torque values to be reached immediately. A TS setting of 65536 will increase the output torque by one unit per PID sample period.

EXAMPLE:

```

MT           'Select torque mode.
T=8000      'Final torque after the TS ramp that we want.
TS=65536    'Increase the torque by 1 unit of T per PID sample.
G           'Begin move

```

NOTE: In Torque mode, the new torque value does not take effect until a G command is issued.

After executing the above code, in the SMI software Terminal window, use the PRINT(TS) command to get the current torque slope value:

```
PRINT (TS)  
65536
```

RELATED COMMANDS:

MT *Mode Torque (see page 620)*

R T=formula *Torque, Open-Loop Commanded (see page 769)*

TSWAIT

Trajectory Synchronized Wait

APPLICATION:	Program execution and flow control
DESCRIPTION:	Suspends program execution during a synchronized move
EXECUTION:	Immediate
CONDITIONAL TO:	PTS or PRTS commands
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M), not ver. 5.32.x.x; 6.x (D/M), M requires EIP option
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The TSWAIT command pauses synchronized program execution. After a GS command has been issued to start a synchronized move, the TSWAIT command can be used to pause program execution until the move has been completed.

Note that a standard TWAIT command would not work in cases where the motor issuing the PTS() and GS commands had a zero-length contribution to the total move. The TSWAIT command was designed to handle this situation. For more details, see Synchronized Motion on page 179.

EXAMPLE:

The next example is a synchronized move in its simplest form. The code could be downloaded to either motor 1 or 2, and it would work the same.

```
ADTS=100           'Set target synchronized accel/decel
VTS=10000          'Set target synchronized velocity
PTS(3000;1,4000;2) 'Set synchronized target positions
GS                'Start synchronized motion
TSWAIT            'Wait for synchronized motion to complete
```

RELATED COMMANDS:

PRTS(...) *Position, Relative Target, Synchronized (see page 685)*

PTS(...) *Position Target, Synchronized (see page 692)*

TWAIT(gen#) *Trajectory Wait (see page 789)*

WAIT=formula *Wait for Specified Time (see page 835)*

TWAIT(gen#)
Trajectory Wait

APPLICATION:	Program execution and flow control
DESCRIPTION:	Suspend command execution while in trajectory
EXECUTION:	Immediate
CONDITIONAL TO:	Position mode (MP) or torque ramp (MT, TS)
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The **TWAIT** command pauses program execution until the Busy Trajectory (Bt) status bit clears. Normally, program execution and trajectory generation are completely independent. Regardless of what the motion is doing, the processor executes code from the top down.

For example, if there were three consecutive motion commands, they would all execute sequentially. Therefore, before the motor could even start to move, the last motion command would dominate. However, using the **TWAIT** command allows each move command to occur and complete.

An alternative to **TWAIT** is:

```
WHILE Bt . . . LOOP
```

The **TWAIT** command and **WHILE Bt** construction terminate when the trajectory ends. Depending on the application, you may wish to perform error checking to ensure that the move was properly completed within a position-error range.

When in MT mode, the **TWAIT** command (and Bt bit) will also wait while a TS ramp is in progress.

The **TWAIT** command will wait for all trajectories to complete. Be aware that dual-trajectory operation may not give the expected result. To access the specific trajectory (1 or 2), use the command form **TWAIT(1)** or **TWAIT(2)**. Also, note there are associated status bits in status word 7 (bits 0 and 8).

EXAMPLE: (Subroutine shows use of DITR, EITR, TMR and TWAIT)

```
C10          'Place a label
  IF PA>47000 'Just before 12 moves
    DITR(0)   'Disable interrupt
    TWAIT     'Wait till reaches 48000
    p=0       'Reset variable p
    PT=p      'Set target position
    G         'Start motion
    TWAIT     'Wait for move to complete
    EITR(0)   'Re-enable interrupt
    TMR(0,1000) 'Restart timer
  ENDIF
GOTO10      'Go back to label
```

RELATED COMMANDS:

^R Bt *Bit, Trajectory In Progress (see page 345)*

TSWAIT *Trajectory Synchronized Wait (see page 788)*

WAIT=formula *Wait for Specified Time (see page 835)*



APPLICATION:	System
DESCRIPTION:	Gets (reads) the motor current applied to the windings
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RUIA ; supports the DS2020 Combitronic system
WRITE:	Read only
LANGUAGE ACCESS:	N/A
UNITS:	milliamperes (mA)
RANGE OF VALUES:	0 to 2147483647 Report for DS2020 Combitronic system: -2147483648 to 2147483647
TYPICAL VALUES:	0 to 20000
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	RUIA:3; x=UIA:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The UIA command gets (reads) the motor current applied to the windings. The value is returned in milliamperes. Therefore, divide by 1000 to convert it to amperes.

The value returned is a measure of current in the motor for thermal limiting. It should not be assumed it is directly related to torque. This current is not a measurement of current from the supply lines; it is measured in the drive bridge. Because the drive is a power-conversion device, the current from the supply is not the same amount of current supplied to the motor windings.

EXAMPLE:

```
i=400           'Motor current to check for
WHILE 1         'While forever
  IF UIA>i      'If motor current in mAmps is > "i"
    GOSUB(100)
    WHILE UIA>i LOOP  'Prevent double trigger
  ENDIF
LOOP
C100
  IF UIA>(i*2)  'If current is twice as much
    GOTO200    'bypass PRINT line below
  ENDIF
  PRINT("Current is above ",i,"mAmps",#13)
C200
  PRINT("Current twice as high as it should be!",#13)
RETURN
```

RELATED COMMANDS:

R Bh *Bit, Overheat (see page 307)*

R TEMP, TEMP(arg) *Temperature, Motor (see page 777)*

R TH=formula *Temperature, High Limit (see page 779)*

R UJA *Bus Voltage (see page 793)*



APPLICATION:	System
DESCRIPTION:	Gets (reads) the bus voltage applied to the motor's drive bridge
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	Class 5 hardware can measure up to 100 volts; the voltage supply should not exceed 48 volts
READ/REPORT:	RUJA ; supports the DS2020 Combitronic system
WRITE:	Read only
LANGUAGE ACCESS:	N/A
UNITS:	millivolts (mV)
RANGE OF VALUES:	0 to 2147483647 Report for DS2020 Combitronic system: 0 to 4294967295
TYPICAL VALUES:	0 to 48000
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	RUJA:3; x=UJA:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The UJA command gets (reads) the bus voltage. The value returned is in millivolts. Therefore, divide the value by 1000 to convert it to volts. For example, 24000 equals 24 volts.

The voltage is measured at the motor's drive bridge; it is not the "control" power supplied for the CPU and electronics. If a DE motor is used, then this is an important difference. If a non-DE motor is used, then both voltage supplies are effectively the same.

EXAMPLE:

```

VT=100000          'Set maximum velocity
PT=1000000        'Set final position
MP                'Set Position Mode
G                 'Start motion
WHILE Bt          'Loop while motion continues
    IF UJA<18500  'If voltage is below 18.5 Volts
        OFF      'Turn motor off
    ENDIF
LOOP              'Loop back to WHILE
END               'Required END

```

RELATED COMMANDS:

R Bh *Bit, Overheat (see page 307)*

R TEMP, TEMP(arg) *Temperature, Motor (see page 777)*

R TH=formula *Temperature, High Limit (see page 779)*

R UIA *Motor Current (see page 791)*



APPLICATION:	System
DESCRIPTION:	Sets one or more user status bits to specified values
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	None
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	Depends on command format and motor model (see details)
TYPICAL VALUES:	Depends on command format and motor model (see details)
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	UO(0):3=1, UO(W,0):3=22 or UO(W,0,15):3=22, where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The UO command sets one or more user status bits to the specified values. These bits are visible in motor status words 12 and 13, which means they can also be used to trigger user program interrupts.

The UO command sets or clears the bit specified by bit#. If the formula's least-significant bit = 1, then it's true (1); otherwise, it's false (0).

- UO(bit#)=formula
If bit 0 in the formula to the right of "=" is 1, then set bit# to a 1; otherwise, when it is even or zero, clear the bit to 0.
- UO(W,word)=formula
Set the group of bits in the specified user word to the bitwise value from the formula.
- UO(W,word[,mask])=formula
Set the group of bits in the specified user word to the bitwise value from the formula. However, leave the bits as is if they are bitwise set to 0 in the bitmask value.

User Word	Associated Status Word	User Bits (individually addressed)
0, e.g., UO(W,0)=x	12, e.g., RW(12) or RB(12,x)	0-15, e.g., UO(15)=0
1, e.g., UO(W,1)=x	13, e.g., RW(13) or RB(13,x)	16-31, e.g., UO(31)=0

User bits allow the programmer to keep track of events or status within an application program. Their functions are defined by the application program in the SmartMotor. User bits are individually addressed starting at 0 (zero based). Likewise, the user-bit words are addressed starting at 0 (zero based).

A powerful feature of user bits is their ability to be addressed over networks such as Combitronic or CANopen. This allows a hosting application to run an interrupt routine in the SmartMotor.

User bits can also be addressed as words, with or without a mask, to define the affected bits.

For more details, see User Status Bits on page 219.

EXAMPLE:

```
UO(0)=a&b      'Sets user bit to 1 if the bit-wise operation
                'result is odd, else sets it to 0.
```

```
UO(W,1,7)=a    'Sets user bits 16, 17 and 18 to the value of
                'the lower three bits in a.
```

RELATED COMMANDS:

$R\ B(\text{word},\text{bit})$ *Status Byte (see page 297)*

$ITR(\text{Int}\#, \text{StatusWord}, \text{Bit}\#, \text{BitState}, \text{Label}\#)$ *Interrupt Setup (see page 517)*

$UR(\dots)$ *User Bits, Reset (see page 801)*

$US(\dots)$ *User Bits, Set (see page 803)*

$R\ W(\text{word})$ *Report Specified Status Word (see page 833)*

Upload Compiled Program and Header

APPLICATION:	Program access
DESCRIPTION:	Upload user EEPROM program with header and raw address information through serial communications.
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The UP command causes a compiled program (runtime code) to be sent out through the serial port where it was requested. The output from the UP command includes a header containing binary information. It also contains special codes created by the compiler to handle program flow statements like GOTO or WHILE LOOP, which are interspersed with the program text. In contrast, the UPLOAD command returns the user program in readable text. For details, see UPLOAD on page 799.



CAUTION: The UP command is not permitted within a user program. Unexpected behavior may result.

Procedure for using this command:

1. Issue the UP command.
2. IF ECHO is enabled, UP and a hex 20 will be returned.
3. 8 bytes of program data will be returned.
4. issue the hex 06 character to request more data.
5. The hex 06 character will be ECHOed if ECHO is enabled.
6. 8 bytes of program data will be returned.
7. Repeat step 5. If less than 8 bytes (including 0 bytes) of program data is remaining, then the end of program was found. The program is ended with a hex character FF. If this character is seen, no further program data is uploaded.

The UP or UPLOAD command does not terminate the current motion mode or trajectory, change motion parameters such as EL, ADT, VT or KP, or alter the current value of the user variables.

The user program may continue running. However, it is recommended to END the program before uploading to avoid communications conflicts with any report or PRINT statements.

The comments in the original source code do not appear when you UP or UPLOAD a program. Comments are removed by the SMI software compiler, which is normal for any compiled computer program.

When uploading a program from a SmartMotor in a daisy chain, use the SILENT and SLEEP commands to prevent the other SmartMotors in the chain from issuing unexpected characters. After the upload is complete, re-enable normal communications to those motors with the WAKE and TALK commands.

RELATED COMMANDS:

SILENT *Silence Outgoing Communications on Communications Port 0 (see page 734)*

SLEEP *Ignore Incoming Commands on Communications Port 0 (see page 744)*

TALK *Talk on Communications Port 0 (see page 771)*

UPLOAD *Upload Standard User Program (see page 799)*

WAKE *Wake Communications Port 0 (see page 837)*

UPLOAD

Upload Standard User Program

APPLICATION:	Program access
DESCRIPTION:	Upload user EEPROM through serial communications
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The UPLOAD command uploads only the text portion of the SmartMotor's™ program as it is shown in the original source file. The program is uploaded to the serial port where it was requested. All comments and blank spaces are removed. In contrast, the UP command uploads the text along with all of the binary information created by the compiler. For details, see UP on page 797.



CAUTION: The UPLOAD command is not permitted within a user program. Unexpected behavior may result.

Procedure for using this command:

1. Issue the UPLOAD command.
2. IF ECHO is enabled, UPLOAD and a hex 20 will be returned.
3. 8 bytes of program data will be returned.
4. issue the hex 06 character to request more data.
5. The hex 06 character will be ECHOed if ECHO is enabled.
6. 8 bytes of program data will be returned.
7. Repeat step 5. If less than 8 bytes (including 0 bytes) of program data is remaining, then the end of program was found. The program is ended with a hex character FF. If this character is seen, no further program data is uploaded.

The UP or UPLOAD command does not terminate the current motion mode or trajectory, change motion parameters such as EL, ADT, VT or KP, or alter the current value of the user variables.

The user program may continue running. However, it is recommended to END the program before uploading to avoid communications conflicts with any report or PRINT statements.

The comments in the original source code do not appear when you UP or UPLOAD a program. Comments are removed by the SMI software compiler, which is normal for any compiled computer program.

When uploading a program from a SmartMotor in a daisy chain, use the SILENT and SLEEP commands to prevent the other SmartMotors in the chain from issuing unexpected characters. After the upload is complete, re-enable normal communications to those motors with the WAKE and TALK commands.

RELATED COMMANDS:

SLEEP *Ignore Incoming Commands on Communications Port 0 (see page 744)*

SILENT *Silence Outgoing Communications on Communications Port 0 (see page 734)*

TALK *Talk on Communications Port 0 (see page 771)*

UP *Upload Compiled Program and Header (see page 797)*

WAKE *Wake Communications Port 0 (see page 837)*



APPLICATION:	System
DESCRIPTION:	Sets one or more user status bits to 0
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	None
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	Depends on command format and motor model (see details)
TYPICAL VALUES:	Depends on command format and motor model (see details)
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	UR(0):3 or UR(W,0):3 or UR(W,0,15):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The UR command sets one or more user status bits to 0. These bits are visible in motor status words 12 and 13, which means they can also be used to trigger user program interrupts.

The UR command sets or clears the bit specified by bit#. If the *expression* least-significant bit = 1, then it's true (1); otherwise, it's false (0).

- UR(bit#)
Clear bit to a 0.
- UR(W,word)
Clear all bits in the specified word.
- UR(W,word[,mask])
Clear all bits in the specified word. However, leave bits as is if they are bitwise set to 0 in the bitmask value.

User Word	Associated Status Word	User Bits (individually addressed)
0, e.g., UR(W,0)	12, e.g., RW(12) or RB(12,x)	0-15, e.g., UR(15)
1, e.g., UR(W,1)	13, e.g., RW(13) or RB(13,x)	16-31, e.g., UR(31)

User bits allow the programmer to keep track of events or status within an application program. Their functions are defined by the application program in the SmartMotor. User bits are individually addressed starting at 0 (zero based). Likewise, the user-bit words are addressed starting at 0 (zero based).

A powerful feature of user bits is their ability to be addressed over networks such as Combitronic or CANopen. This allows a hosting application to run an interrupt routine in the SmartMotor.

User bits can also be addressed as words, with or without a mask, to define the affected bits.

For more details, see User Status Bits on page 219.

EXAMPLE:

```
UR(19)      'RESET User Bit 3 in second User Bit Status Word
```

```
UR(W,0)    'RESET all User Bits in first User Status Word
```

```
UR(W,1,7)  'RESET User bits 16, 17 and 18
```

RELATED COMMANDS:

^R B(word,bit) *Status Byte (see page 297)*

ITR(Int#,StatusWord,Bit#,BitState,Label#) *Interrupt Setup (see page 517)*

UO(...)=formula *User Status Bits (see page 795)*

US(...) *User Bits, Set (see page 803)*

^R W(word) *Report Specified Status Word (see page 833)*



APPLICATION:	System
DESCRIPTION:	Sets one or more user status bits to 1
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	None
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	Depends on command format and motor model (see details)
TYPICAL VALUES:	Depends on command format and motor model (see details)
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	US(0):3 or US(W,0):3 or US(W,0,15):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The US command sets one or more user status bits to 1. These bits are visible in motor status words 12 and 13, which means they can also be used to trigger user program interrupts.

The US command sets or clears the bit specified by bit#. If the *expression* least-significant bit = 1, then it's true (1); otherwise, it's false (0).

- US(bit#)
Set bit to a 1.
- US(W,word)
Set all bits in the specified word.
- US(W,word[,mask])
Set all bits in the specified word. However, leave bits as is if they are bitwise set to 0 in the bitmask value.

User Word	Associated Status Word	User Bits (individually addressed)
0, e.g., US(W,0)	12, e.g., RW(12) or RB(12,x)	0-15, e.g., US(15)
1, e.g., US(W,1)	13, e.g., RW(13) or RB(13,x)	16-31, e.g., US(31)

User bits allow the programmer to keep track of events or status within an application program. Their functions are defined by the application program in the SmartMotor. User bits are individually addressed starting at 0 (zero based). Likewise, the user-bit words are addressed starting at 0 (zero based).

A powerful feature of user bits is their ability to be addressed over networks such as Combitronic or CANopen. This allows a hosting application to run an interrupt routine in the SmartMotor.

User bits can also be addressed as words, with or without a mask, to define the affected bits.

For more details, see User Status Bits on page 219.

EXAMPLE:

```
US (0)      'SET User Bit 0
```

```
US (W, 0, a) 'SET first three User Bits when a=7
```

EXAMPLE: (Fault-handler subroutine, shows use of MTB and US)

```
C0          'Fault handler
MTB:0      'Motor will turn off with Dynamic
           'braking, tell other motors to stop.
US (0) :0   'Set User Status Bit 0 to 1 (Status
           'Word 12 bit zero)
US (ADDR) :0 'Set User Status Bit "address" to 1
           '(Status Word 12 Bit "address")
```

RETURNI

RELATED COMMANDS:

R B(word,bit) Status Byte (see page 297)

ITR(Int#,StatusWord,Bit#,BitState,Label#) Interrupt Setup (see page 517)

UO(...)=formula User Status Bits (see page 795)

UR(...) User Bits, Reset (see page 801)

R W(word) Report Specified Status Word (see page 833)

USB(arg)

USB Status Word

APPLICATION:	Communications control
DESCRIPTION:	Report the value of USB status word or assign it to a variable (see details)
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	No USB= form of this command
READ/REPORT:	RUSB
WRITE:	Read only
LANGUAGE ACCESS:	N/A
UNITS:	See description
RANGE OF VALUES:	See description
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	6.x (D/M); no Class 5
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The USB command is used to report the value of the specified USB status word or assign it to a variable:

- `x=USB(arg)`
Assigns the value of the specified USB status word (specified by `arg`) to the variable `x`
- `RUSB(arg)`
Reports the value of the specified USB status word (specified by `arg`)

RUSB and `x=USB` are also permitted. In these cases, the value of `arg` is 0. Therefore, status word 0 is reported or assigned to a variable.

Arg	Requested Information	Result Value	
		Range	Meaning
0	Report connection state	0-32	0: Detached state 1: Attached state 2: Powered state 4: Default state 8: Address-pending state 16: Address state 32: Configured state

EXAMPLE:

x=**USB** (0) 'Read the status of USB connection.

RELATED COMMANDS:

N/A



APPLICATION:	Motion control
DESCRIPTION:	Gets (reads) the actual (filtered) velocity
EXECUTION:	Next PID sample
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT VALUE:	RVA ; supports the DS2020 Combitronic system
WRITE:	Read only
LANGUAGE ACCESS:	N/A
UNITS:	Scaled encoder counts/sample Report for DS2020 Combitronic system: user increments / sec, see FD=expression on page 461
RANGE OF VALUES:	-2147483648 to 2147483647
TYPICAL VALUES:	-3200000 to 3200000
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	RVA:3, x=VA:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

NOTE: This command is affected by the SCALEV command. For details, see SCALEV(m,d) on page 728. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

The VA command gets (reads) the commanded velocity:

- =VA
Reads the actual velocity measured by the selected encoder. When ENC1 mode is chosen, the external encoder is used.

The SmartMotor can be given a variety of motion commands. Under all types of motion, the actual velocity (rotational) can be reported. This also applies to a back-driven motor when the drive is off.

The value reported is measured from the encoder, so it is based on real-world data rather than an internal calculation. In order to filter the noise from this data, an infinite impulse response (IIR) digital filter is applied. An IIR filter is similar to a moving average but includes all previous data points. The significance of the previous points is diminished while new data points are added. The coefficient of this filter can be adjusted with the VAC command for a longer or shorter time constant. There is a trade-off between quick responsiveness and resolution. The default value for the filter constant should work well for most applications.

Equations for Real-World Units:

Because the encoder resolution and sample rate can vary, the general equations shown in the next table can be applied to converting the value of VA to various units of velocity. These equations force floating-point calculations to avoid overflow and maintain resolution. They can be placed in a user program, or they can be precalculated if the values of SAMP and RES are known (SAMP and RES can be reported from the terminal using the RSAMP and RRES commands, respectively). SAMP can change if the PID command is used. The value of RES can differ between motor models.

Output	Equation
Radians/Sec	$=VA * \pi^2 * ((SAMP / 65536.0) / RES)$
Encoder Counts/Sec	$=VA * (SAMP / 65536.0)$
Rev/Sec	$=VA * ((SAMP / 65536.0) / RES)$
RPM	$=VA * 60.0 * ((SAMP / 65536.0) / RES)$

A useful Scale Factor Multiplier code example, which also illustrates the use of af[], SAMP and RES, is shown in RES on page 702 and SAMP on page 722.

EXAMPLE: (monitor acceleration ramp)

```
O=0                'Set up a velocity move
EL=4000
ADT=10
v=1000000
VT=v
MV
G
WHILE VA<v        'Monitor velocity while
  IF Be           'Accelerating
    BREAK        'Exit if position error
  ENDIF
  GOSUB5         'Report trajectory velocity
LOOP
GOSUB5           'Final report
END

C5
PRINT("PRINT VC = ")
PRINT(VC,#13)    'Get/print commanded velocity
PRINT("RVC = ")
RVC              'Report commanded velocity
WAIT=4000
RETURN
```

Program output is:

```
RUN
PRINT VC = 565
RVC = 1395
PRINT VC = 322065
RVC = 323155
PRINT VC = 643845
RVC = 644915
PRINT VC = 965605
```


RVC = 966675
PRINT VC = 1000000
RVC = 1000000

RELATED COMMANDS:

ENC0 *Encoder Zero (Close Loop on Internal Encoder) (see page 432)*
ENC1 *Encoder Zero (Close Loop on External Encoder) (see page 433)*
R FD=*expression Resolution to Set Units of Position/Velocity/Acceleration (see page 461)*
R RES *Resolution (see page 702)*
R SAMP *Sampling Rate (see page 722)*
VAC(*arg*) *Velocity Actual (filter) Control (see page 810)*
R VC *Velocity Commanded (see page 815)*
R VL=*formula Velocity Limit (see page 818)*
R VT=*formula Velocity Target (see page 828)*

VAC(arg) Velocity Actual (filter) Control

APPLICATION:	Motion control
DESCRIPTION:	Sets the velocity filter
EXECUTION:	Next PID sample
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT VALUE:	None
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	Scaled encoder counts/sample
RANGE OF VALUES:	0 to 65535
TYPICAL VALUES:	0 to 65535
DEFAULT VALUE:	65000
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The VAC(arg) command sets the velocity filter used by the VA command, where arg is the value used to set the time constant of the filter. This command controls the filter used to measure the speed reported from the VA command (RVA, x=VA). A value of VAC(0) turns off this filtering so that VA reports raw data that may be highly quantized (e.g., 0, 65536, 131072). Many applications require finer resolution. Therefore, many samples are averaged in a digital filter.

The maximum value of VAC is 65535. However, the default value of 65000 should work for most applications. Note that higher values provide a smoother filter at the cost of a longer settling time. Therefore, use the default value unless a specific problem with the VA reading requires tuning it.

Refer to the next table for PID sample rate of 8000 Hz (set with the PID2 command).

Value of VAC	Time Constant (PID samples)	Time Constant (msec)*	Cutoff Frequency -3 dB (Hz)*
0	Filter disabled (raw values immediately update VA)		
50000	3.70	0.462	345
60000	11.3	1.42	112
65000	122	15.2	10.5
65400	481	60.2	2.65
65450	762	95.2	1.67
65500	1820	227	0.700

Value of VAC	Time Constant (PID samples)	Time Constant (msec)*	Cutoff Frequency -3 dB (Hz)*
65510	2520	315	0.505
65520	4095	512	0.311
65525	5957	745	0.214
65530	10922	1365	0.117
65535	65536	8192	0.019
*Affected by PID rate			

Notes for previous table:

1. Time Constant is the time required for a step response to reach 63.2% of the final settled value.
2. Cutoff Frequency is the frequency that is attenuated to -3 dB (70.7% the original value). Higher frequencies will be further reduced; lower frequencies will remain stronger than 70.7%.
3. Time Constant (msec) and the Cutoff Frequency (Hz) are affected by the PID rate setting. The table assumes 8000 Hz for the Class 5 motor.
4. Time Constant (PID samples) is not affected by the PID rate setting.

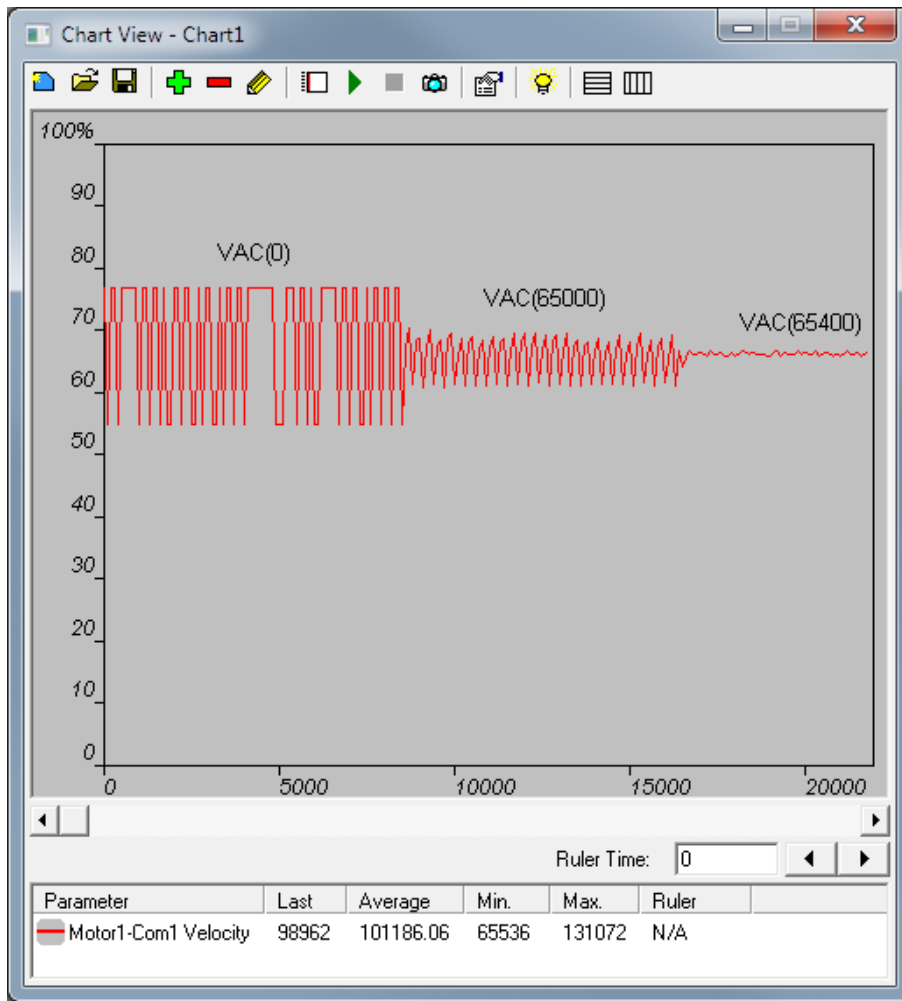
The "Time Constant (PID samples)" should not be construed as the number of averaged samples — it is the number of samples required for an abrupt change (step function) to reach 70.7% of the final value. For instance, refer to the second figure where a step function is approximated by setting a large value for acceleration. The fourth bump has a time constant of 10922 samples. However, it is the same physical move as the others. This means that it takes over one second for the reading to catch up to 70.7% of the actual speed of the motor.

Further, consider this example: the motor used for the next figures has actual cogging; the default setting of VAC(65000) depicts the actual speed fluctuations.

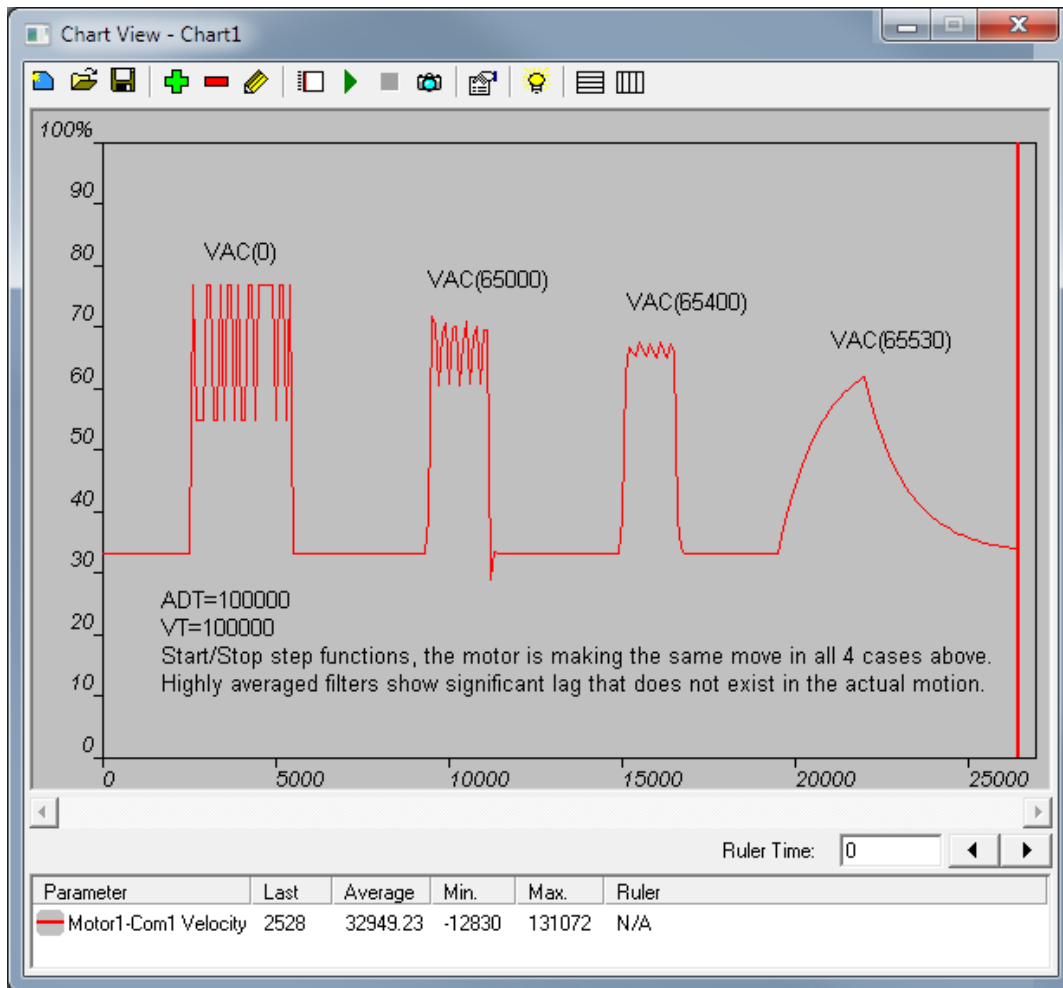
- VAC(0) is dominated by quantization effects, which make the readings jump around by exactly 65536.
- VAC(65400) gives a highly averaged reading in this case. This can be useful in cases where you need a stable reading and are willing to wait long enough for the speed to settle. Referring to the previous table, this rolls off frequencies above 2.65 Hz. A good rule of thumb is to wait five time-constants for a value to settle. In this case, the time constant is 60 msec. Therefore, you would wait for 300 msec ($60 \times 5 = 300$) to allow the value to settle to a valid reading.

To do this in a program:

1. Issue a G command.
2. Wait for the slew status bit to indicate the slew portion of a move.
3. Wait 300 msec for the speed to settle (to obtain a valid reading)
4. Read VA (with RVA or x=VA).



Examples of RVA readings with different settings of VAC()



Dramatic Example of Filter Causing Lag in Readings

EXAMPLE:

```
VAC (65000) 'Set VAC to default value.
VAC (65400) 'Set VAC to higher value to smooth out VA readings,
            'resulting in slower update time.
```

EXAMPLE:

For this example, try changing the VAC() value while polling VA.

```
EIGN(W,0)    'Make all onboard I/O inputs
ZS          'Clear errors.
MV          'Set the SmartMotor to Velocity Mode
VAC(65400)  'Set time constant to 60 milliseconds
ADT=10      'Set a value for accel/decel
VT=200000   'Set a value for velocity target
G           'Start motion.
WHILE B(3,15)==0 LOOP 'Wait for velocity target reached bit to be 1
WAIT=300    'Wait 300 milliseconds
PRINT("Velocity=",VA,#13) 'Print the actual velocity
END
```

RELATED COMMANDS:

R VA *Velocity Actual (see page 807)*
R VC *Velocity Commanded (see page 815)*
R VL=formula *Velocity Limit (see page 818)*
R VT=formula *Velocity Target (see page 828)*



APPLICATION:	Motion control
DESCRIPTION:	Gets (reads) the commanded velocity
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT VALUE:	RVC
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	(encoder counts / sample) * 65536
RANGE OF VALUES:	-2147483648 to 2147483647
TYPICAL VALUES:	-3200000 to 3200000
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	RVC:3, x=VC:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

NOTE: This command is affected by the SCALEV command. For details, see SCALEV(m,d) on page 728. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

When a velocity or position profile move is commanded, the velocity is ramped up and down according to the settings of ADT=, AT=, or DT=, with the value of VT as the maximum value. At any one instant, the calculated velocity of the motion profile can be reported. The sign of this reported velocity is dependent on the direction. A velocity in the negative direction will be reported negative by this command, while a positive velocity is reported as a positive value.

The VC command gets (reads) the commanded velocity:

- =VC
Reads the real-time commanded velocity combined from all trajectory generators.

NOTE: It is not the actual velocity (VA); it is the velocity calculated by the velocity profile at the time the VC command is executed.

Equations for Real-World Units:

Because the encoder resolution and sample rate can vary, the general equations shown in the next table can be applied to converting the value of VC to various units of velocity. These equations force floating-point calculations to avoid overflow and maintain resolution. They can be placed in a user program, or they can be precalculated if the values of SAMP and RES are known (SAMP and RES can be reported from the terminal using the RSAMP and RRES commands, respectively). SAMP can change if the PID command is used. The value of RES can differ between motor models.

Output	Equation
Radians/Sec	$=VC * \pi * 2 * ((SAMP/65536.0)/RES)$
Encoder Counts/Sec	$=VC * (SAMP/65536.0)$
Rev/Sec	$=VC * ((SAMP/65536.0)/RES)$
RPM	$=VC * 60.0 * ((SAMP/65536.0)/RES)$

EXAMPLE: (monitor acceleration ramp)

```
O=0           'Set up a velocity move
EL=4000
ADT=10
v=1000000
VT=v
MV
G
WHILE VA<v    'Monitor velocity while
  IF Be       'Accelerating
    BREAK     'Exit if position error
  ENDIF
  GOSUB5     'Report trajectory velocity
LOOP
GOSUB5       'Final report
END

C5
  PRINT ("PRINT VC = ")
  PRINT (VC, #13)  'Get/print commanded velocity
  PRINT ("RVC = ")
  RVC             'Report commanded velocity
  WAIT=4000
RETURN
```

Program output is:

```
RUN
PRINT VC = 565
RVC = 1395
PRINT VC = 322065
RVC = 323155
PRINT VC = 643845
RVC = 644915
PRINT VC = 965605
RVC = 966675
PRINT VC = 1000000
RVC = 1000000
```


RELATED COMMANDS:

R RES *Resolution (see page 702)*

R SAMP *Sampling Rate (see page 722)*

R VA *Velocity Actual (see page 807)*

R VT=formula *Velocity Target (see page 828)*



VL=formula Velocity Limit

APPLICATION:	Motion control
DESCRIPTION:	Gets (reads) or sets the velocity limit
EXECUTION:	Next PID sample
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT VALUE:	RVL
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	Revolutions per minute (RPM)
RANGE OF VALUES:	Class 5: 0 to 32767 ^a Class 6: 0 to 11000 If this range is exceeded, then VL is forced to the value 0
TYPICAL VALUES:	2000-10000
DEFAULT VALUE:	Factory EEPROM setting (based on model)
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	VL:3=1234 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The VL command gets (reads) or sets the velocity limit in revolutions per minute (RPM):

- =VL
Read the current setting of the limit in RPM.
- VL=formula
Set the velocity fault limit in RPM.

NOTE: If the specified VL value exceeds 32767, this command will force the value to 0.

When the motor exceeds this speed (traveling clockwise or counterclockwise), the motor will fault and motion stops. The speed detection is sensitive on every PID cycle. Therefore, be certain to set enough margin between typical speeds and the shutdown speed to avoid easily tripping the fault.

This command helps provide a safety mechanism for high speeds and other situations that could damage equipment. Of greatest concern are those from gravitational loads that could back drive the motor.

^aFirmware versions, such as older Class 5 or other models, may have a lower maximum value (11000). Therefore, note that the range may be restricted based on your motor's firmware version.

EXAMPLE:

```
VL=3500      'Set Velocity Limit to 3500 RPM
```

RELATED COMMANDS:

R VA *Velocity Actual (see page 807)*

VAC(arg) *Velocity Actual (filter) Control (see page 810)*

R VC *Velocity Commanded (see page 815)*

R VT=formula *Velocity Target (see page 828)*

VLD(variable,number) Variable Load

APPLICATION:	EEPROM (Nonvolatile Memory)
DESCRIPTION:	Sequentially load (transfer) user variables from data EEPROM
EXECUTION:	Immediate
CONDITIONAL TO:	EPTR= variable
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	See details
TYPICAL VALUES:	See details
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The VST() and VLD() commands are used to store data (write) to and load data (read) from the internal nonvolatile RAM (EEPROM), respectively. To read or write into this memory space, a memory address location must first be specified with the EPTR=formula command, where formula requires a value between 0 and 32767. Then use the VST() command to store the data or the VLD() command to retrieve the data.

NOTE: Regardless of the size of the variable being accessed, the EPTR pointer always refers to bytes.

To read in a series of values and assign these values to a sequence of user variables, use the VLD (variable, number) command. The first parameter (variable) specifies the name of the user variable as the start of a sequence of variables to load. The second parameter (number) specifies the number of variables in the sequence of variables to store.

The command interpreter automatically notes the size of the defined variable as 1, 2 or 4 bytes long.

The value parameter is a count of the number of units to transfer. The number of bytes in this units depends on the variable designated (refer to the next table). For example, VSD(aw[0],3) transfers three words in sequence, where each word is two bytes. In this example, the total number of bytes stored is six.

Variable	Bytes*
ab[...]	1
aw[...]	2
al[...]	4
a-z, aa-zz,	4

Variable	Bytes*
aaa-zzz	
af[...]	8
*Required per each variable unit	

When using the data EEPROM, it is important to note that the only the data values are stored. The association of these values to any variable is not retained. Therefore, the only way to retrieve this data is by keeping track of the EPTR value. Also, note that:

- If the data memory access is out of range, the syntax error flag (Bs) will be set.
- The user program will not continue until all bytes have been saved to EEPROM.
- When the EEPROM is busy with a read or write, status word 2, bit 13 will be indicated (1).

EXAMPLE: (Storing and retrieving a single 32-bit standard variable)

```
a=123456789      'Assign a value to the variable "a"
EPTR=100         'Set EEPROM pointer to 100
VST(a,1)         'Store into EEPROM (EPTR incremental
                 'to 104 automatically)
EPTR=100         'Set EEPROM to 100 again
VLD(b,1)         'Load from location 100 into the variable "b"
Rb               'Report result will be: 123456789
```

EXAMPLE: (Storing and retrieving a single 16-bit standard variable)

```
aw[0]=32000      'Assign a value to the 16-bit "array word"(0)
EPTR=100         'Set EEPROM pointer to 100
VST(aw[0],1)     'Store into EEPROM (EPTR incremental
                 'to 102 automatically)
EPTR=100         'Set EEPROM to 100 again
VLD(aw[1],1)     'Load from location 100 into the variable aw[1]
Raw[1]           'Report result will be: 32000
```

EXAMPLE: (Storing and retrieving a single 8-bit standard variable)

```
ab[0]=126        'Assign a value to the 8-bit "array byte"(0)
EPTR=100         'Set EEPROM pointer to 100
VST(ab[0],1)     'Store into EEPROM (EPTR incremental
                 'to 101 automatically)
EPTR=100         'Set EEPROM to 100 again
VLD(ab[1],1)     'Load from location 100 into the variable ab[1]
Rab[1]           'Report result will be: 126
```

EXAMPLE: (Storing and retrieving five consecutive 32-bit standard variables)

```

a=10          'Assign values to the variables "a" thru "e"
b=11
c=12
d=13
e=14
EPTR=100     'Set EEPROM pointer to 100
VST(a,5)     'EPTR will increment to 100+(4*5)=120
              '(4 bytes x 5 stored)
EPTR=100     'Set EEPROM to 100 again
VLD(v,5)     'Load from location 100 into the variable "b"
Rv           'Will report 10
Rw           'Will report 11
Rx           'Will report 12
Ry           'Will report 13
Rz           'Will report 14

```

EXAMPLE: (Storing seven 16-bit numbers into EEPROM)

```

i=10          'Using the variable "i" as index to an array variable
j=7           'Using the variable "j" as the number of sequential
              'variables you wish to store

aw[i]=1111
aw[i+1]=2222
aw[i+2]=3333
aw[i+3]=4444
aw[i+4]=-1111
aw[i+5]=-2222
aw[i+6]=-3333
EPTR=3200    'Set EEPROM memory pointer location to 3200
VST(aw[i],j) 'Starting at address 3200, store "j" or seven
              'sequential variables beginning with aw[i]
              'into EEPROM

```

NOTE: The EEPROM value automatically increments for each value stored or read. The EPTR value after the above execution will be set to 3200+(7 variable * 2 bytes each) or 3214.

EXAMPLE: (Retrieving same data into other variables for later use)

```

EPTR=3200
i=10          'Using the variable "i" as index to an array variable
j=7           'Using the variable "j" as the number of sequential
              'Variables you wish to store

VLD(aw[r],s)
WHILE t<5
    PRINT(#13,aw[t+r]," ")
    t=t+1
LOOP
END           'Output is 111 222 333 444 -1111

```

RELATED COMMANDS:

^R EPTR=formula *EEPROM Pointer (see page 450)*

VST(variable,number) *Variable Save (see page 824)*

VST(variable,number) Variable Save

APPLICATION:	EEPROM (Nonvolatile Memory)
DESCRIPTION:	Sequentially store (transfer) user variables to data EEPROM
EXECUTION:	Immediate
CONDITIONAL TO:	EPTR= variable
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	See details
TYPICAL VALUES:	See details
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The VST() and VLD() commands are used to store data (write) to and load data (read) from the internal nonvolatile RAM (EEPROM), respectively. To read or write into this memory space, a memory address location must first be specified with the EPTR=formula command, where formula requires a value between 0 and 32767. Then use the VST() command to store the data or the VLD() command to retrieve the data.

NOTE: Regardless of the size of the variable being accessed, the EPTR pointer always refers to bytes.

To store a series of values into EEPROM, use the VST(variable, number) command. The first parameter (variable) specifies the name of the first user variable of a sequence of variables containing the data to write. The second parameter (number) specifies the number of variables in the sequence of variables to store.

The command interpreter automatically notes the size of the defined variable as 1, 2 or 4 bytes long.

The value parameter is a count of the number of units to transfer. The number of bytes in this units depends on the variable designated (refer to the next table). For example, VSD(aw[0],3) transfers three words in sequence, where each word is two bytes. In this example, the total number of bytes stored is six.

Variable	Bytes*
ab[...]	1
aw[...]	2
al[...]	4
a-z, aa-zz,	4

Variable	Bytes*
aaa-zzz	
af[...]	8
*Required per each variable unit	

When using the data EEPROM, it is important to note that the only the data values are stored. The association of these values to any variable is not retained. Therefore, the only way to retrieve this data is by keeping track of the EPTR value. Also, note that:

- If the data memory access is out of range, the syntax error flag (Bs) will be set.
- The user program will not continue until all bytes have been saved to EEPROM.
- When the EEPROM is busy with a read or write, status word 2, bit 13 will be indicated (1).

EXAMPLE: (Storing and retrieving a single 32-bit standard variable)

```
a=123456789      'Assign a value to the variable "a"
EPTR=100        'Set EEPROM pointer to 100
VST(a,1)        'Store into EEPROM (EPTR incremental
                'to 104 automatically)
EPTR=100        'Set EEPROM to 100 again
VLD(b,1)        'Load from location 100 into the variable "b"
Rb              'Report result will be: 123456789
```

EXAMPLE: (Storing and retrieving a single 16-bit standard variable)

```
aw[0]=32000     'Assign a value to the 16-bit "array word"(0)
EPTR=100        'Set EEPROM pointer to 100
VST(aw[0],1)   'Store into EEPROM (EPTR incremental
                'to 102 automatically)
EPTR=100        'Set EEPROM to 100 again
VLD(aw[1],1)   'Load from location 100 into the variable aw[1]
Raw[1]         'Report result will be: 32000
```

EXAMPLE: (Storing and retrieving a single 8-bit standard variable)

```
ab[0]=126      'Assign a value to the 8-bit "array byte"(0)
EPTR=100        'Set EEPROM pointer to 100
VST(ab[0],1)   'Store into EEPROM (EPTR incremental
                'to 101 automatically)
EPTR=100        'Set EEPROM to 100 again
VLD(ab[1],1)   'Load from location 100 into the variable ab[1]
Rab[1]         'Report result will be: 126
```

EXAMPLE: (Storing and retrieving five consecutive 32-bit standard variables)

```

a=10          'Assign values to the variables "a" thru "e"
b=11
c=12
d=13
e=14
EPTR=100     'Set EEPROM pointer to 100
VST(a,5)     'EPTR will increment to 100+(4*5)=120
              '(4 bytes x 5 stored)
EPTR=100     'Set EEPROM to 100 again
VLD(v,5)     'Load from location 100 into the variable "b"
Rv           'Will report 10
Rw           'Will report 11
Rx           'Will report 12
Ry           'Will report 13
Rz           'Will report 14

```

EXAMPLE: (Storing seven 16-bit numbers into EEPROM)

```

i=10          'Using the variable "i" as index to an array variable
j=7           'Using the variable "j" as the number of sequential
              'variables you wish to store

aw[i]=1111
aw[i+1]=2222
aw[i+2]=3333
aw[i+3]=4444
aw[i+4]=-1111
aw[i+5]=-2222
aw[i+6]=-3333
EPTR=3200    'Set EEPROM memory pointer location to 3200
VST(aw[i],j) 'Starting at address 3200, store "j" or seven
              'sequential variables beginning with aw[i]
              'into EEPROM

```

NOTE: The EEPROM value automatically increments for each value stored or read. The EPTR value after the above execution will be set to $3200+(7 \text{ variable} * 2 \text{ bytes each})$ or 3214.

EXAMPLE: (Retrieving same data into other variables for later use)

```

EPTR=3200
i=10          'Using the variable "i" as index to an array variable
j=7           'Using the variable "j" as the number of sequential
              'Variables you wish to store

VLD(aw[r],s)
WHILE t<5
    PRINT(#13,aw[t+r]," ")
    t=t+1
LOOP
END           'Output is 111 222 333 444 -1111

```

RELATED COMMANDS:

^R EPTR=formula *EEPROM Pointer (see page 450)*

VLD(variable,number) *Variable Load (see page 820)*



VT=formula Velocity Target

APPLICATION:	Motion control; supports the DS2020 Combitronic system
DESCRIPTION:	Gets/sets the velocity target
EXECUTION:	Buffered until a G command is issued
CONDITIONAL TO:	MP, MV, G, PID n (sample rate), encoder resolution.
LIMITATIONS:	N/A
READ/REPORT:	RVT
WRITE:	Read/write
LANGUAGE ACCESS:	Assignment, formulas and conditional testing
UNITS:	(encoder counts / sample) * 65536 DS2020 Combitronic system: user increments / sec, see FD=e-expression on page 461
RANGE OF VALUES:	-2147483648 to 2147483647
TYPICAL VALUES:	-3200000 to 3200000
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	VT:3=1234, a=VT:3, RVT:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

NOTE: This command is affected by the SCALEV command. For details, see SCALEV(m,d) on page 728. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

The VT command is used to get (read) or set the velocity target:

- =VT
Read the current target velocity
- VT=frm
Set the target velocity

The VT command specifies a target velocity (specifies speed and direction) for velocity moves or a slew speed for position moves. The value must be in the range -2147483648 to 2147483647. Note that in position moves, this value is the unsigned speed of the move and does not imply direction. The value set by the VT command only governs the calculated trajectory of MP and MV modes (position and velocity). In either of these modes, the PID compensator may need to "catch up" if the actual position has fallen behind the trajectory position. In this case, the actual speed will exceed the target speed. The value defaults to zero, so it must be set before any motion can occur. The new value does not take effect until the next G command is issued.

Equations for Real-World Units:

Encoder resolution and sample rate can vary. Therefore, the general equations shown in the next table can be used to convert the real-world units of velocity to a value for VT, where af[0] is already set with the real-world unit value. These equations force floating-point calculations to avoid overflow and maintain resolution. They can be placed in a user program, or they can be precalculated if the values of SAMP and RES are known (SAMP and RES can be reported from the terminal using the RSAMP and RRES commands, respectively). SAMP can change if the PID command is used. The value of RES can differ between motor models.

Input as Value in af[0]	Equation
Radians/Sec	$VT = ((af[0] * RES) / (PI * 2.0 * SAMP)) * 65536$
Encoder Counts/Sec	$VT = (af[0] / (SAMP * 1.0)) * 65536$
Rev/Sec	$VT = ((af[0] * RES) / (SAMP * 1.0)) * 65536$
RPM	$VT = ((af[0] * RES) / (60.0 * SAMP)) * 65536$

A useful Scale Factor Multiplier code example, which also illustrates the use of af[], SAMP and RES, is shown in RES on page 702 and SAMP on page 722.

EXAMPLE: (Shows use of ADT, PT and VT)

```
MP           'Set mode position
ADT=5000    'Set target accel/decel
PT=20000   'Set absolute position
VT=10000   'Set velocity
G           'Start motion
END        'End program
```

EXAMPLE: (Routine homes motor against a hard stop)

```
MDS           'Using Sine mode commutation
KP=3200      'Increase stiffness from default
KD=10200    'Increase damping from default
F           'Activate new tuning parameters
AMPS=100    'Lower current limit to 10%
VT=-10000   'Set maximum velocity
ADT=100     'Set maximum accel/decel
MV          'Set Velocity mode
G           'Start motion
WHILE EA>-100 'Loop while position error is small
LOOP        'Loop back to WHILE
O=-100     'While pressed, declare home offset
S          'Abruptly stop trajectory
MP         'Switch to Position mode
VT=20000   'Set higher maximum velocity
PT=0       'Set target position to be home
G         'Start motion
TWAIT     'Wait for motion to complete
AMPS=1023 'Restore current limit to maximum
END       'End program
```

RELATED COMMANDS:

R FD=expression *Resolution to Set Units of Position/Velocity/Acceleration (see page 461)*

R RES *Resolution (see page 702)*

R SAMP *Sampling Rate (see page 722)*

R VA *Velocity Actual (see page 807)*

VAC(arg) *Velocity Actual (filter) Control (see page 810)*

R VC *Velocity Commanded (see page 815)*

R VL=formula *Velocity Limit (see page 818)*

VTS=formula *Velocity Target, Synchronized Move (see page 831)*

VTS=formula

Velocity Target, Synchronized Move

APPLICATION:	Motion control
DESCRIPTION:	Sets the synchronized (path) velocity target
EXECUTION:	Must be set before issuing PTS or PRTS; will not be effective after that point
CONDITIONAL TO:	PID n
LIMITATIONS:	Must not be negative; 0 is not valid
READ/REPORT:	None
WRITE:	Write only
LANGUAGE ACCESS:	Assignment
UNITS:	(encoder counts / sample) * 65536 in 2D or 3D space
RANGE OF VALUES:	0 to 2147483647
TYPICAL VALUES:	100000 to 3200000
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M), not ver. 5.32.x.x; 6.x (D/M), M requires EIP option
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

NOTE: This command requires a Combitronic-supported motor. Although this command does not support Combitronic syntax, it does use Combitronic communication to pass information between other motors.

NOTE: This command is affected by the SCALEV command. For details, see SCALEV(m,d) on page 728. For the list of SCALE-affected commands, see Commands Affected by SCALE on page 903.

The VTS command sets the maximum velocity target for synchronized moves. The motion along a synchronized move is defined along the path in 2D or 3D space depending on the number of axes defined by PTS or PRTS.

The VTS command is specific to defining the *combined* velocity of all contributing axes. For example, if the move were to occur in an X-Y plane, the velocity set by VTS would not pertain to the just the X- or Y-axis. Rather, it applies to their *combined* motion in the direction of motion.

The value of VTS defaults to zero. Therefore, it must be given a value before any motion can take place.

A useful Scale Factor Multiplier code example, which also illustrates the use of af[], SAMP and RES, is shown in RES on page 702 and SAMP on page 722.

EXAMPLE: (Shows use of ATS, DTS and VTS)

```

EIGN (W, 0)      'Set all I/O as general inputs.
ZS              'Clear errors.
ATS=100         'Set synchronized acceleration target.
DTS=500         'Set synchronized deceleration target.
VTS=100000000  'Set synchronized target velocity.
PTS (500;1,1000;2,10000;3) 'Set synchronized target position
                  'on motor 1, 2 and 3.
GS              'Initiate synchronized move.
TSWAIT         'Wait until synchronized move ends.
END            'Required END.

```

RELATED COMMANDS:

ADTS=formula *Acceleration/Deceleration Target, Synchronized (see page 265)*
 R HM_VTS=formula *Homing Velocity Target to Switch (see page 498)*
 PID# *Proportional-Integral-Differential Filter Rate (see page 654)*
 PRTS(...) *Position, Relative Target, Synchronized (see page 685)*
 PRTSS(...) *Position, Relative Target, Synchronized, Supplemental (see page 688)*
 PTS(...) *Position Target, Synchronized (see page 692)*
 PTSS(...) *Position Target, Synchronized Supplemental (see page 696)*



APPLICATION:	System
DESCRIPTION:	Reports a specified status word
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RW(word) ; supports the DS2020 Combitronic system
WRITE:	Read only
LANGUAGE ACCESS:	Formulas and conditional testing
UNITS:	N/A
RANGE OF VALUES:	Input: 0-17 Output: 0-65535
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	RW(16):3, x=W(16):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The W(word) command reports the specified 16-bit status word. Refer to the next table. Also, see Status Words - SmartMotor on page 921, and see Logical I/O User Read Commands Example for Class 5 M-style Motor on page 510.

Status Word	Purpose
0	Drive state and hardware limits; supports the DS2020 Combitronic system reporting
1	Index capture and software limits; supports the DS2020 Combitronic system reporting
2	Programs and communications; supports the DS2020 Combitronic system reporting
3	PID and motion; supports the DS2020 Combitronic system reporting
4	Timers; supports the DS2020 Combitronic system reporting
5	Interrupts; supports the DS2020 Combitronic system reporting
6	Commutation and bus; supports the DS2020 Combitronic system reporting
7	Trajectory details

Status Word	Purpose
8	Cam and interpolation user bits
9	SD card information (Class 6 M-style only)
10	RxPDO arrival notification
11	Reserved
12	User-controlled bits, word 0 (also used when DMX is active)
13	User-controlled bits, word 1
14-15	Reserved
16	I/O state, word 0
17	I/O state, word 1 (D-style with AD1 option only)

EXAMPLE:

```
ab[10]=W(16)      'Read the status of on-board I/O via
                  'controller's status word.
```

EXAMPLE:

```
      IF (W(0) & 49152) == 49152    'Look at both limits, bits 14 & 15,
                                     'with bit mask 49152 = 32768 + 16384.
GOSUB100      'Execute subroutine.
ENDIF

C100          'Subroutine code here.
RETURN
```

RELATED COMMANDS:

R B(word,bit) *Status Byte (see page 297)*

R FAUSTS(x) *Returns Fault Status Word (see page 459)*

Z *Total CPU Reset (see page 846)*

ZS *Global Reset System State Flag (see page 858)*

WAIT=formula

Wait for Specified Time

APPLICATION:	Program execution and flow control
DESCRIPTION:	Suspends user program execution for specified amount of time
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	milliseconds
RANGE OF VALUES:	0 to 2147483647
TYPICAL VALUES:	0 to 4000
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

WAIT=formula pauses program execution for a specified amount of time. Time is measured in milliseconds (e.g., WAIT=1000 is one second).

EXAMPLE: (Dynamically change from Velocity mode to Torque mode)

```

MV           'Set motor to Velocity mode
VT=100000   'Set velocity to 100000
ADT=1000    'Set accel/decel to 1000
G           'Go (Start moving)
WAIT=2000   'Wait about 2 seconds
T=TRQ       'Set torque to the value the PID filter
            'was commanding in MV
MT G        'Set motor to Torque mode
WAIT=2000   'Wait about 2 seconds
OFF         'Turn the motor off

```

EXAMPLE: (Change commanded speed and acceleration)

```
O=0           'Set current position to zero
MP           'Set to position mode (required if currently
            'in another mode)
VT=100000    'Set velocity to 100000
ADT=1000     'Set accel/decel to 1000
PT=1000000  'Set commanded absolute position to 1000000
G           'Go (Start moving)
WAIT=8000    'Wait about 8 seconds
VT=800000   'Set new velocity of 800000
ADT=500     'Set new accel/decel of 500
G           'Initiate change in speed and acceleration
```

RELATED COMMANDS:

R CLK=formula *Millisecond Clock (see page 369)*

R TMR(timer,time) *Timer (see page 782)*

TWAIT(gen#) *Trajectory Wait (see page 789)*

WAKE

Wake Communications Port 0

APPLICATION:	Communications control; supports the DS2020 Combitronic system
DESCRIPTION:	Motor to execute all channel 0 communications commands
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	WAKE state
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

WAKE clears the SLEEP condition of a SmartMotor™. Except for the WAKE command, a SmartMotor that has been put to SLEEP rejects all other commands received through the primary port (communications channel 0).

WAKE is typically used by a host communicating over the serial channel to isolate individual motors. This may be required when a program is being downloaded or motors are being assigned addresses. The WAKE command can also be used in a program; however, this must be done with caution.

EXAMPLE: (Shows use of SLEEP, SLEEP1, WAKE and WAKE1)

```
'These commands can be sent from the SMI software Terminal
>window to address three SmartMotors:
'0SADDR1
'1ECHO
'1SLEEP
'0SADDR2
'2ECHO
'2SLEEP
'0SADDR3
'3ECHO
'0WAKE
'A host program other than SMI can send the same commands, but the
'prefixed addressing is different. The 0, 1, 2 and 3 are actually
'0x80, 0x81, 0x82 and 0x83, respectively.
'The decimal equivalent of the hex values are 128, 129, 130 and 131.
'The next commands can be sent from a program in motor 1 to
'Motor 2:
PRINT(#130,"SLEEP",#13) 'Cause channel 0 (RS-232) of motor 2 to SLEEP.
PRINT(#130,"WAKE",#13) 'Cause channel 0 (RS-232) of motor 2 to WAKE.
PRINT(#130,"SLEEP1",#13) 'Cause channel 1 (RS-485) of motor 2 to SLEEP.
                          'through channel 0 (RS-232).
PRINT(#130,"WAKE1",#13) 'Cause channel 1 (RS-485) of motor 2 to WAKE
                          'through channel 0 (RS-232).

'Assuming channel 1 (RS-485) is open on all motors with the
'OCHN command, the same commands can be sent with the PRINT1
'command:
OCHN(RS4,1,N,9600,1,8,C) 'Open ports 4 and 5 as RS-485 channel 1.
PRINT1(#130,"SLEEP",#13) 'Cause channel 0 (RS-232) of motor 2 to SLEEP.
PRINT1(#130,"WAKE",#13) 'Cause channel 0 (RS-232) of motor 2 to WAKE.
PRINT1(#130,"SLEEP1",#13) 'Cause channel 1 (RS-485) of motor 2 to SLEEP.
PRINT1(#130,"WAKE1",#13) 'Cause channel 1 (RS-485) of motor 2 to WAKE.
END
```

RELATED COMMANDS:

SLEEP *Ignore Incoming Commands on Communications Port 0 (see page 744)*

SLEEP1 *Ignore Incoming Commands on Communications Port 1 (see page 746)*

WAKE1 *Wake Communications Port 1 (see page 839)*

WAKE1

Wake Communications Port 1

APPLICATION:	Communications control
DESCRIPTION:	Motor to execute all channel 1 communications commands
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	WAKE1 state
FIRMWARE VERSION:	5.0.x, 5.16.x or 5.32.x series (D); 6.4.2.x (D)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

WAKE1 clears the SLEEP1 condition of a SmartMotor™. Except for the WAKE1 command, a SmartMotor that has been put to SLEEP1 rejects all other commands received through the channel 1 serial port.

WAKE1 is typically used by a host communicating over the serial channel to isolate individual motors. This may be required when a program is being downloaded or motors are being assigned addresses. The WAKE1 command can also be used in a program; however, this must be done with caution.

EXAMPLE: (Shows use of SLEEP, SLEEP1, WAKE and WAKE1)

```
'These commands can be sent from the SMI software Terminal
>window to address three SmartMotors:
'0SADDR1
'1ECHO
'1SLEEP
'0SADDR2
'2ECHO
'2SLEEP
'0SADDR3
'3ECHO
'0WAKE
'A host program other than SMI can send the same commands, but the
'prefixed addressing is different. The 0, 1, 2 and 3 are actually
'0x80, 0x81, 0x82 and 0x83, respectively.
'The decimal equivalent of the hex values are 128, 129, 130 and 131.
'The next commands can be sent from a program in motor 1 to
'Motor 2:
PRINT(#130,"SLEEP",#13) 'Cause channel 0 (RS-232) of motor 2 to SLEEP.
PRINT(#130,"WAKE",#13) 'Cause channel 0 (RS-232) of motor 2 to WAKE.
PRINT(#130,"SLEEP1",#13) 'Cause channel 1 (RS-485) of motor 2 to SLEEP.
                        'through channel 0 (RS-232).
PRINT(#130,"WAKE1",#13) 'Cause channel 1 (RS-485) of motor 2 to WAKE
                        'through channel 0 (RS-232).

'Assuming channel 1 (RS-485) is open on all motors with the
'OCHN command, the same commands can be sent with the PRINT1
'command:
OCHN(RS4,1,N,9600,1,8,C) 'Open ports 4 and 5 as RS-485 channel 1.
PRINT1(#130,"SLEEP",#13) 'Cause channel 0 (RS-232) of motor 2 to SLEEP.
PRINT1(#130,"WAKE",#13) 'Cause channel 0 (RS-232) of motor 2 to WAKE.
PRINT1(#130,"SLEEP1",#13) 'Cause channel 1 (RS-485) of motor 2 to SLEEP.
PRINT1(#130,"WAKE1",#13) 'Cause channel 1 (RS-485) of motor 2 to WAKE.
END
```

RELATED COMMANDS:

SLEEP *Ignore Incoming Commands on Communications Port 0 (see page 744)*

SLEEP1 *Ignore Incoming Commands on Communications Port 1 (see page 746)*

WAKE *Wake Communications Port 0 (see page 837)*

WHILE formula

While Condition Program Flow Control

APPLICATION:	Program execution and flow control
DESCRIPTION:	Defines block of code that repeats while formula is true
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	Formula that evaluates true or false
TYPICAL VALUES:	Formula that evaluates true or false
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The WHILE command defines the start of a program loop that repeatedly executes as long as the evaluated condition is true (not equal to zero). Each WHILE formula control block must be terminated with a corresponding LOOP exit statement (see LOOP on page 553). WHILE control blocks may be nested (see the second example).

NOTE: WHILE is not a valid terminal command; it is only valid within a user program.

The WHILE...LOOP control block looks like this:

```
WHILE {formula is true}
    execute program command here
LOOP
```

The "formula" is evaluated the first time WHILE is encountered:

- If true (not zero), program execution is sent back to the WHILE by the corresponding LOOP command, and the formula is evaluated again.
- If false (zero), program execution redirects to the code just below the LOOP command.

Any valid standard formula can be used. In particular, WHILE 1...LOOP is a standard "loop forever" control block.

The formula may be similar to that used when assigning a value to a variable. However, it is strongly recommended to always use a comparison operator such as:

```
==    !=    <    >    <=    >=
```

(for more information, see Math Operators on page 915).

For example, the formula $a=(b+2)*3$ would be applied to a WHILE as:

```
WHILE ( (b+2) *3) !=0
LOOP
```

This is preferred to merely writing "WHILE (b+2)*3". The logical condition being tested is more obvious when the comparison operators are used. It is also possible to combine multiple logical tests when the comparison operators are used:

```
WHILE (a>(b+1)) & (c!=d)
LOOP
```

This statement loops as long as "c" does not equal "d" and "a" is greater than "b+1".

If a BREAK command is encountered while executing a WHILE control block, program execution unconditionally redirects to the program code after the LOOP statement. For details, see BREAK on page 331.

EXAMPLE: (Routine stops motion if voltage drops)

```
EIGN(W,0)      'Disable hardware limits
ZS             'Clear faults
MDS           'Sine mode commutation
ADT=100       'Set maximum accel/decel
VT=100000    'Set maximum velocity
PT=1000000   'Set final position
MP            'Set Position mode
G            'Start motion
WHILE Bt      'Loop while motion continues
  IF UJA<18500 'If voltage is below 18.5 volts
    OFF        'Turn motor off
  ENDIF
LOOP          'Loop back to WHILE
END           'Required END
```

EXAMPLE: (Routine pulses output on a given position)

```

EIGN(W,0)           'Disable limits
ZS                 'Clear faults
ITR(0,4,0,0,1)    'ITR(int#,sw,bit,state,lbl)
ITRE              'Enable all interrupts
EITR(0)           'Enable interrupt 0
OUT(1)=1          'Set I(0)/O B to output, high
ADT=100           'Set maximum accel/decel
VT=100000         'Set maximum velocity
MP               'Set Position mode
'****Main Program Body****
WHILE 1>0
  O=0             'Reset origin for move
  PT=40000        'Set final position
  G              'Start motion
  WHILE PA<20000 'Loop while motion continues
  LOOP           'Wait for desired position to pass
  OUT(1)=0       'Set output low
  TMR(0,400)     'Use timer 0 for pulse width
  TWAIT
  WAIT=1000      'Wait 1 second
LOOP
END
'****Interrupt Subroutine****
C1
  OUT(1)=1       'Set output high again
RETURNI

```

RELATED COMMANDS:

BREAK *Break from CASE or WHILE Loop (see page 331)*
 IF formula *Conditional Program Code Execution (see page 506)*
 LOOP *Loop Back to WHILE Formula (see page 553)*
 SWITCH formula *Switch, Program Flow Control (see page 766)*
 WHILE formula *While Condition Program Flow Control (see page 841)*



Decelerate to Stop

APPLICATION:	Motion control; supports the DS2020 Combitronic system
DESCRIPTION:	Slow motor motion to stop
EXECUTION:	Immediate
CONDITIONAL TO:	Mode dependent—the appropriate deceleration or ramp-down parameter must be set (see details)
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	X:3 or X(0):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The X command immediately abandons the current trajectory mode and causes the motor to slow to a stop using the current deceleration value DT (in a servo mode) or TS (in torque mode).

NOTE: This is different from the S command, which does not consider the DT or TS value.

The X command leaves the motor in its current motion mode. Refer to the next table for the motion modes and appropriate parameters to set.

Motion Mode	Appropriate Parameter to Set
MV	DT or ADT
MP	DT or ADT
MFR	MFD()
MSR	MFD()
MC	MFD()
MT	TS
MH	HM_ADT

The X command halts the homing operation. For more details on SmartMotor homing operations, see the *SmartMotor Homing Procedures and Methods Application Note*.

EXAMPLE:

```
EIGN (W, 0)
ZS
ADT=100
VT=1000000
PT=5000000
G           'Start motion
WHILE Bt   'While trajectory is active
  IF PA>80000 'Set a position to look for
    X           'Decelerate to a stop
    PRINT("Motion Stopped")
  ENDIF
LOOP
END           'Required END
```

Program output is:

Motion Stopped

RELATED COMMANDS:

G *Start Motion (GO)* (see page 473)
S (as command) *Stop Motion* (see page 718)



APPLICATION:	Reset; supports the DS2020 Combitronic system
DESCRIPTION:	Reset the motor to power-up condition
EXECUTION:	Immediate
CONDITIONAL TO:	Serial character transmit completion
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	Z:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The Z command will totally reset the SmartMotor™ as if power were removed and then restored. Consequently, if there is a stored program, it will be run from the beginning. All modes of operation, variables and status bits will be restored back to their defaults. Subsequent to a power up or reset, the SmartMotor will:

1. Initialize the motion mode, status bits and variables.
2. Hold the serial port closed for approximately $\frac{1}{4}$ second.
3. Open and initialize the serial port.
4. Delay for $\frac{1}{2}$ second. At the end of this time, the SmartMotor will examine the communications buffer. The stored program will be aborted only if the specific characters "EE" are found.
5. Run the stored program (unless aborted as previously described).

After a program downloads, using the Z command is a good way to evaluate how your SmartMotor will operate when powered on. The RUN command will execute the stored program, but it will not clear the motor to its default condition. Therefore, the operation after the RUN command will not necessarily mimic what would happen at power up. Using the Z command resets the motor to its power-on state.



CAUTION: The Z command should not be used at or near the top of a user program. Doing so may cause a continuous and repetitive resetting of the CPU and lock out the motor.

NOTE: If you get locked out and are unable to communicate with the SmartMotor, you may be able to recover communications using the SMI software's Communication Lockup Wizard. For more details, see Communication Lockup Wizard on page 31.

EXAMPLE:

```
Z      'Warning issuing this command will cause CPU reset immediately
```

RELATED COMMANDS:

RUN *Run Program (see page 714)*

RUN? *Halt Program Execution Until RUN Received (see page 716)*



Z(word,bit)

Reset Specified Status Bit

APPLICATION:	Reset
DESCRIPTION:	Resets the specified status bit
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	Not all status bits are resettable — will return error if not resettable with Z()
READ/REPORT:	None
WRITE:	Write only
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	Input: word: 0-10 bit: 0-15
TYPICAL VALUES:	Input: word: 0-6 bit: 0-15
DEFAULT VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	Z(2,4):3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

The Z(word,bit) command resets the specified status bit:

- Z(word,bit)
Clears specific status bit in the specific status word.

Refer to the next table, which shows the bits that can be reset with Z(word,bit).

Reset bit with Z(word,bit)?																
Status word	Bit															
	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
0				y	y	y	y	y		y			y	y		
1			y	y			y	y					y	y		
2	y	y	*		y		*						y		y	
3	y			y				y				y				

Reset bit with Z(word,bit)?																
Status word	Bit															
	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
4																
5																
6						y		y						y	y	
7																
8																
9			*					*	*							
10		y	y	y	y	y										
11	N/A															
12-13	Use commands: UR, US, UO															
14-15	N/A															
16-17	Use I/O commands: OR, OS, OUT															
* Class 6 only																

EXAMPLE: (Subroutine prints and resets channel 0 errors)

C9

```

IF CHN(0)          'If CHN0 != 0
  IF CHN(0) &1
    PRINT("BUFFER OVERFLOW")
  ENDIF
  IF CHN(0) &2
    PRINT("FRAMING ERROR")
  ENDIF
  IF CHN(0) &4
    PRINT("COMMAND SCAN ERROR")
  ENDIF
  IF CHN(0) &8
    PRINT("PARITY ERROR")
  ENDIF
  Z(2,0)          'Reset CHN0 errors
ENDIF
RETURN

```

RELATED COMMANDS:

R B(word,bit) *Status Byte (see page 297)*

Z *Total CPU Reset (see page 846)*

ZS *Global Reset System State Flag (see page 858)*

R W(word) *Report Specified Status Word (see page 833)*

Za Reset Overcurrent Flag

APPLICATION:	Reset
DESCRIPTION:	Resets the overcurrent flag (Ba)
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RBa
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
RESET VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

Za resets the overcurrent error flag, Ba, to zero. If the current violation still exists, Ba will again be set to 1.

If the Ba flag is repeatedly set, there may be a problem such as incorrect motor size. Therefore, verify the correct motor has been selected for the current task. For details on motor sizing, see the *Moog Animatics Product Catalog*.

EXAMPLE:

```
IF Ba      'Test flag
    PRINT ("OVER CURRENT")
    Za     'Reset flag
ENDIF
WAIT=4000
IF Ba      'Retest flag
    PRINT ("OVER CURRENT STILL IN EFFECT")
ENDIF
```

RELATED COMMANDS:

R Ba *Bit, Peak Overcurrent (see page 301)*

Z *Total CPU Reset (see page 846)*

Z(word,bit) *Reset Specified Status Bit (see page 848)*

ZS *Global Reset System State Flag (see page 858)*

Ze Reset Position Error Flag

APPLICATION:	Reset
DESCRIPTION:	Resets the position error status bit (Be)
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RBe
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
RESET VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

Ze resets the position error flag, Be, to zero.

EXAMPLE:

```
IF Be                                'Test flag
    PRINT("Position Error")
    Ze                                'Reset flag
ENDIF
```

RELATED COMMANDS:

^R Be *Bit, Position Error Limit (see page 305)*

Z *Total CPU Reset (see page 846)*

Z(word,bit) *Reset Specified Status Bit (see page 848)*

ZS *Global Reset System State Flag (see page 858)*

Zh Reset Temperature Fault

APPLICATION:	Reset
DESCRIPTION:	Resets an excessive temperature fault
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RBh
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The Zh command is used to reset an excessive temperature fault. The motor's present temperature must be at least 5 degrees below the fault threshold (set by TH=, reported by RTH) for this command to take effect.

EXAMPLE:

```
Zh      'Resetting Thermal Fault status bit
```

RELATED COMMANDS:

R Be Bit, Position Error Limit (see page 305)

Z Total CPU Reset (see page 846)

Z(word,bit) Reset Specified Status Bit (see page 848)

ZS Global Reset System State Flag (see page 858)

ZI Reset Historical Left Limit Flag

APPLICATION:	Reset
DESCRIPTION:	Resets the historical left-limit latch (Bl)
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RBI
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
RESET VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

ZI resets the historical left-limit flag, Bl, to zero. If you use Bl to detect the activation of the left limit, be sure to reset it with ZI before rescanning for the bit.

EXAMPLE:

```
IF Bl      'Test flag
    PRINT("Left Limit Latched")
    ZI    'Reset flag
ENDIF
```

RELATED COMMANDS:

^R Bl *Bit, Left Hardware Limit, Historical (see page 316)*

Z *Total CPU Reset (see page 846)*

Z(word,bit) *Reset Specified Status Bit (see page 848)*

ZS *Global Reset System State Flag (see page 858)*

Zls

Reset Left Software Limit Flag, Historical

APPLICATION:	Reset
DESCRIPTION:	Reset historical left/negative software limit latch
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RBlS
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
RESET VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The Zls command resets the left (negative) software limit latch, Bls, to zero. If you use Bls to detect the activation of the left software limit, be sure to reset it with Zls before rescanning for the bit.

EXAMPLE:

```
IF Bls      'Test flag
    PRINT("Left Software Limit Latched")
    Zls 'Reset flag
ENDIF
```

RELATED COMMANDS:

R Bls *Bit, Left Software Limit, Historical (see page 318)*

Z *Total CPU Reset (see page 846)*

Z(word,bit) *Reset Specified Status Bit (see page 848)*

ZS *Global Reset System State Flag (see page 858)*

Zr

Reset Right Limit Flag, Historical

APPLICATION:	Reset
DESCRIPTION:	Resets the historical right-limit latch (Br)
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RBr
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
RESET VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

Zr resets the historical right-limit flag, Br, to zero. If you use Br to detect the activation of the right limit, be sure to reset it with Zr before rescanning for the bit.

EXAMPLE:

```
IF Br                               'Test flag
    PRINT("Right Limit Latched")
    Zr                               'Reset flag
ENDIF
```

RELATED COMMANDS:

^R Br *Bit, Right Hardware Limit, Historical (see page 329)*

Z *Total CPU Reset (see page 846)*

Z(word,bit) *Reset Specified Status Bit (see page 848)*

ZS *Global Reset System State Flag (see page 858)*

Zrs

Reset Right Software Limit Flag, Historical

APPLICATION:	Reset
DESCRIPTION:	Resets the historical right (positive) software limit latch (Brs)
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RBrS
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
RESET VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The Zrs command resets the right (positive) software limit latch, Brs, to zero. If you use Brs to detect the activation of the right software limit, be sure to reset it with Zrs before rescanning for the bit.

EXAMPLE:

```
IF Brs      'Test flag
    PRINT("Right Software Limit Latched")
    Zrs 'Reset flag
ENDIF
```

RELATED COMMANDS:

R Brs Bit, Right Software Limit, Historical (see page 341)

Z Total CPU Reset (see page 846)

Z(word,bit) Reset Specified Status Bit (see page 848)

ZS Global Reset System State Flag (see page 858)

Reset Command Syntax Error Flag

APPLICATION:	Reset
DESCRIPTION:	Reset command scan error latch
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RBs
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
RESET VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

Zs resets the command error latch flag, Bs, to zero. The Zs command can be used with the RBs report command to verify that the current firmware version recognizes what appears to be a valid command and data packet.

EXAMPLE:

```
IF Bs      'Test flag
    PRINT("Syntax Error")
    Zs 'Reset flag
ENDIF
```

RELATED COMMANDS:

R Bs *Bit, Syntax Error (see page 343)*
Z *Total CPU Reset (see page 846)*
Z(word,bit) *Reset Specified Status Bit (see page 848)*
ZS *Global Reset System State Flag (see page 858)*



Global Reset System State Flag

APPLICATION:	Reset; supports the DS2020 Combitronic system
DESCRIPTION:	Reset software system latches to power up state
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	N/A
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
RESET VALUE:	N/A
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M); ds2020_sa_1.0.0_combican (DS2020)
COMBITRONIC:	ZS:3 where ":3" is the motor address — use the actual address or a variable

DETAILED DESCRIPTION:

Almost any event that occurs within a SmartMotor™ gets recorded in system flags. These flags can be read as part of a program or a host inquiry. After a flag is read, it must be reset so it can record the next event. ZS resets all of the latched bits in the S status byte, the W status word, and the status bits such as Ba, Be, Bh, Bi, etc.

ZS performs the flag resets shown in the next table.

Status word and bits	Z letter command	Description
Word 0, bit 3	-	Reset bus voltage fault
Word 0, bit 4	Za	Reset hardware current limit error/fault
Word 0, bit 5	Zh	Reset temperature fault
Word 0, bit 6	Ze	Reset position fault
Word 0, bit 7	Zv	Reset velocity fault
Word 0, bit 9	-	Reset dE/dt limit fault
Word 0, bit 12	Zr	Reset historical right limit fault
Word 0, bit 13	Zl	Reset historical left limit fault
Word 1, bit 2	-	Reset rising edge capture on internal encoder

Status word and bits	Z letter command	Description
Word 1, bit 3	-	Reset falling edge capture on internal encoder
Word 1, bit 6	-	Reset rising edge capture on external encoder
Word 1, bit 7	-	Reset falling edge capture on external encoder
Word 1, bit 12	Zrs	Reset historical right software limit fault
Word 1, bit 13	Zls	Reset historical left software limit fault
Word 2, bit 0	-	Reset COM 0 errors and bits in RCHN(0)
Word 2, bit 1	-	Reset COM 1 errors and bits in RCHN(1)
Word 2, bit 2	-	Reset USB error (Class 6 only)
Word 2, bit 4	-	Reset CAN errors and associated error bits in RCAN
Word 2, bit 6	-	Reset Ethernet errors (Class 6 only)
Word 2, bit 14	Zs	Reset user command syntax error
Word 3, bit 3	Zw	Reset wraparound
Word 3, bit 11	-	Reset modulo rollover flag
Word 6, bit 5	-	Reset feedback fault
Word 6, bit 7	-	Reset drive enable fault
Word 6, bit 13	-	Reset low bus flag
Word 6, bit 14	-	Reset high bus flag

NOTE: In cases where the motor has gone beyond the EL (error limit) but the trajectory generator is still active with the previously calculated trajectory, the ZS command may not clear the Be bit. If you are unable to reset Be with the ZS command, issue an OFF command before issuing the ZS command, which clears the current commanded trajectory and allows the reset to complete.

EXAMPLE:

```
ZS
'Reset error and limit flag latches
```

RELATED COMMANDS:

Z Total CPU Reset (see page 846)
 Z(word,bit) Reset Specified Status Bit (see page 848)
 Za Reset Overcurrent Flag (see page 850)
 Ze Reset Position Error Flag (see page 851)
 Zh Reset Temperature Fault (see page 852)
 Zl Reset Historical Left Limit Flag (see page 853)
 Zls Reset Left Software Limit Flag, Historical (see page 854)
 Zr Reset Right Limit Flag, Historical (see page 855)
 Zrs Reset Right Software Limit Flag, Historical (see page 856)
 Zs Reset Command Syntax Error Flag (see page 857)
 Zv Reset Velocity Limit Fault (see page 860)
 Zw Reset Encoder Wrap Status Flag (see page 861)

Zv

Reset Velocity Limit Fault

APPLICATION:	Reset
DESCRIPTION:	Resets a velocity error fault
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RBv
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

The Zv command is used to reset a velocity error fault.

EXAMPLE:

```
Zv      'Reset Velocity Fault status bit
```

RELATED COMMANDS:

^R Bv *Bit, Velocity Limit (see page 347)*

Z *Total CPU Reset (see page 846)*

Z(word,bit) *Reset Specified Status Bit (see page 848)*

ZS *Global Reset System State Flag (see page 858)*

Zw Reset Encoder Wrap Status Flag

APPLICATION:	Reset
DESCRIPTION:	Resets the encoder wraparound event latch (Bw)
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
READ/REPORT:	RBw
WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
RESET VALUE:	0
FIRMWARE VERSION:	5.x (D/M); 6.x (D/M)
COMBITRONIC:	N/A

DETAILED DESCRIPTION:

Zw resets the encoder wraparound status flag, Bw, to zero. The SmartMotor™ tracks its position as 32-bit data, so a valid position is from -2147483648 to +2147483647. If the motor moves out of this range, the position will overflow or "wraparound".

This is provided for information purposes. Applications with continuous rotation (Velocity mode, Torque mode, Relative Position mode, Cam mode, Follow mode) may experience wraparound. The motor will continue to operate, but the user application may need to know this event occurred.

EXAMPLE: (Test for wraparound and then reset flag)

```
IF Bw      'Test flag
    PRINT("Wraparound Occurred")
    Zw     'Reset flag
ENDIF
```

RELATED COMMANDS:

R Bw Bit, Wrapped Encoder Position (see page 349)

Z Total CPU Reset (see page 846)

Z(word,bit) Reset Specified Status Bit (see page 848)

ZS Global Reset System State Flag (see page 858)

Part 3: Example SmartMotor Programs

Part 3 of this guide provides examples of SmartMotor programs that can be used as reference material for application development. The code examples can be copied and pasted into the SMI program editor.

Move Back and Forth	863
Move Back and Forth with Watch	863
Home Against a Hard Stop (Basic)	864
Home Against a Hard Stop (Advanced)	864
Home Against a Hard Stop (Two Motors)	865
Home to Index Using Different Modes	867
Maintain Velocity During Analog Drift	868
Long-Term Storage of Variables	869
Find Errors and Print Them	869
Change Speed on Digital Input	870
Pulse Output on a Given Position	870
Stop Motion if Voltage Drops	871
Camming - Variable Cam Example	872
Camming - Fixed Cam with Input Variables	873
Camming - Demo XY Circle	875
Chevron Traverse & Takeup	877
CAN Bus - Timed SDO Poll	879
CAN Bus - I/O Block with PDO Poll	880
CAN Bus - Time Sync Follow Encoder	883
Text Replacement in an SMI Program	891

Move Back and Forth

This is a simple program used to set tuning parameters and create an infinite loop, which causes the motor to move back and forth. Note the TWAIT commands that are used to pause program execution during the moves.

```
EIGN (W,0)      'Disable hardware limits
ZS              'Clear faults
ADT=100         'Set maximum accel/decel
VT=1000000     'Set maximum velocity
MP             'Set Position mode
C10            'Place a label
  PT=100000    'Set position
  G            'Start motion
  TWAIT       'Wait for move to complete
  PT=0        'Set position
  G           'Start motion
  TWAIT       'Wait for move to complete
GOTO (10)     'Loop back to label 10
END           'Obligatory END (never reached)
```

Move Back and Forth with Watch

The next example is identical to the previous, except that instead of pausing program execution during the move with the TWAIT, a subroutine is used to monitor for excessive load during the moves. This is an important distinction — most SmartMotor programs should have the ability to react to events during motion.

```
EIGN (W,0)      'Disable hardware limits
ZS              'Clear faults
ADT=100         'Set maximum accel/decel
VT=100000      'Set maximum velocity
MP             'Set Position mode
C1             'Place a label
  PT=100000    'Set position
  G            'Start motion
  GOSUB (10)   'Call wait subroutine
  PT=0        'Set position
  G           'Start motion
  GOSUB (10)   'Call wait subroutine
  GOTO (1)    'Loop back to label 1
END           'Obligatory END (never reached)
' ****Subroutine****
C10
  WHILE Bt     'Loop while trajectory in progress
    IF ABS(EA)>100 'Test for excessive load
      PRINT("Excessive Load",#13) 'Print warning
    ENDIF     'End test
  LOOP       'Loop back to While during motion
RETURN      'Return from subroutine
```

Home Against a Hard Stop (Basic)

Because the SmartMotor has the capability of lowering its own power level and reading its position error, it can be programmed to gently feel for the end of travel. This provides a means to develop a consistent home position subsequent to each power-up.

Machine reliability requires the elimination of potential failure sources. Eliminating a home switch and its associated cable leverages SmartMotor benefits and improves machine reliability.

This program lowers the current limit, moves against a limit, looks for resistance and then declares and moves to a home position located 100 counts from the hard stop.

```
MDS           'Using Sine mode commutation
KP=3200       'Increase stiffness from default
KD=10200     'Increase damping from default
F            'Activate new tuning parameters
AMPS=100     'Lower current limit to 10%
VT=-10000    'Set maximum velocity
ADT=100      'Set maximum accel/decel
MV           'Set Velocity mode
G            'Start motion
WHILE EA>-100 'Loop while position error is small
LOOP         'Loop back to WHILE
O=-100       'While pressed, declare home offset
S            'Abruptly stop trajectory
MP           'Switch to Position mode
VT=20000     'Set higher maximum velocity
PT=0         'Set target position to be home
G            'Start motion
TWAIT       'Wait for motion to complete
AMPS=1000    'Restore current limit to maximum
END          'End Program
```

Home Against a Hard Stop (Advanced)

Because the SmartMotor has the capability of lowering its own power level and reading its position error, it can be programmed to gently feel for the end of travel. This provides a means to develop a consistent home position subsequent to each power-up.

Machine reliability requires the elimination of potential failure sources. Eliminating a home switch and its associated cable leverages SmartMotor benefits and improves machine reliability.

Similar to the previous example, the next program lowers the current limit, moves against a limit, looks for resistance and then declares and moves to a home position just one encoder revolution from the hard stop. However, this example provides a more sophisticated version of the previous method.


```

=====
'Class 5 Home To Hard Stop Example
EIGN(2) EIGN(3) ZS 'Bypass Overtravel Limits
=====
'Set up parameters
    rr=-1           'Home Direction
    vv=100000      'Home Speed
    aa=1000        'Home Accel
    ee=100         'Home Error Limit
    tt=1500        'Home Torque Limit
    hh=4000        'Home Offset
SILENT 'Remove or issue TALK to enable PRINT commands
GOSUB5 'RUN HOME ROUTINE
END
=====
'Home routine (Home to Hard Stop)
C5
PRINT("HOME MOTOR",#13)
VT=vv*rr           'Set Home Velocity
ADT=aa             'Set Home Accel/Decel
MV                'Set to Velocity mode
ZS                'Clear any prior Errors
T=tt*rr           'Preset Torque Values
G                 'Begin move towards Hard Stop
MT
WHILE ABS(EA)<ee LOOP           'Loop, While Position Error within limit
PRINT("HIT HARD STOP",#13)
G                               'Begin move
WAIT=50                         'Wait 50 milliseconds
O=hh*rr                         'Set Origin to home offset
PRINT("MOVING TO ZERO",#13)
MP PT=0 G TWAIT                 'Set Motor to Zero
RETURN
=====

```

Home Against a Hard Stop (Two Motors)

Because the SmartMotor has the capability of lowering its own power level and reading its position error, it can be programmed to gently feel for the end of travel. This provides a means to develop a consistent home position subsequent to each power-up.

Machine reliability requires the elimination of potential failure sources. Eliminating a home switch and its associated cable leverages SmartMotor benefits and improves machine reliability.

Similar to the previous examples, the next program lowers the current limit, moves against a limit, looks for resistance, and then declares and moves to a home position just two encoder revolutions from the hard stop. However, this example sets the home position for two parallel-axis motors with just one program — motor 1 is the controller; motor 2 is the follower.

NOTE: The optional CAN bus and cables are required for SmartMotors linked in parallel.

```

'Class 5 Home To Hard Stop Example
'Note: CAN BUS REQUIRED FOR MOTORS LINKED IN PARALLEL
ECHO 'ECHO on to allow auto addressing downstream
a=1 'set default variable for address 1
WAIT=2000 'wait for boot up time differences
PRINT (#128,"a=a+1",#13) 'each motor prints downstream a=a+1
WAIT=2000 'wait for response time variations
ADDR=a 'Set motor address
WAIT=2000
IF CADDR!=ADDR 'Verify Can Address
    CADDR=ADDR 'Set if not same as motor address
    Z:0 'reset all motors to make can address take effect
ENDIF
WAIT=2000
'Motor 1 will be running ALL code.
EIGN(2) EIGN(3) ZS 'Bypass Over Travel Limits
    'Set up parameters
        rr=-1 'Home Direction
        ii=300 'Home current limit
        vv=100000 'Home Speed
        aa=1000 'Home Accel
        ee=300 'Home Error Limit
        tt=3000 'Home Torque Limit
        hh=8000 'Home Offset
        ll=8000 'Maximum Differential motion between motors
        '(racking limit)
END
C7 'HOME TWO PARALLEL X AXIS MOTORS FROM MOTOR 1 ALONE
zzz=0 'Indicator for racking limit fault
O:0=0 'SET ORIGIN TO ZERO TO CHECK FOR RACKING
AMPS:0=ii 'limit AMPS
VT:0=vv*rr 'Set Home velocity
ADT:0=aa 'Set Home accel/decel
MV:0 'Velocity mode
G:0 'tell both to go
e=0 'Check for motor stopping
WHILE e!=3 'While both have not stopped, loop
    qq=EA:2
    IF ABS (qq)>ee MTB:2 e=e|1 ENDIF 'STOP follower
    IF ABS (EA)>ee MTB e=e|2 ENDIF 'stop controller
        ddd=PA-PA:2
    IF ABS (ddd)>ll 'If racking limit exceeded
        MTB:0 'Stop all motion
        zzz=-1 'Set racking limit variable
        RETURN 'Return out of there
    ENDIF 'Stop the other to prevent racking
LOOP
T:0=tt*rr 'Preset Torque Values
MT:0 'Switch to Torque mode (hold against stop)
WAIT=50 'Wait 50 milliseconds
o=hh*rr 'Calculate home position
O:0=o 'Set Origin to home offset

```

```

AMPS:0=1023           'Set AMPS back to default
MP:0                  'Set to Position mode
PT:0=0                'Set Target to Zero
G:0                   'tell both to go to zero
WHILE B(0,2):2==1 LOOP 'Wait for X follower move to complete
TWAIT                 'Wait for controller to get there
RETURN

```

Home to Index Using Different Modes

Each SmartMotor has an encoder with an index marker at one angle. This marker is useful for establishing a repeatable start-up (home) position.

The next example uses two different modes to home the motor:

- The first method (C1) does this by slowly moving the motor shaft past the index marker and decelerating to a stop, and then moving it back to align with the index marker.
- The second method (C2) does this by abruptly stopping and holding at the index marker.

```

EIGN(W,0,12)         'Assign Travel Limits as General Use Inputs
ZS                    'Clear Any Status Bits
END
'=====
C1                    'Home to index, Relative Position Mode
'Note: This method will go past and move back to index
MP                    'Set to Position mode
Ai(0)                 'Arm Index Capture Register
ADT=3000              'Set Accel/Decel
VT=200000             'Set Velocity
PRT=RES+100          'Set Relative Distance to just past one rev
G TWAIT               'Go, wait until move is complete
PT=I(0)              'Set Target Position to index location
G TWAIT               'Go, wait until move is complete
O=0                   'Set Position to Zero
RETURN
'=====
C2                    'Home to index, Velocity Mode to find it
'Note: This method does an abrupt stop and holds at index.
MV                    'Set to Velocity mode
Ai(0)                 'Arm Index Capture Register
ADT=3000              'Set Accel/Decel
VT=200000             'Set Velocity
G                      'Go
WHILE Bi(0)==0 LOOP  'Wait to see index
X
PT=I(0)              'Set Target Position to index location
MP                    'Switch to Position mode to hold at index mark
G TWAIT               'Go, wait until move is complete
O=0                   'Set Position to Zero
RETURN
'=====

```

Maintain Velocity During Analog Drift

This example causes the SmartMotor's velocity to track an analog input. Analog signals drift and dither, so a dead-band feature has been added to maintain a stable velocity when the operator is not changing the signal. There is also a wait feature to slow the speed of the loop.

```

EIGN (W,0)           'Disable hardware limits
KP=3020             'Increase stiffness from default
KD=10010           'Increase damping from default
F                  'Activate new tuning parameters
ADT=100            'Set maximum accel/decel
MV                 'Set to Velocity mode
d=10               'Analog dead band, 5000 = full scale
o=2500             'Offset to allow negative swings
m=40               'Multiplier for speed
w=10              'Time delay between reads
b=0                'Seed b
C10                'Label to create infinite loop
  a=INA (V1,3) -o  'Take analog 5 Volt FS reading
  x=a-b            'Set x to determine change in input
  IF x>d           'Check if change beyond dead band
    VT=b*m         'Multiplier for appropriate speed
    G              'Initiate new velocity
  ELSEIF x<-d     'Check if change beyond dead band
    VT=b*m         'Multiplier for appropriate speed
    G              'Initiate new velocity
  ENDIF           'End IF statement
  b=a             'Update b for prevention of hunting
  WAIT=w         'Pause before next read
GOTO10            'Loop back to label
END               'Obligatory END (never reached)

```

Long-Term Storage of Variables

Each SmartMotor is equipped with a kind of solid-state disk drive, called EEPROM, reserved just for long term data storage and retrieval. Data stored in the EEPROM will remain even after power cycling, just like the SmartMotor's program itself. However, the EEPROM has limitations. It cannot be written to more than about one million times without being damaged. That may seem like a lot, but if a write command (VST) is used in a fast loop, this number can be exceeded in a short time. Therefore, it is the responsibility of the programmer to see that the memory limitations are considered.

The next example is a subroutine to be called whenever a motion limit is reached. It assumes that the memory locations were pre-seeded with zeros.

NOTE: This example is a subroutine. It would be called with the command GOSUB10.

```

C10          'Subroutine label
EPTR=100    'Set EEPROM pointer in memory
VLD (aa,2)  'Load 2 long variables from EEPROM
IF Br       'If right limit, then...
  aa=aa+1   'Increment variable aa
  Zr        'Reset right limit state flag
ENDIF
IF Bl       'If left limit, then...
  bb=bb+1   'Increment variable bb
  Zl        'Reset left limit state flag
ENDIF
EPTR=100    'Reset EEPROM pointer in memory
VST (aa,2)  'Store variables aa and bb
RETURN      'Return to subroutine call

```

Find Errors and Print Them

This code example looks at different error status bits and prints the appropriate error information to the RS-232 channel.

NOTE: This example is a subroutine. It would be called with the command GOSUB10.

```

C10          'Subroutine label
IF Be       'Check for position error
  PRINT("Position Error",#13)
ENDIF
IF Bh       'Check for over temp error
  PRINT("Over Temp Error",#13)
ENDIF
IF Ba       'Check for over current error
  PRINT("Over Current Error",#13)
ENDIF
RETURN      'Return to subroutine call

```

Change Speed on Digital Input

SmartMotors have digital I/O that can be used for many purposes. In this example, a position move is started, and the speed is increased by 50% if input 0 goes low.

```
EIGN(W,0)           'Disable hardware limit IO
KD=10010            'Changing KD value in tuning
F                  'Accept new KD value
O=0                'Reset origin
ADT=100            'Set maximum accel/decel
VT=10000           'Set maximum velocity
PT=40000           'Set final position
MP                'Set Position mode
G                 'Start motion
WHILE Bt           'Loop while motion continues
  IF IN(0)==0      'If input is low
    IF VT==10000   'Check VT so change happens once
      VT=12000    'Set new velocity
      G           'Initiate new velocity
    ENDIF
  ENDIF
LOOP              'Loop back to WHILE
END
```

Pulse Output on a Given Position

It is often necessary to fire an output when a certain position is reached. There are many ways to do this with a SmartMotor.

This example sets I/O B as an output and makes sure that it comes up to 1 by presetting the output value. After that, the program monitors the encoder position until it exceeds 20000. It pulses the output and then continues monitoring the position.

```

EIGN(W,0)          'Disable limits
ZS
ITR(0,4,0,0,1)    'ITR(int#,sw,bit,state,lbl)
ITRE
EITR(0)
OUT(1)=1          'Set I(0)/O B to output, high
ADT=100           'Set maximum accel/decel
VT=100000        'Set maximum velocity
MP                'Set Position mode
'****Main Program Body****
WHILE 1>0
  O=0              'Reset origin for move
  PT=40000        'Set final position
  G                'Start motion
  WHILE PA<20000  'Loop while motion continues
  LOOP            'Wait for desired position to pass
  OUT(1)=0        'Set output low
  TMR(0,400)     'Use timer 0 for pulse width
  TWAIT
  WAIT=1000      'Wait 1 second
LOOP
END
'****Interrupt Subroutine****
C1
  OUT(1)=1        'Set output high again
RETURNI

```

Stop Motion if Voltage Drops

The voltage, current and temperature of a SmartMotor are always known. These values can be used within a program to react to changes.

In this example, the SmartMotor begins a move and then stops motion if the voltage falls below 18.5 volts.

```

EIGN(W,0)          'Disable hardware limits
ZS                 'Clear faults
MDS                'Sine mode commutation
ADT=100            'Set maximum accel/decel
VT=100000         'Set maximum velocity
PT=1000000        'Set final position
MP                'Set Position mode
G                 'Start motion
WHILE Bt           'Loop while motion continues
  IF UJA<18500     'If voltage is below 18.5 volts
  OFF              'Turn motor off
  ENDIF
LOOP               'Loop back to WHILE
END                'Obligatory END

```

Camming - Variable Cam Example

This example code shows the use of a variable cam (versus the "fixed" cam shown in the previous program example).

When considering use of variable cams, consider these points:

- Whether using variable or fixed-length segments, the base 'cam controller' of the last point is the significant part as far as executing a single cam cycle.
- For fixed, the programmer needs to multiply the number of segments by the segment length.
- For variable, it is in 'absolute' terms. However, the caveat is that the programmer should typically start with the first point as CTW(0,0) — see the next code example. Otherwise, the difference between that and the last point must be calculated in order to know the cam table base length.

```
CTE(1)      ' Erase all cams in flash
CTA(8,0)    ' create 8-point cam (7 segments/intervals). No segment
            ' length defined (variable length segments defined on a
            ' point-by-point basis.

CTW(0,0)    ' First cam point (best practice to set to 0,0 because all cam
            ' base and motor position relative to this).
CTW(100,8000) ' Motor position start +100, cam base start +8000
CTW(200,10000) ' Motor position start +200 (100 greater than previous
            ' point), cam base start + 10000 (relative 2000 greater
            ' than previous point).
CTW(300,14000) ' Peak motor position of 300.
CTW(300,24000)
CTW(200,28000) ' The second parameter (cam base) is always increasing.
            ' The difference from the previous value must be less
            ' than 65535.

CTW(100,30000)
CTW(0,32000)  ' Motor returns to position 0, cam base 32000.

MFSLEW(32000,1) ' Run one cycle of the cam: base 0 through 32000.

MCW(1,0)     ' Select cam table 1, point 0 as the start.
G            ' Start the cam relative to the present motor position.
```



CAUTION: When writing a cam table to EEPROM, structure the program so that the cam table is not frequently rewritten or written from a loop. Repeated erasing and rewriting can burn bits and corrupt data. For details and sample code, refer to Electronic Camming Notes and Best Practices on page 162.

Camming - Fixed Cam with Input Variables

This fixed-cam example uses variables for the values of various program inputs. This method allows quick operational changes versus having the values "hard coded".

```

EIGN(W,0)
ZS
ADDR=1
ECHO
a=0      ' MFA
b=0      ' MFD
m=1      ' keep as 1 for ramp output to match cam input units.
p=8      ' points
ss=3000  ' Segment length
s=p-1    ' segments
sss=ss*s ' total cam input (base) length.
k=sss-(a+b) ' Choose slew length from what is left between the ramp up/down.
IF k<0 PRINT("Ramps too long",#13) END
ENDIF
GOSUB1   ' Write cam table
GOSUB2   ' Run cam operation
END

C1      ' Write cam table one time
IF q==123 RETURN ENDIF
CTE(1)  ' Erase cam table in EEPROM
CTA(p,ss) ' Make sure the number of CTW commands = p.
CTW(0)  'CP=0 {cam pointer or cam index pointer}
CTW(100) 'CP=1
CTW(500) 'CP=2
CTW(2000) 'CP=3
CTW(2000) 'CP=4
CTW(5000) 'CP=5
CTW(1000) 'CP=6
CTW(0)    'CP=7
PRINT("Cam written",#13)
q=123
RETURN

C2
O=0
MFMUL=1
MFDIV=1
MCMUL=1
MCDIV=1
MCE(1)  ' Enable Cam mode
SRC(2)
MFSLEW(k,1) ' Get result in ramp output units to match the cam input units
MFSDC(-1,0)
MFA(a,m)  ' m=1 for ramp output to match cam input units.
MFD(b,m)  ' m=1 for ramp output to match cam input units.

```

MC

MCW(1,0) ' Required to select table.

G

RETURN



CAUTION: When writing a cam table to EEPROM, structure the program so that the cam table is not frequently rewritten or written from a loop. Repeated erasing and rewriting can burn bits and corrupt data. For details and sample code, refer to Electronic Camming Notes and Best Practices on page 162.

Camming - Demo XY Circle

This program makes use of the single-shot (no repeat) MFSDC mode to construct a circle pattern.

```

EIGN(W,0)
ADDR=CADDR
ECHO

'Make a circle. Issue GOSUB1 to write cam for a circle with:
rrr=8000      'Radius in encoder counts (restricted to 0-32767).
ttt=4000      'Time to complete a complete circle in milliseconds
              '(restricted to 0-32767).
ddd=360       'Degrees you want to run, signed value where positive is
              'counter-clockwise (restricted to +/-3239).
aaa=0         'Angle to start at (restricted to 0-359).
END

C5
    GOSUB(1) 'Write the cam.
    GOSUB(0) 'Run the cam.
RETURN

C0 'Run the cam.
    MC 'Cam mode.
    xx=aaa
    yy=(aaa-90)
    IF yy<0
        yy=yy+360 'Modulo sign correction.
    ENDIF

    IF CADDR==2 '*****
        'X axis
        MCW(1,xx) 'Table 1, starting point.
    ENDIF
    IF CADDR==3 '*****
        'Y axis
        MCW(1,yy) 'Table 1, starting point.
    ENDIF
    IF CADDR==1
        END
    ENDIF

    MCE(2) 'Cam table enable.
    SRC(2) 'Source set to virtual axis.

    MFSDC(-1,0) 'Single shot no repeat.

    ss=(ddd/360.0)*28800 '80 counts/controller*360 segments.
    MFMUL=ss/8          '8000/8=1000 gives 1 second time base.
    ss=ABS(ss)         'ABS because you may run reverse.

    sss=ss/2          'Slew will be 1/2 total time base.

```

```

MFSLEW(sss,1)
sss=ss/4      'Accel and decel will be 1/2 that, or 1/4 time base.
MFA(sss,1)
MFD(sss,1)

MFDIV=ttt*2/3  'Divide by 2/3 factor because velocity is not constant
               'throughout the move.

'Largest value in cam table is 32767.
'rrr is radius in counts.
'If it is 32767 or smaller, MCDIV=32767 and it is a straight ratio.
'If it is greater than that, then max MCDIV and ratio it the other way.
IF rrr>32767
    MCMUL=32767
    MCDIV=1073676289.0/rrr  '1073676289=32767^2
ELSE
    MCMUL=rrr
    MCDIV=32767
ENDIF

G TWAIT
ENDIF
RETURN

C1  ' Write the cam.
    WAIT=ADDR*500
    'This writes the sine into EE memory.
    'Normalized table that will be scaled in frequency and amplitude later.
    'Table is +/-32767 in amplitude.

    PRINT("Writing Tables.  Please wait.",#13)
    CTE(1)      'Erase all flash tables.
    CTA(361,80) 'Table will have 361 points, each is 80 ticks
               '(fixed-length data).

    iii=0
    WHILE iii<361
        ppp=32767*SIN(iii)
        CTW(ppp)      'Write a point into table
                     'Length of segment set by CTA command.
        'PRINT("Point: ",iii,", position: ",ppp,#13)
        iii=iii+1    'Update counter.
    LOOP
    PRINT("Done. Motor: ",ADDR,#13)

RETURN

```



CAUTION: When writing a cam table to EEPROM, structure the program so that the cam table is not frequently rewritten or written from a loop. Repeated erasing and rewriting can burn bits and corrupt data. For details and sample code, refer to Electronic Camming Notes and Best Practices on page 162.

Chevron Traverse & Takeup

This program uses four parameters to create a "Chevron" type wrap. For more information on this type of wrap, see Chevron Wrap Example on page 153.

It requires knowing:

- it requires knowing spool width and spool counts/rev, and
- that the cam must not get out of frame at each end.

Therefore, the traverse length must be an even multiple of the counts/rev. Otherwise, it will not work correctly.

Note that you can add a dwell at either end to tweak the frame of reference for the return path to prevent the material from falling into the grooves of the previous layer. That is the purpose of this wrapping method.

```
'Traverse & Takeup Chevron Winding Pattern
' This sample program will perform a traverse & takeup operation to produce
' a "chevron" winding pattern. By winding in a chevron pattern, each layer
' crosses diagonally over the previous layer's position. The purpose of this
' pattern is to prevent the wound material from getting caught in the cracks
' of the previous layer.

EIGN(W,0)
ZS

'System parameters:
' NOTE: To maintain relationship of spool controller to cam, w/c should
' evenly divide with no remainder.
c=5000 'Controller (External) Encoder resolution (counts per 360 deg
      ' turn of spool).
w=10000 'Spool width distance in encoder counts of traversing follower motor.

'Chevron shape (pitch and amplitude)
n=1000 'Follower counts per full (360 deg) turn of controller spool (pitch).
nn=1000 'Follower counts per half (180 deg) turn of controller spool (amp-
litude).

'Prevent overshoot at high end of the spool.
IF nn>n
    s=(w-(nn-n))/n*c 'calculating slew distance for MFSLEW
ELSE
    s=(w/n)*c
ENDIF

cc=c/2 'Calculate 180 deg turn of controller spool.
ITR(1,7,10,0,1)
EITR(1)
ITRE
PAUSE
END
```

```
C1
MCMUL=MCMUL*-1
RETURNI

C123      'Set up and run cam table.
O=0
MC
MFSDC(cc,1)

CTA(3,0,0) ' 3-point table, segment base defined with these points:
CTW(0,0)   ' Controller == 0
CTW(nn,cc) ' Controller == 4000
CTW(n,c)   ' Controller == 8000
MCW(0,0)

MCE(0)     ' Linear cam interpolation (straight lines like true chevron).
'MCE(1)    ' Spline cam interpolation (most likely never needed).
'MCE(2)    ' Spline cam interpolation periodic (sine wave, arc motion).

'Angle of the winding pattern is determined by the cam table points,
' not MFMUL and MFDIV.
MFMUL=1
MFDIV=1

'MCMUL and MCDIV affect cam table. Changing this will change angle of
' winding and chevron shape.
MCMUL=1
MCDIV=1

SRC(2)     'This is for demo. Change to SRC(1) for external encoder
           ' from controller spool.
MFA(0,1)   'No ascend into motion.
MFD(0,1)   'No descend out of motion.
MFSLEW(s,1) 'Slew distance is width of spool in follower motor counts.

G          'Start moving.
MFMUL=-1
RETURN
```

CAN Bus - Timed SDO Poll

This program makes use of one-shot SDO commands to get data while using SmartMotor timer interrupts to poll continuously. This method is used where high-speed polling is not required.

NOTE: For high-speed polling, a PDO is used to do this automatically. For details, see CAN Bus - I/O Block with PDO Poll on page 880.

'Using SmartMotor interrupts and timers to poll via SDO.

'Note PDO mapping is also available; this is just a simplified code example.

```

CANCTL(17,3)      'ENABLE CONTROLLER COMMANDS
ITR(3,4,3,0,300) 'WATCHDOG TIMER INTERRUPT (for polling CAN bus device)
EITR(3)          'ENABLE INTERRUPT 3
ITRE            'ENABLE ALL INTERRUPTS

TMR(3,30)        'Start timer 3 for 30 milliseconds

PAUSE
END
'=====

C300 'CAN bus device has two data packets that will be loaded into x and y.
    x=SDORD(1, 24592,0,2)  'Read 2 bytes from address 1,
                          'object 24592 (0x6179 hex)
                          'object 0x6010, sub-index 0.
    e=CAN(4)              'Trap error codes if any.

    y=SDORD(1, 24608,0,2)  'Read 2 bytes from address 1,
                          'object 24608 (0x6020 hex)
                          'object 0x6020, sub-index 0.
    ee=CAN(4)             'Trap error codes if any.

    IF (e|ee)==0          'Confirm the status of both SDO operations.
                          'Success if they are zero.
    'The variables "x" and "y" now contain values from the CAN bus device.
    ELSE
        'Place error handling code here.
    ENDIF
    TMR(3,30)            'Start timer 3 again for 30 milliseconds.
    'This means the data will be polled every 30 milliseconds.

RETURNI

```

CAN Bus - I/O Block with PDO Poll

This program communicates with a Softlink model RT133-3HF00-CAN I/O block. It uses high-speed PDO polling based on the default mapping found in the .eds file. Also, the follower detects the baud rate.

NOTE: This example uses high-speed PDO polling. For applications that do not require high-speed polling, see CAN Bus - Timed SDO Poll on page 879.

```
' NOTES:
'   - Using Softlink RT133-3HF00-CAN
'   - Using the default PDO mapping found in .eds file
'   - Controller address is 1, follower address is 15, follower detects baud
rate
ADDR=1
CADDR=1
CBAUD=125000
'RUN?
EIGN(W,0) ZS
SILENT

' Enable controller
CANCTL(17,3)

' Ensure pre-operational (not operational) state
NMT(0,128)

' Change analog output from +/-10V voltage(4) to 4-20mA(5)
SDOWR(15,8192,13,1,5)

' Disable transmit and receive PDO for controller to allow changing
' Set bit 31
a=-2147483648          ' 0x80000000
SDOWR(1,5120,1,4,a)   ' 0x1400 rx
SDOWR(1,6144,1,4,a)   ' 0x1800 tx
SDOWR(1,5121,1,4,a)   ' 0x1401 rx
SDOWR(1,6145,1,4,a)   ' 0x1801 tx

' Set mapping number of entries to 0
ab[0]=0
SDOWR(1,5632,0,1,ab[0]) ' 0x1600 rx
SDOWR(1,6656,0,1,ab[0]) ' 0x1a00 tx
SDOWR(1,5633,0,1,ab[0]) ' 0x1601 rx
SDOWR(1,6657,0,1,ab[0]) ' 0x1a01 tx

' Set mapping objects
' ab[2] 0x2220 03 08
a=572523272
SDOWR(1,5632,1,4,a)    ' 0x1600 rx
' ab[3] 0x2220 04 08
a=572523528
SDOWR(1,6656,1,4,a)    ' 0x1a00 tx
```



```

' aw[32] 0x2221 01 10      ' aw[32] is 1st analog voltage input
a=572588304
SDOWR(1,5633,1,4,a)      ' 0x1601 rx
' aw[33] 0x2221 02 10      ' aw[33] is 1st 4-20mA analog output
a=572588560
SDOWR(1,6657,1,4,a)      ' 0x1a01 tx

' Set mapping number of entries to 1
ab[0]=1
SDOWR(1,5632,0,1,ab[0])  ' 0x1600 rx
SDOWR(1,6656,0,1,ab[0])  ' 0x1a00 tx
SDOWR(1,5633,0,1,ab[0])  ' 0x1601 rx
SDOWR(1,6657,0,1,ab[0])  ' 0x1a01 tx

' Enable transmit and receive PDO for controller to allow changing
' Clear bit 31
a=399                    ' 0x0000018f (f = address 15)
SDOWR(1,5120,1,4,a)      ' 0x1400 rx mapped to follower tx

a=527                    ' 0x0000020f (f = address 15)
SDOWR(1,6144,1,4,a)      ' 0x1800 tx mapped to follower rx

a=655                    ' 0x0000028f (f = address 15)
SDOWR(1,5121,1,4,a)      ' 0x1401 rx mapped to follower tx

a=783                    ' 0x0000030f (f = address 15)
SDOWR(1,6145,1,4,a)      ' 0x1801 tx mapped to follower rx

' Tell everyone to go operational
NMT(0,1)

b=1
WHILE 1
    IF B(10,1)==1
        Z(10,1) ' Clear event flag
        PRINT("Rx PDO 1",#13)
    ENDIF
    IF B(10,2)==1
        Z(10,2) ' Clear event flag
        PRINT("Rx PDO 2",#13)
    ENDIF
    IF B(10,3)==1
        Z(10,3) ' Clear event flag
        PRINT("Rx PDO 3",#13)
    ENDIF
    IF B(10,4)==1
        Z(10,4) ' Clear event flag
        PRINT("Rx PDO 4",#13)
    ENDIF

```

```
IF B(10,5)==1
    Z(10,5) ' Clear event flag
    PRINT("Rx PDO 5",#13)
ENDIF
' Set the User Bits in Status Word 12 to reflect the status of the 8 inputs
UO(W,0)=ab[2]&255
' Turn on outputs (continuous count up)
ab[3]=ab[3]+(1*b)
' Set the User Bits in Status Word 13 to reflect the status of the 8 outputs
UO(W,1)=ab[3]&254
WAIT=100
LOOP
END
```

CAN Bus - Time Sync Follow Encoder

This program makes use of CANopen objects to provide following of a CANopen encoder on a CANopen network. For the purposes of an example, one SmartMotor acts as a "controller", and a second SmartMotor can act as the encoder if a CANopen encoder is not available.

```
' Demo with one motor following a CANopen encoder on CANopen network.
' Motor 1 is a controller as far as NMT and SDOs, but will follow data from
' encoder. Motor 2 will act as an encoder, (the PDO mapping to position actual
' instead of an encoder's position object.
' Load this program into both motors.

'++++ HEX Coded Objects for CAN +++++
#define x1000 4096 ' Object 1000h: Device Type
#define x1005 4101 ' Object 1005h: COB-ID Sync
#define x1006 4102 ' Object 1006h: Communication Cycle Period
#define x1400 5120 ' Object 1400h: Receive PDO Communication Parameter
#define x1600 5632 ' Object 1600h: Receive PDO Mapping Parameter 1
#define x1800 6144 ' Object 1800h: Transmit PDO communicating parameter 1
#define x1801 6145 ' Object 1801h: Transmit PDO communicating parameter 2
#define x1A00 6656 ' Object 1A00h: Transmit PDO Mapping Parameter 1
#define x1A01 6657 ' Object 1A01h: Transmit PDO Mapping Parameter 2
#define x2204 8708 ' Object 2204h: Mappable Variables aaa...ddd)
#define x2207 8711 ' Object 2207h: External encoder follow max value (where
' encoder rolls over) i.e., 10-bit encoder would be 1023

#define x2208 8712 ' Object 2208h: External encoder follow input value
#define x2209 8713 ' Object 2209h: External encoder follow control
#define x220A 8714 ' Object 220Ah: External encoder follow MFMUL
#define x220B 8715 ' Object 220Bh: External encoder follow MFDIV
#define x220C 8716 ' Object 220Ch: External encoder follow MFA
#define x220D 8717 ' Object 220Dh: External encoder follow MFD
#define x2304 8964 ' Object 2304h: Motor Status
#define x6002 24578 ' Object 6002h: (encoder profile) Total measuring range
#define x6040 24640 ' Object 6040h: Control word
#define x6060 24672 ' Object 6060h: Trajectory Mode
#define x6064 24676 ' Object 6064h: Position actual value (RPA)
#define x606C 24684 ' Object 606Ch: Velocity actual value (RVA)
#define x608F 24719 ' Object 608Fh: Position encoder resolution
#define x60F4 24820 ' Object 60F4h: Position Error actual value (REA)

' misc values:
#define xffffffff -1

OFF
ADDR=CADDR ' Set serial channel address so you don't have to re-
' address/detect motors.
ECHO ' Enable serial channel echo so you don't have to re-
' address/detect motors.

mmm=1 ' network controller's address
fff=mmm ' The following motor's address. In this demo, it is the network
```

```

eee=2          ' controller, but you can use a 3rd-party encoder's address.
               ' The encoder's address.

EIGN(W,0)     ' Turn off hardware limits!!!!
ZS            ' Clear faults.
O=0

WAIT=100
a=ADDR+128
PRINT(#a)     ' Send our own address char to make sure downstream motors
               ' don't interpret program output as commands.

IF CADDR==mmm
  ' We are the controller, take over the show.

CANCTL(17,3)  ' Enable SDO and NMT commands.

PRINT("Checking for encoder ready: ",#13)
t=0
WHILE 1
  vvv=SDORD(eee,x1000,0,4)  ' Read device type from encoder.
  IF CAN(4)==0  ' Check error for a successful read. A timeout is very
                ' likely until remote device is ready.
  IF (vvv&65535)==406  ' Check lower 16-bits for profile type of device.
    PRINT("Found encoder",#13)
    SWITCH vvv/65536  ' We want to look at the upper 16-bits.
      CASE 1
        PRINT("Single-turn absolute rotary encoder.",#13)
        t=1
        BREAK
      CASE 2
        PRINT("Multi-turn absolute rotary encoder.",#13)
        t=1
        BREAK
      CASE 3
        PRINT("Single-turn absolute rotary encoder ")
        PRINT("with electronic turn-count.",#13)
        t=1
        BREAK
      CASE 4
        PRINT("Incremental rotary encoder.",#13)
        t=1
        BREAK
      CASE 5
        PRINT("Incremental rotary encoder with electronic counting.",#13)
        t=1
        BREAK
      CASE 6
        PRINT("Incremental linear encoder.",#13)
        t=1
        BREAK
      CASE 7

```

```

    PRINT("Incremental linear encoder with electronic counting.",#13)
    t=1
    BREAK
CASE 8
    PRINT("Absolute linear encoder.",#13)
    t=1
    BREAK
CASE 9
    PRINT("Absolute linear encoder with cyclic coding.",#13)
    t=1
    BREAK
CASE 10
    PRINT("Multi-sensor encoder interface.",#13)
    t=0 ' not supported here at this time.
    BREAK
DEFAULT
    PRINT("Unknown type of encoder",#13)
    BREAK
ENDS
BREAK ' Remote device has booted to the point of responding.
ENDIF
IF (vvv&65535)==402 ' Check lower 16-bits for profile type of device.
    PRINT("Motor ",eee," acting as an encoder.",#13)
    ' A motor acting as an encoder.
    t=2
    BREAK
ENDIF
ENDIF
WAIT=500 ' Wait 1/2 second.
PRINT(".")
LOOP
PRINT(#13)

rrr=0 ' Resolution
IF t==1
    rrr=SDORD(eee,x6002,0,4) GOSUB10
    rrr=rrr-1 ' We want the max value, not the number of steps.
    IF CAN(4)!=0
        PRINT("Failed to read encoder range.",#13)
        OFF END
    ENDIF
ENDIF
ENDIF
IF t==2
    rrr=-1 ' SmartMotor, which is the range 0x00000000 to 0xffffffff.
ENDIF
IF t==0
    PRINT("This type not supported, ending now.",#13)
    OFF
    END
ENDIF

NMT(0,128) GOSUB10 ' Network broadcast to go pre-operational state.
' Setup the sync producer/consumers and set time base. Provides time sync so

```

```

' motor clocks keep in step, and data is transmitted/accepted on sync also.
SDWR (mmm,x1006,0,4,10000)  GOSUB10  ' define Cycle period  object 0x1006:0,
                                ' size 4, 10ms
SDWR (eee,x1006,0,4,10000)  GOSUB10  ' define Cycle period  object 0x1006:0,
                                ' size 4, 10ms

IF mmm!=fff
' If follow motor is not the controller.
  SDWR (fff,x1006,0,4,10000)  GOSUB10  ' define Cycle period  object
                                ' 0x1006:0, size 4, 10ms

ENDIF

SDWR (mmm,x1005,0,4,128)    GOSUB10  ' define Cycle ID x0000 0080 (required
                                ' to avoid error in next line.)
SDWR (mmm,x1005,0,4,1073741952) GOSUB10  ' define Cycle ID, producer
                                ' x4000 0080
SDWR (eee,x1005,0,4,128)    GOSUB10  ' define Cycle ID, consumer x0000 0080
IF mmm!=fff
  ' If follow motor is not the controller.
  SDWR (fff,x1005,0,4,128)  GOSUB10  ' define Cycle ID, consumer x0000 0080
ENDIF

' Time sync

' Setup PDO mapping so that controller transmits 32-bit actual position
' (0x6064), and follow motor receives as 32-bit encoder data (0x2208:3)
IF t==2
  ' We need to map a motor completely to get the position data.
  SDWR (eee,x1800,1,4,-1073741439) GOSUB10  ' set Transmit Com Parameter
                                ' START xC000 0181
  SDWR (eee,x1800,2,1,1)           GOSUB10  ' set Transmit Com Parameter
                                ' on every Sync x000 0001
  SDWR (eee,x1A00,0,1,0)           GOSUB10  ' set Transmit map number of
                                ' entries x00
  SDWR (eee,x1A00,1,4,1617166368) GOSUB10  ' set Transmit map x6064 00 20
  SDWR (eee,x1A00,0,1,1)           GOSUB10  ' set Transmit map number of
                                ' entries x01
  SDWR (eee,x1800,1,4,1073742209) GOSUB10  ' set Transmit Com Parameter
                                ' COB-ID x4000 0181

ELSEIF t==1
' Using an encoder. An encoder should have these mappings already.
' Comment these out if the encoder has read-only PDO parameters/mappings.
' Note PDO2 is used; it is default sync type in the encoder 406 profile.
  SDWR (eee,x1801,1,4,-1073741439) GOSUB10  ' set Transmit Com Parameter
                                ' START xC000 0181
  SDWR (eee,x1801,2,1,1)           GOSUB10  ' set Transmit Com Parameter
                                ' on every Sync x000 0001
  SDWR (eee,x1A01,0,1,0)           GOSUB10  ' set Transmit map number of
                                ' entries x00
  SDWR (eee,x1A01,1,4,1617166368) GOSUB10  ' set Transmit map x6064 00 20
  SDWR (eee,x1A01,0,1,1)           GOSUB10  ' set Transmit map number of
                                ' entries x01

```

```

SDOWR(eee,x1801,1,4,1073742209)  GOSUB10 ' set Transmit Com Parameter
                                     ' COB-ID x4000 0181

ENDIF
SDOWR(fff,x1400,1,4,-2147483263)  GOSUB10 ' set Receive Com Parameter start
                                     ' x8000 0181
SDOWR(fff,x1600,0,1,0)             GOSUB10 ' set Receive Map number of
                                     ' entries x00
SDOWR(fff,x1600,1,4,570950432)    GOSUB10 ' set Receive Map x2208 03 20
SDOWR(fff,x1600,0,1,1)             GOSUB10 ' set Receive Map number of
                                     ' entries x01
SDOWR(fff,x1400,1,4,385)           GOSUB10 ' set Receive Com Parameter
                                     ' COB-ID x0000 0181

' Mapping complete.

' Set other objects in follow motor relating to Follow mode.
SDOWR(fff,x2207,0,4,rrr)           GOSUB10 ' set encoder modulo limit
SDOWR(fff,x2209,0,2,0)             GOSUB10 ' set follow control to nominal state.
SDOWR(fff,x220A,0,2,100)           GOSUB10 ' set MFMUL
SDOWR(fff,x220B,0,2,100)           GOSUB10 ' set MFDIV
SDOWR(fff,x220C,0,4,20000)         GOSUB10 ' set MFA control word x2209
                                     ' determines if controller or follower units.
SDOWR(fff,x220D,0,4,10000)         GOSUB10 ' set MFD control word x2209
                                     ' determines if controller or follower units.

NMT(0,1)                            GOSUB10 ' Broadcast to whole network to go
                                     ' to operational state.

SDOWR(fff,x6060,0,1,-11)           GOSUB10 ' set Trajectory Mode to Mixed FOLLOW
SDOWR(fff,x6040,0,2,6)             GOSUB10 ' set Control word in Motor 2
SDOWR(fff,x6040,0,2,7)             GOSUB10 ' set Control word in Motor 2
SDOWR(fff,x6040,0,2,15)            GOSUB10 ' set Control word in Motor 2
PRINT("Setup complete",#13)

IF CADDR==fff
' If the network controller happens to be the follow motor (which is
' expected in this demo by default).
WAIT=3000
PRINT("Follow motor: stopping X.",#13)
X(2) ' Stop following
WAIT=1000
PRINT("Follow motor: re-starting X.",#13)
G(2) ' Restart following
ELSE
' Some other motor is the follow motor.
' ...
ENDIF

ELSEIF CADDR==eee
' We are a motor pretending to be the encoder.
WAIT=500
PRINT("SmartMotor acting as encoder: starting motion.",#13)
MV VT=100000 ADT=50 G ' A simple constant motion.

```

```

    ' Follow motor is already started and will follow exactly.
PRINT("SmartMotor acting as encoder: ") PRINT("running for 10 seconds.",#13)
WAIT=10000
PRINT("SmartMotor acting as encoder: freewheel.",#13)
BRKRLS OFF
ENDIF
END

C10 ' Check for CAN error and display
e=CAN(4)
IF e!=0
    PRINT(#13,"Communication Error: ")
ELSE
    ' PRINT("ok",#13) ' For debugging, to see what commands passed and failed.
    RETURN
ENDIF
SWITCH e
' NOTE: Any error number < 0 is Animatics specific to the SDORD, SDOWR,
' and NMT commands.
CASE -1
    PRINT(" Timeout, no response from remote device.")
BREAK
CASE -2
    PRINT(" Multiple SDO commands simultaneously.")
BREAK
CASE -3
    PRINT(" Controller mode not enabled, see CANCTL command.")
BREAK
CASE -4
    PRINT(" Protocol not supported.")
BREAK
CASE -5
    PRINT(" Transmission failure, check cable, Baud rate.")
BREAK
CASE -6
    PRINT(" Data size returned not expected size.")
BREAK
CASE -20
    PRINT(" Invalid destination address.")
BREAK
CASE -21
    PRINT(" Data size not supported.")
BREAK
CASE -22
    PRINT(" Invalid object index.")
BREAK
CASE -23
    PRINT(" Invalid object sub-index.")
BREAK
CASE -24
    PRINT(" Invalid NMT command value.")
BREAK
' NOTE: any error number > 0 is specific to CANopen's specific list of

```



```
' errors per 301 specification.
CASE 84082688
    PRINT(" Toggle bit not alternated.")
BREAK
CASE 84148224
    PRINT(" SDO protocol timed out.")
BREAK
CASE 84148225
    PRINT(" Client/server command specifier not valid or unknown.")
BREAK
CASE 84148226
    PRINT(" Invalid block size (block mode only).")
BREAK
CASE 84148227
    PRINT(" Invalid sequence number (block mode only).")
BREAK
CASE 84148228
    PRINT(" CRC error (block mode only).")
BREAK
CASE 84148229
    PRINT(" Out of memory.")
BREAK
CASE 100728832
    PRINT(" Unsupported access to an object.")
BREAK
CASE 100728833
    PRINT(" Attempt to read a write only object.")
BREAK
CASE 100728834
    PRINT(" Attempt to write a read only object.")
BREAK
CASE 100794368
    PRINT(" Object does not exist in the object dictionary.")
BREAK
CASE 100925505
    PRINT(" Object cannot be mapped to the PDO.")
BREAK
CASE 100925506
    PRINT(" Exceeds PDO length.")
BREAK
CASE 100925507
    PRINT(" General parameter incompatibility reason.")
BREAK
CASE 100925511
    PRINT(" General internal incompatibility in the device.")
BREAK
CASE 101056512
    PRINT(" Access failed due to an hardware error.")
BREAK
CASE 101122064
    PRINT(" Data type mismatch, length of service parameter.")
BREAK
CASE 101122066
```

```

    PRINT(" Data type mismatch, length of service parameter, high.")
BREAK
CASE 101122067
    PRINT(" Data type mismatch, length of service parameter, low.")
BREAK
CASE 101253137
    PRINT(" Sub-index does not exist.")
BREAK
CASE 101253168
    PRINT(" Value range of parameter exceeded (write access).")
BREAK
CASE 101253169
    PRINT(" Value of parameter written too high.")
BREAK
CASE 101253170
    PRINT(" Value of parameter written too low.")
BREAK
CASE 101253174
    PRINT(" Maximum value is less than minimum value.")
BREAK
CASE 134217728
    PRINT(" General error.  (CANopen)")
BREAK
CASE 134217760
    PRINT(" Data can't be sent to the application.")
BREAK
CASE 134217761
    PRINT(" Data can't be sent to the application, local control.")
BREAK
CASE 134217762
    PRINT(" Data can't be sent to the application, device state.")
BREAK
CASE 134217763
    PRINT(" Object dictionary dynamic generation fails.")
BREAK

DEFAULT
    PRINT(" Unknown error.")
    IF e>0
        ' One of the CANopen errors
        PRINT(" Consult CANopen error list.")
    ' ELSE
        ' One of our error, but not included in the list above.
    ENDIF
BREAK
ENDS
PRINT(#13)
RETURN

```

Text Replacement in an SMI Program

This example shows possible uses of the `#define` command. That command is used by SMI to simply replace characters of code with an associated named value.

NOTE: `#define` is not an executable command; it is used for compile-time translation from SMI.

NOTE: Uploads of programs with define statements will show the alternate or replaced text only.

```
'Examples of using #define for text replacement in a program.
'NOTE: #define is used by SMI to replace characters of code with an
'associated named value.
#define DisableTravelLimits  EIGN(W,12)      'Assign inputs 2 and 3
                                     'as general inputs.
#define ClearFaultBits      ZS              'Issue ZS command.
#define GoInput              1-IN(6)        'Invert input 6.
#define 4BitBinaryInputs    15-(IN(W,0)&15) 'Binary mask inputs 0
                                     'through 3.
#define FindHome             GOSUB(100)
#define GearOutputPosition  RES*40.0        'Suppose 40:1 gear reducer.
#define NormalSpeed         500000
#define JogSpeed             10000
#define PositionError       B(0,6)         'Position Error Status Bit.
```

'The code below shows examples of using the above "define"
'variables.

```
DisableTravelLimits      'Disable travel limits.
ClearFaultBits           'Issue ZS command.
IF GoInput                'If Input 6 goes low - note that it was
                           'defined as 1-IN(6).
    PRINT("Go Received",#13)
    x=4BitBinaryInputs    'Returns value from 0 to 15 for inputs
                           '0, 1, 2 and 3.
    GOSUB(x)
ENDIF

FindHome                  'Run the home routine.

VT=NormalSpeed           'Set the speed to 500000.

PRT=22.5*GearOutputPosition 'Results in 22.5 rotations of gear head
                           'output shaft.

VT=JogSpeed              'Set the speed to 10000.

END

C100                      'Place the home routine code here
    PRINT("Home Routine Called",#13)
RETURN

C0
```

```
C1  
C2  
C3  
C4  
C5  
C6  
C7  
C8  
C9  
C10  
C11  
C12  
C13  
C14  
C15  
PRINT("Subroutine",x," called",#13)  
RETURN
```

Appendix

This appendix provides related information for use with the SmartMotor. With the exception of the Motion Command Quick Reference on page 895, all other topics are listed in alphabetical order.

Motion Command Quick Reference	895
Array Variable Memory Map	897
ASCII Character Set	899
Binary Data	900
Commands Affected by SCALE	903
Command Error Codes	906
Decoding the Error	906
Finding the Error Source	907
Glossary	908
Math Operators	915
Moment of Inertia	916
Matching Motor to Load	916
Improving the Moment of Inertia Ratio	916
RCAN, RCHN and RMODE Status	917
RCAN Status Decoder	917
RCHN Status Decoder	917
Clearing Serial Port Errors	918
RMODE Status Decoder	918
Mode Status Example	918
Scale Factor Calculation	919
Sample Rates	919
PID Sample Rate Command	919
Encoder Resolution and the RES Parameter	919
Native Velocity and Acceleration Units	920
Velocity Calculations	920
Acceleration Calculations	920
Status Words - SmartMotor	921
Status Word 0: Primary Fault/Status Indicator	921
Status Word 1: Index Registration and Software Travel Limits	922
Status Word 2: Communications, Program and Memory	922

Status Word 3: PID State, Brake, Move Generation Indicators	923
Status Word 4: Interrupt Timers	923
Status Word 5: Interrupt Status Indicators	924
Status Word 6: Drive Modes	924
Status Word 7: Multiple Trajectory Support	925
Status Word 8: Cam Support	926
Status Word 9: No Bits Defined (Class 5 Only)	926
Status Word 9: SD Card and DMX Information (Class 6 Only)	926
Status Word 10: RxPDO Arrival Notification	927
Status Word 12: DMX Information (Class 5 Only)	928
Status Word 12: User Bits Word 0 (Class 6 Only)	928
Status Word 13: User Bits Word 1	929
Status Word 16: On Board Local I/O Status: D-Style Motor	929
Status Word 16: On Board Local I/O Status: M-Style Class 5 Motor	930
Status Word 16: On Board Local I/O Status - Class 6 Motor	930
Status Word 17: Expanded I/O Status - D-Style AD1 Motor	931
Fault and Status Words - DS2020 Combitronic System	932
Fault Words	932
Fault Tables	932
Status Words	934
Torque Curves	938
Understanding Torque Curves	938
Peak Torque	938
Continuous Torque	938
Ambient Temperature Effects on Torque Curves and Motor Response:	939
Supply Voltage Effects on Torque Curves and Motor Response:	939
Example 1: Rotary Application	940
Example 2: Linear Application	940
Dyno Test Data vs. the Derated Torque Curve	940
Proper Sizing and Loading of the SmartMotor	941
SmartMotor Troubleshooting	943
Troubleshooting - First Steps	943

Motion Command Quick Reference

The next table provides a quick reference for the primary Class 5 motion commands. For the complete list of motion control commands and links to their descriptions, see Motion Control on page 957.

		Absolute Position	Relative Position	Velocity	Accel and Decel Together	Accel	Decel	Following Error	DE/Dt Derivative Error Limit	Over Speed Limit
Report	Actual	RPA	RPRA	RVA	N/A	N/A		REA	RDEA	
Report	End Target	RPT	RPRT	RVT	RAT	RAT	RDT	REL	RDEL	RVL
Report	Commanded	RPC	RPRC	RVC	RAC	RAC				
Assign	End Target	PT=	PRT=	VT=	ADT=	AT=	DT=			
Assign	Command	N/A	N/A	N/A	N/A	N/A	N/A	EL=	DEL=	VL=

In the chart above, you will notice Actual, End Target, and Commanded:

- Actual: The value of the parameter as the processor sees it in real time at the shaft, regardless of anything commanded by the trajectory generator
- Target: The requested trajectory target to reach and/or maintain at any given time
- Commanded: The compensated value of the trajectory generator at any time in its attempt to reach the target

For example, in terms of the position commands:

- Position Target (PT): The desired target position you are shooting for; what you have specified as a target position value
- Position Actual (PA): The current position in real time (right now), regardless of target or where it is being told to go
- Position Commanded (PC): The position the controller processor is actually commanding it to go to at the time

NOTE: Any difference between Position Commanded (PC) and Position Actual (PA) is due to position error.

There are two position types:

- Absolute: The finite position value in reference to position zero
- Relative: A relative distance from the present position at the time

All commands shown above are associated with both Mode Position (MP) and Mode Velocity (MV). They may also be used in dual trajectory mode when running either of those modes on top of gearing or camming.

All distance parameters are in encoder counts. Encoder resolution may be obtained and used in a program through the RES command. The RRES command will report encoder resolution. You can also use the RES command directly in math formulas.

EXAMPLE:

If you want it the axis to move to location 1234, then you would issue:

```
PT=1234
```

While moving there:

- RPC would report the commanded position from the processor.
- RPA would report actual position of the encoder or motor shaft.
- $x=PC-PA$ would calculate position error at that moment.
- REA would report actual position error at that moment.
- RBt would report a 1 (while moving) because the trajectory is active.

After the move has completed, RBt would report a 0 (to indicate the trajectory is no longer active).

Array Variable Memory Map

Integer Array Memory:

NOTE: Overlapping memory `aw[0]` is the least significant word of `a[0]`; likewise, `ab[0]` is the least significant byte of `aw[0]` and `a[0]`.

Long (32-bit signed)		Word (16-bit signed)			Bytes (8-bit signed)			
a[n] where n is:		aw[n] where n is:			ab[n] where n is:			
		LSb	MSb		LSb	middle bytes	MSb	
0		0	1		0	1	2	3
1		2	3		4	5	6	7
2		4	5		8	9	10	11
3		6	7		12	13	14	15
4		8	9		16	17	18	19
5		10	11		20	21	22	23
6		12	13		24	25	26	27
7		14	15		28	29	30	31
8		16	17		32	33	34	35
9		18	19		36	37	38	39
10		20	21		40	41	42	43
11		22	23		44	45	46	47
12		24	25		48	49	50	51
13		26	27		52	53	54	55
14		28	29		56	57	58	59
15		30	31		60	61	62	63
16		32	33		64	65	66	67
17		34	35		68	69	70	71
18		36	37		72	73	74	75
19		38	39		76	77	78	79
20		40	41		80	81	82	83
21		42	43		84	85	86	87
22		44	45		88	89	90	91
23		46	47		92	93	94	95
24		48	49		96	97	98	99
25		50	51		100	101	102	103
26		52	53		104	105	106	107
27		54	55		108	109	110	111
28		56	57		112	113	114	115
29		58	59		116	117	118	119
30		60	61		120	121	122	123
31		62	63		124	125	126	127
32		64	65		128	129	130	131
33		66	67		132	133	134	135
34		68	69		136	137	138	139

Long (32-bit signed)		Word (16-bit signed)			Bytes (8-bit signed)			
al[n] where n is:		aw[n] where n is:			ab[n] where n is:			
		LSb	MSb		LSb	middle bytes	MSb	
35		70	71		140	141	142	143
36		72	73		144	145	146	147
37		74	75		148	149	150	151
38		76	77		152	153	154	155
39		78	79		156	157	158	159
40		80	81		160	161	162	163
41		82	83		164	165	166	167
42		84	85		168	169	170	171
43		86	87		172	173	174	175
44		88	89		176	177	178	179
45		90	91		180	181	182	183
46		92	93		184	185	186	187
47		94	95		188	189	190	191
48		96	97		192	193	194	195
49		98	99		196	197	198	199
50		100	101		200	201	202	203
Overlapping is "little endian" for byte and word order								

Integer Variable Memory Non-Overlapping:

Name	Quantity	Type
a-z	26	32-bit signed
aa-zz	26	32-bit signed
aaa-zzz	26	32-bit signed
78 total letter variables		

Float Variable Memory:

Name	Quantity	Type
af[0]-af[7]	8	64 bit IEEE-754
8 total floating-point variables		

ASCII Character Set

ASCII is an acronym for American Standard Code for Information Interchange. It refers to the convention established to relate characters, symbols and functions to binary data. If a SmartMotor is asked its position over the RS-232 connection, and it is at position 1, it will not return a byte of value one, but instead will return the ASCII code for 1 which is binary value 49. That is why it appears on a Terminal window as the numeral 1. The ASCII character set is shown in the next table (Dec=decimal, Hex=hexadecimal).

Dec	Hex	ASCII	Dec	Hex	ASCII	Dec	Hex	ASCII	Dec	Hex	ASCII
0	00	NUL	35	23	#	70	46	F	105	69	i
1	01	SOH	36	24	\$	71	47	G	106	6A	j
2	02	STX	37	25	%	72	48	H	107	6B	k
3	03	ETX	38	26	&	73	49	I	108	6C	l
4	04	EOT	39	27	'	74	4A	J	109	6D	m
5	05	ENQ	40	28	(75	4B	K	110	6E	n
6	06	ACK	41	29)	76	4C	L	111	6F	o
7	07	BEL	42	2A	*	77	4D	M	112	70	p
8	08	BS	43	2B	+	78	4E	N	113	71	q
9	09	HT	44	2C	,	79	4F	O	114	72	r
10	0A	LF	45	2D	-	80	50	P	115	73	s
11	0B	VT	46	2E	.	81	51	Q	116	74	t
12	0C	FF	47	2F	/	82	52	R	117	75	u
13	0D	CR	48	30	0	83	53	S	118	76	v
14	0E	SO	49	31	1	84	54	T	119	77	w
15	0F	SI	50	32	2	85	55	U	120	78	x
16	10	DLE	51	33	3	86	56	V	121	79	y
17	11	DC1	52	34	4	87	57	W	122	7A	z
18	12	DC2	53	35	5	88	58	X	123	7B	{
19	13	DC3	54	36	6	89	59	Y	124	7C	
20	14	DC4	55	37	7	90	5A	Z	125	7D	}
21	15	NAK	56	38	8	91	5B	[126	7E	~
22	16	SYN	57	39	9	92	5C	\	127	7F	Del
23	17	ETB	58	3A	:	93	5D]			
24	18	CAN	59	3B	;	94	5E	^			
25	19	EM	60	3C	<	95	5F	_			
26	1A	SUB	61	3D	=	96	60	'			
27	1B	ESC	62	3E	>	97	61	a			
28	1C	FC	63	3F	?	98	62	b			
29	1D	GS	64	40	@	99	63	c			
30	1E	RS	65	41	A	100	64	d			
31	1F	US	66	42	B	101	65	e			
32	20	SP	67	43	C	102	66	f			
33	21	!	68	44	D	103	67	g			
34	22	"	69	45	E	104	68	h			

Binary Data

The SmartMotor language allows the programmer to access data at the binary level. Understanding binary data is useful when programming the SmartMotor or any electronic device. The section provides an explanation of how binary data works.

All digital computer data is stored as binary information. A binary element is one that has only two states, commonly described as "on" and "off" or "one" and "zero." A light switch is a binary element. It can either be "on" or "off." A computer's memory is nothing but a vast array of binary switches called "bits".

The power of a computer comes from the speed and sophistication with which it manipulates these bits to accomplish higher tasks. The first step towards these higher goals is to organize these bits in such a way that they can describe things more complicated than "off" or "on."

Different quantities of bits are used to make up the building blocks of data. They are most commonly described as:

Four bits = Nibble
 Eight bits = Byte
 Sixteen bits = Word
 Thirty two bits = Long

One bit has two possible states, on or off. Every time a bit is added, the possible number of states is doubled. Two bits have four possible states:

00 off-off
 01 off-on
 10 on-off
 11 on-on

A nibble has 16 possible states. A byte has 256, a word has 65536 and a long has billions of possible combinations.

Because a byte of information has 256 possible states, it can reflect a number from zero to 255. This is done by assigning each bit a value of twice the one before it, starting with one. Each bit value becomes:

Bit	Value
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

If all their values are added together the result is 255. By leaving particular bits out, any sum between zero and 255 can be created. Look at the next example bytes and their decimal values:

Byte	Value
0 0 0 0 0 0 0 0	0
0 0 0 0 0 0 0 1	1
0 0 0 0 0 0 1 0	2
0 0 0 0 0 0 1 1	3
0 0 0 1 0 0 0 0	16
0 0 0 1 1 1 1 0	30
0 0 1 1 1 1 0 0	60
1 0 0 0 0 0 0 0	128
1 0 0 1 1 1 0 1	157
1 1 1 1 1 1 1 1	255

To make use of the limited memory available with micro controllers that can fit into a SmartMotor, there are occasions where every bit is used. One example is Status Word 0. A single value can be uploaded from a SmartMotor and be binary coded with eight, sixteen or thirty-two independent bits of information. The next table shows Status Word 0 and its 16 bits of coded information:

Name	Description	Bit	Value
	Drive ready	0	1
Bo	Motor OFF	1	2
Bt	Trajectory in progress	2	4
	Bus voltage fault	3	8
Ba	Over current	4	16
Bh	Excessive temperature fault	5	32
Be	Excessive position error fault	6	64
Bv	Velocity limit fault	7	128
	Real-time temperature limit	8	256
	Derivative of position error limit	9	512
	Hardware right (+) limit enabled	10	1024
	Hardware left (-) limit enabled	11	2048
Br	Historical right (+) limit fault	12	4096
Bl	Historical left (-) limit fault	13	8192
Bp	Real time right (+) limit	14	16384
Bm	Real time left (-) limit	15	32768

There are three useful mathematical operators that work on binary data:

- **&** (bit-wise and) compares the two operands (bytes, words or longs) and looks for what they have in common. The resulting data has ones only where there were ones in both operands.
- **|** (bit-wise or) results in a one for each bit corresponding to a one in either operand.
- **!** (bit-wise exclusive or) results in a one for each bit corresponding to a one in either operand. It produces a one for each bit when the corresponding bits in the two operands are different and a zero when they are the same.

These operations are illustrated in the next examples:

A	B	A&B	A B	A!B
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Knowing how the binary data works will allow you to write shorter and faster code. The next two code examples check if both limit inputs are high. The first example does this without taking advantage of a binary operator, while the second example shows how using a binary operator makes the code shorter and faster.

Example 1:

```
IF Bm          'Look for - limit high
  IF Bp        'Look for + limit high
    GOSUB100   'Execute subroutine
  ENDIF
ENDIF
```

Example 2:

```
IF (W(0) & 49152) == 49152  'Look at both limits, bits 14 & 15,
                           'w/bit mask 49152 = 32768 + 16384
  GOSUB100                 'Execute subroutine
ENDIF
```

Both examples will execute subroutine 100 if both limit inputs are high. Example 2 uses less code than Example 1 and will run faster as a part of a larger program loop.

The next two examples show how the use of an I/O word and mask can improve program size and execution speed:

Example 3:

```
IF IN(0)       'Look for input 0
  GOSUB200     'Execute subroutine
ENDIF
IF IN(1)       'look for input 1
  GOSUB200     'Execute subroutine
ENDIF
```

Example 4:

```
IF IN(W,0,3)   'Look at both inputs 0 and 1
  GOSUB200     'Execute subroutine
ENDIF
```

Both examples 3 and 4 accomplish the same task with different levels of efficiency.

Commands Affected by SCALE

The next table provides a list of the commands that are affected by the SCALEA, SCALEP and SCALEV commands.

Command	Scaled?	Direction	Scale config	Notes
Accel/Decel				
AT=	yes	set	SCALEA	
RAT	yes	report	SCALEA	
DT=	yes	set	SCALEA	
RDT	yes	report	SCALEA	
ADT=	yes	set	SCALEA	
RAC	yes	report	SCALEA	
Velocity				
VT=	yes	set	SCALEV	
RVT	yes	report	SCALEV	
RVC	yes	report	SCALEV	
RVA	yes	report	SCALEV	
Position				
PT=	yes	set	SCALEP	
RPT	yes	report	SCALEP	
RPC / RPC(0)	yes	report	SCALEP	
RPA	yes	report	SCALEP	
PRT=	yes	set	SCALEP	
RPRT	yes	report	SCALEP	
RPRC	yes	report	SCALEP	
RPRA	yes	report	SCALEP	
O=, O(0)=, O(1)=, O(2)=	yes	set	SCALEP	
OSH=, OSH(0)=, OSH(1)=, OSH(2)=	yes	set	SCALEP	
Synchronized Motion				
PTS(...)	n/a	set	SCALEP	Special case, see notes 1, 2
PTSS(...)	n/a	set	SCALEP	
PRTS(...)	n/a	set	SCALEP	
PRTSS(...)	n/a	set	SCALEP	
VTS=	n/a	set	SCALEV	
ATS=	n/a	set	SCALEA	Special case, see notes 1, 2

Commands Affected by SCALE

Command	Scaled?	Direction	Scale config	Notes
DTS=	n/a	set	SCALEA	Special case, see notes 1, 2
ADTS=	n/a	set	SCALEA	Special case, see notes 1, 2
Error Limit				
EL=	yes	set	SCALEP	Enabled by default. SYSCTL(6,0) to disable, SYSCTL(6,1) to enable
REL	yes	report	SCALEP	
REA	yes	report	SCALEP	
Index, Rising/Falling Edge, Internal Encoder				
RI(0)	Only when ENCO	report	SCALEP	Scaling disabled on RI(0) / RJ(0) when ENC1 mode active, then scale moves to RI(1), RJ(1)
RJ(0)	Only when ENCO	report	SCALEP	
Modulo Position				
PMT=	yes	set	SCALEP	
RPMT	yes	report	SCALEP	
PML=	yes	set	SCALEP	
RPML	yes	report	SCALEP	
RPMA	yes	report	SCALEP	
Software Limit				
SLP=	yes	set	SCALEP	Software position limits
RSLP	yes	report	SCALEP	
SLN=	yes	set	SCALEP	
RSLN	yes	report	SCALEP	
Follow Mode				
MFA	yes	set	SCALEP	Follower units only; controller units not related to scaling
MFD	yes	set	SCALEP	
MFSLEW	yes	set	SCALEP	
MFLTP=	yes	set	SCALEP	
RMFLTP	yes	report	SCALEP	
MFHTP=	yes	set	SCALEP	
RMFHTP	yes	report	SCALEP	
MFL	yes	set	SCALEP	Follower units only; controller units not related to scaling
MFH	yes	set	SCALEP	
External Encoder				

Commands Affected by SCALE

Command	Scaled?	Direction	Scale config	Notes
RI(1)	yes	report	SCALEP	Scaled only for ENC1 (external encoder)
RJ(1)	yes	report	SCALEP	

Notes:

1. PTS commands position, velocity, and acceleration units must agree. I.e., if position is in mm, then velocity must be mm/sec, acceleration must be mm/(sec²).
2. User may provide in scaled units, but the PTS command itself does not apply scaling. The values are simply passed to the target.

Command Error Codes

When a command results in error, the Command error bit (Status Word 2, bit 14) will be tripped. It is up to the user to catch this and read the error code.

NOTE: If multiple command managers are executing commands at the same time, it is possible for one to overwrite the other.

Decoding the Error

The RERRC or =ERRC commands reply with a numeric value describing the most recent error. To decode the error, refer to the next table.

Code	Description	Notes
0	NO_ERROR	
1	BAD_LENGTH	Only used for CAN when the command is too long
2	BAD_ARGUMENT	
3	BAD_PACKET	
4	BAD_OPERATION	
5	MISSING_ARGUMENT	
6	NOT_USED	
7	ERROR_PARENTHESES	
8	NOT_USED	
9	LENGTH_VIOLATION	Embedded address was not found within the 64-character buffer for IF, SWITCH, GOTO, etc. (should never happen)
10	BAD_ARRAY_ADDR	Array index outside the defined range
11	DIVIDE_BY_ZERO	Attempt to divide by 0
12	STACK_OVERFLOW	No room on stack for another 10 GOSUB and INTERRUPTS use the same stack
13	STACK_UNDERFLOW	RETURN or RETURNI with no place to return
14	BAD_LABEL	Label does not exist for GOSUB or GOTO
15	NESTED_SWITCH	Too many nested SWITCH (> 4)
16	BAD_FORMULA	
17	BAD_WRITE_LENGTH	VST command amount written too long
18	NOT_USED	
19	BAD_BIT	Z{letter} command issued for a bit that cannot be reset
20	INVALID_INTERRUPT	EITR command for interrupt not defined)
21	NO_PERMISSION	Operation or memory range is not user-accessible
22	OPERATION_FAILED	General error
23	MATH_OVERFLOW	
24	CMD_TIMEOUT	Combitronics timeout
25	IO_NOT_PRESENT	
26	NO_CROSS_AXIS_SUPPORT	
27	BAD_MOTOR_STATE	
28	BAD_CROSS_AXIS_ID	
29	BAD_COMBITRONIC_FCODE	
30	BAD_COMBITRONIC_SF_CODE	
31	EE_WRITE_QUEUE_FULL	
32	CAM_FULL	

Finding the Error Source

The RERRW or =ERRW commands reply with a numeric code describing the source of the error. To decode the error source, refer to the next table.

Code	Description	Notes
0	CMD_COMM0	RS-232 for D-style, RS-485 for M-style
1	CMD_COMM1	RS-485 for D-style only
2	CMD_PROG	From downloaded program
3	CMD_CAN	CAN port (CANopen, DeviceNet , Combitronic) or PROFIBUS
4	CMD_MB0	N/A
5	CMD_MB1	N/A

Glossary

This section provides a glossary of industrial motion terms.

Acceleration

A change in velocity as a function of time. Acceleration usually refers to increasing velocity, and deceleration to decreasing velocity.

Accuracy

A measure of the difference between expected position and actual position of a motor or mechanical system. Motor accuracy is usually specified as an angle representing the maximum deviation from expected position.

Address

A unique identifier assigned to a network device that differentiates it from other devices operating on the same network.

Ambient Temperature

The temperature of the cooling medium, usually air, immediately surrounding the motor or another device.

Angular Accuracy

The measure of shaft positioning accuracy on a servo or stepping motor.

Back EMF (BEMF)

The voltage generated when a permanent magnet motor is rotated. This voltage is proportional to motor speed and is present regardless of whether the motor winding(s) are energized or not.

Breakaway Torque

The torque required to start a machine in motion. Almost always greater than the running torque.

Brushless Motor

Class of motors that operate using electronic commutation of phase currents rather than electromechanical (brush-type) commutation. Brushless motors typically have a permanent magnet rotor and a wound stator.

Closed Loop

A broadly-applied term relating to any system in which the output is measured and compared to the input. The output is then adjusted to reach the desired condition. In motion control, the term typically describes a system utilizing a velocity and/or position transducer to generate correction signals in relation to desired parameters.

Cogging (Cogging Torque)

A term used to describe nonuniform angular velocity. Cogging appears as jerkiness, especially at low speeds.

Commutation

A term which refers to the action of steering currents or voltages to the proper motor phases so as to produce optimum motor torque. Proper commutation means the relationship of the Rotor to the Stator must be known at all times.

- In brush-type motors, commutation is done electromechanically through the brushes and commutator.
- In brushless motors, commutation is done by the switching electronics using rotor position information obtained by Hall sensors, single turn absolute encoder or a resolver.

Controller

A term describing a functional block containing an amplifier, power supplies and possibly position-control electronics for operating a servomotor or step motor.

Current at Peak Torque (IPK) (Amperes)

The amount of input current required to develop peak torque. This is often outside the linear torque/current relationship.

Current, Rated

The maximum allowable continuous current a motor can handle without exceeding motor temperature limits.

Detent Torque

The maximum torque that can be applied to a non-energized step motor without causing continuous rotating motion.

Duty Cycle

For a repetitive cycle, the ratio of on time to total cycle time.

$$\text{Duty cycle (\%)} = [\text{On time} / (\text{On time} + \text{Off time})] \times 100\%$$

Dynamic Braking

A passive technique for stopping a permanent magnet brush or brushless motor. The motor windings are shorted together through a resistor which results in motor braking with an exponential decrease in speed.

Efficiency

The ratio of power output to power input.

Electrical Time Constant (te) (Seconds)

The time required for current to reach 63.2% of its final value for a fixed voltage level. It can be calculated from the relationship

$$t_e = L/R$$

Where L is inductance (henries) and R is resistance (ohms).

Encoder

A feedback device that converts mechanical motion into electronic signals. The most commonly used rotary encoders output digital pulses corresponding to incremental angular motion. For example, a 1000-line encoder produces 1000 pulses every mechanical revolution. The encoder consists of a glass or metal wheel with alternating transparent and opaque stripes, detected by optical sensors to produce the digital outputs.

Feedback

A signal which is transferred from the output back to the input for use in a closed loop system.

Form Factor

The ratio of RMS current to average current. This number is a measure of the current ripple in a SCR or other switch-mode type of drive. Because motor heating is a function of RMS current while motor torque is a function of average current, a form factor greater than 1.00 means some fraction of motor current is producing heat but not torque.

Four Quadrant

Refers to a motion system which can operate in all four quadrants, i.e., velocity in either direction and torque in either direction. This means that the motor can accelerate, run and decelerate in either direction.

Friction

A resistance to motion caused by contact with a surface. Friction can be constant with varying speed (Coulomb friction) or proportional to speed (viscous friction).

Hall Sensor

A feedback device which is used in a brushless servo system to provide information for the amplifier to electronically commutate the motor. The device uses a magnetized wheel and Hall effect sensors to generate the commutation signals.

Holding Torque

Holding torque (sometimes called static torque) specifies the maximum external torque that can be applied to a stopped, energized motor without causing the rotor to move. Typically used as a feature specification when comparing motors.

Horsepower

A unit of measure of power. One horsepower is equal to 746 watts. The measurement of rotary power must take speed and torque into account. Horsepower is a measure of a motor's torque and speed capability. The formula is:

$$\text{HP} = \text{Torque (lb-in.)} \times \text{Speed (RPM)} / 63,025$$

$$\text{HP} = \text{Torque (lb-ft.)} \times \text{Speed (RPM)} / 5,252$$

$$\text{HP} = \text{Volts} \times \text{Amps} \times \text{Efficiency} / 746$$

Inductance (L) (mH - millihenries line-to-line)

The property of a circuit that has a tendency to resist current flow when no current is flowing, and when current is flowing has a tendency to maintain that current flow. It is the electrical equivalent to mechanical inertia.

Inductance (mutual)

Mutual inductance is the property that exists between two current-carrying conductors (or coils) when a change in current in one induces a voltage in the other.

Inertia

The property of an object to resist change in velocity unless acted on by an outside force. Higher-inertia objects require larger torques to accelerate and decelerate. Inertia is dependent on the mass and shape of the object.

Inertial Match

The reflected inertia of the load is equal to the rotor inertia of the motor. For most efficient operation, a system coupling ratio should be selected that provides this condition.

Controller (/ Follower)

The device that is controlling the downstream device(s), or follower(s). For example, it could be a PLC controlling one or more SmartMotors and other devices (e.g., through fieldbus communications), a SmartMotor controlling other SmartMotors (e.g., Combitronic communications). Controller, in Follow mode or Cam mode, refers to the encoder source input.

Open-loop

A system in which there is no feedback. Motor motion is expected to faithfully respond to the input command. Stepping motor systems are an example of open-loop control.

Overload Capacity

The ability of a drive to withstand currents above its continuous rating. It is defined by NEMA (National Electrical Manufacturers Association) as 150% of the rated full-load current for "standard industrial DC motors" for one minute.

Peak torque (T_{pk}) (lb-in.)

The maximum torque a brushless motor can deliver for short periods of time. Operating permanent-magnet motors above the maximum torque value can cause demagnetization of the rare-earth magnets. This is an irreversible effect that will alter the motor characteristics and degrade performance. It is also known as peak current. This should not be confused with system peak torque, which is often determined by amplifier peak-current limitations, where peak current is typically two times continuous current.

Poles

Refers to the number of magnetic poles arranged on the rotor of the brushless motor. Unlike an AC motor, the number of poles has no direct relationship to the base speed of the motor.

Power

The rate at which work is done. In motion control, power is equal to torque multiplied by speed.

Power (watts) = force x distance/time

Power = voltage x current

Power Factor

Ratio of true power (kW) to apparent power (kVA).

Pulse Rate

The frequency of the step pulses applied to a step motor driver. The pulse rate, multiplied by the resolution of the motor/driver combination (in steps per revolution), yields the rotational speed in revolutions per second.

Pulse Width Modulation (PWM)

Describes a switch-mode (as opposed to linear) control technique used in amplifiers and drivers to control motor voltage and current.

Ramp, Ramp Up, Ramp Down

The whole or ascend/descend parts, respectively, of the Trapezoidal Move Profile (TMP), which is associated with follow or cam modes. The complete motion profile is defined by an ascend (ramp up) part, the slew part, and the descend (ramp down) part. See the diagram Trapezoidal Move Profile (TMP) and Output Position Diagrams on page 144.

Read / Read-Write

Read (or read-only): the system can read data from a file or device but not write data to the file or device.

Read-Write: the system can read data from or write data to a file or device.

Regeneration

The action during motor braking, in which the motor acts as a generator and takes kinetic energy from the load, converts it to electrical energy and returns it to the amplifier.

Repeatability

The degree to which a parameter such as position or velocity can be duplicated.

Resolution

The smallest increment into which a parameter can be broken down. For example, a 1000-line encoder has a resolution of 1/1000 of a revolution.

Resonance

Oscillatory behavior caused by mechanical or electromechanical harmonics and limitations.

Ringling

Oscillation of a system after a sudden change in state.

RMS Current - Root Mean Square Current

In an intermittent duty-cycle application, the RMS current is equal to the value of steady state current that would produce the equivalent motor heating over a period of time.

RMS Torque - Root Mean Square Torque

In an intermittent duty-cycle application, the RMS torque is equal to the value of steady state torque that would produce the equivalent motor heating over a period of time.

Rotor

The moving part of the motor, consisting of the shaft and magnets. These magnets are analogous to the field winding of a brush-type DC motor.

Settling Time

The time required for a parameter to stop oscillating or ringing and reach its final value.

Follower (/ Controller)

One or more devices that are being controlling by an upstream device, or controller. For example, it could be one or more SmartMotors and other devices being controlled by a PLC (e.g., via fieldbus communications), one or more SmartMotors being controlled by a SmartMotor (e.g., Combitronic communications). Follower, in Follow mode or Cam mode, refers to the intermediate counts produced by the trapezoidal move profile.

Speed

Describes the linear or rotational velocity of a motor or other object in motion.

Stall Torque

The amount of torque developed with voltage applied and the shaft locked or not rotating. Also known as locked-rotor torque.

Stator

The nonmoving part of the motor. Specifically, it is the iron core with the wire winding in it that is pressed into the frame shell. The winding pattern determines the voltage constant of the motor.

Stiffness

The ability to resist movement induced by an applied torque. Stiffness is often specified as a torque displacement curve, indicating the amount a motor shaft will rotate upon application of a known external force when stopped.

Torque

A measure of angular force which produces rotational motion. This force is defined by a linear force multiplied by a radius, e.g., lb-in. The formula is:

$$\text{Torque (lb-ft.)} = 5,250 \times \text{HP/RPM}$$

Torque Constant (KT = lb-ft./A)

An expression of the relationship between input current and output torque. For each ampere of current, a fixed amount of torque is produced.

NOTE: Torque constants *are not* linear over the operating range of a motor. They apply best at approximately 75% of no-load maximum speed or where the peak and continuous torque curves meet.

Torque-to-Inertia Ratio

Defined as the motor's holding torque divided by the inertia of its rotor. The higher the ratio, the higher a motor's maximum acceleration capability will be.

Trapezoidal Move Profile (TMP)

The motion profile defined by an ascend (ramp up) part, a slew part and descend (ramp down) part, which are associated with follow or cam modes. Also, referred to in whole as "ramp", and as "ramp up" and "ramp down" for the ascend and descend parts, respectively. See the diagram Trapezoidal Move Profile (TMP) and Output Position Diagrams on page 144.

Velocity

The change in position as a function of time. Velocity has both a magnitude and sign.

Voltage Constant (KE) (V/kRPM peak, line-to-line)

May also be termed Back-EMF constant. When a motor is operated, it generates a voltage proportional to speed but opposing the applied voltage. The shape of the voltage waveform depends on the specific motor design. For example, in a brushless motor, the wave shape may be trapezoidal or sinusoidal.

Math Operators

The next table shows the math operators that are available for the SmartMotor.

Operator	Description
Basic operations:	
+	Add
-	Subtract
*	Multiply
/	Divide
Logical operations:	
>	Greater than
<	Less than
==	Equal to
<=	Less than or equal to
>=	Greater than or equal to
Integer operations:	
^	Power limited to 4th power and below, integers only
&	Bitwise AND
	Bitwise inclusive OR
!	Bitwise exclusive OR
!=	Not equal to
%	Modulo (remainder) division
SQRT(value)	Integer square root
ABS(value)	Integer absolute value
Floating-point functions:	
FSQRT(value)	Floating-point square root
FABS(value)	Floating-point absolute value
SIN(value)	Floating-point sine
COS(value)	Floating-point cosine
TAN(value)	Floating-point tangent
ASIN(value)	Floating-point arcsine
ACOS(value)	Floating-point arccosine
ATAN(value)	Floating-point arctangent
PI	Floating-point representation of pi

Moment of Inertia

A basic understanding of Moment of Inertia serves well in ensuring proper SmartMotor™ sizing. It is one thing to look at static points on torque curves, but it is altogether different when considering the dynamic aspects of loads being accelerated at high rates.

- The inertial mass of an object is a measure of its resistance to a change in its velocity.
- The Moment of Inertia of an object is at a point of reference of rotation, which is at the pivot point or axis of rotation.
- The Moment of Inertia can, therefore, be thought of as a measure of the resistance to any change in rotational speed.

For linear systems, the rate of change of speed (acceleration) is proportional to the force applied. Double the mass requires double the force to achieve the same acceleration. Similarly, for rotational systems, the angular acceleration of the load is proportional to the torque applied. Double the Moment of Inertia and the torque needs to be doubled for the same angular acceleration. Moment of Inertia is, therefore, a measure of a load's resistance to angular speed change, or how much effort (torque) is required to cause acceleration or deceleration.

Matching Motor to Load

A common rule of thumb for SmartMotor sizing is that the load should have no more than ten times the Moment of Inertia of the motor rotor that is driving it. This provides a good starting point and typically allows for safe sizing over a wide range of applications.

A rotating load wants to maintain the same velocity. Therefore, when a motor attempts to accelerate the load, it must overcome the Moment of Inertia of that load by applying additional torque to increase the speed. As a result, it takes more torque to change speed than it does to maintain a given speed.

In the same manner, for the motor to decelerate the load, the load's Moment of Inertia wants to keep the motor going the same speed and will, in effect, back-drive the motor, which turns it into a generator.



CAUTION: In extreme cases, back-drive can result in overvoltage damage to the motor's drive stage.

Improving the Moment of Inertia Ratio

Adding gear reduction to a motor gives it more leverage to prevent back-driving and also provides an advantage in accelerating a load up to speed.

Any given change in gear reduction results in a proportional change in speed and static torque, but results in a squared change in acceleration and dynamic rate of change of torque. The result is that by adding gear ratio you gain a squared decrease in the ratio of Moment of Inertia between motor and load.

Therefore, through gear reduction, the motor has a greater advantage in both accelerating and decelerating the load. Gear reduction adds protection against damage to the overall system.

RCAN, RCHN and RMODE Status

This section provides the information for decoding the RCAN status, RCHN status and RMODE status.

RCAN Status Decoder

The next table provides a "decoder" for the RCAN (CAN port) status. When a general CAN port error is flagged, you can use the information in the table to decode the RCAN command result or the {variable}=CAN result in a program.

Description	Value	Bit
CAN Power Okay	1	0
DeviceNet Com fault occurred	2	1*
DeviceNet Power ignore option enabled	4	2
Reserved	8	3
User attempted to do Combitronics read from broadcast address	16	4*
Combitronics debug, internal issue	32	5*
Timeout (Combitronics Client)	64	6*
Combitronics server ran out of buffer slots	128	7*
Errors reached warning level	256	8*
Receive Errors reached warning level	512	9*
Transmit Errors reached warning level	1024	10*
Receive Passive Error	2048	11*
Transmit Passive Error (cable issue)	4096	12*
Bus Off Error	8192	13*
RX buffer 1 overflowed	16384	14*
RX buffer 0 overflowed	32768	15*

*Indicates an error bit

For more details on SmartMotor homing operations, see the *SmartMotor Homing Procedures and Methods Application Note*.

RCHN Status Decoder

The next table provides a "decoder" for the RCHN (serial port) status. Like the RCAN (CAN port) status, you can use this information to decode general RS-232 or RS-485 serial port status information. Note that:

- RCHN(n) or =CHN(n) where n is the id of the serial port.
- It returns an integer; only the lower four bits indicate the state of the error.

Description	Value	Bit
Buffer Overflow Error	1	0
Framing Error	2	1
N/A	4	2
Parity Error	8	3
Timeout	16	4

Clearing Serial Port Errors

It is up to the user to read which port caused the error using RCHN(n).

- Use Z(2,0) to clear communication errors on port 0
- Use Z(2,1) to clear communication errors on port 1

The Status Word 2 "Error channel 0" and "Error channel 1" bits will indicate if any error has occurred on that port. Those status bits will be cleared when the communications error is cleared using one of the commands listed above.

RMODE Status Decoder

The next table provides a "decoder" for the RMODE, RMODE(1) and RMODE(2) status. The MODE command contains integer values for the current operating mode. Issuing RMODE or {variable}=MODE will return a value according to the present drive mode status.

NOTE: When running dual trajectory generators, RMODE(1) applies to Trajectory Generator 1, and RMODE(2) applies to trajectory Generator 2. RMODE, RMODE(1) and RMODE(2) all use the same values for mode descriptions where they apply.

Description	Value	Mode
Mixed Mode	-5	Mixed Mode
Electronic Camming	-4	Cam Mode
Step and Direction Mode	-3	STEPDIR
Electronic Gearing	-2	FOLLOW
Future Use	-1	N/A
N/A	0	NOMODE
(MP command, default on startup)	1	POSITION
Future Use	2	N/A
(MV command)	3	VELOCITY
(MT command)	4	TORQUE
Future Use	5	N/A
Future Use	6	HOMING
(Profiling via CANopen only, at this point)	7	INTPOSITION
Future Use	8	N/A
Future Use	9	N/A

Mode Status Example

These are some examples of capturing the current value of MODE:

```
x=MODE          'Sets the variable x equal to the value of the current mode.
IF MODE==1      'Executes the IF structure when mode equals the specified value.
WHILE MODE==4   'Executes the WHILE loop when the mode equals the specified
value.
```

Scale Factor Calculation

This section provides information on using Sample Rates and Encoder Resolution to calculate scale factors.

Sample Rates

NOTE: For Class 6 SmartMotors, the sample rate is fixed at 16,000 Hz (16 kHz).

Native units for all SmartMotors are in Encoder Counts per sample. A "sample" is considered the time period during which encoder position data is collected. These are related commands:

- SAMP is a read-only command that returns the value of the sample rate in cycles per second and is affected by the PID command. For example:

```
x=SAMP 'Set x equal to the sample rate
```

NOTE: There is no SAMP= command.

- RSAMP reports the sample rate in values of samples per second to the SMI terminal window or other user interface. For example:

```
RSAMP 'Report the sample rate to the user's screen.
```

For example, if you issues RSAMP and it returns 8000, then the motor collects position samples 8000 times per second. The default sample rate is 8000 samples per second, but it can be adjusted, as described in the next section.

PID Sample Rate Command

NOTE: This feature is not available for Class 6 SmartMotors.

The SmartMotor controllers default to 8000 samples per second, but it is adjustable by use of the PID command. The command PID2 is the default. However, the commands PID1, PID4, and PID8 are also available.

This table provides a list of sample rates for each of the PID commands:

PID Command	Samples per Second	Value SAMP	RSAMP Returns:
PID8	2000	2000	2000
PID4	4000	4000	4000
PID2	8000	8000	8000
PID1	16000	16000	16000

Encoder Resolution and the RES Parameter

These commands are used to report encoder resolution:

- RES is a read-only command that reports the encoder resolution that will be experienced by the user (i.e., the change in RPA as the motor makes one shaft revolution). The values stored in the EEPROM fields for encoder resolution may not be a reliable source of information due to pre-scaling in the encoder firmware. For example:

```
x=RES 'Set x equal to the encoder resolution.
```

NOTE: There is no RES= command.

- RRES reports encoder resolution to the SMI terminal window or other user interface. This value is set at the factory and cannot be changed. For example:

```
RRES 'Report encoder resolution to the user's
screen.
```

NOTE: The typical resolution is 4000 for SmartMotors with a NEMA 23 or smaller frame size; it is 8000 for the larger SmartMotors. However, depending on purchase options, this may not always be the case. Therefore, always issue the RRES command to verify the resolution.

Native Velocity and Acceleration Units

The next table shows the units for native Velocity Target (VT) and native Acceleration Target (AT).

Native Velocity Target units (VT) are: (Encoder Counts/sample) * 65536

Native Acceleration Target units (AT) are: (Encoder Counts/sample/sample) * 65536

Velocity Calculations

If you know the desired revolutions per second (RPS) and want to set Velocity Target (VT), use this equation:

$$VT = \frac{RPS \times RES \times 65536}{SAMP}$$

If you wish to calculate velocity in real world units from native units:

$$RPS = \frac{VT \times SAMP}{RES \times 65536}$$

NOTE: The system value is Actual Velocity (VA). You can issue RVA to report actual velocity of the system in real time, but it will return native units.

The previous equation may be used with either VT or VA.

Acceleration Calculations

If you know your desired Revolutions per second per second (RPSS) and want to set Accel Target (AT):

$$AT = \frac{RPSS \times RES \times 65536}{SAMP^2}$$

NOTE: The same calculation works for acceleration or deceleration, so it may be applied to AT, DT, and combined ADT parameters.

If you wish to calculate Acceleration in real world units from native units:

$$RPSS = \frac{AT \times SAMP^2}{RES \times 65536}$$

NOTE: At this time, there is no method for reporting actual real-time acceleration in Class 5 SmartMotors.

The previous equation may be used with either AT or DT.

Status Words - SmartMotor

This section provides the descriptions of the status words for the SmartMotor. For DS2020 Combitronic system fault and status words, see Fault and Status Words - DS2020 Combitronic System on page 932.

To use the status words:

- The RB(sw,b) command will report the status bit "b" from status word "sw".
- The RW(x) command will report the 16-bit value of Status Word x. For example, RW(0) will report the 16-bit value of Status Word 0, RW(1) will report the 16-bit value of Status Word 1, and so on.
- You can assign the 16-bit result to a variable: a=W(x).

For example, the next code snippet prints the status "Drive ready and OFF." if the "value" (from the Value column) is equal to "3" (value 1, Drive Ready + value 2, Bo: Motor is off). See the next table (Status Word: 0) for these values.

```
IF W(0) & 3
  PRINT ("Drive ready and OFF.")
ENDIF
```

Status Word 0: Primary Fault/Status Indicator

Description	Bit	To Clear	Value	Bit	To read	Assign	Related Commands
Drive Ready			1	0	RB(0,0)	=B(0,0)	
Bo: Motor is off (indicator)	Bo		2	1	RB(0,1)	=B(0,1)	OFF
Bt: Trajectory in progress (indicator)	Bt		4	2	RB(0,2)	=B(0,2)	G, TWAIT, GS, TSWAIT
Servo Bus Voltage Fault		Z(0,3), ZS	8	3	RB(0,3)	=B(0,3)	
Peak Over Current occurred	Ba	Z(0,4), Za, ZS	16	4	RB(0,4)	=B(0,4)	
Excessive Temperature fault latch	Bh	Z(0,5), Zh, ZS	32	5	RB(0,5)	=B(0,5)	TH={temp in degrees C} (85 Deg. Default)
Excessive Position Error Fault	Be	Z(0,6), Ze, ZS	64	6	RB(0,6)	=B(0,6)	EA, REA, EL, EL={value in encoder counts}
Velocity Limit Fault	Bv	Z(0,7), Zv, ZS	128	7	RB(0,7)	=B(0,7)	VL={value in RPM}, RVL
Real-time temperature limit			256	8	RB(0,8)	=B(0,8)	
Derivative Error Limit (dE/dt) Fault		Z(0,9), ZS	512	9	RB(0,9)	=B(0,9)	RDEL, DEL={value in velocity units}
Hardware Limit Positive Enabled			1024	10	RB(0,10)	=B(0,10)	EILP(2), EIGN(2), EIGN(W,2), FSA()
Hardware Limit Negative Enabled			2048	11	RB(0,11)	=B(0,11)	EILN(3), EIGN(3), EIGN(W,3), FSA()
Historical Right Limit (+ or Positive)	Br	Z(0,12), Zr, ZS	4096	12	RB(0,12)	=B(0,12)	EILP(2), EIGN(2), EIGN(W,2), FSA()
Historical Left Limit (- or Negative)	Bl	Z(0,13), Zl, ZS	8192	13	RB(0,13)	=B(0,13)	EILN(3), EIGN(3), EIGN(W,3), FSA()
Right (+ or Positive) Limit Asserted	Bp		16384	14	RB(0,14)	=B(0,14)	RIN(2)
Left Limit (- or Negative) Asserted	Bm		32768	15	RB(0,15)	=B(0,15)	RIN(3)

Status Word 1: Index Registration and Software Travel Limits

Description	Bit	To Clear	Value	Bit	To read	Assign	Related Commands
Rise Capture Encoder(0) Armed			1	0	RB(1,0)	=B(1,0)	Ai(0)
Fall Capture Encoder(0) Armed			2	1	RB(1,1)	=B(1,1)	Aj(0)
Rising edge captured ENC(0) (historical bit)	Bi(0)	Z(1,2), ZS	4	2	RB(1,2)	=B(1,2)	
Falling edge captured ENC(0) (historical bit)	Bj(0)	Z(1,3), ZS	8	3	RB(1,3)	=B(1,3)	
Rise Capture Encoder(1) Armed			16	4	RB(1,4)	=B(1,4)	Ai(1)
Fall Capture Encoder(1) Armed			32	5	RB(1,5)	=B(1,5)	Aj(1)
Rising edge captured ENC(1) (historical bit)	Bi(1)	Z(1,6), ZS	64	6	RB(1,6)	=B(1,6)	
Falling edge captured ENC(1) (historical bit)	Bj(1)	Z(1,7), ZS	128	7	RB(1,7)	=B(1,7)	
Capture input state 0 (indicator)	Bx(0)		256	8	RB(1,8)	=B(1,8)	
Capture input state 1 (indicator)	Bx(1)		512	9	RB(1,9)	=B(1,9)	
Software Travel Limits Enabled			1024	10	RB(1,10)	=B(1,10)	SLE, SLD, SLM(mode)
Soft limit mode (indicator): 0-Don't Stop. 1-Cause Fault. Default is 1			2048	11	RB(1,11)	=B(1,11)	SLM(mode)
Historical positive software overtravel limit	Brs	Z(1,12), Zrs, ZS	4096	12	RB(1,12)	=B(1,12)	SLP=formula, RSLP
Historical negative software overtravel limit	Bls	Z(1,13), Zls, ZS	8192	13	RB(1,13)	=B(1,13)	SLN=formula, RSLN
Real time positive soft limit (indicator)	Bps		16384	14	RB(1,14)	=B(1,14)	RBps
Real time negative soft limit (indicator)	Bms		32768	15	RB(1,15)	=B(1,15)	RBms

Status Word 2: Communications, Program and Memory

Description	Bit	To Clear	Value	Bit	To read	Assign	Related Commands
Error on Communications Channel 0		Z(2,0), ZS	1	0	RB(2,0)	=B(2,0)	RCHN(0), OCHN(), CCHN(), BAUD(0)=
Error on Communications Channel 1		Z(2,1), ZS	2	1	RB(2,1)	=B(2,1)	RCHN(1), OCHN(), CCHN(), BAUD(1)=
USB Error (Class 6 only)		Z(2,2), ZS	4	2	RB(2,2)	=B(2,2)	
Reserved 3			8	3	RB(2,3)	=B(2,3)	
CAN Port Error		Z(2,4), ZS	16	4	RB(2,4)	=B(2,4)	RCAN, CBAUD, CADDR
Reserved 5			32	5	RB(2,5)	=B(2,5)	
Ethernet Error (Class 6 only)		Z(2,6)	64	6	RB(2,6)	=B(2,6)	
I ² C Running			128	7	RB(2,7)	=B(2,7)	PRINT1(), GETCHR1, OCHN(), CCHN()
Watchdog Event			256	8	RB(2,8)	=B(2,8)	
ADB (Animatics Data Block) Bad Checksum			512	9	RB(2,9)	=B(2,9)	
Program Running			1024	10	RB(2,10)	=B(2,10)	RUN, RUN?, PAUSE, RESUME, END, GOTO, GOSUB, RETURN, Z
Trace in Progress			2048	11	RB(2,11)	=B(2,11)	
EE Write Buffer Overflow		Z(2,12)	4096	12	RB(2,12)	=B(2,12)	EPTR=, VST(), VLD()
EE Busy			8192	13	RB(2,13)	=B(2,13)	EPTR=, VST(), VLD()
Command Syntax Error	Bs	Z(2,14), Zs, ZS	16384	14	RB(2,14)	=B(2,14)	RERRC, =ERRC, RERRW, =ERRW
Program Checksum Error	Bk		32768	15	RB(2,15)	=B(2,15)	

Status Word 3: PID State, Brake, Move Generation Indicators

Description	Bit	To Clear	Value	Bit	To read	Assign	Related Commands
Reserved 0			1	0	RB(3,0)	=B(3,0)	
Torque Saturation			2	1	RB(3,1)	=B(3,1)	
Voltage Saturation			4	2	RB(3,2)	=B(3,2)	
Wraparound Occurred	Bw	Z(3,3), Zw, ZS	8	3	RB(3,3)	=B(3,3)	
KG Enabled			16	4	RB(3,4)	=B(3,4)	KG=
Velocity Direction			32	5	RB(3,5)	=B(3,5)	VT=
Torque Direction			64	6	RB(3,6)	=B(3,6)	
I/O Fault Latch		Z(3,7)	128	7	RB(3,7)	=B(3,7)	
Relative Position Mode			256	8	RB(3,8)	=B(3,8)	
Reserved 9			512	9	RB(3,9)	=B(3,9)	X, X(1), X(2), MFSDC()
Peak Current Saturation			1024	10	RB(3,10)	=B(3,10)	
Modulo Rollover		Z(3,11), ZS	2048	11	RB(3,11)	=B(3,11)	PML, PMA
Brake Asserted			4096	12	RB(3,12)	=B(3,12)	BRKSRV, BRKTRJ, BRKENG, BRKRLS, EOBK()
Brake OK			8192	13	RB(3,13)	=B(3,13)	
External Go Enabled			16384	14	RB(3,14)	=B(3,14)	EISM(6)
Velocity Target Reached			32768	15	RB(3,15)	=B(3,15)	VT=

Status Word 4: Interrupt Timers

Description	Bit	To Clear	Value	Bit	To read	Assign	Related Commands
Timer 0			1	0	RB(4,0)	=B(4,0)	TMR(0,#)
Timer 1			2	1	RB(4,1)	=B(4,1)	TMR(1,#)
Timer 2			4	2	RB(4,2)	=B(4,2)	TMR(2,#)
Timer 3			8	3	RB(4,3)	=B(4,3)	TMR(3,#)
Reserved			16	4	RB(4,4)	=B(4,4)	
Reserved			32	5	RB(4,5)	=B(4,5)	
Reserved			64	6	RB(4,6)	=B(4,6)	
Reserved			128	7	RB(4,7)	=B(4,7)	
Timer 8			256	8	RB(4,8)	=B(4,8)	TMR(8,#)
Drive enabled			512	9	RB(4,9)	=B(4,9)	
Command request timeout			1024	10	RB(4,10)	=B(4,10)	
Fault handling enabled			2048	11	RB(4,11)	=B(4,11)	
Group fault			4096	12	RB(4,12)	=B(4,12)	
Group ready			8192	13	RB(4,13)	=B(4,13)	
Remote fault			16384	14	RB(4,14)	=B(4,14)	
Timeout event			32768	15	RB(4,15)	=B(4,15)	

Status Word 5: Interrupt Status Indicators

Description	Bit	To Clear	Value	Bit	To read	Assign	Related Commands
Interrupt 0 enabled			1	0	RB(5,0)	=B(5,0)	ITR(), EITR(), ITRE, ITRD
Interrupt 1 enabled			2	1	RB(5,1)	=B(5,1)	ITR(), EITR(), ITRE, ITRD
Interrupt 2 enabled			4	2	RB(5,2)	=B(5,2)	ITR(), EITR(), ITRE, ITRD
Interrupt 3 enabled			8	3	RB(5,3)	=B(5,3)	ITR(), EITR(), ITRE, ITRD
Interrupt 4 enabled			16	4	RB(5,4)	=B(5,4)	ITR(), EITR(), ITRE, ITRD
Interrupt 5 enabled			32	5	RB(5,5)	=B(5,5)	ITR(), EITR(), ITRE, ITRD
Interrupt 6 enabled			64	6	RB(5,6)	=B(5,6)	ITR(), EITR(), ITRE, ITRD
Interrupt 7 enabled			128	7	RB(5,7)	=B(5,7)	ITR(), EITR(), ITRE, ITRD
Reserved			256	8	RB(5,8)	=B(5,8)	
Reserved			512	9	RB(5,9)	=B(5,9)	
Reserved			1024	10	RB(5,10)	=B(5,10)	
Reserved			2048	11	RB(5,11)	=B(5,11)	
Reserved			4096	12	RB(5,12)	=B(5,12)	
Reserved			8192	13	RB(5,13)	=B(5,13)	
Reserved			16384	14	RB(5,14)	=B(5,14)	
Interrupts Enabled			32768	15	RB(5,15)	=B(5,15)	ITRE, ITRD

Status Word 6: Drive Modes

Description	Bit	To Clear	Value	Bit	To read	Assign	Related Commands
Trap Mode			1	0	RB(6,0)	=B(6,0)	MDT
Enhanced Trap Mode			2	1	RB(6,1)	=B(6,1)	MDE
Sine Mode			4	2	RB(6,2)	=B(6,2)	MDS
Vector Control			8	3	RB(6,3)	=B(6,3)	
Reserved 4			16	4	RB(6,4)	=B(6,4)	
Feedback Fault ¹		Z(6,5), ZS	32	5	RB(6,5)	=B(6,5)	MDT, MDE, MDS, MDC
MDH mode active			64	6	RB(6,6)	=B(6,6)	
Drive Enable Fault		Z(6,7), ZS	128	7	RB(6,7)	=B(6,7)	
Angle Match			256	8	RB(6,8)	=B(6,8)	
TOB Enabled			512	9	RB(6,9)	=B(6,9)	MDE, MDB
Inverted			1024	10	RB(6,10)	=B(6,10)	MINV()
MTB Active			2048	11	RB(6,11)	=B(6,11)	MTB, FSA()
ABS Battery Fault			4096	12	RB(6,12)	=B(6,12)	ENCCTL()
Low Bus Voltage		Z(6,13), ZS	8192	13	RB(6,13)	=B(6,13)	
High Bus Voltage		Z(6,14), ZS	16384	14	RB(6,14)	=B(6,14)	
Regen Active			32768	15	RB(6,15)	=B(6,15)	

1. Defaults as health check of internal encoder. Other use depends on encoder mode (ENC0 or ENC1) and firmware version. Set due to hardware problems with internal encoder (or external encoder when in ENC1 mode). Contact factory for more details.

Status Word 7: Multiple Trajectory Support

Description	Bit	To Clear	Value	Bit	To read	Assign	Related Commands
TG1 In Progress			1	0	RB(7,0)	=B(7,0)	G(1), X(1), MP(1), MV(1)
TG1 Accel/Ascend			2	1	RB(7,1)	=B(7,1)	G(1), X(1), MP(1), MV(1)
TG1 Slewing			4	2	RB(7,2)	=B(7,2)	G(1), X(1), MP(1), MV(1)
TG1 Decel/Descend			8	3	RB(7,3)	=B(7,3)	G(1), X(1), MP(1), MV(1)
TG1 Reserved/Dwell			16	4	RB(7,4)	=B(7,4)	
Reserved 5			32	5	RB(7,5)	=B(7,5)	
Reserved 6			64	6	RB(7,6)	=B(7,6)	
Reserved 7			128	7	RB(7,7)	=B(7,7)	
TG2 In Progress			256	8	RB(7,8)	=B(7,8)	G(2), X(2), MFR(2), MC(2), MFSDC(), MFSLEW()
TG2 Accel/Ascend			512	9	RB(7,9)	=B(7,9)	G(2), X(2), MFR(2), MC(2), MFSDC(), MFSLEW()
TG2 Slewing			1024	10	RB (7,10)	=B(7,10)	G(2), X(2), MFR(2), MC(2), MFSDC(), MFSLEW()
TG2 Decel/Descend			2048	11	RB (7,11)	=B(7,11)	G(2), X(2), MFR(2), MC(2), MFSDC(), MFSLEW()
TG2 Dwell (or higher Dwell in progress)			4096	12	RB (7,12)	=B(7,12)	G(2), X(2), MFR(2), MC(2), MFSDC(), MFSLEW(), MFH, MFHTP
Traverse Direction, 0:Fwd, 1:Rev			8192	13	RB (7,13)	=B(7,13)	MFCTP()
Traverse Lower Dwell in Progress			16384	14	RB (7,14)	=B(7,14)	MFSDC(), MFL, MFLTP
TS WAIT			32768	15	RB (7,15)	=B(7,15)	PTS(), PRTS(), GS, X, TSWAIT

Status Word 8: Cam Support

Description	Bit	To Clear	Value	Bit	To read	Assign	Related Commands
Cam User Bit 0			1	0	RB(8,0)	=B(8,0)	CTW()
Cam User Bit 1			2	1	RB(8,1)	=B(8,1)	CTW()
Cam User Bit 2			4	2	RB(8,2)	=B(8,2)	CTW()
Cam User Bit 3			8	3	RB(8,3)	=B(8,3)	CTW()
Cam User Bit 4			16	4	RB(8,4)	=B(8,4)	CTW()
Cam User Bit 5			32	5	RB(8,5)	=B(8,5)	CTW()
Cam Mode 0			64	6	RB(8,6)	=B(8,6)	MCE()
Cam Mode 1			128	7	RB(8,7)	=B(8,7)	MCE()
IP User Bit 0			256	8	RB(8,8)	=B(8,8)	
IP User Bit 1			512	9	RB(8,9)	=B(8,9)	
IP User Bit 2			1024	10	RB(8,10)	=B(8,10)	
IP User Bit 3			2048	11	RB(8,11)	=B(8,11)	
IP User Bit 4			4096	12	RB(8,12)	=B(8,12)	
IP User Bit 5			8192	13	RB(8,13)	=B(8,13)	
IP Mode 0			16384	14	RB(8,14)	=B(8,14)	
IP Mode 1			32768	15	RB(8,15)	=B(8,15)	

Status Word 9: No Bits Defined (Class 5 Only)

Description	Bit	To Clear	Value	Bit	To read	Assign	Related Commands
Reserved			1	0			
Reserved			2	1			
Reserved			4	2			
Reserved			8	3			
Reserved			16	4			
Reserved			32	5			
Reserved			64	6			
Reserved			128	7			
Reserved			256	8			
Reserved			512	9			
Reserved			1024	10			
Reserved			2048	11			
Reserved			4096	12			
Reserved			8192	13			
Reserved			16384	14			
Reserved			32768	15			

Status Word 9: SD Card and DMX Information (Class 6 Only)

For more details on DMX, see the *Moog Animatics SmartMotor™ DMX Guide*.

Description	Bit	To Clear	Value	Bit	To read	Assign	Related Commands
SD card present			1	0	RB(9,0)	=B(9,0)	
SD card busy			2	1	RB(9,1)	=B(9,1)	
SD card access error		Z(9,2)	4	2	RB(9,2)	=B(9,2)	
SD card user program found (.smx)			8	3	RB(9,3)	=B(9,3)	
SD card param. ee found			16	4	RB(9,4)	=B(9,4)	
SD card encrypted user program found (.smxe)			32	5	RB(9,5)	=B(9,5)	
DMX packets seen within the last second ¹		Timeout 1 sec of no valid packets	64	6	RB(9,6)	=B(9,6)	
DMX data received ²		Z(9,7)	128	7	RB(9,7)	=B(9,7)	
DMX end of packet		Z(9,8)	256	8	RB(9,8)	=B(9,8)	

Appendix: Status Word 10: RxPDO Arrival Notification

Description	Bit	To Clear	Value	Bit	To read	Assign	Related Commands
received ³							
Reserved			512	9			
Reserved			1024	10			
Reserved			2048	11			
Reserved			4096	12			
Reserved			8192	13			
Reserved			16384	14			
Reserved			32768	15			

1. Set on arrival of any start code; may/not be relevant data.
 2. Set when the last expected motor channel arrives, not when entire packet arrives. E.g., if the set channel quantity is 2, the flag is set when second byte is saved to aw[1]. Allows reaction ASAP on a per motor basis.
 3. Set on arrival of last expected host-capable channel when that channel is received. Therefore, a full 512 channel packet will set this bit at the end of the packet. Allows multiple motors to react to the same event simultaneously (on a higher DMX slot/channel after getting their data).

Status Word 10: RxPDO Arrival Notification

Description	Bit	To Clear	Value	Bit	To read	Assign	Related Commands
Controller enabled			1	0	RB(10,0)	=B(10,0)	
RxPDO 1 arrived		Z(10,1)	2	1	RB(10,1)	=B(10,1)	
RxPDO 2 arrived		Z(10,2)	4	2	RB(10,2)	=B(10,2)	
RxPDO 3 arrived		Z(10,3)	8	3	RB(10,3)	=B(10,3)	
RxPDO 4 arrived		Z(10,4)	16	4	RB(10,4)	=B(10,4)	
RxPDO 5 arrived		Z(10,5)	32	5	RB(10,5)	=B(10,5)	
Reserved			64	6	RB(10,6)	=B(10,6)	
Reserved			128	7	RB(10,7)	=B(10,7)	
Reserved			256	8	RB(10,8)	=B(10,8)	
Reserved			512	9	RB(10,9)	=B(10,9)	
Reserved			1024	10	RB(10,10)	=B(10,10)	
Reserved			2048	11	RB(10,11)	=B(10,11)	
Reserved			4096	12	RB(10,12)	=B(10,12)	
Reserved			8192	13	RB(10,13)	=B(10,13)	
Reserved			16384	14	RB(10,14)	=B(10,14)	
Reserved			32768	15	RB(10,15)	=B(10,15)	

The user program should clear these status bits with a Z(10,bit) command, where bit is values 1-5, after the event handler part of the user program is executed. Bit 0 cannot be cleared—it is an indication of the controller status.
 NOTE: The ZS command will have no effect on these bits.

Status Word 12: DMX Information (Class 5 Only)

NOTE: If DMX is used for Class 5, do not use the spare bits as user-controlled bits. For more details on DMX, see the *Moog Animatics SmartMotor™ DMX Guide*.

Description	Bit	To Clear	Value	Bit	To read	Assign	Related Commands
DMX packets seen within the last second ¹		Timeout 1 sec of no valid packets	1	0	RB(12,0)	=B(12,0)	
DMX data received ²		User: use command UR(1).	2	1	RB(12,1)	=B(12,1)	
DMX end of packet received ³		User: use command UR(2).	4	2	RB(12,2)	=B(12,2)	
Reserved			8	3	RB(12,3)	=B(12,3)	
Reserved			16	4	RB(12,4)	=B(12,4)	
Reserved			32	5	RB(12,5)	=B(12,5)	
Reserved			64	6	RB(12,6)	=B(12,6)	
Reserved			128	7	RB(12,7)	=B(12,7)	
Reserved			256	8	RB(12,8)	=B(12,8)	
Reserved			512	9	RB(12,9)	=B(12,9)	
Reserved			1024	10	RB(12,10)	=B(12,10)	
Reserved			2048	11	RB(12,11)	=B(12,11)	
Reserved			4096	12	RB(12,12)	=B(12,12)	
Reserved			8192	13	RB(12,13)	=B(12,13)	
Reserved			16384	14	RB(12,14)	=B(12,14)	
Reserved			32768	15	RB(12,15)	=B(12,15)	

1. Set on arrival of any start code; may/not be relevant data.

2. Set when the last expected motor channel arrives, not when entire packet arrives. E.g., if the set channel quantity is 2, the flag is set when second byte is saved to aw[1]. Allows reaction ASAP on a per motor basis.

3. Set on arrival of last expected host-capable channel when that channel is received. Therefore, a full 512 channel packet will set this bit at the end of the packet. Allows multiple motors to react to the same event simultaneously (on a higher DMX slot/channel after getting their data).

Status Word 12: User Bits Word 0 (Class 6 Only)

Description	Bit	To Clear	Value	Bit	To read	Assign	Related Commands
User Bit 0			1	0	RB(12,0)	=B(12,0)	US(0), UR(0)
User Bit 1			2	1	RB(12,1)	=B(12,1)	US(1), UR(1)
User Bit 2			4	2	RB(12,2)	=B(12,2)	US(2), UR(2)
User Bit 3			8	3	RB(12,3)	=B(12,3)	US(3), UR(3)
User Bit 4			16	4	RB(12,4)	=B(12,4)	US(4), UR(4)
User Bit 5			32	5	RB(12,5)	=B(12,5)	US(5), UR(5)
User Bit 6			64	6	RB(12,6)	=B(12,6)	US(6), UR(6)
User Bit 7			128	7	RB(12,7)	=B(12,7)	US(7), UR(7)
User Bit 8			256	8	RB(12,8)	=B(12,8)	US(8), UR(8)
User Bit 9			512	9	RB(12,9)	=B(12,9)	US(9), UR(9)
User Bit 10			1024	10	RB(12,10)	=B(12,10)	US(10), UR(10)
User Bit 11			2048	11	RB(12,11)	=B(12,11)	US(11), UR(11)
User Bit 12			4096	12	RB(12,12)	=B(12,12)	US(12), UR(12)
User Bit 13			8192	13	RB(12,13)	=B(12,13)	US(13), UR(13)
User Bit 14			16384	14	RB(12,14)	=B(12,14)	US(14), UR(14)
User Bit 15			32768	15	RB(12,15)	=B(12,15)	US(15), UR(15)

Status Word 13: User Bits Word 1

Description	Bit	To Clear	Value	Bit	To read	Assign	Related Commands
User Bit 16			1	0	RB(13,0)	=B(13,0)	US(16), UR(16)
User Bit 17			2	1	RB(13,1)	=B(13,1)	US(17), UR(17)
User Bit 18			4	2	RB(13,2)	=B(13,2)	US(18), UR(18)
User Bit 19			8	3	RB(13,3)	=B(13,3)	US(19), UR(19)
User Bit 20			16	4	RB(13,4)	=B(13,4)	US(20), UR(20)
User Bit 21			32	5	RB(13,5)	=B(13,5)	US(21), UR(21)
User Bit 22			64	6	RB(13,6)	=B(13,6)	US(22), UR(22)
User Bit 23			128	7	RB(13,7)	=B(13,7)	US(23), UR(23)
User Bit 24			256	8	RB(13,8)	=B(13,8)	US(24), UR(24)
User Bit 25			512	9	RB(13,9)	=B(13,9)	US(25), UR(25)
User Bit 26			1024	10	RB(13,10)	=B(13,10)	US(26), UR(26)
User Bit 27			2048	11	RB(13,11)	=B(13,11)	US(27), UR(27)
User Bit 28			4096	12	RB(13,12)	=B(13,12)	US(28), UR(28)
User Bit 29			8192	13	RB(13,13)	=B(13,13)	US(29), UR(29)
User Bit 30			16384	14	RB(13,14)	=B(13,14)	US(30), UR(30)
User Bit 31			32768	15	RB(13,15)	=B(13,15)	US(31), UR(31)

Status Word 16: On Board Local I/O Status: D-Style Motor

Description	Bit	To Clear	Value	Bit	To read	Assign	Related Commands
On Board I/O 0			1	0	RB(16,0)	=B(16,0)	OS(0), OR(0), OUT(0)=
On Board I/O 1			2	1	RB(16,1)	=B(16,1)	OS(1), OR(1), OUT(1)=
On Board I/O 2			4	2	RB(16,2)	=B(16,2)	OS(2), OR(2), OUT(2)=
On Board I/O 3			8	3	RB(16,3)	=B(16,3)	OS(3), OR(3), OUT(3)=
On Board I/O 4			16	4	RB(16,4)	=B(16,4)	OS(4), OR(4), OUT(4)=
On Board I/O 5			32	5	RB(16,5)	=B(16,5)	OS(5), OR(5), OUT(5)=
On Board I/O 6			64	6	RB(16,6)	=B(16,6)	OS(6), OR(6), OUT(6)=
Reserved 7			128	7	RB(16,7)	=B(16,7)	
Reserved 8			256	8	RB(16,8)	=B(16,8)	
Reserved 9			512	9	RB(16,9)	=B(16,9)	
Reserved 10			1024	10	RB(16,10)	=B(16,10)	
Reserved 11			2048	11	RB(16,11)	=B(16,11)	
Reserved 12			4096	12	RB(16,12)	=B(16,12)	
Reserved 13			8192	13	RB(16,13)	=B(16,13)	
Reserved 14			16384	14	RB(16,14)	=B(16,14)	
Reserved 15			32768	15	RB(16,15)	=B(16,15)	

Status Word 16: On Board Local I/O Status: M-Style Class 5 Motor

Description	Bit	To Clear	Value	Bit	To read	Assign	Related Commands
On Board I/O 0			1	0	RB(16,0)	=B(16,0)	OS(0), OR(0), OUT(0)=
On Board I/O 1			2	1	RB(16,1)	=B(16,1)	OS(1), OR(1), OUT(1)=
On Board I/O 2			4	2	RB(16,2)	=B(16,2)	OS(2), OR(2), OUT(2)=
On Board I/O 3			8	3	RB(16,3)	=B(16,3)	OS(3), OR(3), OUT(3)=
On Board I/O 4			16	4	RB(16,4)	=B(16,4)	OS(4), OR(4), OUT(4)=
On Board I/O 5			32	5	RB(16,5)	=B(16,5)	OS(5), OR(5), OUT(5)=
On Board I/O 6			64	6	RB(16,6)	=B(16,6)	OS(6), OR(6), OUT(6)=
On Board I/O 7			128	7	RB(16,7)	=B(16,7)	OS(7), OR(7), OUT(7)=
On Board I/O 8			256	8	RB(16,8)	=B(16,8)	OS(8), OR(8), OUT(8)=
On Board I/O 9			512	9	RB(16,9)	=B(16,9)	OS(9), OR(9), OUT(9)=
On Board I/O 10			1024	10	RB(16,10)	=B(16,10)	OS(10), OR(10), OUT(10)=
Not Fault Output			2048	11	RB(16,11)	=B(16,11)	
Drive Enable Input			4096	12	RB(16,12)	=B(16,12)	
Reserved 13			8192	13	RB(16,13)	=B(16,13)	
Reserved 14			16384	14	RB(16,14)	=B(16,14)	
Reserved 15			32768	15	RB(16,15)	=B(16,15)	

Status Word 16: On Board Local I/O Status - Class 6 Motor

Description	Bit	To Clear	Value	Bit	To read	Assign	Related Commands
On Board input 0			1	0	RB(16,0)	=B(16,0)	
On Board input 1			2	1	RB(16,1)	=B(16,1)	
On Board input 2			4	2	RB(16,2)	=B(16,2)	
On Board input 3			8	3	RB(16,3)	=B(16,3)	
On Board I/O 4*			16	4	RB(16,4)	=B(16,4)	OS(4), OR(4), OUT(4)=
On Board I/O 5*			32	5	RB(16,5)	=B(16,5)	OS(5), OR(5), OUT(5)=
On Board input 6			64	6	RB(16,6)	=B(16,6)	
Drive Enable Input			128	7	RB(16,7)	=B(16,7)	
Brake output (default), see EOBK for options.			256	8	RB(16,8)	=B(16,8)	EOBK(), BRKRLS, BRKENG, BRKSRV, BRKTRJ. OS(8), OR(8), OUT(8)=
Not Fault Output (default), see EOFT for options.			512	9	RB(16,9)	=B(16,9)	EOFT(), OS(9), OR(9), OUT(9)=
Reserved 10			1024	10	RB(16,10)	=B(16,10)	
Reserved 11			2048	11	RB(16,11)	=B(16,11)	
Reserved 12			4096	12	RB(16,12)	=B(16,12)	
Reserved 13			8192	13	RB(16,13)	=B(16,13)	
Reserved 14			16384	14	RB(16,14)	=B(16,14)	
Reserved 15			32768	15	RB(16,15)	=B(16,15)	

*Consult specific product guide; not all Class 6 models support output on this particular I/O.

Status Word 17: Expanded I/O Status - D-Style AD1 Motor

Description	Bit	To Clear	Value	Bit	To read	Assign	Related Commands
Exp I/O 16			1	0	RB(17,0)	=B(17,0)	OS(16), OR(16), OUT(16)=
Exp I/O 17			2	1	RB(17,1)	=B(17,1)	OS(17), OR(17), OUT(17)=
Exp I/O 18			4	2	RB(17,2)	=B(17,2)	OS(18), OR(18), OUT(18)=
Exp I/O 19			8	3	RB(17,3)	=B(17,3)	OS(19), OR(19), OUT(19)=
Exp I/O 20			16	4	RB(17,4)	=B(17,4)	OS(20), OR(20), OUT(20)=
Exp I/O 21			32	5	RB(17,5)	=B(17,5)	OS(21), OR(21), OUT(21)=
Exp I/O 22			64	6	RB(17,6)	=B(17,6)	OS(22), OR(22), OUT(22)=
Exp I/O 23			128	7	RB(17,7)	=B(17,7)	OS(23), OR(23), OUT(23)=
Exp I/O 24			256	8	RB(17,8)	=B(17,8)	OS(24), OR(24), OUT(24)=
Exp I/O 25			512	9	RB(17,9)	=B(17,9)	OS(25), OR(25), OUT(25)=
Reserved 10			1024	10	RB(17,10)	=B(17,10)	
Reserved 11			2048	11	RB(17,11)	=B(17,11)	
Reserved 12			4096	12	RB(17,12)	=B(17,12)	
Reserved 13			8192	13	RB(17,13)	=B(17,13)	
Reserved 14			16384	14	RB(17,14)	=B(17,14)	
Reserved 15			32768	15	RB(17,15)	=B(17,15)	

Fault and Status Words - DS2020 Combitoric System

This section provides the descriptions of the fault and status words for the DS2020 Combitoric system. For SmartMotor status words, see Status Words - SmartMotor on page 921.

The DS2020 Combitoric system contains the fault/status information:

- Fault words, 0-2, which are reported through the RFAUSTS command, and whose fault reaction is set through the FSAD command.
- Status words, 0-6, which are reported and assigned like other SmartMotor status words.

Refer to the next sections for details on the Fault and Status words.

Fault Words

During its operation, the DS2020 Combitoric system can signal more than 70 faults, which are organized into three 32-bit fault words.

Command	Description	Report Command	Notes
-	Returns the fault status word x	RFAUSTS(x)	x: 0 to 2
FSAD(n,m)	Set reaction m (see the list below) to fault number n (see the fault columns in the Fault Tables section)	RFSAD(n)	n: 1 to 73 m: 0 to 4

For each fault, the desired reaction m can be selected:

- | m | Reaction |
|-----|---|
| 0 | None |
| 1 | Send CANopen emergency message |
| 2 | Disable power stage |
| 3 | Slow down ramp |
| 4 | Quick stop ramp |
| 127 | Disable power stage for hard faults, not selectable or editable |
- For hard faults, the reaction is always disable power stage. These faults respond to RFSAD(n) with 127; they behave exactly like faults with code 2.
 - Reactions 2, 3, 4 and 127 also send an emergency message.
 - Reactions 3, 4, once the ramp is terminated, disable the drive. These two commands act like X, S with that difference.

Fault Tables

The full list of standard DS2020 faults are shown in these tables; there is one table for each fault word 0 through 2. Note that:

- Red faults cannot occur in the DS2020 Combitoric system with resolver motor.
- The Fault column is the number that identifies the fault in the FSAD and RFSAD commands; the reaction to a particular fault is based on the m value specified with the FSAD command.

- Hard faults, 1-8, 12, 13 and 19-21 (shown in Fault Word 0, bold text), are coded with reaction 127 and cannot be changed by the user.

Fault Word 0

bit	Fault	Fault Name	Description	Default Reaction
0	0	Reserved	Reserved	-
1	1	Short-circuit phase U low	IGBT fault phase U low	127
2	2	Short-circuit phase U high	IGBT fault phase U high	127
3	3	Short-circuit phase V low	IGBT fault phase V low	127
4	4	Short-circuit phase V high	IGBT fault phase V high	127
5	5	Short-circuit phase W low	IGBT fault phase W low	127
6	6	Short-circuit phase W high	IGBT fault phase W high	127
7	7	Short-circuit IGBT recovery	Recovery IGBT fault	127
8	8	Gate undervoltage	Under voltage of IGBT gate	127
9	9	DC link undervoltage	Bus voltage too low	2
10	10	DC link overvoltage	Bus voltage too high	2
11	11	Drive overtemperature	Temperature of the drive is too high	2
12	12	STO 1 low voltage	Supply of STO 1 circuit not detected (fault active in every CiA402 state)	127
13	13	STO 2 low voltage	Supply of STO 2 circuit not detected (fault active in every CiA402 state)	127
14	14	EEPROM fault	Reading error on the EEPROM or content not valid	2
15	15	Software watchdog	Software alarm	2
16	16	Parameter init. error	Initialization error	2
17	17	Node ID data memory	Not used	2
18	18	User data memory	"Customer parameters" region of the memory corrupted / not configured	2
19	19	Restore data memory	Not used	127
20	20	Factory data memory	"Factory parameters" region of the memory corrupted / not configured	127
21	21	Calibration data memory	Not used	127
22	22	Diagnosis data memory	"Diagnostic parameters" region of memory corrupted/not configured	0
23	23	Brake feedback fault	Brake status signal inconsistent	2
24	24	Motor temperature warning	Motor temperature warning	1
25	25	Motor overtemperature	Motor temperature fault	2
26	26	Missing feedback config.	Standard transducer interface enabled but not configured	2
27	27	Transducer general fault	Initialization error or transducer not detected	2
28	28	Sincos values fault	Sinusoidal signal amplitude inconsistent	2
29	29	Hiperface position conflict	Digital position (Hiperface protocol) inconsistent with calculated position	2
30	30	Hiperface status error	Encoder status error (Hiperface protocol)	2
31	31	Hiperface transmit error	Encoder transmission error (Hiperface protocol)	2

Fault Word 1

bit	Fault	Fault Name	Description	Default Reaction
0	32	Hiperface receive error	Encoder reception error (Hiperface protocol)	2
1	33	EnDat 22 warning message	Warning message from EnDat 22 encoder	2
2	34	EnDat 22error 1 message	Type 1 error message from EnDat 22 encoder	2
3	35	EnDat 22 error 2 message	Type 2 error message from EnDat 22 encoder	2
4	36	EnDat 22 CRC error	CRC error from EnDat 22 encoder	2
5	37	EnDat 22 position not ready	Position error - not ready from EnDat 22 encoder	2
6	38	EnDat 22 not ready for strobe	Strobe error - not ready from EnDat 22 encoder	2
7	39	Resolver sync. fault	Resolver signal synchronization error (phase)	2
8	40	Resolver signals fault	Signal resolver level error (amplitude)	2
9	41	Synchronization error	Irregular frequency of internal interrupt	2
10	42	Interrupt time exceeded	Internal interrupt signal not detected	2
11	43	Task time exceeded	The execution time of the task has exceeded maximum time	2
12	44	Velocity limit	Maximum speed exceeded	2
13	45	Excessive position error	Following position error	2

bit	Fault	Fault Name	Description	Default Reaction
14	46	Position reference limit	Not used	1
15	47	EtherCAT_link_fault	Link EtherCAT not detected	2
16	48	EtherCAT_communication_fault	EtherCAT communication generic error	2
17	49	EtherCAT_rpdo_time_out	Time out received PDO (EtherCAT)	2
18	50	EtherCAT_rpdo_data	Data error received PDO (EtherCAT)	2
19	51	EtherCAT_tpdo_time_out	Time out transmitted PDO (EtherCAT)	2
20	52	EtherCAT_tpdo_data	Data error transmitted PDO (EtherCAT)	2
21	53	internal_communication_fault	Not Used	2
22	54	internal_communication_heartbeat_error	Not Used	2
23	55	internal_receive_pdo_time_out	Not Used	2
24	56	internal_transmit_pdo_time_out	Not Used	2
25	57	Phases_not_ok	Not Used	2
26	58	Overcurrent occurred	Overcurrent fault	2
27	59	CAN communication fault	CAN communication generic error	2
28	60	CAN RPDO0 timed out	Time out one received PDO (CAN)	2
29	61	CAN RPDO1 timed out	Time out two received PDO (CAN)	2
30	62	CAN RPDO0 data	Data error one received PDO (CAN)	2
31	63	CAN RPDO1 data	Data error two received PDO (CAN)	2

Fault Word 2

bit	Fault	Fault Name	Description	Default Reaction
0	64	CAN TPDO0 timed out	Time out one transmitted PDO (CAN)	2
1	65	CAN TPDO1 timed out	Time out two transmitted PDO (CAN)	2
2	66	CAN TPDO0 data	Data error one transmitted PDO (CAN)	2
3	67	CAN TPDO1 data	Data error two transmitted PDO (CAN)	2
4	68	CAN life guard error	Lifeguard time expired (CAN)	2
5	69	CAN sync consumer timed out	Sync message timeout (CAN)	2
6	70	External signal fault	Fault triggered by digital input	2
7	71	Overfrequency fault	Over frequency (no-dual use limitation)	1
8	72	Historical positive H/W limit	Positive limit switch asserted	4
9	73	Historical negative H/W limit	Negative limit switch asserted	4

Status Words

This section describes the DS2020 Combitoric system's 16-bit status words, which can be reported and assigned in the same manner as the other SmartMotor status words. For more details, see Status Words - SmartMotor on page 921.

Status Word 0: Primary Fault/Status Indicator

Status Word 0 is compliant to SmartMotor status word 0, except for bit 9 (dE/dt is not available on the DS2020 Combitoric system).

bit	Fault	Fault Name	Description
0	-	Drive ready	Drive is correctly powered, not in fault, and the motor can be moved
1	-	Motor is off	Power stage is disabled (no PWM is generated)
2	-	Trajectory in progress	Trajectory generator is generating a trajectory
3	9, 10	Bus voltage fault	Overvoltage or undervoltage condition happened (check SW2, bit 8,9 to know which of them)
4	58	Overcurrent occurred	Drive tries to draw more current than its maximum
5	11, 25	Excessive temperature fault	Motor or drive temperature is too high (check SW2, bit 10,11 to know which of them)
6	45	Excessive position error	Position error EA exceeded the limit EL
7	44	Velocity limit	Velocity exceeded the maximum motor speed.
8	-	Real-time temperature limit	Motor or drive temperature exceed the limit value.

Appendix: Status Word 1: Current CiA DS402 State

bit	Fault	Fault Name	Description
9	-	-	-
10	-	Positive H/W limit enabled	Positive H/W limit switch is enabled and checked
11	-	Negative H/W limit enabled	Negative H/W limit switch is enabled and checked
12	72	Historical positive H/W limit	Positive H/W limit switch has been hit (latched)
13	73	Historical negative H/W limit	Negative H/W limit switch has been hit (latched)
14	-	Positive H/W limit asserted	Positive H/W limit switch is asserted
15	-	Negative H/W limit asserted	Negative H/W limit switch is asserted

Status Word 1: Current CiA DS402 State

Status word 1 indicates the current CiA DS402 state (see the *Moog Animatics DS2020 Combitoric™ Installation and Startup Guide*).

bit	Fault	Fault Name	Description
0	-	CiA402: Not ready to switch on	CiA DS402 state
1	-	CiA402: Switch on disabled	CiA DS402 state
2	-	CiA402: Ready to switch on	CiA DS402 state
3	-	CiA402: Switched on	CiA DS402 state
4	-	CiA402: Operation enabled	CiA DS402 state
5	-	CiA402: Quick stop active	CiA DS402 state
6	-	CiA402: Fault reaction active	CiA DS402 state
7	-	CiA402: Fault	CiA DS402 state
8	-	Position mode	Drive is in Position Mode
9	-	Velocity mode	Drive is in Velocity Mode
10	-	TG: Acceleration	Acceleration part of trajectory is in execution
11	-	TG: Constant speed	Constant speed part of trajectory is in execution
12	-	TG: Deceleration	Deceleration part of trajectory is in execution
13	12	STO 1 low voltage fault	On if supply of STO 1 circuit not detected (historical)
14	13	STO 2 low voltage fault	On if supply of STO 2 circuit not detected (historical)
15	-	-	-

Status Word 2: Control and Hardware Faults

Status word 2 reports control and hardware faults.

bit	Fault	Fault Name	Description
0	1	Short-circuit phase U low	IGBT fault phase U low
1	2	Short-circuit phase U high	IGBT fault phase U high
2	3	Short-circuit phase V low	IGBT fault phase V low
3	4	Short-circuit phase V high	IGBT fault phase V high
4	5	Short-circuit phase W low	IGBT fault phase W low
5	6	Short-circuit phase W high	IGBT fault phase W high
6	7	Short-circuit IGBT recovery	Recovery IGBT fault
7	8	Gate undervoltage	IGBT gate voltage too low
8	9	DC link undervoltage	Bus voltage too low
9	10	DC link overvoltage	Bus voltage too high
10	11	Drive overtemperature	Drive temperature too high
11	24	Motor temperature warning	Motor temperature near limit
12	25	Motor overtemperature	Motor temperature too high
13	70	External signal fault	Fault triggered by digital input
14	71	Overfrequency fault	Generated current frequency exceeded "no dual-use" regulation limit
15	-	-	-

Status Word 3: Position/Velocity sensor and Brake Feedback Faults

Status word 3 reports all the faults related to position/velocity sensor (that can be resolver or encoder) and brake feedback.

Appendix: Status Word 4: Communication Faults

bit	Fault	Fault Name	Description
0	23	Brake feedback fault	Brake status signal inconsistent
1	26	Missing feedback config.	Standard transducer interface enabled but not configured
2	27	Transducer general fault	Initialization error or transducer not detected
3	28	Sincos values fault	Sinusoidal signal amplitude inconsistent
4	29	Hiperface position conflict	Digital position (Hiperface protocol) inconsistent with calculated position
5	30	Hiperface status error	Encoder status error (Hiperface protocol)
6	31	Hiperface transmit error	Encoder transmission error (Hiperface protocol)
7	32	Hiperface receive error	Encoder reception error (Hiperface protocol)
8	33	EnDat 22 warning message	Warning message from EnDat 22 encoder
9	34	EnDat 22 error 1 message	Type 1 error message from EnDat 22 encoder
10	35	EnDat 22 error 2 message	Type 2 error message from EnDat 22 encoder
11	36	EnDat 22 CRC error	CRC error from EnDat 22 encoder
12	37	EnDat 22 position not ready	Position error - not ready from EnDat 22 encoder
13	38	EnDat 22 not ready for strobe	Strobe error - not ready from EnDat 22 encoder
14	39	Resolver sync. fault	Resolver signal synchronization error (phase)
15	40	Resolver signals fault	Signal resolver level error (amplitude)

Status Word 4: Communication Faults

Status word 4 reports all the communication faults.

bit	Fault	Fault Name	Description
0	59	CAN communication fault	CAN communication generic error
1	60	CAN RPDO0 timed out	Time out first received PDO (CAN)
2	61	CAN RPDO1 timed out	Time out second received PDO (CAN)
3	62	CAN RPDO0 data	Data error first received PDO (CAN)
4	63	CAN RPDO1 data	Data error second received PDO (CAN)
5	64	CAN TPDO0 timed out	Time out first transmitted PDO (CAN)
6	65	CAN TPDO1 timed out	Time out second transmitted PDO (CAN)
7	66	CAN TPDO0 data	Data error first transmitted PDO (CAN)
8	67	CAN TPDO1 data	Data error second transmitted PDO (CAN)
9	68	CAN life guard error	Life guard time expired (CAN)
10	69	CAN sync consumer timed out	Sync message timeout (CAN)
11	-	-	-
12	-	-	-
13	-	-	-
14	-	-	-
15	-	-	-

Status Word 5: Software and Memory Faults

Status word 5 reports software and memory faults.

bit	Fault	Fault Name	Description
0	14	EEPROM fault	Reading error on the EEPROM or content not valid.
1	18	User data memory	"Customer parameters" region of memory corrupted / not configured
2	20	Factory data memory	"Factory parameters" region of memory corrupted / not configured
3	22	Diagnosis data memory	"Diagnostic parameters" region of memory corrupted/not configured
4	41	Synchronization error	Irregular frequency of internal interrupt
5	42	Interrupt time exceeded	Internal interrupt signal not detected
6	43	Task time exceeded	The execution time of the task has exceeded maximum time
7	-	-	-
8	-	-	-
9	-	-	-
10	-	-	-
11	-	-	-
12	-	-	-
13	-	-	-

bit	Fault	Fault Name	Description
14	-	-	-
15	-	-	-

Status Word 6: I/O States

Status word 6 reports I/O states.

bit	Fault	Fault Name	Description
0	-	I/O 0	On when 24V applied
1	-	I/O 1	On when 24V applied
2	-	I/O 2	On when 24V applied
3	-	I/O 3	On when 24V applied
4	-	I/O 4	On when 24V applied
5	-	I/O 5	On when 24V applied
6	-	-	-
7	-	-	-
8	-	-	-
9	-	-	-
10	-	-	-
11	-	-	-
12	-	-	-
13	-	-	-
14	-	STO 1 status	On when STO 1 is powered
15	-	STO 2 status	On when STO 2 is powered

Torque Curves

Understanding Torque Curves

Each set of torque curves depicts the limits of both continuous and peak torque for a given SmartMotor™ over the full range of speed.

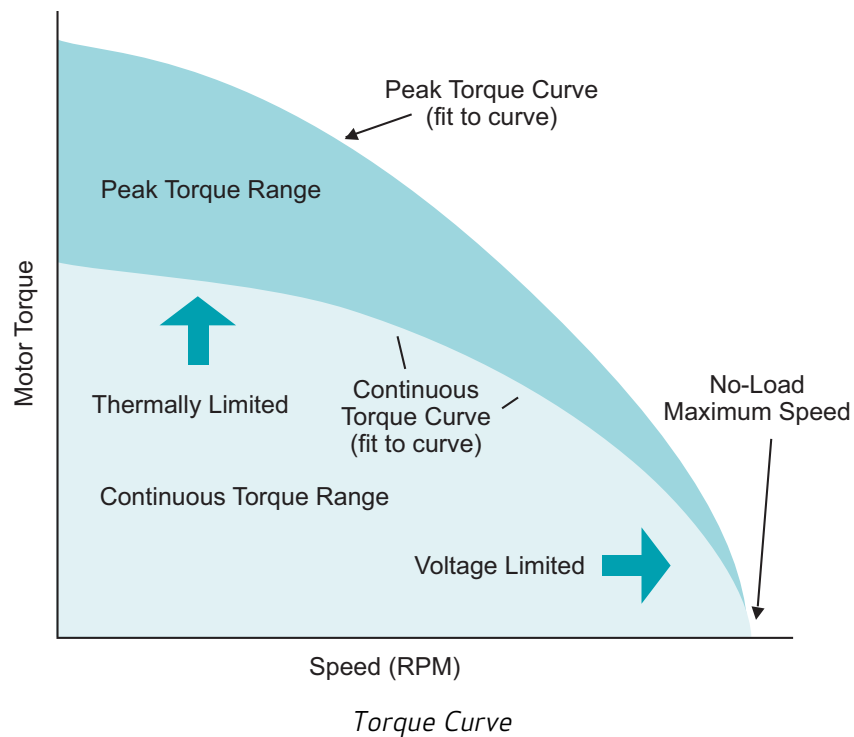
Peak Torque

The peak torque curve is derived from dyno (dynamometer) testing. It is the point at which peak current limit hardware settings of the drive prevent further torque in an effort to protect the drive-stage components.

Continuous Torque

The continuous torque curve is also derived from dyno testing. It is the point at which the temperature rises from an ambient of 25°C to the designed thermal limit.

For example, the motor will be placed on the dyno tester and set to operate at 1000 RPM continuously with the load slowly increased until the controller reaches its maximum sustained thermal limit. This limit is either 70°C or 85°C depending on the model number. All Class 5 SmartMotor servos are set to 85°C.



The lower-right side of the curve is limited by supply voltage. This is the point at which Back EMF suppresses any further speed increase. Higher supply voltages will shift the zero torque point of the curve further to the right.

Ambient Temperature Effects on Torque Curves and Motor Response:

If the motor is operated in an environment warmer than 25°C, it will reach its thermal limit faster for a given load and further limit continuous torque. Therefore, any given motor torque curve *must be* linearly derated for a given ambient temperature from 25°C to 85°C for all Class 5 SmartMotor servos.

Supply Voltage Effects on Torque Curves and Motor Response:

Higher voltages have a two-fold effect on torque curves. As mentioned previously, raising the voltage shifts the curve to the right; it also allows higher current into the drive. However, torque curves depict maximum allowable torque at a given velocity.

If you double the supply voltage, the motor can sustain twice the original velocity. Acceleration is also increased due to an increase in the peak torque curve. This may potentially be a significant reduction of time to complete moves due to the $a \cdot t^2$ term in kinematic equations. This is useful for high-speed indexing and fast start/stop motion.

NOTE: All torque curves shown in the *Moog Animatics Product Catalog* also show the shaft output power curves.

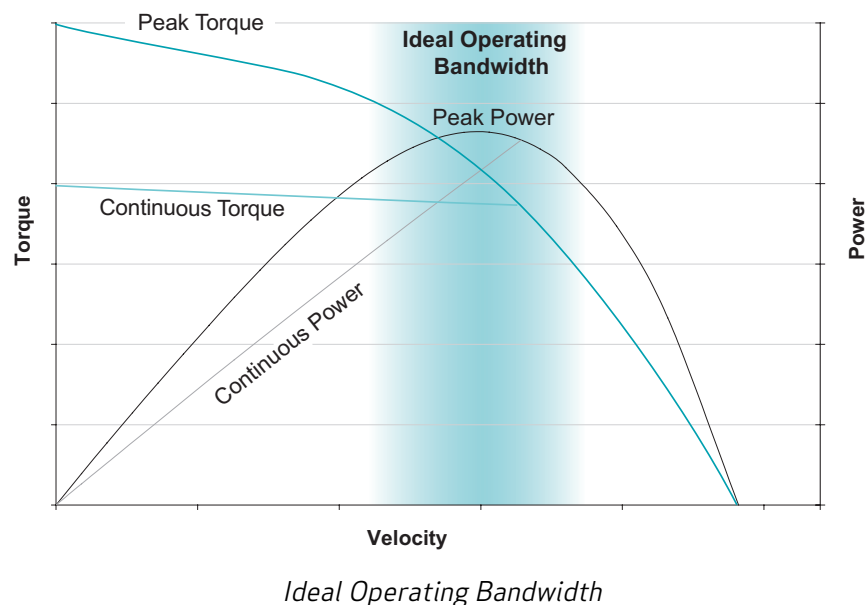
Power can be calculated with this equation:

$$\text{Power (W}^*) = \text{Torque (N.m)} \times \text{Speed (RPM)} / 9.5488$$

**In some versions of Moog Animatics literature, this was incorrectly shown as "kW".*

For any given mechanical system being moved by a SmartMotor, it is ideal to ensure the motor is running within its optimum performance range (see the next figure). Through proper mechanical system design, this can be achieved by adjusting one or more of these items:

- Gear reduction
- Belt reduction
- Lead screw pitch
- Pinion gear diameter



Ideal Operating Bandwidth

Example 1: Rotary Application

Suppose you have a load that requires 300 RPM at the output of a gear head, and the optimum speed range for the motor is 2100 RPM.

Divide the optimum operating speed by the load speed to get the ideal gear reduction. In this case: $2100 \text{ RPM} / 300 \text{ RPM} = 7$. So a 7:1 gear reduction would allow the motor to operate in its most efficient range.

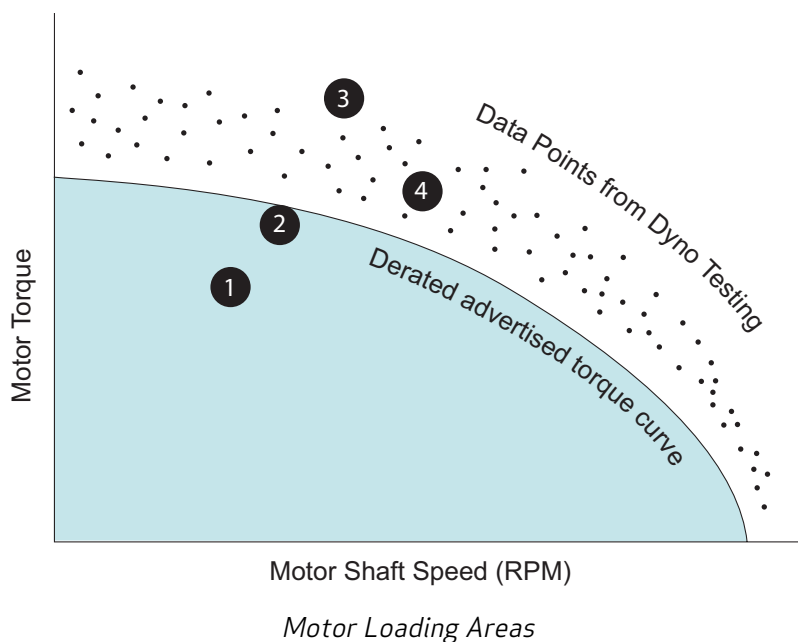
Example 2: Linear Application

Suppose you need to run at 100 mm/sec using a ball screw, and the motor has an ideal range of 3000 RPM. $3000 \text{ RPM} / 60 = 50$ rotations per second (RPS). 100 mm/sec divided by 50 RPS is 2 mm per rotation. Therefore, an ideal pitch would be 2 mm.

Dyno Test Data vs. the Derated Torque Curve

NOTE: For any given product model number, there may be variations of as much as $\pm 10\%$.

The next figure depicts data points collected from dyno testing of a given SmartMotor model. A best-fit torque curve is created from these data points and is then derated to at least 5% below the worst case data points. The derated curve is shown in the *Moog Animatics Product Catalog*. This means that within any given model number, *every* motor sold will perform at or higher than the advertised torque. Theoretically, *all* motors should be no less than 5% higher than advertised and may be more than 20% higher.



The diagram shows motor loading in four areas:

1. This is ideal and depicts a load within the normal operating range of the motor. The motor should operate well and have no problems for many years.
2. The load is very close to the operating limit. The motor will run quite warm as compared to Point 1.

3. The load exceeds the advertised level and exceeds +10% expected range of possible torque capabilities. In this case, the motor will most likely either overheat quickly and fault out or immediately generate a position error because it simply does not have enough power to support the load demand.



WARNING: Using an undersized motor can cause unpredictable machine performance and is a potential safety hazard (see Motor Sizing on page 32).

4. The load exceeds the advertised operating limit of the motor. However, due to data scatter and derating, there may be some motors that will work and others that do not. This is because it falls within the range of $\pm 10\%$ variation for motors for a given size. This can result in major problems for the machine builder.

For example, imagine designing a machine that operates in this range. Then you replicate that machine with many of them running on a production floor. One day, a motor at the lower end of the $\pm 10\%$ expected variation is placed on a new machine and that motor generates spurious drive faults. It appears as though the motor is malfunctioning because "all the other motors work just fine." This is unfortunate because, in reality, all motors were undersized in the machine design and are now operating outside of their advertised limits.

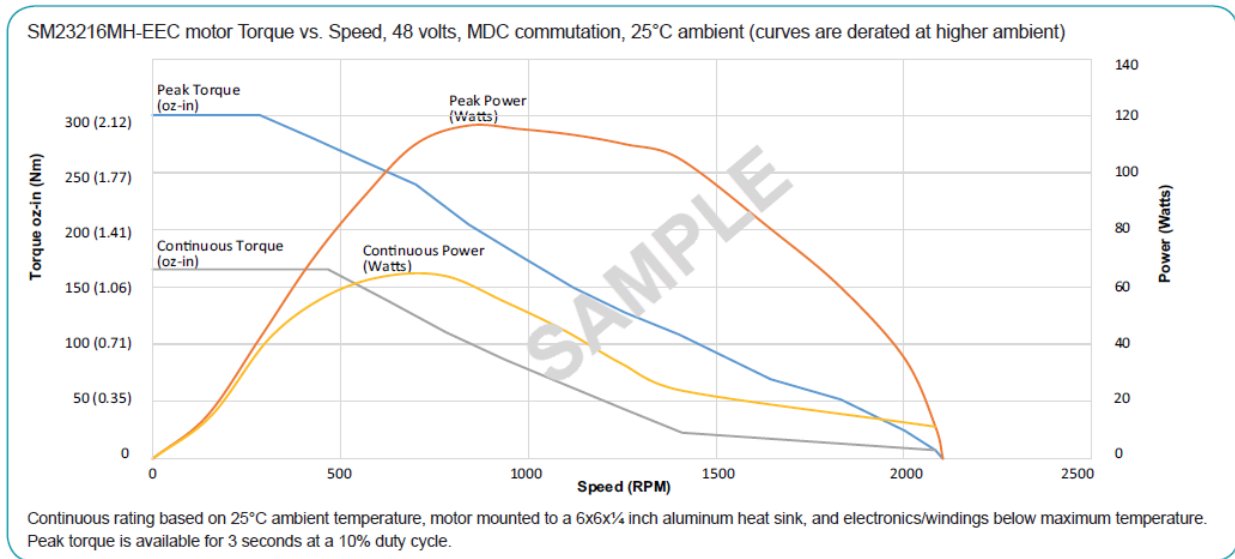
That is why it is important to properly calculate load torque to ensure the correct motor is designed into the application (refer to the next section). Never assume that testing of one motor means all motors of that size will work — it is simply not the case. You should never proceed without performing proper load calculation and motor sizing. The goal is to have all motors operating below the advertised limits, which will ensure reliable operation and long motor life.

Proper Sizing and Loading of the SmartMotor

It is important to properly calculate load torque to ensure the correct SmartMotor is selected and designed into the application. Consider the next sample figure. If properly sized/loaded, the motor can run at or under the Continuous Torque limit continuously, assuming 25°C ambient temperature. Further, the motor can tolerate intermittent operation above the Continuous Torque limit up to the Peak Torque limit for brief periods. However, that additional capacity may reduce as a function of time when operating above the Continuous Torque limit.

In order to protect the motor, Moog Animatics has designed in safeguards to limit current that may engage when the motor is operated for a sustained or accumulated brief periods above the continuous ratings (i.e., operating above capacity for torque/time). This could lead to position error or position error faults.

NOTE: These safeguards DO NOT indicate a defective motor. Rather, they are an indication that the motor may not be properly sized for the intended application, or that other design or environmental factors are affecting motor performance, such as unintended axial or radial forces acting on the load, elevated ambient air temperature, improper mounting that prevents adequate heat sinking, etc.



Sample Power Chart

To ensure that your SmartMotor successfully performs as intended:

- Select the proper power supply
- Use the proper electrical interface
- Properly size the motor for the intended application
- Consider the thermal environment
- Adhere to the proper mechanical and environmental implementation

For more details on these items, see the SmartMotor Success Checklist in the *Moog Animatics Product Catalog*.

Also, refer to these topics (in this section):

- Understanding Torque Curves on page 938
- Dyno Test Data vs. the Derated Torque Curve on page 940

SmartMotor Troubleshooting

This topic provides information on SmartMotor troubleshooting "first steps" and also a troubleshooting flowchart.

NOTE: This information is extracted from the *Troubleshooting with Status Bits and SMI Tools Application Note*. For full details, please see that document on the Moog Animatics website.

Troubleshooting - First Steps

When diagnosing any problem, perform these basic first steps to reset and check the equipment.

1. Restart the motor.
2. Restart any associated devices (PLCs, HMIs, PCs).
3. Download a fresh copy of the program (especially if you've been making changes).
4. Check the wiring – make sure all connectors are secure and visually inspect all cables for any signs of wear.

After completing those steps, if the problem still hasn't been resolved, it's time for a deeper investigation to isolate the root cause. If you are unsure of where to start, you can use the steps describe below. If you are trying to troubleshoot a specific error, please skip the next section.

NOTE: If possible, remove the motor from the machine and connect it to SMI using separate power supplies and cables.

1. Are LEDs on?

If not, double check your power supply and cables to make sure proper connections are being made. If you are measuring the appropriate voltage at the motor end of your cable, it's likely the motor has been damaged and requires an RMA. If yes, consult the Understand the LEDs topic in your [SmartMotor installation guide](#).

NOTE: Class 5 D-style motors with the DE option and all M-style motors require separate control and amplifier power. For Class 6 D-style motors, separate power is not required but is highly recommended. See your [SmartMotor installation guide](#) for details.

- If you are using a Class 5 -DE option motor, please ensure power is being supplied to both pin 15 and A1.
- If you are stuck in the bootloader, [contact Moog Animatics](#) for assistance.
- If the LEDs indicate you are not stuck in the bootloader, please move to the next step.

2. Can you establish communications?

If not, make sure your communication settings and hardware are correct. Run the Lockup Wizard in SMI.

NOTE: If you can only establish communication using the SMI Lockup Wizard, your user program may be setting up the port incorrectly. Clearing the EEPROM will reset the port settings. However, be aware that clearing the EEPROM will also erase the user program.

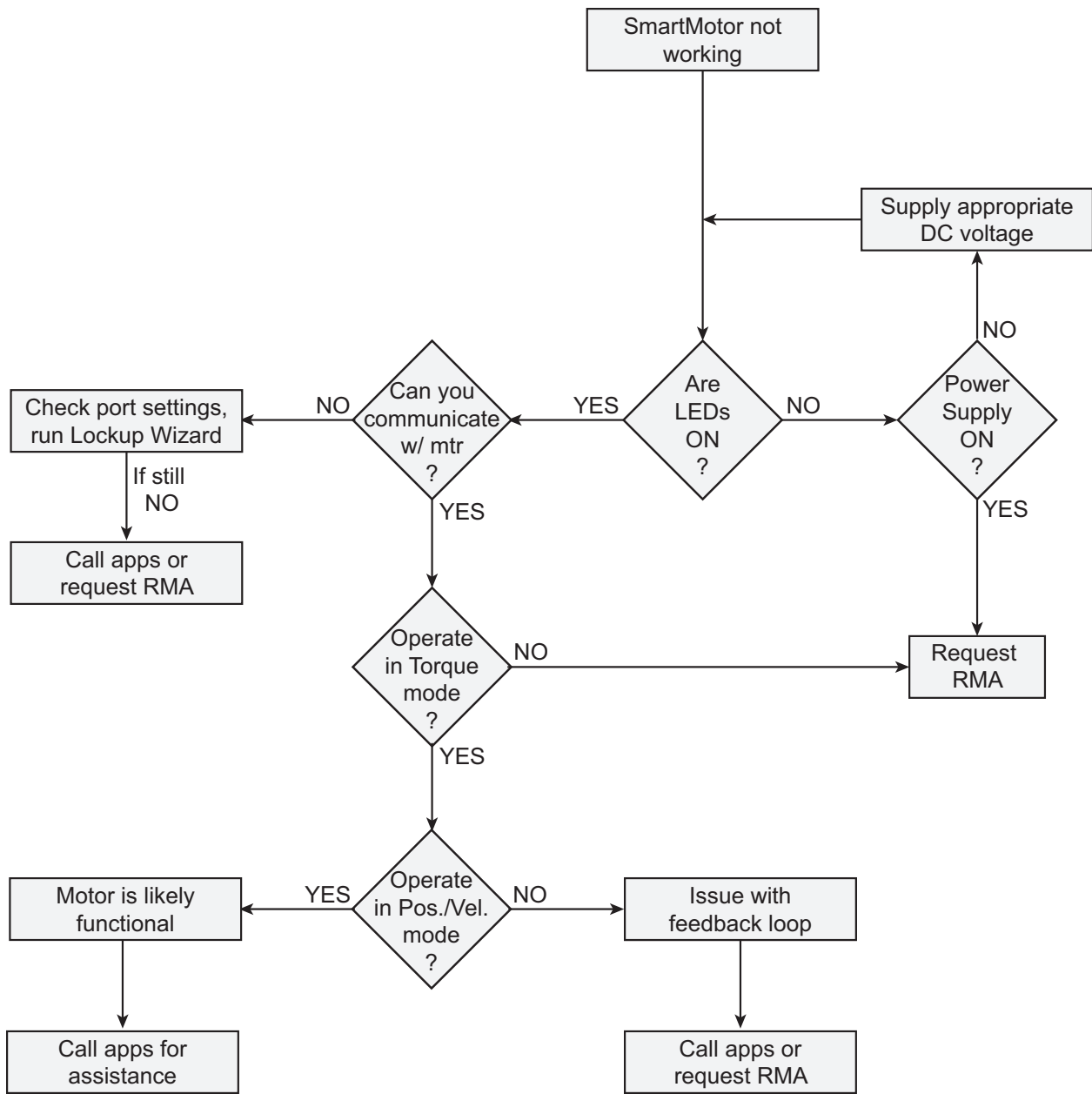
If you still can't establish communications, then it's possible the motor has been damaged and requires an RMA. If you can, then move to the next step.

3. Will the motor operate in torque mode?

Open up the SmartMotor Playground in SMI and move the motor in torque mode. Be aware, torque mode is open loop and can cause undesirable motion if precaution is not taken. Please be conscious of safety mechanisms like hardware limit switches, software limits, and E-stops. If not, make sure that there are no errors and the drive ready bit is active. Also, make sure that there is nothing physically impeding the shaft. If you cannot achieve motion, it's likely the motor requires an RMA. If you can achieve motion, then the drive stage is likely functional. Move to the next step.

4. Will the motor operate in position mode?

Again, use the SmartMotor Playground to command motion in either position mode or velocity mode. If the motor fails to complete a move, it's possible the motor is damaged and requires an RMA. If this step works, it's safe to assume the motor is functional. At this point, it's time to start exploring other components in the system or the specific status bits described in the next section.



SmartMotor Troubleshooting Flowchart

For more details on this process, see the *Troubleshooting with Status Bits and SMI Tools Application Note* on the Moog Animatics website.

Commands Listed Alphabetically

This section shows an alphabetical listing of all available commands and their descriptions.

NOTE: A superscript "R" character preceding the command indicates there is a corresponding "report" version of that command.

(Single Space Character) *Single Space Delimiter and String Terminator (see page 248)*

R a...z *32-Bit Variables (see page 249)*

R aa...zz *32-Bit Variables (see page 249)*

R aaa...zzz *32-Bit Variables (see page 249)*

R ab[index]=formula *Array Byte [index] (see page 252)*

R ABS(value) *Absolute Value of () (see page 255)*

R AC *Acceleration Commanded (see page 256)*

R ACOS(value) *Arccosine (see page 259)*

R ADDR=formula *Address (for RS-232 and RS-485) (see page 261)*

ADT=formula *Acceleration/Deceleration Target (see page 263)*

ADTS=formula *Acceleration/Deceleration Target, Synchronized (see page 265)*

R af[index]=formula *Array Float [index] (see page 267)*

Ai(enc) *Arm Index Rising Edge (see page 270)*

Aij(enc) *Arm Index Rising Edge Then Falling Edge (see page 272)*

Aj(enc) *Arm Index Falling Edge (see page 274)*

Aji(enc) *Arm Index Falling Edge Then Rising Edge (see page 276)*

R al[index]=formula *Array Long [index] (see page 278)*

R AMPS=formula *Amps, PWM Limit (see page 281)*

R ASIN(value) *Arcsine (see page 284)*

R AT=formula *Acceleration Target (see page 286)*

R ATAN(value) *Arctangent (see page 289)*

R ATOF(index) *ASCII to Float (see page 291)*

ATS=formula *Acceleration Target, Synchronized (see page 292)*

R aw[index]=formula *Array Word [index] (see page 294)*

R B(word,bit) *Status Byte (see page 297)*

R Ba *Bit, Peak Overcurrent (see page 301)*

R BAUD(channel)=formula *Set BAUD Rate (RS-232 and RS-485) (see page 303)*

R Be *Bit, Position Error Limit (see page 305)*

R Bh *Bit, Overheat (see page 307)*

R Bi(enc) *Bit, Index Capture, Rising (see page 309)*

R Bj(enc) *Bit, Index Capture, Falling (see page 312)*

R Bk *Bit, Program EEPROM Data Status (see page 315)*

R Bl *Bit, Left Hardware Limit, Historical (see page 316)*

R Bls *Bit, Left Software Limit, Historical (see page 318)*

R Bm *Bit, Left Hardware Limit, Real-Time (see page 320)*
 R Bms *Bit, Left Software Limit, Real-Time (see page 322)*
 R Bo *Bit, Motor OFF (see page 324)*
 R Bp *Bit, Right Hardware Limit, Real-Time (see page 325)*
 R Bps *Bit, Right Software Limit, Real-Time (see page 327)*
 R Br *Bit, Right Hardware Limit, Historical (see page 329)*
 BREAK *Break from CASE or WHILE Loop (see page 331)*
 BRKENG *Brake Engage (see page 333)*
 BRKRLS *Brake Release (see page 335)*
 BRKSRV *Brake Servo, Engage When Not Servoing (see page 337)*
 BRKTRJ *Brake Trajectory, Engage When No Active Trajectory (see page 339)*
 R Brs *Bit, Right Software Limit, Historical (see page 341)*
 R Bs *Bit, Syntax Error (see page 343)*
 R Bt *Bit, Trajectory In Progress (see page 345)*
 R Bv *Bit, Velocity Limit (see page 347)*
 R Bw *Bit, Wrapped Encoder Position (see page 349)*
 R Bx(enc) *Bit, Index Input, Real-Time (see page 351)*
 C{number} *Command Label (see page 353)*
 R CADDR=formula *CAN Address (see page 355)*
 R CAN, CAN(arg) *CAN Bus Status (see page 357)*
 CANCTL(function,value) *CAN Control (see page 359)*
 CASE formula *Case Label for SWITCH Block (see page 360)*
 R CBAUD=formula *CAN Baud Rate (see page 363)*
 CCHN(type,channel) *Close Communications Channel (RS-232 or RS-485) (see page 365)*
 R CHN(channel) *Communications Error Flag (see page 367)*
 R CLK=formula *Millisecond Clock (see page 369)*
 COMCTL(function,value) *Serial Communications Control (see page 370)*
 R COS(value) *Cosine (see page 372)*
 R CP *Cam Pointer for Cam Table (see page 374)*
 CTA(points,seglen[,location]) *Cam Table Attribute (see page 376)*
 CTE(table) *Cam Table Erase (see page 378)*
 R CTR(enc) *Counter, Encoder, Step and Direction (see page 380)*
 R CTT *Cam Table Total in EEPROM (see page 382)*
 CTW(pos[,seglen][,user]) *Cam Table Write Data Points (see page 383)*
 R DEA *Derivative Error, Actual (see page 386)*
 DEFAULT *Default Case for SWITCH Structure (see page 388)*
 R DEL=formula *Derivative Error Limit (see page 390)*
 DELM(arg) *Derivative Error Limit Mode (see page 392)*
 R DFS(value) *Dump Float, Single (see page 393)*
 DITR(int) *Disable Interrupts (see page 394)*
 R DT=formula *Deceleration Target (see page 396)*

DTS=formula *Deceleration Target, Synchronized (see page 399)*
R EA *Error Actual (see page 401)*
ECHO *Echo Incoming Data on Communications Port 0 (see page 403)*
ECHO0 *Echo Incoming Data on Communications Port 0 (see page 405)*
ECHO1 *Echo Incoming Data on Communications Port 1 (see page 406)*
ECHO_OFF *Turn Off Echo on Communications Port 0 (see page 407)*
ECHO_OFF0 *Turn Off Echo on Communications Port 0 (see page 408)*
ECHO_OFF1 *Turn Off Echo on Communications Port 1 (see page 409)*
ECS(counts) *Encoder Count Shift (see page 410)*
EIGN(...) *Enable as Input for General-Use (see page 412)*
EILN *Enable Input as Limit Negative (see page 415)*
EILP *Enable Input as Limit Positive (see page 417)*
EIRE *Enable Index Register, Encoder Capture (see page 419)*
EIRI *Enable Index Register, Input Capture (see page 421)*
EISM(x) *E-Configure Input as Sync Controller (see page 423)*
EITR(int) *Enable Interrupts (see page 424)*
R EL=formula *Error Limit (see page 426)*
ELSE *IF-Structure Command Flow Element (see page 428)*
ELSEIF formula *IF-Structure Command Flow Element (see page 430)*
ENC0 *Encoder Zero (Close Loop on Internal Encoder) (see page 432)*
ENC1 *Encoder Zero (Close Loop on External Encoder) (see page 433)*
ENCCTL(function,value) *Encoder Control (see page 435)*
ENCD(in_out) *Set Encoder Bus Port as Input or Output (see page 437)*
END *End Program Code Execution (see page 439)*
ENDIF *End IF Statement (see page 441)*
ENDS *End SWITCH Structure (see page 443)*
EOBK(IO) *Enable Output, Brake Control (see page 445)*
EOFT(IO) *Enable Output, Fault Indication (see page 447)*
EIDX(number) *Encoder, Output Index (see page 449)*
R EPTR=formula *EEPROM Pointer (see page 450)*
R ERRC *Error Code, Command (see page 451)*
R ERRW *Communication Channel of Most Recent Command Error (see page 453)*
R ETH(arg) *Get Ethernet Status and Errors (see page 455)*
ETHCTL(function,value) *Control Industrial Ethernet Network Features (see page 456)*
F *Force Into PID Filter (see page 457)*
R FABS(value) *Floating-Point Absolute Value of () (see page 463)*
R FAUSTS(x) *Returns Fault Status Word (see page 459)*
R FD=expression *Resolution to Set Units of Position/Velocity/Acceleration (see page 461)*
FSA(cause,action) *Fault Stop Action (see page 465)*
R FSAD(n,m) *Set Reaction to Fault (see page 467)*
R FSQRT(value) *Floating-Point Square Root (see page 469)*
R FW *Firmware Version (see page 471)*
G *Start Motion (GO) (see page 473)*
R GETCHR *Next Character from Communications Port 0 (see page 476)*

R GETCHR1 *Next Character from Communications Port 1 (see page 478)*
 GOSUB(label) *Subroutine Call (see page 480)*
 GOTO(label) *Branch Program Flow to a Label (see page 482)*
 R GROUP(function,value) *Group Address Settings (see page 484)*
 GS *Start Synchronized Motion (GO Synchronized) (see page 487)*
 R HEX(index) *Decimal Value of a Hex String (see page 489)*
 R HM_ADT=formula *Homing Accel/Decel Target (see page 491)*
 R HM_MTHD=formula *Homing Method (see page 492)*
 R HM_OSET=formula *Homing Offset (see page 496)*
 R HM_VTS=formula *Homing Velocity Target to Switch (see page 498)*
 R HM_VTZ=formula *Homing Velocity Target to Zero (see page 500)*
 R I(enc) *Index, Rising-Edge Position (see page 502)*
 R IDENT=formula *Set Identification Value (see page 504)*
 IF formula *Conditional Program Code Execution (see page 506)*
 R IN(...) *Specified Input (see page 509)*
 R INA(...) *Specified Input, Analog (see page 512)*
 IPCTL(function,"string") *Set IP Address, Subnet Mask or Gateway (see page 515)*
 ITR(Int#,StatusWord,Bit#,BitState,Label#) *Interrupt Setup (see page 517)*
 ITRD *Interrupt Disable, Global (see page 520)*
 ITRE *Enable Interrupts, Global (see page 522)*
 R J(enc) *Index, Falling-Edge Position (see page 524)*
 R KA=formula *Constant, Acceleration Feed Forward (see page 526)*
 R KD=formula *Constant, Derivative Coefficient (see page 528)*
 R KG=formula *Constant, Gravitational Offset (see page 530)*
 R KI=formula *Constant, Integral Coefficient (see page 532)*
 R KII=formula *Current Control Loop: Integrator (see page 534)*
 R KL=formula *Constant, Integral Limit (see page 535)*
 R KP=formula *Constant, Proportional Coefficient (see page 537)*
 R KPI=formula *Current Control Loop: Proportional (see page 539)*
 R KS=formula *Constant, Velocity Filter Option (for KD) (see page 540)*
 R KV=formula *Constant, Velocity Feed Forward (see page 542)*
 R LEN *Length of Character Count in Communications Port 0 (see page 544)*
 R LEN1 *Length of Character Count in Communications Port 1 (see page 545)*
 R LFS(value) *Load Float Single (see page 547)*
 LOAD *Download Compiled User Program to Motor (see page 548)*
 LOCKP *Lock Program (see page 551)*
 LOOP *Loop Back to WHILE Formula (see page 553)*
 MC *Mode Cam (Electronic Camming) (see page 555)*
 R MCDIV=formula *Mode Cam Divisor (see page 557)*
 MCE(arg) *Mode Cam Enable () (see page 558)*
 R MCMUL=formula *Mode Cam Multiplier (see page 560)*

MCW(table,point) *Mode Cam Where (Start Point) (see page 562)*
 MDB *Enable TOB Feature (Commutation Mode) (see page 564)*
 MDC *Mode Current (Commutation Mode) (see page 566)*
 MDE *Mode Enhanced (Commutation Mode) (see page 568)*
 MDH *Mode Hybrid (Commutation Mode) (see page 570)*
 MDHV *Mode Hybrid Velocity (Commutation Mode) (see page 572)*
 MDS *Mode Sine (Commutation Mode) (see page 574)*
 MDT *Mode Trap (Commutation Mode) (see page 576)*
 MFO *Mode Follow, Zero External Counter (see page 578)*
 MFA(distance[,m/s]) *Mode Follow Ascend (see page 580)*
 MFCTP(arg1,arg2) *Mode Follow Control Traverse Point (see page 583)*
 MFD(distance[,m/s]) *Mode Follow Descend (see page 585)*
 R MFDIV=formula *Mode Follow Divisor (see page 588)*
 MFH(distance[,m/s]) *Mode Follow, High Ascend/Descend Rate (see page 590)*
 R MFHTP=formula *Mode Follow, High Traverse Point (see page 592)*
 MFL(distance[,m/s]) *Mode Follow, Low Ascend/Descend Rate (see page 594)*
 R MFLTP=formula *Mode Follow, Low Traverse Point (see page 596)*
 R MFMUL=formula *Mode Follow Multiplier (see page 598)*
 MFR *Mode Follow Ratio (see page 600)*
 MFSDC(distance,mode) *Mode Follow, Stall-Dwell-Continue (see page 603)*
 MFSLEW(distance[,m/s]) *Mode Follow Slew (see page 605)*
 R MH *Mode, Homing (see page 607)*
 MINV(arg) *Mode Inverse (Commutation Inverse) (see page 608)*
 R MODE *Mode Operating (see page 610)*
 MP *Mode Position (see page 613)*
 MSO *Mode Step, Zero External Counter (see page 616)*
 MSR *Mode Step Ratio (see page 618)*
 MT *Mode Torque (see page 620)*
 MTB *Mode Torque Brake (see page 622)*
 MV *Mode Velocity (see page 624)*
 NMT *Send NMT State (see page 626)*
 O=formula, O(trj#)=formula *Origin (see page 628)*
 R OC(...) *Output Condition (see page 630)*
 OCHN(...) *Open Channel (see page 632)*
 R OF(...) *Output Fault (see page 634)*
 OFF *Off (Drive Stage Power) (see page 636)*
 OR(value) *Output, Reset (see page 638)*
 OS(...) *Output, Set (see page 640)*
 OSH=formula, OSH(trj#)=formula *Origin Shift (see page 642)*
 OUT(...)=formula *Output, Activate/Deactivate (see page 644)*
 R PA *Position, Actual (see page 646)*
 PAUSE *Pause Program Execution (see page 648)*
 R PC, PC(axis) *Position, Commanded (see page 650)*
 R PI *Pi Constant (see page 653)*

PID# *Proportional-Integral-Differential Filter Rate (see page 654)*
 R PMA *Position, Modulo Actual (see page 657)*
 R PML=formula *Modulo Position Limit (see page 659)*
 R PMT=formula *Position, Modulo Target (see page 661)*
 R PRA *Position, Relative Actual (see page 663)*
 R PRC *Position, Relative Commanded (see page 666)*
 PRINT(...) *Print Data to Communications Port (see page 669)*
 PRINT0(...) *Print Data to Communications Port 0 (see page 673)*
 PRINT1(...) *Print Data to Communications Port 1 (see page 677)*
 PRINT8(...) *Print Data to USB Port (see page 680)*
 R PRT=formula *Position, Relative Target (see page 683)*
 PRTS(...) *Position, Relative Target, Synchronized (see page 685)*
 PRTSS(...) *Position, Relative Target, Synchronized, Supplemental (see page 688)*
 R PT=formula *Position, (Absolute) Target (see page 690)*
 PTS(...) *Position Target, Synchronized (see page 692)*
 R PTSD *Position Target, Synchronized Distance (see page 695)*
 PTSS(...) *Position Target, Synchronized Supplemental (see page 696)*
 R PTST *Position Target, Synchronized Time (see page 698)*
 R RANDOM=formula *Random Number (see page 699)*
 RCKS *Report Checksum (see page 701)*
 R RES *Resolution (see page 702)*
 RESUME *Resume Program Execution (see page 704)*
 RETURN *Return From Subroutine (see page 706)*
 RETURNI *Return Interrupt (see page 708)*
 RSP *Report Sampling Rate and Firmware Revision (see page 710)*
 RSP1 *Report Firmware Compile Date (see page 712)*
 RSP5 *Report Network Card Firmware Version (see page 713)*
 RUN *Run Program (see page 714)*
 RUN? *Halt Program Execution Until RUN Received (see page 716)*
 S (as command) *Stop Motion (see page 718)*
 SADDR# *Set Address (see page 720)*
 R SAMP *Sampling Rate (see page 722)*
 SCALEA(m,d) *Scale Acceleration Value (see page 724)*
 SCALEP(m,d) *Scale Position Value (see page 726)*
 SCALEV(m,d) *Scale Velocity Value (see page 728)*
 SDORD(...) *SDO Read (see page 730)*
 SDOWR(...) *SDO Write (see page 732)*
 SILENT *Silence Outgoing Communications on Communications Port 0 (see page 734)*
 SILENT1 *Silence Outgoing Communications on Communications Port 1 (see page 736)*
 R SIN(value) *Sine (see page 738)*
 SLD *Software Limits, Disable (see page 740)*
 SLE *Software Limits, Enable (see page 742)*
 SLEEP *Ignore Incoming Commands on Communications Port 0 (see page 744)*
 SLEEP1 *Ignore Incoming Commands on Communications Port 1 (see page 746)*

R SLM(mode) *Software Limit Mode (see page 748)*

R SLN=formula *Software Limit, Negative (see page 750)*

R SLP=formula *Software Limit, Positive (see page 752)*

SNAME("string") *Set PROFINET Station Name (see page 754)*

R SP2 *Bootloader Version (see page 755)*

R SP6 *Serial Number (see page 756)*

R SQRT(value) *Integer Square Root (see page 757)*

SRC(enc_src) *Source, Follow and/or Cam Encoder (see page 759)*

STACK *Stack Pointer Register, Clear (see page 761)*

STDOUT=formula *Set Device Output (see page 764)*

SWITCH formula *Switch, Program Flow Control (see page 766)*

R T=formula *Torque, Open-Loop Commanded (see page 769)*

TALK *Talk on Communications Port 0 (see page 771)*

TALK1 *Talk on Communications Port 1 (see page 773)*

R TAN(value) *Tangent (see page 775)*

R TEMP, TEMP(arg) *Temperature, Motor (see page 777)*

R TH=formula *Temperature, High Limit (see page 779)*

R TMR(timer,time) *Timer (see page 782)*

R TRQ *Torque, Real-Time (see page 784)*

R TS=formula *Torque Slope (see page 786)*

TSWAIT *Trajectory Synchronized Wait (see page 788)*

TWAIT(gen#) *Trajectory Wait (see page 789)*

R UIA *Motor Current (see page 791)*

R UJA *Bus Voltage (see page 793)*

UO(...)=formula *User Status Bits (see page 795)*

UP *Upload Compiled Program and Header (see page 797)*

UPLOAD *Upload Standard User Program (see page 799)*

UR(...) *User Bits, Reset (see page 801)*

US(...) *User Bits, Set (see page 803)*

R USB(arg) *USB Status Word (see page 805)*

R VA *Velocity Actual (see page 807)*

VAC(arg) *Velocity Actual (filter) Control (see page 810)*

R VC *Velocity Commanded (see page 815)*

R VL=formula *Velocity Limit (see page 818)*

VLD(variable,number) *Variable Load (see page 820)*

VST(variable,number) *Variable Save (see page 824)*

R VT=formula *Velocity Target (see page 828)*

VTS=formula *Velocity Target, Synchronized Move (see page 831)*

R W(word) *Report Specified Status Word (see page 833)*

WAIT=formula *Wait for Specified Time (see page 835)*

WAKE *Wake Communications Port 0 (see page 837)*

WAKE1 *Wake Communications Port 1 (see page 839)*

WHILE formula *While Condition Program Flow Control (see page 841)*
X *Decelerate to Stop (see page 844)*
Z *Total CPU Reset (see page 846)*
Z(word,bit) *Reset Specified Status Bit (see page 848)*
Za *Reset Overcurrent Flag (see page 850)*
Ze *Reset Position Error Flag (see page 851)*
Zh *Reset Temperature Fault (see page 852)*
Zl *Reset Historical Left Limit Flag (see page 853)*
Zls *Reset Left Software Limit Flag, Historical (see page 854)*
Zr *Reset Right Limit Flag, Historical (see page 855)*
Zrs *Reset Right Software Limit Flag, Historical (see page 856)*
Zs *Reset Command Syntax Error Flag (see page 857)*
ZS *Global Reset System State Flag (see page 858)*
Zv *Reset Velocity Limit Fault (see page 860)*
Zw *Reset Encoder Wrap Status Flag (see page 861)*

Commands Listed by Function

The section provides a functional listing of all available commands and their descriptions.

NOTE: A superscript "R" character preceding the command indicates there is a corresponding "report" version of that command.

Communications Control	955
Data Conversion	956
EEPROM (Nonvolatile Memory)	956
I/O Control	956
Math Function	957
Motion Control	957
Program Access	960
Program Execution and Flow Control	960
Reset Commands	961
System	961
Variables	962

Communications Control

- R ADDR=formula *Address (for RS-232 and RS-485) (see page 261)*
- R BAUD(channel)=formula *Set BAUD Rate (RS-232 and RS-485) (see page 303)*
- R CADDR=formula *CAN Address (see page 355)*
- R CAN, CAN(arg) *CAN Bus Status (see page 357)*
- CANCTL(function,value) *CAN Control (see page 359)*
- R CBAUD=formula *CAN Baud Rate (see page 363)*
- CCHN(type,channel) *Close Communications Channel (RS-232 or RS-485) (see page 365)*
- R CHN(channel) *Communications Error Flag (see page 367)*
- COMCTL(function,value) *Serial Communications Control (see page 370)*
- ECHO *Echo Incoming Data on Communications Port 0 (see page 403)*
- ECHO0 *Echo Incoming Data on Communications Port 0 (see page 405)*
- ECHO1 *Echo Incoming Data on Communications Port 1 (see page 406)*
- ECHO_OFF *Turn Off Echo on Communications Port 0 (see page 407)*
- ECHO_OFF0 *Turn Off Echo on Communications Port 0 (see page 408)*
- ECHO_OFF1 *Turn Off Echo on Communications Port 1 (see page 409)*
- R ETH(arg) *Get Ethernet Status and Errors (see page 455)*
- ETHCTL(function,value) *Control Industrial Ethernet Network Features (see page 456)*
- R GETCHR *Next Character from Communications Port 0 (see page 476)*
- R GETCHR1 *Next Character from Communications Port 1 (see page 478)*
- R GROUP(function,value) *Group Address Settings (see page 484)*
- IPCTL(function,"string") *Set IP Address, Subnet Mask or Gateway (see page 515)*
- R LEN *Length of Character Count in Communications Port 0 (see page 544)*
- R LEN1 *Length of Character Count in Communications Port 1 (see page 545)*
- NMT *Send NMT State (see page 626)*
- OCHN(...) *Open Channel (see page 632)*
- SADDR# *Set Address (see page 720)*
- SDORD(...) *SDO Read (see page 730)*
- SDOWR(...) *SDO Write (see page 732)*
- SILENT *Silence Outgoing Communications on Communications Port 0 (see page 734)*
- SILENT1 *Silence Outgoing Communications on Communications Port 1 (see page 736)*
- SLEEP *Ignore Incoming Commands on Communications Port 0 (see page 744)*
- SLEEP1 *Ignore Incoming Commands on Communications Port 1 (see page 746)*
- SNAME("string") *Set PROFINET Station Name (see page 754)*
- STDOUT=formula *Set Device Output (see page 764)*
- TALK *Talk on Communications Port 0 (see page 771)*
- TALK1 *Talk on Communications Port 1 (see page 773)*
- R USB(arg) *USB Status Word (see page 805)*
- WAKE *Wake Communications Port 0 (see page 837)*
- WAKE1 *Wake Communications Port 1 (see page 839)*

Data Conversion

- R ATOF(index) *ASCII to Float (see page 291)*
- R DFS(value) *Dump Float, Single (see page 393)*
- R HEX(index) *Decimal Value of a Hex String (see page 489)*
- R LFS(value) *Load Float Single (see page 547)*
- PRINT(...) *Print Data to Communications Port (see page 669)*
- PRINT0(...) *Print Data to Communications Port 0 (see page 673)*
- PRINT1(...) *Print Data to Communications Port 1 (see page 677)*
- PRINT8(...) *Print Data to USB Port (see page 680)*

EEPROM (Nonvolatile Memory)

- R EPTR=formula *EEPROM Pointer (see page 450)*
- R IDENT=formula *Set Identification Value (see page 504)*
- VLD(variable,number) *Variable Load (see page 820)*
- VST(variable,number) *Variable Save (see page 824)*

I/O Control

- Ai(enc) *Arm Index Rising Edge (see page 270)*
- Aij(enc) *Arm Index Rising Edge Then Falling Edge (see page 272)*
- Aj(enc) *Arm Index Falling Edge (see page 274)*
- Aji(enc) *Arm Index Falling Edge Then Rising Edge (see page 276)*
- EIGN(...) *Enable as Input for General-Use (see page 412)*
- EILN *Enable Input as Limit Negative (see page 415)*
- EILP *Enable Input as Limit Positive (see page 417)*
- EIRE *Enable Index Register, Encoder Capture (see page 419)*
- EIRI *Enable Index Register, Input Capture (see page 421)*
- EISM(x) *E-Configure Input as Sync Controller (see page 423)*
- ENCD(in_out) *Set Encoder Bus Port as Input or Output (see page 437)*
- EOBK(IO) *Enable Output, Brake Control (see page 445)*
- EOFT(IO) *Enable Output, Fault Indication (see page 447)*
- EOIDX(number) *Encoder, Output Index (see page 449)*
- R I(enc) *Index, Rising-Edge Position (see page 502)*
- R IN(...) *Specified Input (see page 509)*
- R INA(...) *Specified Input, Analog (see page 512)*
- R J(enc) *Index, Falling-Edge Position (see page 524)*
- R OC(...) *Output Condition (see page 630)*
- R OF(...) *Output Fault (see page 634)*
- OR(value) *Output, Reset (see page 638)*
- OS(...) *Output, Set (see page 640)*
- OUT(...)=formula *Output, Activate/Deactivate (see page 644)*

Math Function

- R ABS(value) *Absolute Value of () (see page 255)*
- R ACOS(value) *Arccosine (see page 259)*
- R ASIN(value) *Arcsine (see page 284)*
- R ATAN(value) *Arctangent (see page 289)*
- R COS(value) *Cosine (see page 372)*
- R FABS(value) *Floating-Point Absolute Value of () (see page 463)*
- R FSQRT(value) *Floating-Point Square Root (see page 469)*
- R PI *Pi Constant (see page 653)*
- R RANDOM=formula *Random Number (see page 699)*
- R SIN(value) *Sine (see page 738)*
- R SQRT(value) *Integer Square Root (see page 757)*
- R TAN(value) *Tangent (see page 775)*

Motion Control

- R AC *Acceleration Commanded (see page 256)*
- ADT=formula *Acceleration/Deceleration Target (see page 263)*
- ADTS=formula *Acceleration/Deceleration Target, Synchronized (see page 265)*
- R AMPS=formula *Amps, PWM Limit (see page 281)*
- R AT=formula *Acceleration Target (see page 286)*
- ATS=formula *Acceleration Target, Synchronized (see page 292)*
- BRKENG *Brake Engage (see page 333)*
- BRKRLS *Brake Release (see page 335)*
- BRKSRV *Brake Servo, Engage When Not Servoing (see page 337)*
- BRKTRJ *Brake Trajectory, Engage When No Active Trajectory (see page 339)*
- R CP *Cam Pointer for Cam Table (see page 374)*
- CTA(points,seglen[,location]) *Cam Table Attribute (see page 376)*
- CTE(table) *Cam Table Erase (see page 378)*
- R CTR(enc) *Counter, Encoder, Step and Direction (see page 380)*
- R CTT *Cam Table Total in EEPROM (see page 382)*
- CTW(pos[,seglen][,user]) *Cam Table Write Data Points (see page 383)*
- R DEA *Derivative Error, Actual (see page 386)*
- R DEL=formula *Derivative Error Limit (see page 390)*
- DELM(arg) *Derivative Error Limit Mode (see page 392)*
- R DT=formula *Deceleration Target (see page 396)*
- DTS=formula *Deceleration Target, Synchronized (see page 399)*
- R EA *Error Actual (see page 401)*
- ECS(counts) *Encoder Count Shift (see page 410)*
- R EL=formula *Error Limit (see page 426)*
- ENCO *Encoder Zero (Close Loop on Internal Encoder) (see page 432)*

ENC1 *Encoder Zero (Close Loop on External Encoder) (see page 433)*
 ENCCTL(function,value) *Encoder Control (see page 435)*
 F *Force Into PID Filter (see page 457)*
 R FAUSTS(x) *Returns Fault Status Word (see page 459)*
 R FD=expression *Resolution to Set Units of Position/Velocity/Acceleration (see page 461)*
 FSA(cause,action) *Fault Stop Action (see page 465)*
 R FSAD(n,m) *Set Reaction to Fault (see page 467)*
 G *Start Motion (GO) (see page 473)*
 GS *Start Synchronized Motion (GO Synchronized) (see page 487)*
 R HM_ADT=formula *Homing Accel/Decel Target (see page 491)*
 R HM_MTHD=formula *Homing Method (see page 492)*
 R HM_OSET=formula *Homing Offset (see page 496)*
 R HM_VTS=formula *Homing Velocity Target to Switch (see page 498)*
 R HM_VTZ=formula *Homing Velocity Target to Zero (see page 500)*
 R KA=formula *Constant, Acceleration Feed Forward (see page 526)*
 R KD=formula *Constant, Derivative Coefficient (see page 528)*
 R KG=formula *Constant, Gravitational Offset (see page 530)*
 R KI=formula *Constant, Integral Coefficient (see page 532)*
 R KII=formula *Current Control Loop: Integrator (see page 534)*
 R KL=formula *Constant, Integral Limit (see page 535)*
 R KP=formula *Constant, Proportional Coefficient (see page 537)*
 R KPI=formula *Current Control Loop: Proportional (see page 539)*
 R KS=formula *Constant, Velocity Filter Option (for KD) (see page 540)*
 R KV=formula *Constant, Velocity Feed Forward (see page 542)*
 MC *Mode Cam (Electronic Camming) (see page 555)*
 R MCDIV=formula *Mode Cam Divisor (see page 557)*
 MCE(arg) *Mode Cam Enable () (see page 558)*
 R MCMUL=formula *Mode Cam Multiplier (see page 560)*
 MCW(table,point) *Mode Cam Where (Start Point) (see page 562)*
 MDB *Enable TOB Feature (Commutation Mode) (see page 564)*
 MDC *Mode Current (Commutation Mode) (see page 566)*
 MDE *Mode Enhanced (Commutation Mode) (see page 568)*
 MDH *Mode Hybrid (Commutation Mode) (see page 570)*
 MDHV *Mode Hybrid Velocity (Commutation Mode) (see page 572)*
 MDS *Mode Sine (Commutation Mode) (see page 574)*
 MDT *Mode Trap (Commutation Mode) (see page 576)*
 MF0 *Mode Follow, Zero External Counter (see page 578)*
 MFA(distance[,m/s]) *Mode Follow Ascend (see page 580)*
 MFCTP(arg1,arg2) *Mode Follow Control Traverse Point (see page 583)*
 MFD(distance[,m/s]) *Mode Follow Descend (see page 585)*
 R MFDIV=formula *Mode Follow Divisor (see page 588)*
 MFH(distance[,m/s]) *Mode Follow, High Ascend/Descend Rate (see page 590)*
 R MFHTP=formula *Mode Follow, High Traverse Point (see page 592)*

MFL(distance[,m/s]) *Mode Follow, Low Ascend/Descend Rate (see page 594)*
 R MFLTP=formula *Mode Follow, Low Traverse Point (see page 596)*
 R MFMUL=formula *Mode Follow Multiplier (see page 598)*
 MFR *Mode Follow Ratio (see page 600)*
 MFSDC(distance,mode) *Mode Follow, Stall-Dwell-Continue (see page 603)*
 MFSLEW(distance[,m/s]) *Mode Follow Slew (see page 605)*
 R MH *Mode, Homing (see page 607)*
 MINV(arg) *Mode Inverse (Commutation Inverse) (see page 608)*
 R MODE *Mode Operating (see page 610)*
 MP *Mode Position (see page 613)*
 MSO *Mode Step, Zero External Counter (see page 616)*
 MSR *Mode Step Ratio (see page 618)*
 MT *Mode Torque (see page 620)*
 MTB *Mode Torque Brake (see page 622)*
 MV *Mode Velocity (see page 624)*
 O=formula, O(trj#)=formula *Origin (see page 628)*
 OFF *Off (Drive Stage Power) (see page 636)*
 OSH=formula, OSH(trj#)=formula *Origin Shift (see page 642)*
 R PA *Position, Actual (see page 646)*
 R PC, PC(axis) *Position, Commanded (see page 650)*
 PID# *Proportional-Integral-Differential Filter Rate (see page 654)*
 R PMA *Position, Modulo Actual (see page 657)*
 R PML=formula *Modulo Position Limit (see page 659)*
 R PMT=formula *Position, Modulo Target (see page 661)*
 R PRA *Position, Relative Actual (see page 663)*
 R PRC *Position, Relative Commanded (see page 666)*
 R PRT=formula *Position, Relative Target (see page 683)*
 PRTS(...) *Position, Relative Target, Synchronized (see page 685)*
 PRTSS(...) *Position, Relative Target, Synchronized, Supplemental (see page 688)*
 R PT=formula *Position, (Absolute) Target (see page 690)*
 PTS(...) *Position Target, Synchronized (see page 692)*
 R PTSD *Position Target, Synchronized Distance (see page 695)*
 PTSS(...) *Position Target, Synchronized Supplemental (see page 696)*
 R PTST *Position Target, Synchronized Time (see page 698)*
 S (as command) *Stop Motion (see page 718)*
 SCALEA(m,d) *Scale Acceleration Value (see page 724)*
 SCALEP(m,d) *Scale Position Value (see page 726)*
 SCALEV(m,d) *Scale Velocity Value (see page 728)*
 SLD *Software Limits, Disable (see page 740)*
 SLE *Software Limits, Enable (see page 742)*
 R SLM(mode) *Software Limit Mode (see page 748)*
 R SLN=formula *Software Limit, Negative (see page 750)*
 R SLP=formula *Software Limit, Positive (see page 752)*

SRC(enc_src) *Source, Follow and/or Cam Encoder (see page 759)*
 R T=formula *Torque, Open-Loop Commanded (see page 769)*
 R TRQ *Torque, Real-Time (see page 784)*
 R TS=formula *Torque Slope (see page 786)*
 R VA *Velocity Actual (see page 807)*
 VAC(arg) *Velocity Actual (filter) Control (see page 810)*
 R VC *Velocity Commanded (see page 815)*
 R VL=formula *Velocity Limit (see page 818)*
 R VT=formula *Velocity Target (see page 828)*
 VTS=formula *Velocity Target, Synchronized Move (see page 831)*
 X *Decelerate to Stop (see page 844)*

Program Access

LOAD *Download Compiled User Program to Motor (see page 548)*
 LOCKP *Lock Program (see page 551)*
 RCKS *Report Checksum (see page 701)*
 UP *Upload Compiled Program and Header (see page 797)*
 UPLOAD *Upload Standard User Program (see page 799)*

Program Execution and Flow Control

(Single Space Character) *Single Space Delimiter and String Terminator (see page 248)*
 BREAK *Break from CASE or WHILE Loop (see page 331)*
 C[number] *Command Label (see page 353)*
 CASE formula *Case Label for SWITCH Block (see page 360)*
 DEFAULT *Default Case for SWITCH Structure (see page 388)*
 DITR(int) *Disable Interrupts (see page 394)*
 EITR(int) *Enable Interrupts (see page 424)*
 ELSE *IF-Structure Command Flow Element (see page 428)*
 ELSEIF formula *IF-Structure Command Flow Element (see page 430)*
 END *End Program Code Execution (see page 439)*
 ENDIF *End IF Statement (see page 441)*
 ENDS *End SWITCH Structure (see page 443)*
 GOSUB(label) *Subroutine Call (see page 480)*
 GOTO(label) *Branch Program Flow to a Label (see page 482)*
 IF formula *Conditional Program Code Execution (see page 506)*
 ITR(Int#,StatusWord,Bit#,BitState,Label#) *Interrupt Setup (see page 517)*
 ITRD *Interrupt Disable, Global (see page 520)*
 ITRE *Enable Interrupts, Global (see page 522)*
 LOOP *Loop Back to WHILE Formula (see page 553)*
 PAUSE *Pause Program Execution (see page 648)*
 RESUME *Resume Program Execution (see page 704)*
 RETURN *Return From Subroutine (see page 706)*
 RETURNI *Return Interrupt (see page 708)*
 RUN *Run Program (see page 714)*

RUN? *Halt Program Execution Until RUN Received (see page 716)*
 STACK *Stack Pointer Register, Clear (see page 761)*
 SWITCH formula *Switch, Program Flow Control (see page 766)*
 R TMR(timer,time) *Timer (see page 782)*
 TSWAIT *Trajectory Synchronized Wait (see page 788)*
 TWAIT(gen#) *Trajectory Wait (see page 789)*
 WAIT=formula *Wait for Specified Time (see page 835)*
 WHILE formula *While Condition Program Flow Control (see page 841)*

Reset Commands

Z *Total CPU Reset (see page 846)*
 Z(word,bit) *Reset Specified Status Bit (see page 848)*
 Za *Reset Overcurrent Flag (see page 850)*
 Ze *Reset Position Error Flag (see page 851)*
 Zh *Reset Temperature Fault (see page 852)*
 Zl *Reset Historical Left Limit Flag (see page 853)*
 Zls *Reset Left Software Limit Flag, Historical (see page 854)*
 Zr *Reset Right Limit Flag, Historical (see page 855)*
 Zrs *Reset Right Software Limit Flag, Historical (see page 856)*
 Zs *Reset Command Syntax Error Flag (see page 857)*
 ZS *Global Reset System State Flag (see page 858)*
 Zv *Reset Velocity Limit Fault (see page 860)*
 Zw *Reset Encoder Wrap Status Flag (see page 861)*

System

R B(word,bit) *Status Byte (see page 297)*
 R Ba *Bit, Peak Overcurrent (see page 301)*
 R Be *Bit, Position Error Limit (see page 305)*
 R Bh *Bit, Overheat (see page 307)*
 R Bi(enc) *Bit, Index Capture, Rising (see page 309)*
 R Bj(enc) *Bit, Index Capture, Falling (see page 312)*
 R Bk *Bit, Program EEPROM Data Status (see page 315)*
 R Bl *Bit, Left Hardware Limit, Historical (see page 316)*
 R Bls *Bit, Left Software Limit, Historical (see page 318)*
 R Bm *Bit, Left Hardware Limit, Real-Time (see page 320)*
 R Bms *Bit, Left Software Limit, Real-Time (see page 322)*
 R Bo *Bit, Motor OFF (see page 324)*
 R Bp *Bit, Right Hardware Limit, Real-Time (see page 325)*
 R Bps *Bit, Right Software Limit, Real-Time (see page 327)*
 R Br *Bit, Right Hardware Limit, Historical (see page 329)*
 R Brs *Bit, Right Software Limit, Historical (see page 341)*
 R Bs *Bit, Syntax Error (see page 343)*

- R Bt *Bit, Trajectory In Progress (see page 345)*
- R Bv *Bit, Velocity Limit (see page 347)*
- R Bw *Bit, Wrapped Encoder Position (see page 349)*
- R Bx(enc) *Bit, Index Input, Real-Time (see page 351)*
- R CLK=formula *Millisecond Clock (see page 369)*
- R ERRC *Error Code, Command (see page 451)*
- R ERRW *Communication Channel of Most Recent Command Error (see page 453)*
- R FW *Firmware Version (see page 471)*
- R RES *Resolution (see page 702)*
- RSP *Report Sampling Rate and Firmware Revision (see page 710)*
- RSP1 *Report Firmware Compile Date (see page 712)*
- RSP5 *Report Network Card Firmware Version (see page 713)*
- R SAMP *Sampling Rate (see page 722)*
- R SP2 *Bootloader Version (see page 755)*
- R SP6 *Serial Number (see page 756)*
- R TEMP, TEMP(arg) *Temperature, Motor (see page 777)*
- R TH=formula *Temperature, High Limit (see page 779)*
- R UIA *Motor Current (see page 791)*
- R UJA *Bus Voltage (see page 793)*
- UO(...)=formula *User Status Bits (see page 795)*
- UR(...) *User Bits, Reset (see page 801)*
- US(...) *User Bits, Set (see page 803)*
- R W(word) *Report Specified Status Word (see page 833)*

Variables

- R a...z *32-Bit Variables (see page 249)*
- R aa...zz *32-Bit Variables (see page 249)*
- R aaa...zzz *32-Bit Variables (see page 249)*
- R ab[index]=formula *Array Byte [index] (see page 252)*
- R af[index]=formula *Array Float [index] (see page 267)*
- R al[index]=formula *Array Long [index] (see page 278)*
- R aw[index]=formula *Array Word [index] (see page 294)*

Commands for Combitronic

This section provides an alphabetical listing of all available Combitronic commands and their descriptions. Refer to each command description for supported features, units or value range differences, etc.

NOTE: A superscript "R" character preceding the command indicates there is a corresponding "report" version of that command.

- R a...z *32-Bit Variables (see page 249)*
- R aa...zz *32-Bit Variables (see page 249)*
- R aaa...zzz *32-Bit Variables (see page 249)*
- R ab[index]=formula *Array Byte [index] (see page 252)*
- R AC *Acceleration Commanded (see page 256)*
- ADT=formula *Acceleration/Deceleration Target (see page 263)*
- R af[index]=formula *Array Float [index] (see page 267)*
- Ai(enc) *Arm Index Rising Edge (see page 270)*
- Aij(enc) *Arm Index Rising Edge Then Falling Edge (see page 272)*
- Aj(enc) *Arm Index Falling Edge (see page 274)*
- Aji(enc) *Arm Index Falling Edge Then Rising Edge (see page 276)*
- R al[index]=formula *Array Long [index] (see page 278)*
- R AMPS=formula *Amps, PWM Limit (see page 281)*
- R AT=formula *Acceleration Target (see page 286)*
- R aw[index]=formula *Array Word [index] (see page 294)*
- R B(word,bit) *Status Byte (see page 297)*
- BRKENG *Brake Engage (see page 333)*
- BRKRLS *Brake Release (see page 335)*
- BRKSRV *Brake Servo, Engage When Not Servoing (see page 337)*
- BRKTRJ *Brake Trajectory, Engage When No Active Trajectory (see page 339)*
- R CLK=formula *Millisecond Clock (see page 369)*
- R CTR(enc) *Counter, Encoder, Step and Direction (see page 380)*
- R DEA *Derivative Error, Actual (see page 386)*
- R DEL=formula *Derivative Error Limit (see page 390)*
- DITR(int) *Disable Interrupts (see page 394)*
- R DT=formula *Deceleration Target (see page 396)*
- R EA *Error Actual (see page 401)*
- ECS(counts) *Encoder Count Shift (see page 410)*
- EIGN(...) *Enable as Input for General-Use (see page 412)*
- EILN *Enable Input as Limit Negative (see page 415)*
- EILP *Enable Input as Limit Positive (see page 417)*
- EITR(int) *Enable Interrupts (see page 424)*
- R EL=formula *Error Limit (see page 426)*
- ENC0 *Encoder Zero (Close Loop on Internal Encoder) (see page 432)*

ENC1 *Encoder Zero (Close Loop on External Encoder) (see page 433)*
 END *End Program Code Execution (see page 439)*
 EOBK(IO) *Enable Output, Brake Control (see page 445)*
 EOFT(IO) *Enable Output, Fault Indication (see page 447)*
 F *Force Into PID Filter (see page 457)*
 R FAUSTS(x) *Returns Fault Status Word (see page 459)*
 R FD=expression *Resolution to Set Units of Position/Velocity/Acceleration (see page 461)*
 FSA(cause,action) *Fault Stop Action (see page 465)*
 R FW *Firmware Version (see page 471)*
 R FSAD(n,m) *Set Reaction to Fault (see page 467)*
 G *Start Motion (GO) (see page 473)*
 GOSUB(label) *Subroutine Call (see page 480)*
 GOTO(label) *Branch Program Flow to a Label (see page 482)*
 R GROUP(function,value) *Group Address Settings (see page 484)*
 R HM_ADT=formula *Homing Accel/Decel Target (see page 491)*
 R HM_MTHD=formula *Homing Method (see page 492)*
 R HM_OSET=formula *Homing Offset (see page 496)*
 R HM_VTS=formula *Homing Velocity Target to Switch (see page 498)*
 R HM_VTZ=formula *Homing Velocity Target to Zero (see page 500)*
 R I(enc) *Index, Rising-Edge Position (see page 502)*
 R IDENT=formula *Set Identification Value (see page 504)*
 R IN(...) *Specified Input (see page 509)*
 R INA(...) *Specified Input, Analog (see page 512)*
 ITRD *Interrupt Disable, Global (see page 520)*
 ITRE *Enable Interrupts, Global (see page 522)*
 R J(enc) *Index, Falling-Edge Position (see page 524)*
 R KA=formula *Constant, Acceleration Feed Forward (see page 526)*
 R KD=formula *Constant, Derivative Coefficient (see page 528)*
 R KG=formula *Constant, Gravitational Offset (see page 530)*
 R KI=formula *Constant, Integral Coefficient (see page 532)*
 R KL=formula *Constant, Integral Limit (see page 535)*
 R KP=formula *Constant, Proportional Coefficient (see page 537)*
 R KS=formula *Constant, Velocity Filter Option (for KD) (see page 540)*
 R KV=formula *Constant, Velocity Feed Forward (see page 542)*
 MC *Mode Cam (Electronic Camming) (see page 555)*
 R MCDIV=formula *Mode Cam Divisor (see page 557)*
 R MCMUL=formula *Mode Cam Multiplier (see page 560)*
 MDB *Enable TOB Feature (Commutation Mode) (see page 564)*
 MDC *Mode Current (Commutation Mode) (see page 566)*
 MDE *Mode Enhanced (Commutation Mode) (see page 568)*
 MDH *Mode Hybrid (Commutation Mode) (see page 570)*

MDHV *Mode Hybrid Velocity (Commutation Mode) (see page 572)*
MDS *Mode Sine (Commutation Mode) (see page 574)*
MDT *Mode Trap (Commutation Mode) (see page 576)*
MFO *Mode Follow, Zero External Counter (see page 578)*
MFA(distance[,m/s]) *Mode Follow Ascend (see page 580)*
MFCTP(arg1,arg2) *Mode Follow Control Traverse Point (see page 583)*
MFD(distance[,m/s]) *Mode Follow Descend (see page 585)*
R MFDIV=formula *Mode Follow Divisor (see page 588)*
MFH(distance[,m/s]) *Mode Follow, High Ascend/Descend Rate (see page 590)*
R MFHTP=formula *Mode Follow, High Traverse Point (see page 592)*
MFL(distance[,m/s]) *Mode Follow, Low Ascend/Descend Rate (see page 594)*
R MFLTP=formula *Mode Follow, Low Traverse Point (see page 596)*
R MFMUL=formula *Mode Follow Multiplier (see page 598)*
MFR *Mode Follow Ratio (see page 600)*
MFSDC(distance,mode) *Mode Follow, Stall-Dwell-Continue (see page 603)*
MFSLEW(distance[,m/s]) *Mode Follow Slew (see page 605)*
R MH *Mode, Homing (see page 607)*
MINV(arg) *Mode Inverse (Commutation Inverse) (see page 608)*
R MODE *Mode Operating (see page 610)*
MP *Mode Position (see page 613)*
MSO *Mode Step, Zero External Counter (see page 616)*
MSR *Mode Step Ratio (see page 618)*
MT *Mode Torque (see page 620)*
MTB *Mode Torque Brake (see page 622)*
MV *Mode Velocity (see page 624)*
O=formula, O(trj#)=formula *Origin (see page 628)*
OFF *Off (Drive Stage Power) (see page 636)*
OR(value) *Output, Reset (see page 638)*
OS(...) *Output, Set (see page 640)*
OSH=formula, OSH(trj#)=formula *Origin Shift (see page 642)*
OUT(...)=formula *Output, Activate/Deactivate (see page 644)*
R PA *Position, Actual (see page 646)*
PAUSE *Pause Program Execution (see page 648)*
R PC, PC(axis) *Position, Commanded (see page 650)*
R PMA *Position, Modulo Actual (see page 657)*
R PML=formula *Modulo Position Limit (see page 659)*
R PMT=formula *Position, Modulo Target (see page 661)*
R PRT=formula *Position, Relative Target (see page 683)*
R PT=formula *Position, (Absolute) Target (see page 690)*
R RES *Resolution (see page 702)*
RESUME *Resume Program Execution (see page 704)*
RUN *Run Program (see page 714)*
S (as command) *Stop Motion (see page 718)*
R SAMP *Sampling Rate (see page 722)*

SCALEA(m,d) *Scale Acceleration Value (see page 724)*
SCALEP(m,d) *Scale Position Value (see page 726)*
SCALEV(m,d) *Scale Velocity Value (see page 728)*
SLD *Software Limits, Disable (see page 740)*
SLE *Software Limits, Enable (see page 742)*
R SLM(mode) *Software Limit Mode (see page 748)*
R SLN=formula *Software Limit, Negative (see page 750)*
R SLP=formula *Software Limit, Positive (see page 752)*
R SP2 *Bootloader Version (see page 755)*
R SP6 *Serial Number (see page 756)*
SRC(enc_src) *Source, Follow and/or Cam Encoder (see page 759)*
R T=formula *Torque, Open-Loop Commanded (see page 769)*
R TEMP, TEMP(arg) *Temperature, Motor (see page 777)*
R TH=formula *Temperature, High Limit (see page 779)*
R TMR(timer,time) *Timer (see page 782)*
R TRQ *Torque, Real-Time (see page 784)*
R TS=formula *Torque Slope (see page 786)*
R UIA *Motor Current (see page 791)*
R UJA *Bus Voltage (see page 793)*
UO(...)=formula *User Status Bits (see page 795)*
UR(...) *User Bits, Reset (see page 801)*
US(...) *User Bits, Set (see page 803)*
R VA *Velocity Actual (see page 807)*
R VC *Velocity Commanded (see page 815)*
R VL=formula *Velocity Limit (see page 818)*
R VT=formula *Velocity Target (see page 828)*
R W(word) *Report Specified Status Word (see page 833)*
X *Decelerate to Stop (see page 844)*
Z *Total CPU Reset (see page 846)*
Z(word,bit) *Reset Specified Status Bit (see page 848)*
ZS *Global Reset System State Flag (see page 858)*

Commands for DS2020 Combitronic

The section provides an alphabetical listing of all available DS2020 Combitronic system commands and their descriptions. However, not all of the commands shown are Combitronic-supported commands (i.e., not all of the listed commands support Combitronic addressing/syntax). Refer to each command description for supported features, units or value range differences, etc.

NOTE: On the command description page: if the full command is supported, the DS2020 Combitronic system support will be noted in the APPLICATION row of the table; if only the report form of the command is supported, the DS2020 Combitronic system support will be noted in the READ/REPORT row of the table.

NOTE: A superscript "R" character preceding the command indicates there is a corresponding "report" version of that command. However, refer to the previous NOTE regarding support specific to the DS2020 Combitronic system.

- R ADDR=formula *Address (for RS-232 and RS-485) (see page 261)*
- ADT=formula *Acceleration/Deceleration Target (see page 263)*
- Ai(enc) *Arm Index Rising Edge (see page 270)*
- R AMPS=formula *Amps, PWM Limit (see page 281)*
- R AT=formula *Acceleration Target (see page 286)*
- R B(word,bit) *Status Byte (see page 297)*
- R BAUD(channel)=formula *Set BAUD Rate (RS-232 and RS-485) (see page 303)*
- R Be *Bit, Position Error Limit (see page 305)*
- R Bi(enc) *Bit, Index Capture, Rising (see page 309)*
- R CADDR=formula *CAN Address (see page 355)*
- R CBAUD=formula *CAN Baud Rate (see page 363)*
- R DT=formula *Deceleration Target (see page 396)*
- R EA *Error Actual (see page 401)*
- ECHO *Echo Incoming Data on Communications Port 0 (see page 403)*
- ECHO_OFF *Turn Off Echo on Communications Port 0 (see page 407)*
- EIGN(...) *Enable as Input for General-Use (see page 412)*
- EILN *Enable Input as Limit Negative (see page 415)*
- EILP *Enable Input as Limit Positive (see page 417)*
- EISM(x) *E-Configure Input as Sync Controller (see page 423)*
- R EL=formula *Error Limit (see page 426)*
- F *Force Into PID Filter (see page 457)*
- R FAUSTS(x) *Returns Fault Status Word (see page 459)*
- R FD=expression *Resolution to Set Units of Position/Velocity/Acceleration (see page 461)*
- R FSAD(n,m) *Set Reaction to Fault (see page 467)*
- G *Start Motion (GO) (see page 473)*
- R I(enc) *Index, Rising-Edge Position (see page 502)*
- R IN(...) *Specified Input (see page 509)*
- R KD=formula *Constant, Derivative Coefficient (see page 528)*
- R KI=formula *Constant, Integral Coefficient (see page 532)*

R KP=formula *Constant, Proportional Coefficient (see page 537)*
R KV=formula *Constant, Velocity Feed Forward (see page 542)*
R MODE *Mode Operating (see page 610)*
MP *Mode Position (see page 613)*
MV *Mode Velocity (see page 624)*
O=formula, O(trj#)=formula *Origin (see page 628)*
OCHN(...) *Open Channel (see page 632)*
OFF *Off (Drive Stage Power) (see page 636)*
OUT(...)=formula *Output, Activate/Deactivate (see page 644)*
R PA *Position, Actual (see page 646)*
R PT=formula *Position, (Absolute) Target (see page 690)*
R RES *Resolution (see page 702)*
RSP *Report Sampling Rate and Firmware Revision (see page 710)*
S (as command) *Stop Motion (see page 718)*
SADDR# *Set Address (see page 720)*
SLEEP *Ignore Incoming Commands on Communications Port 0 (see page 744)*
R TEMP, TEMP(arg) *Temperature, Motor (see page 777)*
R TH=formula *Temperature, High Limit (see page 779)*
R UIA *Motor Current (see page 791)*
R UJA *Bus Voltage (see page 793)*
R VA *Velocity Actual (see page 807)*
R VT=formula *Velocity Target (see page 828)*
R W(word) *Report Specified Status Word (see page 833)*
WAKE *Wake Communications Port 0 (see page 837)*
X *Decelerate to Stop (see page 844)*
Z *Total CPU Reset (see page 846)*
ZS *Global Reset System State Flag (see page 858)*

TAKE A CLOSER LOOK

Moog Animatics, a sub-brand of Moog Inc. since 2011, is a global leader in integrated automation solutions. With over 30 years of experience in the motion control industry, the company has U.S. operations and international offices in Germany and Japan as well as a network of Automation Solution Providers worldwide.

Americas - West
Moog Animatics
2581 Leghorn Street
Mountain View, CA 94043
United States

Americas - East
Moog Animatics
1995 NC Hwy 141
Murphy, NC 28906
United States

Europe
Moog GmbH
Memmingen Branch
Allgaeustr. 8a
87766 Memmingerberg
Germany

Asia
Moog Animatics
Kichijoji Nagatani City Plaza 405
1-20-1, Kichijojihoncho
Musashino-city, Tokyo 180-0004
Japan

Tel: +1 650-960-4215
Email: animatics_sales@moog.com

Tel: +49 8331 98 480-0
Email: info.mm@moog.com

Tel: +81 (0)422 201251
Email: mcg.japan@moog.com

For Animatics product information, visit www.animatics.com

For more information or to find the office nearest you, email animatics_sales@moog.com

Moog is a registered trademark of Moog Inc. and its subsidiaries.
All trademarks as indicated herein are the property of Moog Inc. and its subsidiaries.
©2001-2022 Moog Inc. All rights reserved. All changes are reserved.

Moog Animatics SmartMotor™ Developer's Guide,
Rev. R, July 2022, PN: SC80100003-002